

Vemund H. Vestreng

# Automatic identification of cost drivers from 3D part models of sheet metal components

Master's thesis in Mechanical engineering

Supervisor: Andrei Lobov

January 2022



Vemund H. Vestreng

# **Automatic identification of cost drivers from 3D part models of sheet metal components**

Master's thesis in Mechanical engineering  
Supervisor: Andrei Lobov  
January 2022

Norwegian University of Science and Technology  
Faculty of Engineering  
Department of Mechanical and Industrial Engineering



## Abstract

To accommodate the increasing requirements for agility in the product development process, automated product assessments are becoming ever more important. Such assessments lay the foundation to enable quick and automatic cost evaluations, which are necessary to monitor and compare different product concepts and versions.

In the automotive industry, multiple complex electrical and mechanical systems need to be assembled into one. In turn, this generates multiple product revisions that need to be evaluated in terms of costs. These products typically contain sheet metal components, making the assessment of these particularly important. However, such assessments entail manual processes performed by several experts who employ multiple different software.

In this work, an architecture for a system to automatically identify key production cost drivers from 3D part models of sheet metal components is first proposed. The system architecture is designed based on evaluations of identified cost drivers in terms of their dependencies and possible methods to automatically identify them, outlining the flow of information through various evaluation modules. The modular system relies primarily on learning-based methods, employing a modified multi-view convolutional neural network architecture that accounts for spatial information. In turn, this enables flexibility and adaptability compared to traditional rule-based product assessments.

The system modules are developed with an iterative approach, testing and validating different techniques and concepts whilst building towards a proof-of-concept for the employed methods. The successfully developed modules are finally assembled into a prototype system that automatically identifies key cost drivers from a CAD native representation of a sheet metal component. The developed prototype does not require expert handling and demonstrates the functionality of the proposed system for automatic product assessments targeting cost estimation.

## Sammendrag

For å imøtekomme de økende kravene til smidighet i produktutviklingsprosessen, er automatiserte produktevalueringer viktigere enn noen gang. Slike evalueringer legger grunnlaget for å kunne utføre raske og automatiserte kostnadsestimater, noe som er nødvendig for å sammenligne ulike produktkonsepter og versjoner.

I bilindustrien må flere komplekse elektriske og mekaniske systemer settes sammen til ett. Dette fører igjen til utallige produktrevisjoner, som alle må evalueres med hensyn til kostnader. Disse produktene består normalt sett av platekomponenter, noe som gjør analysen av slike komponenter spesielt viktig. Disse analysene medfører imidlertid manuelle prosesser utført av flere eksperter som tar i bruk ulike programvarer.

I dette arbeidet foreslås først en arkitektur for et system rettet mot automatisert analyse av platekomponenter for å identifisere kostdrivere. Systemarkitekturen er basert på evalueringer av identifiserte kostdrivere med utgangspunkt i deres avhengigheter og mulige måter å automatisk identifisere dem på gjennom analyse av 3D-modeller. Det foreslåtte modulære systemet baserer seg primært sett på læringbaserte metoder, noe som tillater fleksibilitet og endringsdyktighet sammenlignet med mer tradisjonelle regelbaserte produktevalueringer.

De ulike systemmodulene utvikles med en iterativ tilnærming, gjennom å teste og validere ulike teknikker og konsepter på veien mot å oppnå et resultat som tilstrekkelig beviser dyktigheten til de anvendte metodene. De utviklede modulene settes til slutt sammen til en prototype som automatisk identifiserer viktige kostdrivere fra en 3D-modell av en platekomponent. Den utviklede prototypen krever ingen spesialkompetanse og demonstrerer funksjonaliteten til det foreslåtte systemet for automatisert produktevaluering rettet mot kostnadsestimering.

## Preface

I would like to thank my supervisor, Andrei Lobov, as well as my colleagues in the BMW Group, Felix Schumacher, Maximilian Jüngling, and Octavian Knoll, for guidance, advice, and constructive discussions throughout this work.

Thank you, Maya, for your unconditional support and encouragement.

# Contents

<b>List of figures</b>	<b>i</b>
<b>List of tables</b>	<b>v</b>
<b>List of abbreviations</b>	<b>vi</b>
<b>Terms</b>	<b>viii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Problem description . . . . .	2
1.2 Limitations . . . . .	3
1.3 Structure . . . . .	4
1.3.1 Reading guide . . . . .	6
<b>2 Background</b>	<b>9</b>
2.1 Sheet metal components . . . . .	9
2.2 Cost estimation . . . . .	11
2.2.1 Automatic cost estimation . . . . .	11
2.3 Geometrical analysis . . . . .	12
2.3.1 Rule-based analysis . . . . .	12
2.3.2 Learning-based analysis . . . . .	14
2.4 Problem verification . . . . .	16
<b>3 Methods</b>	<b>18</b>
3.1 The STL format and its analysis . . . . .	18
3.2 Artificial neural networks . . . . .	19
3.3 Convolutional neural networks . . . . .	31
3.3.1 CNN architectures . . . . .	38
3.3.2 Representation of 3D objects . . . . .	42
<b>4 Approach</b>	<b>49</b>
4.1 Outline . . . . .	49
4.2 Cost drivers . . . . .	50
4.2.1 Welding information . . . . .	51



4.2.2	Blank size . . . . .	52
4.2.3	Part mass . . . . .	53
4.2.4	Material type . . . . .	54
4.2.5	Sheet thickness . . . . .	54
4.2.6	Part size . . . . .	55
4.2.7	Fastener information . . . . .	56
4.2.8	Production method . . . . .	58
4.2.9	Number of parts per stroke . . . . .	60
4.2.10	Production volume . . . . .	61
4.2.11	Place of production . . . . .	61
4.2.12	Number of operations . . . . .	61
4.2.13	Process plan . . . . .	62
4.2.14	Part surface area . . . . .	66
4.2.15	Cost driver summary . . . . .	66
4.3	Method choices and development . . . . .	67
4.3.1	Choice of methods . . . . .	67
4.3.2	Learning-based method development . . . . .	71
<b>5</b>	<b>Implementation</b>	<b>76</b>
5.1	System architecture . . . . .	76
5.1.1	Dependencies . . . . .	76
5.1.2	Proposed system architecture . . . . .	79
5.2	Data acquisition and preparation . . . . .	84
5.2.1	Data acquisition . . . . .	84
5.2.2	Data preparation . . . . .	86
5.3	Development of each module . . . . .	91
5.3.1	Rule-based and API modules . . . . .	91
5.3.2	Learning-based modules . . . . .	97
<b>6</b>	<b>Results and discussion</b>	<b>116</b>
6.1	Classification #2 - Fastener ID . . . . .	116
6.1.1	Data . . . . .	116
6.1.2	Testing approach . . . . .	120

6.1.3	Results and discussion . . . . .	121
6.1.4	Summary and future directions . . . . .	131
6.2	Classification #3 - Production method . . . . .	132
6.2.1	Data . . . . .	132
6.2.2	Testing approach . . . . .	137
6.2.3	Results and discussion . . . . .	139
6.2.4	Summary and future directions . . . . .	146
6.3	Classification #1 - Part type . . . . .	147
6.3.1	Data . . . . .	147
6.3.2	Testing approach . . . . .	150
6.3.3	Results and discussion . . . . .	151
6.3.4	Summary and future directions . . . . .	155
6.4	Classification #4 - Number of operations . . . . .	155
6.4.1	Data . . . . .	155
6.4.2	Testing approach . . . . .	158
6.4.3	Results and discussion . . . . .	160
6.4.4	Summary and future directions . . . . .	164
6.5	Regression #2 - Blank size . . . . .	164
6.5.1	Data . . . . .	165
6.5.2	Testing approach . . . . .	167
6.5.3	Results and discussion . . . . .	170
6.5.4	Summary and future directions . . . . .	175
6.6	Regression #1 - Welding points . . . . .	176
6.6.1	Data . . . . .	176
6.6.2	Testing approach . . . . .	179
6.6.3	Results and discussion . . . . .	182
6.6.4	Summary and future directions . . . . .	190
6.7	Classification #5 - Parts per stroke . . . . .	192
6.7.1	Data . . . . .	192
6.7.2	Testing approach . . . . .	194
6.7.3	Results and discussion . . . . .	195
6.7.4	Summary and future directions . . . . .	197

6.8	Prototype . . . . .	197
6.8.1	Case study . . . . .	200
6.9	Summary . . . . .	203
<b>7</b>	<b>Conclusion and future work</b>	<b>208</b>
<b>8</b>	<b>References</b>	<b>210</b>

## List of figures

1	A proposed reading guide for this thesis. . . . .	7
2	Example of a sheet metal component labeled as a small assembly. . . . .	10
3	The same part in the native CAD format (left) and the STL format (right). . . . .	18
4	An illustration of a fully-connected neural network. . . . .	20
5	The ReLU activation function. . . . .	22
6	Illustration of a network without dropout (top) and with dropout (bottom) applied. . . . .	31
7	Illustration of how a binary black and white photo looks to a human (left) and a computer (right). . . . .	33
8	Rough outline of a typical CNN architecture. . . . .	33
9	Illustration of 2D convolution operation from [53]. . . . .	34
10	Illustration of average and max pooling with a pooling filter of size 2x2 using a stride of 2. . . . .	36
11	The VGG-16 and VGG-19 architecture variations, as presented in [54]. . . . .	39
12	The ResNet architecture with descriptions of the number of block repetitions that are included in the ResNet-50 and the ResNet-152. As presented in [56]. . . . .	40
13	Inception module, as presented in [58]. . . . .	42
14	Illustration of a multi-view representation of a cube. . . . .	43
15	The late fully-connected (top) and score (bottom) multi-view fusion techniques. . . . .	45
16	Illustration of 2D bounding boxes. . . . .	55
17	Illustration of fasteners. . . . .	56
18	Traditional CNN architecture for image classification with the convolutional and prediction blocks indicated. . . . .	72
19	The spatial CNN architecture used in this work. . . . .	73
20	The spatial score multi-view CNN architecture used in this work. . . . .	74
21	The spatial late fully-connected multi-view CNN architecture used in this work. . . . .	75
22	N-squared diagram outlining the dependencies of each cost driver. . . . .	77
23	The proposed architecture for an automated system to identify cost drivers from 3D representations of sheet metal components. . . . .	80
24	Illustration of the different views employed in this work. . . . .	87
25	Illustration of different augmentation techniques in 2D. . . . .	90

26	An irregular tetrahedron (left) and the matrix equation to calculate the volume (right).	93
27	Benchmark comparison of different convolutional neural network architectures on the ImageNet-1k validation dataset from [93]. The ball size is the <i>model complexity</i> , which is a measurement relating to the number of trainable parameters in the architecture. . . . .	102
28	Examples of fasteners and their IDs. . . . .	117
29	Spatial distribution of fasteners. . . . .	118
30	Example of augmented fastener. . . . .	119
31	The best models with each architecture in Classification #2 - Stage 1. . . . .	122
32	The initial results when changing the target from fastener type to ID in Classification #2. . . . .	123
33	The impact of an augmented dataset on the validation accuracy when training the standard VGG-16 to target the fastener ID in Classification #2. . . . .	124
34	The validation accuracies of the standard and spatial VGG-16 models trained on the augmented dataset in Classification #2 - Stage 2. . . . .	125
35	Examples of errors made by the spatial model tested in Classification #2 - Stage 2. . . . .	127
36	Results for the late fully-connected multi-view fusion technique in Classification #2 - Stage 3. . . . .	129
37	Fine-tuning of the spatial single-view model in Classification #2 - Stage 4. . . . .	130
38	Examples of sheet metal parts manufactured with the different production methods distinguished in this work. . . . .	133
39	Spatial distribution of parts manufactured with the different production methods. . . . .	135
40	Number of parts in the dataset for each production method in percent. . . . .	136
41	Number of image examples for each production method in the base dataset. . . . .	137
42	Validation results for the best model on the original problem in Classification #3. . . . .	140
43	Initial validation results on original problem in Classification #3. To the left are the results from training on the base dataset, to the right from training on the augmented dataset. . . . .	141
44	The change in system architecture following the problem redefinition in Classification #3. . . . .	143

45	Initial validation results after problem redefinition in Classification #3. On the top row are the results for the standard VGG-16 and ResNet-50 configurations, on the bottom row are the results for the spatial models. . . . .	144
46	The best validation results obtained upon fine-tuning of the spatial VGG and ResNet on the redefined problem in Classification #3. . . . .	145
47	The test results of the final model developed in Classification #3. The characteristics of this model are presented in Table 7 . . . . .	147
48	Examples of each class in Classification #1. . . . .	148
49	Spatial distribution of fasteners and sheet metal parts. . . . .	148
50	Number of parts for each class in the base dataset of Classification #1. . . . .	149
51	Best validation results achieved with the different imbalance techniques in Classification #1 - Stage 1. . . . .	152
52	Validation results of the two strongest models developed in Classification #1. . . . .	154
53	The errors made by the regular fully-connected and weighted product-score fusion multi-view models in Classification #1. . . . .	154
54	Examples of Press line parts manufactured with the different numbers of operations. . . . .	156
55	Spatial distribution of parts manufactured with different number of operations. . . . .	158
56	The distribution of image examples in each set in Classification #4. . . . .	159
57	Performance of the best model in Classification #4 on the validation data (left) and test data (right). . . . .	161
58	Distribution of blank areas by part area (top) and global bounding box diagonal (bottom) in Regression #2. . . . .	166
59	The results of the regular (left) and spatial (right) models after the initial rough tuning in Regression #2. . . . .	171
60	The two parts with the largest blank sizes in the dataset in Regression #2. . . . .	172
61	The validation results when training on the 16x augmented dataset (left) and subsequently fine-tuning the model on the same augmented dataset (right) in Regression #2. . . . .	173
62	The best validation results for the different fusion techniques in Regression #2. . . . .	173
63	The validation and test results on the ProgDie parts in Regression #2. . . . .	174
64	Distribution of the number of welding points in the full dataset in Regression #1. . . . .	177
65	Distribution in percent in each set in Regression #1. . . . .	178

66	Illustration of the network to be developed in Regression #1. . . . .	179
67	Illustration of the final network developed for Regression #1. . . . .	183
68	Validation results for the top 10 configurations after the optimization loops in Regression #1. . . . .	184
69	Validation results for the final network developed for Regression #1. . . . .	186
70	The normal distribution with a mean of 0 and a variance of 1. . . . .	188
71	Examples of the distorted datasets used for the sensitivity analysis in Regression #1.	189
72	Results from the sensitivity analysis in Regression #1. . . . .	191
73	Distribution of the number of parts per stroke for the Press parts (left) and the ProgDie parts (right) in Classification #5. . . . .	193
74	Distribution of the number of parts per stroke in the Press part sets (left) and ProgDie sets (right) in Classification #5. . . . .	193
75	Confusion matrices of the final networks developed for Classification #5. . . . .	196
76	Prototype system. . . . .	198
77	The part that is included as a case study. . . . .	201
78	The actual and predicted fastener in case study. . . . .	203

## List of tables

1	Overview of all identified cost drivers and their area of importance. . . . .	50
2	Overview of methods to identify the various cost drivers. . . . .	66
3	Overview of which methods that are pursued for automatic identification of each cost driver in this work. . . . .	68
4	Overview of CNN architectures used in previous relevant works. . . . .	101
5	The final configuration of the network in Classification #2. . . . .	122
6	Comparison of the results for different multi-view fusion techniques in Classification #2. . . . .	128
7	The final configuration of the network in Classification #3. . . . .	139
8	The final configuration of the network in Classification #1. . . . .	152
9	The final configuration of the network in Classification #4. . . . .	160
10	The results from the linear regressions in Regression #2. . . . .	167
11	The final configuration of the networks in Regression #2. . . . .	170
12	The final configuration of the network in Regression #1. . . . .	182
13	The best performing configurations from the optimization loop in Regression #1. . .	184
14	The final configuration of the networks in Classification #5. . . . .	195
15	Case study results. . . . .	202
16	Summary of the developed learning-based modules. . . . .	204



## List of abbreviations

Below are declarations of abbreviations used in the thesis.

**AAG:** Attributed adjacency graph

**ACAPP:** Automated computer-aided process planning

**AFR:** Automatic feature recognition

**ANN:** Artificial neural network

**API:** Application programming interface

**BB:** Bounding box

**CA:** Contact area

**CAPP:** Computer-aided process planning

**CBR:** Case-based reasoning

**CNN:** Convolutional neural network

**FC:** Fully-connected

**FEA:** Finite element analysis

**FEM:** Finite element method

**KBE:** Knowledge-based engineering

**LCS:** Largest common subgraph

**LR:** Learning rate

**MAE:** Mean absolute error

**MAPE:** Mean absolute percentage error

**MDG:** Model dependency graph

**MSE:** Mean squared error

**N.A.:** Not applicable

**NSD:** N-squared diagram or  $N^2$  diagram

**PPS:** Parts per stroke

**RQ:** Research question

**SGD:** Stochastic gradient descent

**STL:** STereoLitography or standard tessellation language or standard triangle language

**WP:** Welding point

## Terms

Below is a list of definitions of terms used throughout the thesis.

**Cost driver:** A cost driver is a component entity that is of importance to the cost. Examples include the mass and production method. See more in Section 4.2.

**Geometrical information:** Geometrical information is used in this work to describe the information relating to the shapes in a part.

**Individual accuracy:** A term used to describe the accuracy on one specific class in a classification problem. This is sometimes also referred to as the *precision* in other works.

**Learning-based method:** Any algorithm that tunes its own parameters through a learning process. See Section 2.3.2.

**Precision:** See definition of *individual accuracy*.

**Production costs:** Costs related to the purchase of raw material and operation of machines used during manufacturing. Described in the first paragraph of Section 2.2.

**Rule-based method:** Any algorithm that employs predefined criteria to solve a task. See Section 2.3.1.

**Small assembly:** A sheet metal component consisting of a smaller number of individual sheet metal parts, potentially including fasteners. Described in the first paragraph in Section 2.1.

**Spatial information:** Spatial information is used in this work to describe the information relating to the size of a part.

**Tooling costs:** Costs related to the purchase and manufacturing of the tools required in the production process. Described in the first paragraph of Section 2.2.

## Neural network terms

Neural network terms are highlighted in bold where they are explained in Sections 3.2 (general terms) and 3.3 (convolutional neural network terms).

# 1 Introduction

To satisfy and react to an ever more complex and unpredictable market, modern product development processes need to be agile. Such agile processes demand a high degree of efficiency, adaptability, and flexibility. However, the products to be developed also keep becoming more complex, combining a wide range of different technologies that need to be assembled into a finished product. These technology interfaces generate the need to align several different system requirements, which in turn results in multiple product revisions. In addition, the visions of multiple product stakeholders, such as investors, management, sales, marketing, engineering, and production, need to be aligned.

In order to accommodate the visions of all stakeholders, meet the various system demands, and react to market changes, multiple product concepts and versions are required along the way from an idea to a finished product that can be delivered to the customer. Each of these concepts and versions must be assessed in terms of cost, to evaluate if the design is feasible or not within the set budget. However, such assessments are often time-consuming and entail manual product evaluations by several experts before a cost estimate is provided.

The time-consuming process of manually assessing a product in order to derive a cost estimate is not consistent with the requirements of modern product development. The ability to react and adapt is lost whilst waiting for products to be assessed, which in turn can lead to lost revenue. To ensure a more efficient and flexible development process, automated product assessments are becoming increasingly important.

System dependencies and differing visions become very prominent in the automotive industry, where multiple complex electrical and mechanical systems need to be assembled into one. The market is additionally in ever more change, demanding greener alternatives without losing the comfort or performance of traditional vehicles. In order to deliver the desired products in time to satisfy the customer and beat competitors, the development process needs to be efficient and flexible. In the automotive industry, especially the assessment of sheet metal components is of importance. However, an easily adaptable and completely automated system for assessing sheet metal components in terms of cost driving attributes is not readily available.

## 1.1 Problem description

To evaluate and compare different product concepts and versions in the automotive industry, cost estimation of sheet metal components must be performed. To fit into the modern product development process, such an assessment should be efficient and adaptable. In turn, this implies that the cost estimation should be performed in an automated manner by a flexible system.

To perform automated cost estimation, previous works have employed both direct and 2-step approaches. In the direct approach, low-level product information, such as geometry, is directly linked to cost using learning-based methods. This means that a cost evaluation can be performed directly on a 3D model of a component, potentially with some supplementary data. In the 2-step approach, on the other hand, this low-level data is first evaluated with the purpose to determine certain higher-level entities of relevance to the cost, namely the *cost drivers*. These cost drivers are subsequently evaluated to finally determine the costs.

To train the learning-based models required by the direct approach, large cost databases are necessary that link the low-level product data together with cost data. Generating such databases is a time-consuming process that often needs to be manually carried out by experts. No such databases are readily available to this work, and the generation of such a database is not feasible within the scope of this work. As a result, the direct approach is not considered a viable option.

For the 2-step approach, both rule- and learning-based methods can be employed in both steps, meaning there are four main variations. However, performing the cost driver evaluation with a learning-based method suffers from the same limitation as the direct approach, namely the requirement that a cost database exists. Although such a database likely can be smaller in this case due to the abstraction of the low-level product data into higher-level data (cost drivers), it is still not feasible within the scope of this work to generate a sufficient cost database. As a result, the rule-based, or analytical, approach to cost driver evaluation is the remaining option. Such an analytical cost estimation model, which can be automated if its inputs are known, is readily available and used in practice at the company where this work is carried out. The inputs to the model are, however, found by manual labor performed by several experts that employ multiple software. The problem at hand can hence be reduced to automatically identifying the inputs required by the existing analytical cost estimation tool.

To fully automate a component evaluation, the analysis needs to begin with a component representation that is readily available. This work is carried out at a leading German automotive manufacturer where that representation is the 3D CAD part model of the component. Additionally, to ensure flexibility and rigidity, all analysis, where possible, should be performed on a purely geometrical and spatial basis. That is, the analysis should not rely on existing design documentation, but rather be performed directly on the base information in the 3D CAD model. Although the intended target users of the system are development engineers who need to perform a quick cost evaluation of their developed product, it is assumed that the users have no expert knowledge to contribute to the analysis. The analysis should hence be performed completely automatically without the need for any expert handling.

Rule-based methods for geometrical analysis have been frequently employed in previous works. However, these are often found to be error-prone because it is difficult, if not impossible, to find simple general rules in some cases. Additionally, rule-based methods are often inflexible since a set of rules often apply specifically to a certain use case. This means that new rules, which can be difficult to determine, could need to be developed from scratch in cases where the problem only slightly changes. As a result, state-of-the-art works on geometrical analysis have started to explore learning-based methods to evaluate components. These approaches are, however, often not focused on sheet metal components, and they do not cover all necessary inputs to the existing analytical cost estimation tool. Additionally, individual approaches to identify specific cost drivers need to be assembled into a complete automated system that accepts a 3D CAD part model of a sheet metal component as an input and outputs the relevant cost drivers.

The above discussion on the necessity of automated cost estimation and the limitations of previous works poses the research question:

**RQ:** How to automatically identify cost drivers from a 3D CAD part model of a sheet metal component?

## 1.2 Limitations

Following are some limitations that are imposed on the presented work.

First of all, it should be noted that this work is on the subject of geometrical analysis. Although it is aimed towards cost estimation, the targets of this work are cost drivers (see Section 4.2),

not an actual cost estimate. That is, the developed system aims only to automatically evaluate a component to identify its cost drivers, not to evaluate these in terms of cost. Some theoretical background and discussions relating to cost estimation are, however, included.

Secondly, it should be noted that the system proposed and developed in this work targets only production costs, not tooling costs. The reason for this choice is given in Section 4.3.1. Note, however, that the tooling cost drivers still are outlined, evaluated, and discussed in Section 4.2.

This work also limits itself to aim for the development of a proof-of-concept system rather than a finalized and readily implementable system. This is because it is not feasible within the scope of this work to optimize all aspects of the proposed system, and that it is considered to be a greater contribution to evaluate multiple system modules to proof-of-concept level performance than to only optimize a select few. This limitation is particularly relevant for learning-based system modules, where optimization tuning can be performed at length to further increase performance.

Only individual sheet metal components or small assemblies, as described in Section 2.1, are considered in this work. This entails the exclusion of larger assemblies where multiple small assemblies or individual components are joined in assembly lines. This limitation is imposed both because the cost considerations related to such assembly processes differ from those of the smaller assemblies and individual components, which do not require assembly lines, and to limit the scope of the work.

The system presented in this work is developed specifically for the company where the work is carried out. Although it is believed that most aspects are transferable to other applications, it should be noted that deviations can occur.

### **1.3 Structure**

The remainder of the thesis is organized as follows. First, some background on sheet metal parts, cost estimation, and geometrical analysis is presented. Subsequently, the presented previous works are briefly discussed in light of the problem description to verify that the research question holds in light of the presented state-of-the-art works. Together, the problem description in Section 1.1, background in Section 2, and the problem verification in Section 2.4 are intended as an introduction to the problem at hand and a work justification.

After presenting some background, the most relevant methods employed in this work are presented in Section 3. Learning-based methods are a central topic in the work, and the primary part of the Methods chapter is therefore devoted to these. The chapter is written with the assumption that the reader does not possess extensive previous knowledge on the topic of learning-based methods, specifically artificial neural networks and convolutional neural networks. The reason for this is that a fundamental understanding of learning-based methods is considered important to understand the argumentation and discussions in this work. The chapter is hence written as an attempt to lay such a foundation within the work itself rather than briefly mention certain topics and reference books where the reader is expected to perform literary studies before proceeding. Readers with previous expertise on the subject may find some parts redundant and can, in that case, use the section either to briefly refresh their knowledge on some topics or entirely skip it and rather return to look up key terms and specific information if the technical discussions require this. The Background chapter and Methods chapter are together intended to represent an introduction into the state of the art on the topic of this work.

Following the Methods chapter, an outline of the approach employed to solve the problem in this work is presented in Section 4.1. After this, a detailed introduction and evaluation of the various cost drivers, which are the target entities to identify in this work, is provided in Section 4.2. This section also briefly discusses some state-of-the-art works that are relevant to the different specific cost drivers. Several of these works are also mentioned in Section 2 but are here discussed either with some more detail or a different focus. Other works are presented for the first time in the section. The reason for not including all discussion of state-of-the-art works in Section 2 is to avoid confusion for the reader and to enable more continuous reading without the need to go back and forth between sections. Once the cost drivers have been established and evaluated, the specific methods to pursue for identification of the various cost drivers are defined and developed in Section 4.3.

After the methods to pursue are defined, the implementation of the presented approach is discussed in Section 5. First, a theorized system that enables automatic identification of relevant cost drivers from a 3D CAD representation of a sheet metal component is proposed in Section 5.1. Subsequently, the work performed to acquire and prepare data to develop the modules of the proposed system is presented in Section 5.2, before the development of the modules themselves is discussed in Section 5.3. For the rule-based parts of the system, their complete development is discussed in Section



5.3.1. For the learning-based modules, the discussion in Section 5.3.2 rather outlines the general testing approach that is employed in their development. The reason for this is that the development of these modules is iterative and performed through testing and validation of various techniques and concepts. Their detail development is hence more suitably presented as results in Section 6. To support the discussion of the employed testing approach, multiple previous works on the topics of artificial neural networks and convolutional neural networks are consulted and discussed in Section 5.3.2. This discussion is not included together with the state of the art in Sections 2 and 3 in order to avoid confusion for the reader and to clarify the specific sources from which the information to make the design choices is obtained.

The results from the development of the learning-based modules in the proposed system are presented in chronological order in Section 6. Relevant information on the data used in the development and the specific testing approach employed is also presented for each module. Each development is also briefly summarized independently with propositions for future works on the specific subject. In Section 6.8, an assembled prototype system of the successfully developed modules is presented together with a case study to demonstrate its functionality. Finally, the results of all developments are tabulated and briefly summarized in Section 6.9. For the reader to whom the exact technical execution is not of particular interest, it is possible to skip several parts of Section 6 and focus primarily on sections 6.8 and 6.9.

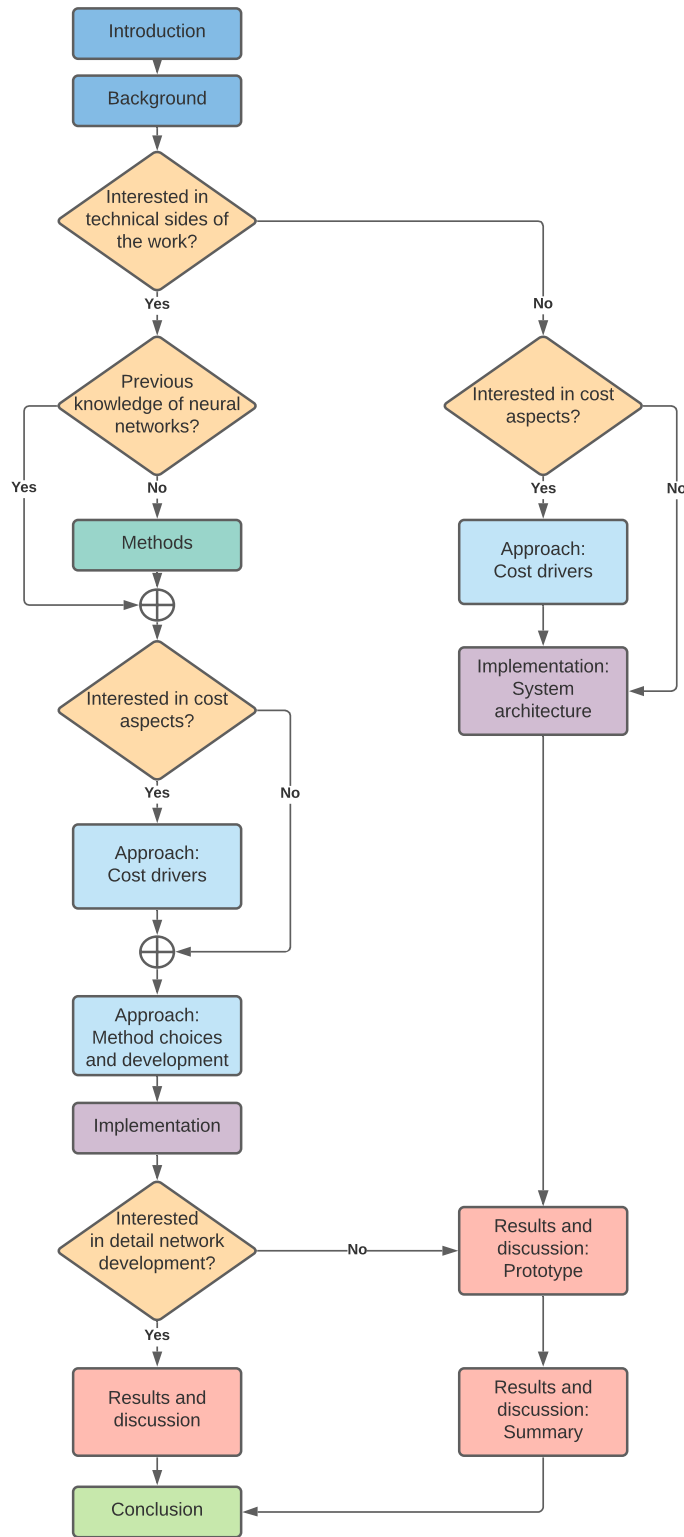
The thesis is finalized with a conclusion and propositions for future works.

### **1.3.1 Reading guide**

This is a large work, containing more than 200 pages of content. However, part of the reason for this is that the work is intended to be useful to different people with differing purposes. That is, not all sections are necessarily of importance to all readers.

Figure 1 presents a suggested reading guide for this thesis. The intention of this graph is to aid the reader in identifying which sections that may be of particular interest. The yellow shapes ask questions relating to what the reader aims to gain from the work, and different paths can subsequently be followed to the sections that are considered the most relevant for the differing applications.

There are two main paths that can be followed when reading this work. One path is for the reader



**Figure 1** – A proposed reading guide for this thesis.

to whom the technical sides of the work are of interest, and the other for the reader who is more interested in simply gaining an overview of the work. It should be noted that the presented guide merely is a suggestion and that the individual reader may find it appropriate to deviate from this. It could, for example, be useful in certain cases to return to a skipped section to gain a deeper insight into the current discussion.

## 2 Background

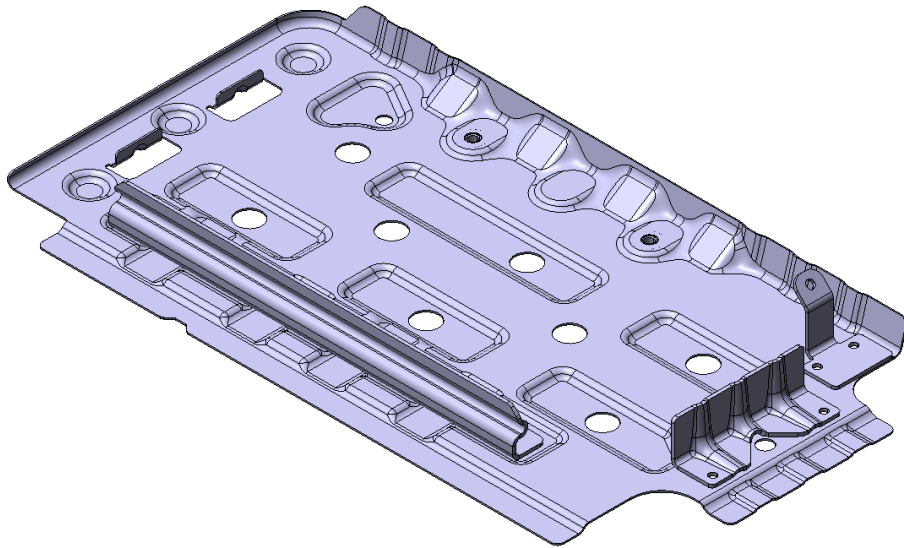
This section will provide an overview of relevant topics and state-of-the-art works on these. First, sheet metal components are introduced, before the field of cost estimation is presented together with previous works. Finally, the field of geometrical analysis, both rule- and learning-based, is discussed.

### 2.1 Sheet metal components

A sheet metal component consists of one or more parts manufactured from thin metal sheets. These are typically joined using either welding or other joining techniques, such as gluing, to create the final component. The individual sheet metal parts can additionally contain fasteners, such as bolts and nuts, that are mounted on the sheets either by welding or through mechanical force from the fastener being pressed into the sheet. In this work, only components that are labeled as either small assemblies or individual components are considered. A *small assembly* is defined as a component that consists of one primary sheet metal part, on which some additional sheet metal parts, like brackets, are mounted. An example of such a part is provided in Figure 2. Here, three smaller brackets are mounted on a main plate to create a complete component. An *individual component* consists of only a single sheet metal part. Both of these component types can additionally include fasteners.

Sheet metal parts are frequently employed in industry. The primary reasons for this are their abilities to take on complex free-form shapes and that their manufacturing bids high productivity through repetitive processes in mass production [1]. Especially the automotive industry finds sheet metal parts useful, and these components therefore constitute a significant amount of the car body [2]. Because this work is carried out at a leading German automotive manufacturer, these parts are the focus of this work.

In order to create a sheet metal component, specialized manufacturing techniques, such as punching, cutting, and bending, are performed to transform a flat metal sheet, called the blank, into 2D or 3D objects [3]. Work related to the analysis of sheet metal parts in order to specify these techniques and their parameters is often considered both a complex and experience-based task [4]. In turn, this means that a lot of effort is invested by a number of experts in order to evaluate such components in terms of cost.



**Figure 2** – Example of a sheet metal component labeled as a small assembly.

## 2.2 Cost estimation

Cost estimation is the process of predicting the final expenses related to some product. These expenses have multiple dependencies, such as machine, material, and tool specifications [5]. Many classifications and definitions relating to the costs of a part exist. In this work, the costs of a component are split into production costs and tooling costs, following the practice where the work is performed. The production costs include the material and manufacturing costs. The material costs are the expenses associated with purchasing the raw material that is required to produce a part, whilst the manufacturing costs include expenses related to operating the machines required in the production process, such as the man-hours, electricity, and maintenance that is demanded. The tooling costs are the expenses associated with the production and purchasing of the various tools that are needed to produce the part.

Cost estimation of sheet metal components is an intricate process that generally requires expert knowledge. This often results in a high time consumption, both because of the time it takes to perform the work itself and because of the waiting time related to the limited capacity of experts [2]. However, costs are of pivotal importance in the development process, and it is often necessary to perform drastic design changes following a cost estimate in order to stay within budget [6]. To limit the extent of these design changes and aim to design a cost-optimal product, it is desired to make frequent cost estimations throughout the development. However, strict deadlines make this challenging. To enable a more integrated and continuous cost focus in the development process, quick and automatic cost estimation is therefore desired.

### 2.2.1 Automatic cost estimation

To automatically estimate the costs of sheet metal parts manufactured through laser cutting, punching, and bending, [3] tests both regression models and neural networks. These models are provided certain factors of influence to the costs, such as the mass, sheet thickness, and bounding box dimensions, to finally predict the costs of the parts. [7] proposes an approach that first analyzes a sheet metal part with a combination of rule- and learning-based methods to determine cost driving attributes of the part, such as the contour length and mechanical features. This information is then translated into machining technique specific base units by a knowledge-based system to finally establish a cost estimate from the base units. [6] bases the cost estimate of sheet metal parts on some machining features, such as holes and bends, and base information relating

to these. This information is then analyzed in a rule-based manner to establish the cost estimate. In [2], a sheet metal part is first evaluated with a manual knowledge-based system to establish a base profile of the part. Case-based reasoning in which this base information is compared to that of other parts in a database is subsequently used to establish a proposed process plan, from which a cost estimate is made.

In recent work, also convolutional neural network (CNN) approaches are employed to estimate the costs. [8] estimates the manufacturing costs of basic parts by first evaluating the part geometry with a CNN to obtain geometrical information about the part. This information is then used as input to learning-based models that finally deliver a cost estimate. In [9], a more direct method is proposed. This approach removes the intermediate step that describes the part in terms of cost driving attributes by employing a CNN that takes a 3D voxel part representation as input to directly estimate the manufacturing costs of basic parts. A similar method, which also directly evaluates a voxel model to predict the cost but additionally considers material and volume data, is presented in [10].

## **2.3 Geometrical analysis**

Multiple previous works on automated cost estimation involve 2-step approaches where a component is first evaluated to determine certain entities of importance to the cost. In order to perform such component evaluations, the geometry of the part must be analyzed. There are two main approaches that are employed for geometrical analysis, namely rule- and learning-based. This section will present some relevant previous works related to each.

### **2.3.1 Rule-based analysis**

The term rule-based analysis refers in this work to any method in which a human designer defines specific criteria, or rules, to transform some input into the desired output. A simple example of such a rule could be "if volume is above  $1m^3$ , then label as large". The algorithm then does exactly what it is programmed to do, namely to label all parts with volumes above  $1m^3$  as large. The rules can be either such basic if-then type rules or more complex rules that employ graph or finite element methods to analyze a part.

To classify basic sheet metal parts, [11] employs both a rule-based algorithm and Fourier descriptors. These methods are used to create an index for a part, from which queries are made to identify similar

parts through the application of fuzzy logic. In [12], rule-based analysis of STEP files is employed to perform a manufacturability evaluation of sheet metal parts produced with bending and shearing processes. The approach is completed in two stages, first by analyzing the STEP representation to extract certain part features, and subsequently to evaluate this information with a rule-based algorithm. [13] analyzes a STEP part representation to determine machining features in sheet metal parts. Their approach determines the 3D central plane of the part and evaluates this to extract features like holes and embossments. [14] analyzes STEP files to recognize deformation features in sheet metal parts. The STL format has also previously been used for identification of machining features in sheet metal parts. In [15], STL analysis is performed using the adjacency relations between triangles in an STL part representation to identify different surface types. Features are subsequently determined through evaluation of patterns between neighboring surfaces. [16] also employs a triangulated part representation to calculate the medial axis, from which important features in sheet metal parts are identified.

In order to compare solid models of milled components, [17] evaluates their features. The proposed method relies on the model dependency graph (MDG), which is a graphical representation of a part based on its features and the relations between these. From the MDGs, the largest common subgraph between two parts is identified and used to evaluate their similarity. In [18], aerospace sheet metal parts are also compared in a feature-based manner in order to identify differences with the aim of information reuse.

For rule-based feature recognition, the attributed adjacency graph (AAG) has also been frequently employed [19]. This graph-based method represents a part through the neighboring relations of different surfaces, separating concave and convex edges. The AAG is subsequently used to identify features through patterns within these surface interrelations.

Multiple previous works are focused on finite element analysis (FEA) for simulation of sheet metal forming processes [20]. In [21], for example, the finite element method is used to simulate a deep drawing process as part of an approach for optimizing the springback. [22] discusses the inverse finite element approach, which can be employed to predict the initial blank of a sheet metal part. This approach starts out with a finite element representation of a part and performs an inverse simulation of the forming process to estimate its initial workpiece.

Finite element analysis can, to some extent, be performed automatically using, for example, an API



[23]. API is an abbreviation for application programming interface, and is a tool that provides an interface between a software application and various programming languages. In turn, this allows for automatic control of the software using a programming language. That is, instead of manually defining processes inside of some application, this can be done automatically using a script with a predefined set of processes to complete. CAD software APIs can also be directly used for rule-based geometrical analysis. In [24], for example, a CAD software API is used to investigate and classify the edges in a 3D CAD part model.

### 2.3.2 Learning-based analysis

The other main approaches employed for geometrical analysis relate to learning-based methods. Although rules often can be simple to implement in certain cases and provide a clear overview of exactly what happens inside of a system, general rules are not always known and can be extremely difficult to determine [25]. In such cases, learning-based methods are often consulted.

The term *learning-based methods* refers in this work to any algorithm that relies on tuning its own parameters through some sort of training process. That is, instead of creating some predefined set of rules to perform an evaluation, a human designer creates a model where the "rules" are determined by the model itself through a learning process in which the model evaluates examples of the task at hand. For example, instead of programming an algorithm that labels all parts with a volume larger than  $1m^3$  as large, a model is designed with some predetermined functionality and a set of tunable parameters. This model could subsequently be presented with examples of parts that are labeled as both large and not large, and tune its parameters such that it finally, after successful training, would label parts with a volume larger than  $1m^3$  as large. In this example, the general rule is simple to implement when knowing both the volume and the threshold above which parts should be labeled as large. However, this is not necessarily always the case. For example, instead of the volume, the available data could be only the vertices of a part, and the threshold for labeling a part as large could be unknown.

Multiple previous works employ learning-based analysis for the purpose of evaluating sheet metal parts and properties [26]. Many such approaches focus, for example, on predicting values that traditionally are obtained with FEA because of the high complexity of the nonlinear calculations in finite element models [27]. In [28], the springback angle in a wipe-bending process is predicted using an artificial neural network (ANN) approach that evaluates sheet metal part properties, such as

the thickness and material properties. Springback in U-shaped bending is predicted with a similar approach in [29]. Springback is a phenomenon that happens in large deformation operations, such as bending and drawing, where the final shape is slightly changed after the operation because the material contracts. Another phenomenon that can happen in sheet metal forming is wrinkling. [30] uses neural networks to evaluate if wrinkling will occur during production of sheet metal parts. To determine the optimum clearance between the punch and die in sheet metal blanking operations, [25] employs a neural network that takes the material elongation as an input. In [31], a neural network solution is used instead of constitutive nonlinear material models to describe stress-strain relationships of large deformations in sheet metal parts.

To optimize the tool design in a sheet metal air bending process, [32] employs a neural network that evaluates the material thickness and mechanical properties. [33] employs a neural network to establish the tool path in an incremental sheet metal forming process for L-shaped profiles. To evaluate more complex parts, [34] employs the Gaussian curvatures at each vertex of a triangulated no-thickness mesh representation as network inputs.

Several learning-based approaches have also been employed to identify the machining features in a part. Multiple inputs, such as graph-based, face-coding, contour syntactic, and volume decomposition methods, have been tested [35]. The most recent approaches, however, employ more direct geometrical inputs. In [36], a 3D CNN is developed, called FeatureNet, which takes a full voxel model as input. A voxel model is a 3D model format in which a part is represented by a number of cubes, or *voxels*. Whereas a pixel in an image has two dimensions, the voxel has three and can hence represent 3-dimensional objects. The voxel model is evaluated by the CNN to identify various milled machining features. To improve the accuracy on intersecting features, [37] and later [38] have been developed. These employ CNNs that evaluate images captured from multiple views of a part to perform the task of milled machining feature recognition.

Convolutional neural networks are also frequently applied for other applications in geometrical analysis. [39] and [40], for example, identify different types of fasteners from real life images with such networks. In [41] and [42], sheet metal part types are identified with CNNs.

## 2.4 Problem verification

The high degree of complexity related to the analysis of sheet metal components generates a need for an efficient and automated process. To automatically estimate the cost of components, both 2-step [2] [3] [7] [6] and direct [9] [10] approaches have previously been used. The direct approach employs CNNs to directly evaluate a component in terms of cost, rather than first performing an abstraction of low-level product information into cost drivers, as is done in the 2-step approach. However, to train the CNNs that are used to estimate the costs with the direct approach, large amounts of cost data, which is not feasible to acquire within the scope of this work, is required. Similar data is also required by the 2-step approach variation where a learning-based algorithm is used to evaluate cost drivers [3]. As discussed in Section 1.1, this renders a 2-step approach with an analytical cost driver evaluation as the best option in this work.

In order to translate low-level product information into cost drivers, either rule- or learning-based geometrical analysis can be performed. Previous works use both STEP [12] [13] [14] and triangulated [15] [16] part representations to analyze sheet metal components in a rule-based manner. However, limitations in the approaches that can require manual intervention are sometimes reported [15]. The sheet metal components that are analyzed in works that do not report similar limitations are often found to be of a less complex nature [11] [12]. The limitations to rule-based analysis in complex sheet metal components typically stem from the previously discussed issues with identifying definite general rules that are flexible and true in all events [25] [36].

To combat the limitations of rule-based approaches, several recent works employ learning-based methods to perform geometrical analysis. The presented results of these approaches are promising, although their applications often are concerned only with milled parts [36] [37] [38]. These have multiple differences from the free-form sheet metal components that are the focus of this work, which are typically considered to be more complex. The works focused on learning-based analysis of sheet metal components are often found to be limited to specific component types. [42], for example, is concerned only with laser cutting components.

No flexible and completely automated system that evaluates 3D CAD part models of typical sheet metal components to identify key cost drivers is found to be readily available. However, certain previous works are of particular importance as inspiration and potential solutions to identify individual cost drivers in such a system. This is further discussed in Section 4.2.

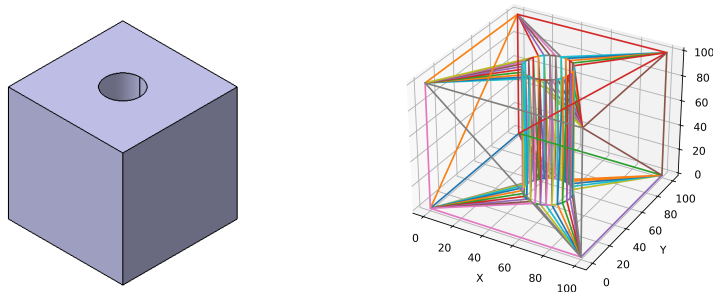
From the above discussion, the research question stands.

### 3 Methods

This section will elaborate on the primary methods that are employed in this work. First, a brief introduction to the STL format is provided, before a more detailed presentation of neural networks and convolutional neural networks is made. The reasons for using approaches relating to these methods are discussed in Section 4.3.

#### 3.1 The STL format and its analysis

STL is an abbreviation for STereoLitography, standard tessellation language, or standard triangle language. All of these refer to the same file format that approximates 3D surface geometry using triangles. As an illustration of the STL format, a cube with a hole through its center is presented in Figure 3 in both its native CAD and the STL part representation. Upon investigation of the STL representation to the right, the triangles that approximate the geometry can be observed. The front face of the cube, for example, consists of two triangles.



**Figure 3** – The same part in the native CAD format (left) and the STL format (right).

Each triangle, also called facet, in an STL file consists of three vertices, represented in the format as three points in 3D space. A normal vector is additionally assigned to each triangle, indicating which side of the triangle that represents the surface of the part. This normal vector points outwards, meaning away from the solid object.

Simplicity and independence from any particular 3D software are often considered to be the primary strengths of the STL format. Analysis of STL files generally entails investigations into the relations between the vertices and normal vectors of the different triangles [43]. Surface types are, for example, identified in [15] through analysis of, amongst other things, the angles between the neighboring triangles. Typical base information that can be extracted from the STL format includes

the number of triangles in the STL representation, the part volume, surface area, and bounding box [44].

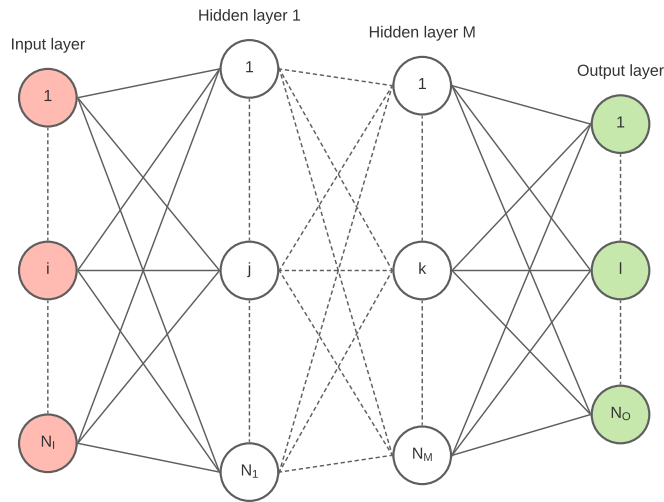
### 3.2 Artificial neural networks

Analysis with learning-based methods, specifically artificial neural networks (ANNs), henceforth primarily referred to as just *neural networks*, constitutes a significant part of this work. This subsection is therefore dedicated to elaboration on this method. Unless otherwise cited, the discussed theory is obtained from the books [45], [46], and [47].

This section is intended to provide some theoretical background on the topic of neural networks so that it is possible to follow the discussions and argumentation throughout the thesis. For more details, the above-mentioned books can be consulted. Argumentation and discussion as to why different design choices are made are generally not included in this section, as its aim is to explain the fundamental concepts. The reasons for making different design choices, direct citations to different works on specific topics, and more specific details are provided in Section 5.3.2.

Neural networks are a subset of machine learning inspired by the functions of the human brain. Like the brain, the traditional **fully-connected neural networks** contain multiple **neurons** that are interconnected and communicate with each other. An illustration of such a network is provided in Figure 4. Here, each node represents a neuron, whilst each line connecting two nodes represents a connection between two neurons. The network consists of several *layers*, starting with the input layer to the left and ending with the output layer to the right. The **input layer** is merely an illustration of the inputs that are provided to the network, and the values that exit the **output layer** are the final network output. The layers between the input and output layers are called **hidden layers**, and their number can range from 1 to, theoretically, any natural number. However, 1 or 2 hidden layers are usually sufficient. In fact, a neural network with only a single hidden layer can approximate any continuous function, and neural networks are therefore often said to be *universal approximators* [48].

The number of neurons in both the input and output layers of a neural network is determined by the problem. That is, some problem with a certain number of outputs is attempted solved using some certain number of inputs. If, for example, determining if a part should be labeled as large or not based on the volume, there would be one input neuron, namely the volume, and two output



**Figure 4** – An illustration of a fully-connected neural network.

neurons, indicating either "large" or "not large". The number of neurons in the hidden layers can, on the other hand, be adjusted. This number is a tunable parameter of the network, and the optimal amount of hidden layers depends on the problem complexity. A simple problem can, for example, often be sufficiently solved with a single layer of only a couple of neurons, whereas a more complex one could require a larger amount of neurons and possibly more hidden layers.

In a fully-connected network, all neurons in the previous layer are connected to all neurons in the next. That is, information is passed from the input layer, processed through all neurons in the hidden layers, and finally exits the network as some output after being processed through the output layer. Between the layers, the output from the previous layer is multiplied with tunable parameters, called **weights**. These weights are part of what is tuned during training of the network, and they determine how much the output from some neuron in the previous layer should be weighted by a particular neuron in the next. If, for example, the aim of a network is to make a prediction about the weather, and the inputs that are provided are the time of year, atmospheric pressure, the weather over the past 3 days, and the age of the network designer, the weights associated with the latter input will likely be very small since it presumably is not of particular importance to the weather. In the illustration in Figure 4, each connection between two neurons can be considered to represent multiplication with such a weight. The outputs from the previous layer into some single neuron in the next are multiplied with their respective weights and then summed together. Additionally, another parameter, called the **bias**, is added to this sum. The biases are the other

parameters, in addition to the weights, that are tuned during training of the network. The value that is passed into some neuron is then:

$$z_k^l = \sum_{j=1}^{N_{l-1}} (w_{kj}^l a_j^{l-1}) + b_k^l \quad (1)$$

where  $z_k^l$  is the input to the neuron  $k$  in the current layer,  $l$ ,  $N_{l-1}$  is the number of neurons in the previous layer,  $w_{kj}^l$  is the weight connecting the neuron  $j$  in the previous layer to the neuron  $k$  in the current layer,  $a_j^{l-1}$  is the output of neuron  $j$  in the previous layer, and  $b_k^l$  is the bias in neuron  $k$  in the current layer.

Inside of all neurons except the input neurons, which merely illustrate the network inputs, there is a function that transforms the input  $z_k^l$  in Equation 1 into some output. These functions are called **activation functions**. The output from neuron  $k$  in the current layer,  $l$ , is therefore:

$$a_k^l = f(z_k^l) \quad (2)$$

where  $f(z)$  is the activation function. Typically, all neurons in all hidden layers will employ the same activation function, but this activation function often differs from that in the output layer.

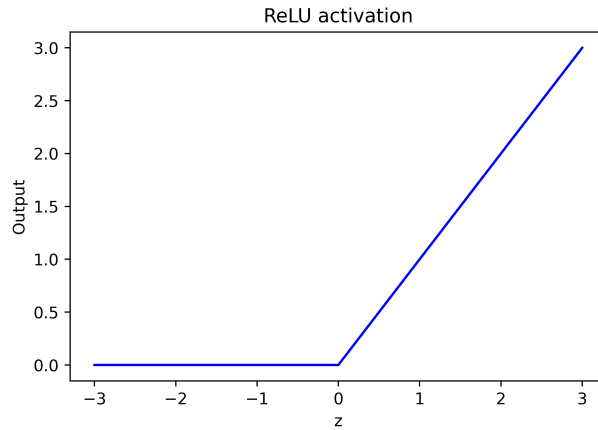
There are many different types of activation functions, most of which have different advantages and disadvantages. In general, neural networks are still somewhat "black boxes", which means that the mathematical tools to fully describe and prove the universal efficiency of various network concepts are not available. As a result, much of network theory, for example relating to activation functions, is based on hypotheses and experimental results in various applications. This often means that many different arguments exist as to which activation function to use when, without the existence of any universal truth. This discussion will not dig deep into these arguments but rather focus on the activation functions that are employed in this work.

In the hidden layers of networks in this work, the **rectified linear unit**, or **ReLU**, activation is used. This activation function is often found to perform well, especially in deep learning problems with many hidden layers (such as convolutional neural networks, which are elaborated on in Section 3.3). The ReLU activation is defined as:



$$f(z_k^l) = \max(0, z_k^l) \quad (3)$$

That is, when the input to neuron  $k$ , as calculated in Equation 1, is positive, the output of the neuron is simply the same as this input. If the input is negative, the output is zero. The output of the ReLU activation function for some different inputs,  $z$ , is plotted in Figure 5.



**Figure 5** – The ReLU activation function.

The output layer neurons also contain activation functions. In these, however, there is less discussion as to which activation functions to use. In general, the activation in the output neurons is primarily dependent on the problem type. That is, the choice depends on whether the problem at hand is a classification or regression problem. A **classification problem** means that it is desired to evaluate some input to determine the class to which this input belongs. In the case of weather prediction, for example, the output could be either "good weather" or "bad weather". This would mean two output neurons, one relating to each class. Usually, it is considered beneficial that the output relating to a particular class takes on a probabilistic value. That is, that the output relates to the predicted probability, or *confidence* of the network, that the input belongs to a particular class. For example, that the prediction adheres to an 80% confidence of good weather for the set of inputs. In order to achieve such an output behavior, **softmax activation** is typically used in classification problems. This activation function is defined as:

$$f(z_k^l) = \frac{e^{z_k^l}}{\sum_{i=1}^{N_l} e^{z_i^l}} \quad (4)$$

where  $z_k^l$  is the input to neuron  $k$  in the current (output) layer,  $l$ ,  $N_l$  is the number of neurons in the current layer, and  $z_i^l$  is the input to neuron  $i$  in the current layer. The sum of all activations in the output layer for the softmax activation,  $\sum_{k=1}^{N_l} f(z_k^l)$ , is 1, and the output hence adheres to a probability distribution. There are also some more reasons than those presented as to why it can be beneficial to use softmax activation in the output layer, but these will not be detailed here.

A **regression problem** means that it is desired to establish some specific value rather than a class. That is, the desired output can not be discretized on a predetermined interval. In the case of weather prediction, for example, the desired output could be the temperature. This output can take on any (within reason) floating-point value, and a class can therefore not be assigned to all possible outputs. In regression problems, the choice of output activation is typically the **linear activation**, defined as:

$$f(z_k^l) = z_k^l \tag{5}$$

That is, the output of the neuron is simply equal to its input, defined in Equation 1. A regression problem can have both one and multiple output neurons, depending on the problem. For example, if it is desired to predict both the temperature and the amount of rain, the weather problem would have two output neurons, both with linear activation.

Once a network is fully defined with some initial number of hidden layers, some initial number of neurons in each of these layers, and some activation function associated with all neurons, the network is ready to be trained. During training, the network will tune the weights and biases associated with all connections in the network in order to attempt to determine the correct output from the provided input. Initially, all weights and biases in the network are usually generated randomly, often with some mean value around 0 and a variance around 1.

There are two main types of training, namely supervised and unsupervised. In **supervised learning**, the network is provided both a training example, which is a set of inputs, and the answer, or *label*, associated with this set of inputs. That is, the network is provided with both an input and the expected output, or target value, for this input. In **unsupervised learning**, the network is not provided with these target values. Instead, roughly speaking, the network evaluates multiple training examples and finds common patterns amongst these, making its own classes with

input sets that display similar patterns. In this work, only supervised learning is employed, and all subsequent discussion therefore refers to supervised learning.

Before training a neural network, the available labeled data needs to be prepared. This entails both preprocessing the data in such a way that it can be used efficiently as input to the network, as well as sectioning it into subcategories to be used for different parts of the development. The preprocessing stage can, for example, entail normalization of the various inputs, such that their sizes are of comparable magnitudes. This is often found to improve the overall performance of the network. The dataset is usually split into three subcategories, namely training, validation, and test data. This is done in order to ensure that labeled data is available to evaluate the network performance on data that it has never "seen" before. That is, validating that the patterns the network finds to link the input to the output are true in the general case, and not just true for the data that the network has used for training, namely the **training data**.

It can often happen that the network ends up merely learning non-general patterns that only exist in the training data. This is a phenomenon called **overfitting**. Overfitting is possible because neural networks typically have such large amounts of tunable parameters that they can start to "memorize" the training examples instead of learning the general patterns that connect the input and the output. Overfitting will almost always happen to some extent in a network. This is intuitive also when considering how humans learn. If, for example, a human is tasked with identifying different species of plants from images and provided some set of images depicting each different species as training examples, they would generally be more certain that a specific image provided as an example of a species depicts that species, than some other image that they have not previously seen. The same is true for networks. Although some overfitting usually is inevitable, it is desired to limit the extent of overfitting that happens in the network.

One way to limit the extent of overfitting is to increase the amount and quality of training data. That is, increase the total size of the labeled dataset and the variation within this. Data can often be considered as the most important factor to achieve success in neural network applications. Even the best and most complex state-of-the-art networks can not achieve a high performance if they are trained on insufficient data. By increasing the amount and quality of training data, the network will tune its weights based on more examples that cover a larger portion of the possible input space, which, in turn, would allow it to identify more general patterns. It is harder to merely "memorize"

a larger and more varied dataset, which means overfitting typically would be reduced.

In order to validate that the network is actually learning general patterns, and also to evaluate the degree of overfitting in the network, it is desired to keep certain amounts of data out of the training process. Because of this, validation and test data is separated from the training data. The **validation data** is used continuously to validate the network performance on "unseen" data throughout the development process. Based on the validation data results, different design choices can be made during development, such as altering the number of neurons in a hidden layer. The **test data** is subsequently used to validate the final network configuration once the development process is assumed to be complete. The reason for separating the validation and test data is that some overfitting also can occur to the validation data. Since the development process aims to optimize the validation data performance, making different design choices based on the results on this dataset, it could happen that some of these choices only benefit the validation data, and not the actual general case. As a result, in order to validate that the assumed performance in the general case is true, the test data is typically kept aside and out of consideration until a final configuration is reached. There are, however, some cases where it could be beneficial to consult the test data performance also during the development.

The aim of the network training process is to tune the weights and biases in such a way that when the network is provided with some set of inputs, it determines the correct output. During training, the network is, in other words, trying to match its own output for a specific training example with the adhering label, or target value, associated with this example. In order to evaluate how close the network is to achieving its task, it needs some measurement that describes the distance between the prediction and the target. That is, how close the current weights and biases are to those that achieve a completely correct answer. This measurement of how close the prediction and target are, is provided by the **loss function**. This function is often also referred to as the *cost function*, but the term loss function is used consistently throughout this work.

Like for the activation functions, multiple different loss functions exist, without there necessarily being a universal "right" or "wrong" loss function. However, a typical choice in classification problems is the *crossentropy loss function*. The **categorical crossentropy loss function** is used on all classification problems in this work and is defined for a single data point as:

$$L(\hat{y}) = - \sum_{i=1}^{N_c} y_i \times \log(\hat{y}_i) \quad (6)$$

where  $N_c$  is the number of classes,  $y_i$  is the target value for class  $i$ , and  $\hat{y}_i$  is the predicted value for class  $i$ . In cases where multiple data points are evaluated at once, the mean of their individual losses is used<sup>1</sup>.

The number of output neurons in classification problems is, as previously mentioned, equal to the number of classes in the problem at hand. When softmax activation is used in these output neurons, the  $\hat{y}_i$  values will be some number on the interval  $[0, 1]$ , adhering to the network's confidence that the input belongs to class  $\hat{y}_i$ , relative to the other classes. The label, or target, of a training example is then some vector in which all entries are 0, apart from the one that adheres to the correct class. This entry will be 1, indicating 100% confidence. If, for example, there are two classes, "good weather" and "bad weather", and the example is a set of inputs where the correct answer is "good weather", the target vector is:

$$y = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad (7)$$

Since all the entries,  $y_i$ , in the target vector are 0 apart from the one that adheres to the correct class, and because the logarithm of 1 is 0, it follows that the categorical crossentropy is 0 when the network predicts the correct class with 100% confidence. In other words, the loss is 0 when the network prediction is completely correct. This is a general property of a loss function.

For regression problems, a common choice of loss function, and also the one that is used for regression problems in this work, is the **mean squared error (MSE) loss**. The MSE loss for a set of data points is defined as:

$$L(\hat{y}) = \frac{1}{N_b} \sum_{j=1}^{N_b} (y_j - \hat{y}_j)^2 \quad (8)$$

---

<sup>1</sup>There are also other ways to evaluate multiple data points, but this work considers to the mean.

where  $N_b$  is the number of data points evaluated,  $y_j$  is the target for one of those data points, and  $\hat{y}_j$  is the predicted output for that data point. If only a single data point is to be evaluated,  $N_b$  would be 1, and the summation would subsequently disappear. As for the categorical crossentropy loss function described above, it can be noticed that also the MSE loss is 0 when the predictions and targets are equal.

The loss function allows the network to measure how far its prediction is from the correct answer during training. However, the network also needs a tool that evaluates this loss and subsequently determines how to change the weights and biases in order to achieve a smaller loss in the future. In other words, something that defines how the weights and biases should be tuned in order for them to approach their optimum value, where the network prediction is equal to the target for all data points. This tool is usually the combination of the backpropagation algorithm and an optimizer. The **backpropagation** algorithm aims to approximate the gradient of the loss function for all weights and biases in the network. That is, determine how the loss function changes when a change is made to some weight or bias inside the network. In other words, its output allows for an investigation of how a weight or bias should be changed in order to achieve a smaller loss, meaning that they are approaching their optimum. In order to do this, the algorithm first predicts the result of a given input, called the *feedforward* process, before it *backpropagates* the *output error*, to finally determine the partial derivatives of all weights and biases (that together make up the gradients) for that input. Further details on the exact workings of the backpropagation algorithm are not provided here, and it is rather referred to the books cited at the beginning of the chapter if more detail is desired. There are also other algorithms and methods to train neural networks, but the combination of backpropagation and optimizer is the common choice, and also what is used and considered in this work.

The gradients that are calculated by the backpropagation algorithm are evaluated by the **optimizer**, or *learning algorithm*. The optimizer is what finally determines how to update all the weights and biases in the network. Like for the activation functions and losses, there are multiple different optimizers to choose from. There is also generally no "correct" choice of optimizer, but rather many different ones with different advantages and disadvantages. Common optimizers are, however, typically based on the principle of gradient descent. The base idea of gradient descent is simply to evaluate the gradients acquired through backpropagation in order to find out how to change the weights and biases so that the change in loss between the new and old configuration is

negative. That is:

$$\Delta L = L_{new} - L_{old} < 0 \quad (9)$$

In other words, the new weights and biases achieve a solution that is closer to the target. To ensure this, gradient descent takes advantage of the fact that a change in some function,  $f(\vec{x})$ , can be approximated by:

$$\Delta f(\vec{x}) \approx \sum_{i=1}^{N_v} \left( \frac{\partial f}{\partial x_i} \cdot \Delta x_i \right) = \nabla_{\vec{x}} f \cdot \Delta \vec{x} \quad (10)$$

where  $\vec{x}$  is some vector of variables of the function  $f$ ,  $N_v$  is the number of variables,  $\frac{\partial f}{\partial x_i}$  is the partial derivative of the function with respect to one of the variables,  $x_i$ ,  $\Delta x_i$  is a change in that variable,  $\nabla_{\vec{x}} f$  is the gradient of  $f$  with respect to  $\vec{x}$ , and  $\Delta \vec{x}$  is the vector representing the changes in all variables. By selecting the change in variables,  $\Delta \vec{x}$ , as:

$$\Delta \vec{x} = \vec{x}_{new} - \vec{x}_{old} = -a \nabla_{\vec{x}} f \quad (11)$$

where  $a$  is some positive constant, the approximated change in the function  $f(\vec{x})$  reduces to:

$$\Delta f(\vec{x}) \approx -a (\nabla_{\vec{x}} f)^2 < 0 \quad (12)$$

where  $(\nabla_{\vec{x}} f)^2$  is the square of the gradient of  $f$  with respect to  $\vec{x}$ . Since the square of any real number is positive, the complete expression is therefore strictly negative. In other words, it is approximated that the function  $f(\vec{x})$  is decreasing by updating the variables,  $\vec{x}$ , according to:

$$\vec{x}_{new} = \vec{x}_{old} - a \nabla_{\vec{x}} f \quad (13)$$

Translating this to the case of weights, biases, and the loss function, gradient descent finds that in order to achieve a solution where the updated values generate a smaller loss than the previous ones, the weights and biases should be updated as:

$$\mathbf{w}_{new} = \mathbf{w}_{old} - \eta \nabla_{\mathbf{w}} L \tag{14}$$

$$\mathbf{b}_{new} = \mathbf{b}_{old} - \eta \nabla_{\mathbf{b}} L \tag{15}$$

where  $\mathbf{w}$  and  $\mathbf{b}$  are the tensors of the weights and biases, and  $\nabla_{\mathbf{w}} L$  and  $\nabla_{\mathbf{b}} L$  are the gradients of the loss function with respect to the weights and biases. In these equations,  $a$  is substituted with  $\eta$ , which is typically called the learning rate.

The **learning rate** is, as can be observed in Equations 14 and 15, a parameter that describes how much the weights and biases should be changed for every update. Having a suitable learning rate is very important in order to achieve satisfactory results. However, as for most other network settings, there is no universally "correct" or optimal learning rate that can be applied in every case. Rather, some initial learning rate is typically set by the designer, and then this is subsequently tuned in an iterative manner through either trial and error or some optimization algorithm. The learning rate and other similar parameters that are set and tuned by a designer or an external optimization algorithm are often referred to as **hyperparameters**.

To solve Equations 14 and 15, a learning rate, which is set by a designer prior to training, the current weights and biases, which are randomly initiated at the start of training, and the gradients, which are found through backpropagation, are required. Since all of these are known, the equations can hence be solved to update the weights and biases. In classic gradient descent, the gradients of the weights and biases are estimated as the mean of the calculated gradients for all training examples. However, this often results in a high computational effort because all training examples need to be evaluated for every single update. As a result, it is more common to use optimizers that are based on a gradient descent variation called **stochastic gradient descent (SGD)**. In this variation, the gradient over all data points is estimated by the mean over a subset of data points, called a **batch**, or mini-batch<sup>2</sup>. An update of the weights and biases can subsequently be made for each of these estimated gradients, rather than evaluating all training examples for every update.

---

<sup>2</sup>If the batch size is larger than 1, SGD is often referred to as mini-batch SGD.



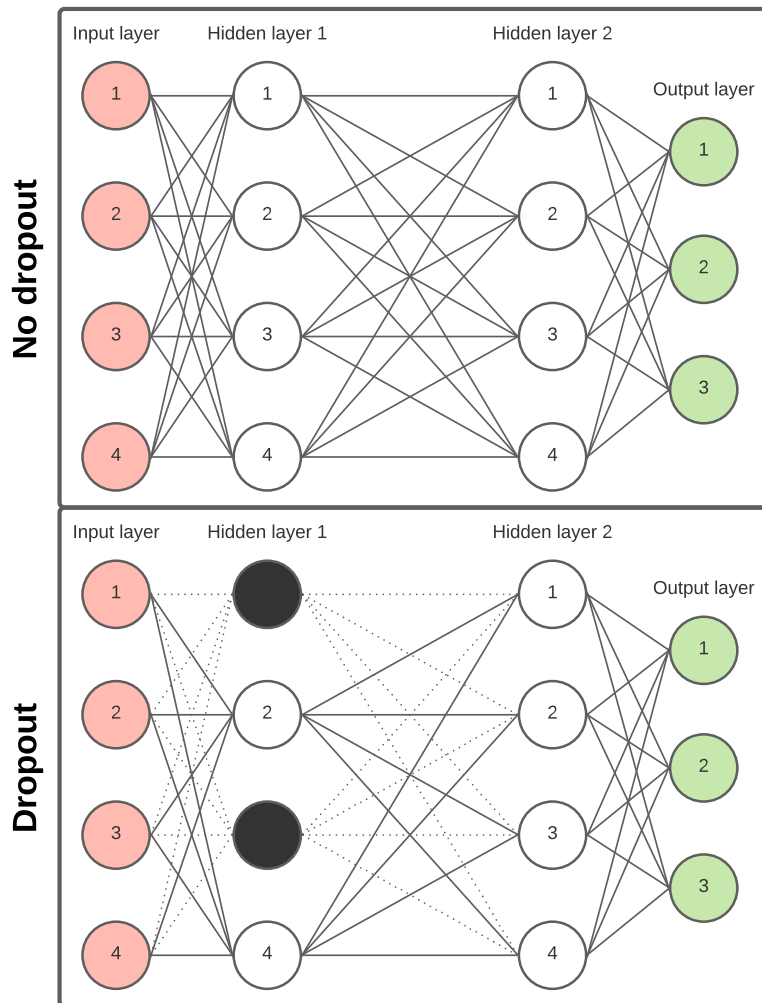
The **batch size**, which is the number of training examples in each batch, is another hyperparameter that can be tuned by the designer.

When all training examples have been evaluated once, a so-called **epoch** is finished. The training data is then typically shuffled and re-split into batches, and another training epoch is commenced. The number of epochs to train for can either be predetermined, as a hyperparameter, or determined dynamically based on the results obtained during training. That is, the training can, for example, be stopped once no improvement is observed on the validation data for some number of epochs. This technique is called **early stopping** and is used throughout this work.

Using early stopping has three primary advantages. For one thing, the training is seized once the network is assumed to have achieved a near-optimal solution for the current configuration. In other words, it is ensured within reason that the network does not train too short to achieve representative results, whilst it, at the same time, does not train redundantly long, increasing the time effort of the training process. The second advantage is that early stopping removes a hyperparameter that can be difficult to set, namely the number of epochs to train for. Although the number of epochs to "wait" for an improvement must be specified, this parameter is often easier to determine. The third advantage is that early stopping helps to reduce overfitting. Increasing amounts of overfitting are often observed if a network trains for too long. In these cases, the performance on the validation data stagnates, or even decreases, whilst the performance on training data keeps improving as the network starts to learn more non-general patterns that are unique to the training data.

Another technique that is often used to reduce the overfitting in a neural network is **dropout** [49][50]. Dropout can be applied to either all or some fully-connected network layers, and it is a technique where some of the neurons in the layer on which dropout is applied are randomly *dropped* with some probability during training. That is, for each training example or batch, some random neurons are, in effect, removed from the network. An illustration of dropout in a simple network with two hidden layers is provided in Figure 6. Here, the top network is the original network without any dropout applied, whereas the bottom is an illustration of a training setting where 50% dropout is applied to "Hidden layer 1". As can be observed, neurons 1 and 3 are "removed" from the network, meaning their outputs are not considered by the subsequent layer. For the next training example or batch, another set of neurons is likely dropped, such as 2 and 4 or 1 and 2.

A reason as to why dropout often can help to reduce overfitting is, roughly speaking, that the



**Figure 6** – Illustration of a network without dropout (top) and with dropout (bottom) applied.

network is forced to learn independent patterns. That is, it tunes the weights and biases with a number of different architecture configurations, which in turn somewhat limits its ability to merely ”memorize” certain combinations of neuron activations and forces it to learn more general and rigid patterns. There are also other aspects to dropout, which can be found in the papers that introduce the technique, namely [49] and [50].

### 3.3 Convolutional neural networks

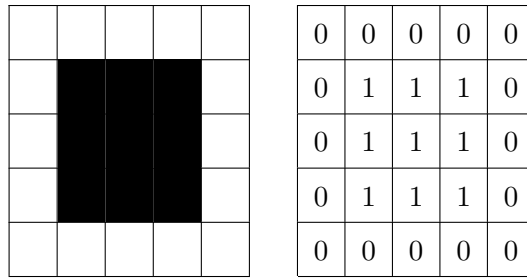
Convolutional neural networks (CNNs) are a subset of artificial neural networks and part of what often is referred to as *deep learning*. CNNs have multiple differences from the traditional fully-connected networks, although most of the basic principles are the same. The field of

convolutional neural networks is fairly young, not being thoroughly introduced until 1998 in [51]. Still after that, several years passed before their use started to gain traction in the mid 2000s, and even more before their use started to take off in the early 2010s [52]. The reason for this delayed development relates in large to the limited computing power that was generally available in the early years after the introduction of CNNs. The theory in this chapter, unless otherwise cited, is obtained from the same books that are cited at the beginning of Section 3.2, namely [45], [46], and [47]. If the reasoning behind the different design choices is not given here, it is provided in Section 5.3.2, unless otherwise stated.

Convolutional neural networks are primarily and traditionally used for image recognition. That is, they are specially designed to recognize patterns in 2D space. Whereas a fully-connected network takes some 1D array of inputs, the regular CNNs evaluate 2D matrices. In fact, state-of-the-art CNNs evaluate color images, which means their base inputs are three 2D matrices, one for each color in a traditional RGB image. Although traditional fully-connected networks also can perform image recognition by transforming the images into 1D arrays, CNNs have proven more suitable for this task because of their ability to account for the spatial structure of images.

In order to understand how a neural network can evaluate an image, it is first important to realize that the computer representation of an image simply is a matrix, or tensor, of numbers. To humans, an image consists of patterns relating to the perceived colors and contrasts between the different elements in the image. To a CNN trained for image recognition, it consists of patterns relating to the different numbers in the computerized image representation. In the simplest case of a binary black and white photo, for example, white is typically represented by the number 0, whereas black is represented by the number 1. An illustration of a black square in this format, as perceived by humans and computers, is presented in Figure 7. Here, each pixel is separated by borders, and it can be observed that what to humans is perceived as a black square, is a collection of 1s to a computer. The same principle applies in more complex color images.

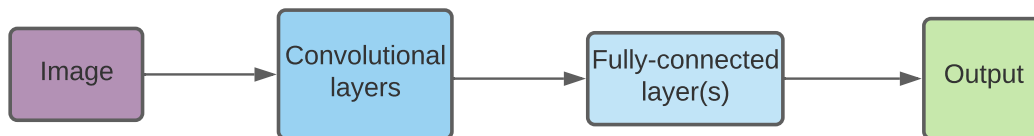
A rough outline of a typical convolutional network architecture is provided in Figure 8. First, the input image is passed through the *convolutional layers* of the network, before the final output from these is passed through one or more fully-connected layers to make the final prediction of the network. Inside the convolutional layers, the network attempts to identify patterns in the input image, and these patterns are then subsequently evaluated by the fully-connected layer(s) to



**Figure 7** – Illustration of how a binary black and white photo looks to a human (left) and a computer (right).

connect them with some output.

The base operation in convolutional neural networks is the **convolution operation**. In this operation, a **filter**, or **kernel**<sup>3</sup>, is used to transform some input image into some new representation, called the **feature map**. An illustration of a 2D convolution operation with a 3x3 filter is presented in Figure 9. Here, the input image,  $I$ , to the left, is convoluted with the filter,  $F$ , to make the feature map,  $I * F$ . In mathematics, the  $*$  sign is typically the operator used to denote a convolution. The CNN convolution operation is an element-wise multiplication between a region in the input matrix or tensor<sup>4</sup> and the filter, finally summed to create a single output. The resulting feature map is then subject to an activation function and is passed as input to subsequent layers. After being subjected to an activation function, the feature map is sometimes referred to as the *activation map*. This work will, however, strictly use the term feature map for simplicity.



**Figure 8** – Rough outline of a typical CNN architecture.

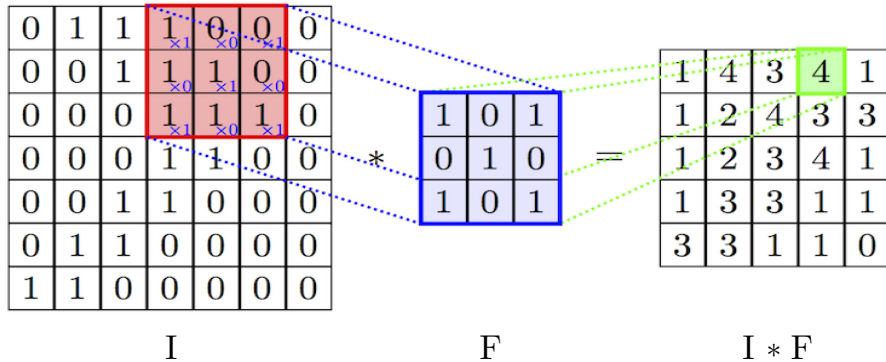
In Figure 9, the convolution operation is illustrated by the filter’s convolution with the red region in the input. The output of this convolution is the sum:

<sup>3</sup>These terms are used synonymously in many works, although some claim they have somewhat different meanings. This work will use *filter*.

<sup>4</sup>To the first convolutional layer, this is the input image.

$$\begin{aligned}
I_{1,4} * F &= (1 \times 1) + (0 \times 0) + (0 \times 1) \\
&+ (1 \times 0) + (1 \times 1) + (0 \times 0) \\
&+ (1 \times 1) + (1 \times 0) + (1 \times 1) = 4
\end{aligned}
\tag{16}$$

which is illustrated by the green field in the feature map. The filter is usually passed over the input image with a *stride* of 1. This means that it starts in the upper left corner of the image, and then moves horizontally towards the right, one pixel at a time, until it reaches the upper right corner, where it moves one pixel down and again starts to move from the left towards the right on this row. This is then repeated until it reaches the lower right corner. In all, this results in a size reduction because the pixels on the edges are used less than those in the center. This can sometimes lead to incompatible sizes in between the layers of the CNNs. In such cases, *padding* is often used to increase the dimension of the feature map. When an image is padded, this means that an extra border, or multiple borders, of zeros are added around the feature map to increase its dimensions. Other strides than 1 can also be used for the purpose of spatial manipulation of the feature map, typically with the aim of spatial reduction.



**Figure 9** – Illustration of 2D convolution operation from [53].

The convolution between the input tensor and the filter can be considered as the CNN equivalent to the process where the outputs of the previous layer are multiplied by their respective weights in a fully-connected network. However, one important difference is that the numbers inside a single filter, namely its weights, are the same throughout the convolution. That is, the same filter is used on the entire image for all strides. Therefore, these weights are often referred to as *shared weights*.

Based on the weights in a filter, different patterns, or features, in the input image are highlighted in

the feature map. That is, some set of filter weights could, for example, highlight horizontal edges, whereas another one could highlight vertical ones. In a convolutional layer, the shared weights inside a filter are tuned during training, typically together with a single shared bias for each feature map. In other words, the different filters are tuned to recognize various patterns in the input image that finally are of use to make an evaluation as to what is depicted in the image. The filters can have different sizes in different layers, allowing them to, amongst other things, highlight different types of patterns. Typically, the patterns that are recognized have a more and more complex interpretation, the more layers that are passed. The final output of the convolutional layers of a CNN is then some geometrical description of the input image in terms of identified patterns, or features. This data is then, as shown in Figure 8, passed to some fully connected layers that evaluate it to make the final prediction.

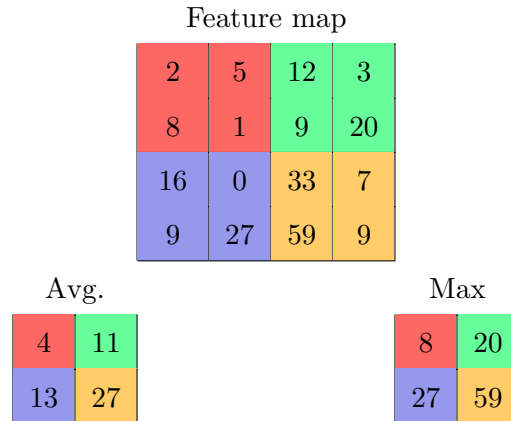
There will typically be multiple filters in each convolutional layer, each recognizing different patterns. This means that multiple feature maps of the same input, one for each filter, are created. This often causes the number of parameters in the network to explode. If, for example, a single black and white input image is subject to three different filters, this results in 3 output feature maps. If the next layer also has three filters, the output of this layer will be 9 feature maps, and so on. Often, there are many more than three filters in each layer. In order to reduce the number of parameters inside the network, **pooling layers** are often used after some number of convolutional layers. The base idea of a pooling operation is to reduce the size of the feature map by "summarizing" the identified patterns. There are also other potential benefits of the pooling than merely parameter reduction, but these will not be discussed here.

Two commonly used pooling operations are average pooling and max pooling. These are illustrated in Figure 10. Here, both pooling operations use a pooling filter of size 2x2 that is moved across the feature map with a stride of 2. In the **average pooling** operation, the region summarized by the pooling filter is reduced by taking the average of all numbers in this region. In Figure 10, for example, the red region is summarized by

$$avg_{1,1} = \frac{2 + 5 + 8 + 1}{4} = \frac{16}{4} = 4 \tag{17}$$

which is illustrated by the red field in the average pooling output to the lower left.

In the **max pooling** operation, the region is summarized by simply extracting the largest number inside the region. In the green region in Figure 10, for example, the largest number, or maximum, is 20, which means this is the output of the max pooling operation on this region.



**Figure 10** – Illustration of average and max pooling with a pooling filter of size 2x2 using a stride of 2.

The pooling layers have no parameters that are tunable to the network during training, which means that they remain constant with the same functionality throughout the training process. The type of pooling, size of pooling filter (the region that is summarized), and pooling stride are, however, all subject to selection by a network designer, without there being a clear "right" or "wrong". The size and stride are often chosen in order of achieving some desired output dimension, for example to ensure compatibility with a subsequent layer. The type is often chosen through trial and error. In this work, CNNs are not built from scratch, which means that the convolutional and pooling layer configurations and architectures will adhere to those used in previously successful architectures, which are presented below. Although the previous discussion separates convolutional and pooling layers, the term *convolutional layers* is sometimes used somewhat freely in this work to also include pooling layers. This is, for example, the case in Figure 8. The reason for this is to increase simplicity, whilst it, at the same time, is assumed to not be a root cause for any misunderstandings.

Even though pooling helps to somewhat reduce the complexity in a network, CNNs can still be difficult to train. That is, a lot of training examples are often required in order to tune all weights and biases from their randomized initial state to achieve a satisfactory performance. Luckily,

however, there is a technique that can aid in the training of CNNs and often allows for successful training with many fewer examples than what otherwise would be required. This technique is called **transfer learning**.

The base idea of transfer learning is that similar applications can inherit CNN parameters from each other. Since the basic task of a CNN is to recognize patterns in an input image, these patterns will often be the same, or at least similar, across different tasks with some similarities. Recognition of a straight edge, for example, could be of use both to identify some specific brand of car and to identify different types of furniture. This is merely an illustrative example, since the patterns recognized in CNNs generally are more complex, but the basic idea is the same in transfer learning. That is, all parameters in a CNN are first tuned through training, or **pre-training**, on a large available dataset that is, at least assumed, to share certain similarities to the current task. Subsequently, the weights and biases in the convolutional layers of this tuned model are transferred, or copied, to a "blank" initialization of the same CNN architecture, and training on the current task can commence. This training process is typically performed in 2 steps. First, all the parameters in the convolutional layers are "frozen", meaning they are locked and not subject to tuning. The training is then performed only on the final fully-connected layers, whose weights and biases are randomly initialized. That is, the same patterns that were established through pre-training on the larger available dataset are identified in the input image, and the final layers are tuned to connect these patterns to the targets of the current problem. Once this training step is finished, for example because training is ceased by early stopping, a second step, called **fine-tuning**, is often performed. In fine-tuning, the convolutional layers are "defrosted", and training continues on all parameters in the entire architecture, starting from the previously tuned weights and biases. This fine-tuning is often carried out at a lower learning rate than the one initially used to train the other layers of the network. The reason for this is that smaller adjustments of the parameters typically are required if the patterns in the pre-training dataset are somewhat adhering to those in the current task.

Transfer learning does, however, not solve all problems with training CNNs. Sometimes, the only way to achieve sufficient network performance is through increasing the amounts and quality of the training data. As previously discussed, data can often be considered as the most important factor to a successful network. As such, increasing the dataset size and variation is often a key focus in network development. However, labeling further existing data can be a very expensive process and is not always feasible nor possible. As a result, techniques to artificially increase the dataset size



and variation, such as **data augmentation** and synthetic data generation, are often consulted. These are discussed in Section 5.2.2 on data preparation.

### 3.3.1 CNN architectures

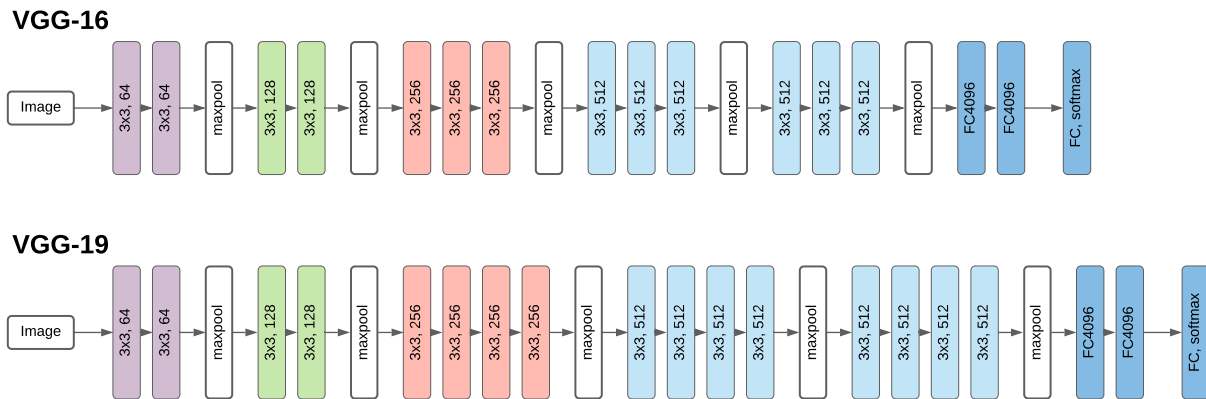
This work will, as previously mentioned, not build CNNs from scratch. Whereas it is common to build "customized" fully-connected network architectures for the specific application, meaning the number of hidden layers and neurons in these typically are tuned on a case-by-case basis, convolutional neural network applications often employ pre-built architectures that have performed well in previous works. The reasons for this are the high complexity of the CNN architectures, the solid work that has been performed to develop highly efficient architectures, and the fundamental similarity between CNN applications. The fundamental similarity refers to the fact that the basic task of a CNN used for image recognition is to identify patterns in the input image, regardless of whether that image depicts a dog or a plant. The recognized patterns will often differ, but an architecture's ability to recognize certain patterns of certain complexities will remain. However, some architectures can be better at recognizing certain types of patterns than others. Different problems have different complexities and traits, meaning different architectures might suit some problems better than others. As a result, it is rarely clear which architecture, out of all available ones, that will perform the best on the specific application.

In this subsection, the three architectures that are used in this work are briefly presented. The reasons for using these exact ones are provided in Section 5.3.2. For more detail on the different architectures, it is referred to the papers presenting them, which are cited for each below.

#### VGG

The VGG architecture is presented in [54]. In some ways, the VGG can be considered a classic, straightforward CNN architecture. Whereas many of the other CNN architectures frequently used in state-of-the-art works employ special elaborate techniques to try and improve performance, the VGG has a more traditional linear approach. That is not to say, however, that the VGG can not outperform these other architectures on certain points, nor that its development did not constitute a significant improvement compared to previous CNN architectures when it was introduced. The base idea that the VGG builds on is that smaller filter sizes than previously used can achieve a superior performance by increasing the *depth* of the network, meaning the number of convolutional layers. Its 3x3 filters allow for efficient architectures with both 16 and 19 weight layers, 13 and

16 of which are convolutional layers. This is compared to, for example, the then state-of-the-art network, AlexNet [55], which uses five convolutional layers and filter sizes up to 11x11.

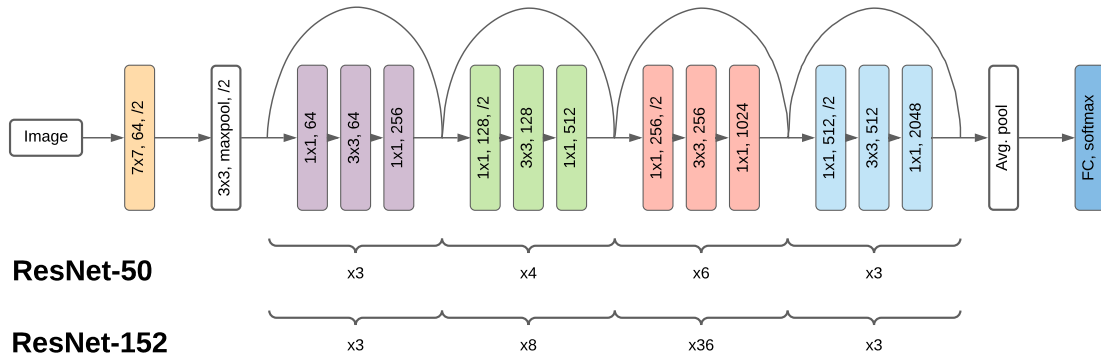


**Figure 11** – The VGG-16 and VGG-19 architecture variations, as presented in [54].

Figure 11 shows a schematic illustration of the VGG-16 and VGG-19 architecture variations. There also exist more variations with fewer layers, but these are the ones that will be employed in this work. Each colored box in the figure illustrates weight layers, meaning a neural network layer that contains trainable weights. Apart from the final three layers, all layers are convolutional layers. In the illustration, the filter sizes, 3x3, are denoted together with the number of output channels from the respective layer. The number of channels is the *width* of the convolutional layers, meaning the number of independent stacks of feature maps to be evaluated. As can be observed in the figure, the VGG-19 has one additional layer with 256 output channels (red) and two additional layers with 512 channels (light blue), one on each side of the penultimate max pooling layer, as compared to the VGG-16. The final three layers of the VGG architecture are fully-connected (FC) layers. The first two have 4096 neurons and ReLU activation, whilst the final layer has softmax activation. The number of neurons in the final (output) layer is not indicated since the number of output neurons, as previously discussed, depends on the problem at hand.

## ResNet

The ResNet architecture is introduced in [56]. The inspiration for the development of the ResNet was a phenomenon observed when adding more and more convolutional layers to "plain" networks (such as the VGG). After a certain amount of added convolutional layers, the network performance is observed to decrease substantially. This phenomenon is called the *degradation problem*. The degradation problem is counterintuitive because adding more layers generally is considered to enable



**Figure 12** – The ResNet architecture with descriptions of the number of block repetitions that are included in the ResNet-50 and the ResNet-152. As presented in [56].

deciphering of more complex patterns and hence increase the performance up until a point of saturation. Moreover, if considering one shallow and one deep variation of the same network architecture, it is counterintuitive that the deeper variation achieves a higher error. This is because the added layers should be able to merely transfer the information provided as input as their output. As such, the deeper variation should not achieve a higher error than the shallow one. Still, experimental results show that this often is the case when using the current CNN tools, which inspired the concept of *residual learning* introduced by the ResNet architecture.

To combat the degradation problem, the ResNet architecture includes *residual connections*, or *skip connections*, or *shortcut connections*. In Figure 12, which depicts the general ResNet architecture, these are the connections that pass over the different blocks. In short, the base idea is that these connections should allow to build deeper architectures since the input to a block is passed both through the block itself and directly past<sup>5</sup> the block to its output. As such, if the block is found during training to decrease the performance, the weights inside the filters can merely be tuned such that the output of the block approximates its input, hence theoretically preventing that deeper networks degrade.

The ResNet architecture in Figure 12 can be observed to consist of four main building blocks, colored in purple, green, red, and light blue, and a final fully-connected output layer with softmax activation, colored in darker blue. The four main building blocks are repeated different numbers of times depending on the ResNet variation. In this work, the ResNet-50 and ResNet-152 are

<sup>5</sup>In certain cases, the input is somewhat manipulated in order to achieve compatible sizes with the output. See [56] for details.

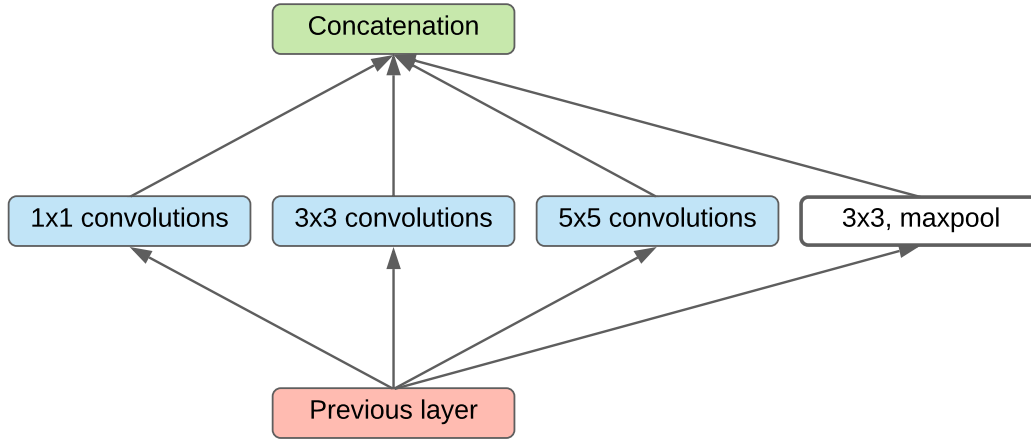
employed, and their numbers of block repetitions are therefore the ones included in the figure. Note that the residual connections also are repeated when these blocks are repeated. That is, for example, if two of the purple blocks are repeated, this would mean that there are two residual connections, one passing over each block, not one residual connection passing over both. The "/2" included in some of the layer descriptions indicate a *downsampling* with a stride of 2. Apart from this, the descriptions of each layer are consistent with those for the VGG in Figure 11.

As opposed to the VGG, it is worth noting that the ResNet architecture only has a single fully-connected layer between the final convolutional layer and the network output. That is, the VGG architecture has two extra fully-connected layers after the convolutional layers. Additionally, it can be observed that there are no pooling layers in between the blocks and that the final pooling operation relies on average pooling rather than max pooling.

### **Inception-ResNet-v2**

The Inception-ResNet-v2 is presented in [57]. The architecture of this network is based on the combination of two concepts, namely the residual connections discussed above for the ResNet and *inception modules*. The concept of inception modules was introduced in [58] by the same team that later developed the Inception-ResNet-v2. The base idea of these modules, which are named after the movie with the same name, is, roughly speaking, to allow the network to explore multiple convolutions and subsequently "choose" which ones, or which combinations, it finds the most preferable to solve specific tasks. Similar to the ResNet, the motivation behind the development of the inception architecture is to create larger, meaning deeper and wider, networks in order of increasing performance. The base idea is inspired by enabling sparse connections within the network, which means that not all of the inputs necessarily connect to each output, as in the traditional ANNs and CNNs. This would allow to go deeper through not "wasting" computing power on near-zero weights that are unimportant.

An illustration of the initial inception module is presented in Figure 13. As can be observed, instead of merely being subject to a single convolution operation, the output of the previous layer is subjected to 3 different convolutions, with filter sizes 1x1, 3x3, and 5x5, in addition to a max pooling operation. These are subsequently concatenated in a *filter concatenation* and passed to the subsequent layer. The illustration in Figure 13 is simplified, since this module also entails multiple dimension reductions in order to achieve compatibility across layers.



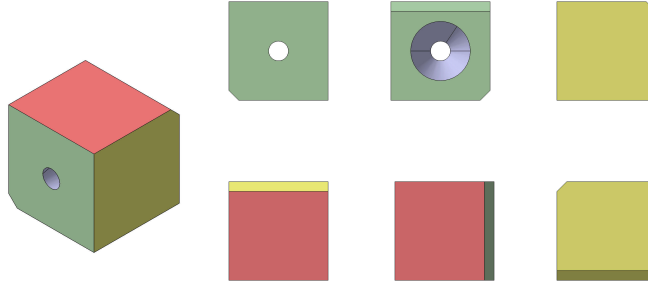
**Figure 13** – Inception module, as presented in [58].

The complete architecture of the Inception-ResNet-v2 is too large and too complex to include in the discussion here. Instead, it is referred to the paper presenting the architecture ([57]) for more detail. However, the base principle in the architecture is, as previously mentioned, to combine the residual connections in Figure 12 and the inception module in Figure 13. This is achieved by including a residual connection (direct connection) between the previous layer and the concatenation in several manipulated inception modules. The final output of the convolutional layers is connected to a single fully-connected softmax activation output layer, in the same fashion as the ResNet in Figure 12.

### 3.3.2 Representation of 3D objects

At the beginning of the section on CNNs, it is stated that CNNs primarily and traditionally are used for image recognition. The reason for this perhaps vague formulation is that recent works also explore CNNs that take 3D inputs instead of the traditional 2D images. Some of these works, ones that employ voxel model inputs, are already presented in Section 2.

The reason for this new exploration is induced by the increasing application of artificial intelligence to multiple real-world tasks. Since the real world is 3-dimensional, sufficient detail about real objects is not always achieved through single 2D images. Multiple different methods of representing 3D objects, or shapes, to artificial neural networks have emerged over the last years, including the previously mentioned voxel representation [9] [10] [36], point cloud representation [59], mesh representation [60], and multi-view representation [61]. Several different methods of representing



**Figure 14** – Illustration of a multi-view representation of a cube.

3D objects to neural networks are reviewed in [62]. Of the reviewed approaches, the multi-view representation is considered amongst the best. Point cloud representation methods also perform well in the study, but the effort involved in implementing and modifying the state-of-the-art models with this representation is considered significantly higher than for the multi-view approach. Because of these considerations, this work will focus on the multi-view representation. This section will therefore aim to provide an overview of the multi-view method. Many previous works on this subject are also evaluated in Section 5.3.2.

The **multi-view CNN** was introduced in [61]. The base concept of this method is to represent a 3D object through multiple images captured from different views around the object. That is, rather than attempting to present the network with some 3D representation directly, the 3D object is represented by a number of 2D inputs. Compared to, for example, the voxel representation, this drastically reduces the number of inputs. A  $224 \times 224$  color image, for example, constitutes  $224 \times 224 = 150528$  pixels, or inputs. Multiplied by, for example, 7 views, this results in a total of around 1 million inputs. Even though that is a high number of inputs, it is substantially less than the over 11 million voxels in a  $224 \times 224 \times 224$  voxel model. The multi-view CNN additionally gains from having the potential to train parts of its architecture on single images, before extending the single-view model into multi-view.

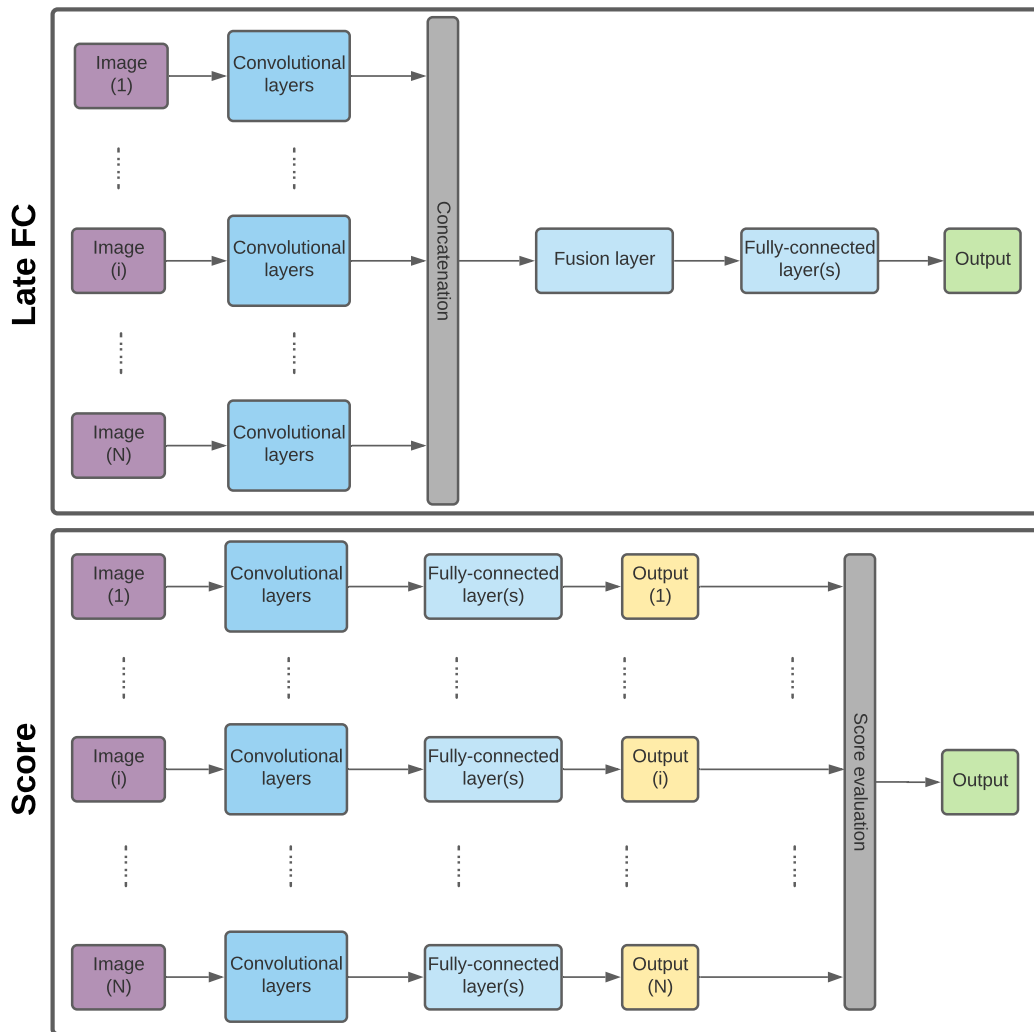
An illustration of a multi-view representation of a cube is available in Figure 14. Here, the corresponding faces on each side of the cube have the same color, and some features are included to highlight the different sides.

## Multi-view fusion

An important question when using multi-view CNNs to evaluate 3D objects is how to combine the information from the different images in order to gain more information from the multi-view representation than from a mere single-view image of the object. That is, how the information collected from each image can be fused into a unified network output in the best way. There are many different methods to do this, several of which are reviewed in [63]. In all of these fusion techniques, the convolutional architecture of choice can be employed. In other words, the fusion techniques are not specific to a certain architecture but rather general methods to fuse information from multiple CNNs. This work will focus on two main fusion techniques, namely late fully-connected (FC) fusion and score fusion. The reason for choosing these is that they are two of the techniques that perform the best in [63]. Additionally, they are fundamentally different, and both have clear advantages and disadvantages, which are discussed below.

Since this work will consider late fully-connected and score fusion, only these will be further discussed here to provide an overview. For more details on these and other fusion techniques, [63] can be consulted. The discussions in this subsection are based on topics from that paper and from the multi-view papers in Table 4 in Section 5.3.2.

An illustration of the late fully-connected and score fusion techniques are presented in Figure 15. In **late fully-connected fusion**, presented on the top, the different view images of the 3D object are fused using a fully-connected *fusion layer*. First, the N input images are processed through all convolutional layers in the CNN architecture of choice, before the output from the last layers for each view are concatenated, meaning stacked together. This concatenated array is subsequently input into the fully-connected fusion layer, and the output from this layer is then passed through the final fully-connected layer, or layers, in the complete CNN architecture to get a unified output. In the case of the VGG (Figure 11), for example, the *convolutional layers* in Figure 15 would be all layers up to, and including, the final maxpool layer, and the *fully-connected layer(s)* would be the 2 FC4096 layers and the final softmax output layer. [63] finds this fusion technique to be the strongest out of all the ones that are tested. Since the late FC fusion technique includes a trainable fusion layer, it enables the model to evaluate and learn relations across the different images. That is, the weights in the fusion layer can be tuned so that the model can recognize patterns also in between the views. On the other hand, this also results in more tunable parameters than, for example, the score fusion techniques.



**Figure 15** – The late fully-connected (top) and score (bottom) multi-view fusion techniques.



**Score fusion** is illustrated in the lower part of Figure 15. In fact, score fusion is not a unique technique in itself but a collective term for several different possible evaluations that can be made on the outputs of a complete CNN for each individual view in order to determine a unified output. That is, in score fusion, all  $N$  views are evaluated independently in full by a stand-alone CNN to provide some prediction for each view. These individual predictions are subsequently subject to a score evaluation, in which some unified output is determined. The stand-alone CNNs for each view are generally, and always in this work, merely copies of the same CNN that has been trained on single-view images. For classification problems, this work will focus on two score evaluations, namely product-score and what will be termed *democratic-score*. For regression problems, the evaluation will be different, something that is discussed in Section 6.5.

In **product-score fusion**, the final output is determined by an element-wise multiplication of the individual output vectors from each view. That is:

$$y_{\odot} = \prod_{i=1}^N y_i \quad (18)$$

where  $y_{\odot}$  is the product-score output,  $N$  is the number of views, and  $y_i$  is the output of view  $i$ . The final predicted class is then the index of  $y_{\odot}$  with the highest value. This fusion technique performs the second best in [63], after the fully-connected fusion.

In **democratic-score fusion**, the final output class is the one that is predicted the most across each individual view. That is, for example, if class 1 is predicted by 5/7 views in a 7-view multi-view problem, then the final output is chosen as class 1. Instead of considering the individual performance on each class in each view, like the product-score, this fusion technique hence evaluates the actual prediction from each view. The democratic-score fusion technique is included because it is considered to be of interest to explore how the results for this technique deviate from the product-score models. Additionally, it constitutes little additional effort to consider also democratic-score when already considering product-score.

As opposed to late fully-connected fusion, is score fusion, as presented here, a "stupid" fusion technique. That is, no intelligent evaluation based on the interrelations between different views is made. Instead, a simple rule-based evaluation is performed to fuse the information from the different views. It could, however, technically be possible to train a network layer that evaluates

the concatenated vector of the outputs from each individual view to predict the final class, and, as such, create what could be considered an intelligent score fusion. This is not done in this work, and score fusion will hence be considered a stupid fusion technique throughout this work.

Even though the presented score fusion techniques are not able to evaluate patterns in between the views, they also have advantages when compared to late FC fusion. Since the score fusion techniques merely perform a rule-based evaluation, this means that there are no additional tunable parameters in the network compared to a single-view model. Rather, a CNN can be trained fully on single-view images and merely duplicated to create the multi-view model. This significantly reduces the training effort compared to late FC fusion models, which need to also be trained in multi-view. A reduced training effort means both higher efficiency and, more importantly, that fewer training examples potentially would be required to train such models to achieve satisfactory performance. An important difference between training in single-view and multi-view is that the single-view training can benefit from considering every image from every view of an object as an independent training example. This effectively means that there are many more training examples available for the single-view training than for the multi-view training. On the other hand, the multi-view image sets that are evaluated during multi-view training provide more complete information per example since the object is viewed from multiple sides.

### **Number of views**

A final question when using multi-view images to represent a 3D object is how many views to consider. That is, how many images from different sides of the object that should be evaluated by the network.

In this work, a 3D object will be represented by seven views. Somewhat following the approach in [37], six views were initially considered. Since the simplest 3D object, namely a cuboid, can encapsulate any 3D object (the bounding box), imaging a component from each side of such a cuboid would capture all sides. However, upon investigating these six views of a component created in the global coordinate system, it is observed that the six views in several cases only capture single faces each. That is, no view presents a more complete overview of the component as a whole. This can be observed in the six images to the right in Figure 14. As a result, it is considered preferable to also include a view in which more faces are visible at the same time. The isometric view, which is included to the left in Figure 14, is such a view and is therefore also included.

The number of views,  $N$ , in Figure 15 is hence 7 in this work. In the event that the results obtained during the development performed in this work are found to be insufficient, more views can be tested to increase the performance.

## 4 Approach

In this section, an outline of the approach used to solve the problem in this work is first presented. Subsequently, the target entities of the work, namely the cost drivers, are defined, elaborated on, and evaluated. Finally, the choice of method to pursue for identification of each cost driver and the development of these methods are presented and discussed.

### 4.1 Outline

To develop a system for automatic identification of cost drivers from 3D CAD representations of sheet metal components, the following approach is applied:

1. Identification of cost drivers
2. Evaluation of each cost driver
3. Development of methods to identify cost drivers
4. Development of system architecture
5. Data procurement and preparation
6. Development of system

First, the various component entities that are of importance to estimating the cost, namely the cost drivers, are established. Second, each cost driver is evaluated to determine its dependencies and the most feasible methods to automatically identify it, starting from a CAD representation of a component. Subsequently, the specific methods to pursue for identification of each cost driver are determined and developed.

Once the base information about all cost drivers and the methods that will be employed to identify them are established, the implementation can begin by theorizing an architecture for the automated evaluation system which outlines the flow of data and information through various evaluation modules. With the evaluation modules clearly defined, the data that will be required for their development can be identified, which enables the start of the process to obtain and prepare this data. Finally, with the data acquired, the technical sides to the work can begin by developing and testing each of the modules in the system architecture.

## 4.2 Cost drivers

The first stage in this work is to identify which component entities that are of importance to estimating the cost. These component entities are henceforth referred to as cost drivers, and they are the targets of the automated evaluation system to be developed. Since the system is intended to be assembled with an existing analytical cost estimation model, many of the required system outputs are predefined. However, it is still important to break down this model and identify the most important and actually cost driving model inputs in order to discard data that is only or primarily included for human readability or used as supporting information to other non-relevant systems. The model analysis and identification of key cost drivers are performed in collaboration with cost engineers at the automotive company where this work is carried out.

Cost driver	Area of importance	
	Production costs	Tooling costs
Welding information	X	
Blank size	X	
Part mass	X	
Material type	X	X
Sheet thickness	X	X
Part size	X	X
Fastener information	X	X
Production method	X	X
Number of parts per stroke	X	X
Production volume	X	X
Place of production	X	X
Number of operations	X	X
Process plan		X
Part surface area		X

**Table 1** – Overview of all identified cost drivers and their area of importance.

The final overview of the identified key cost drivers can be found in Table 1. In this table, the first column lists the names of the various cost drivers that are identified. The other two columns indicate which area of cost the specific cost driver is of importance to. The table is sorted to start

with the cost drivers that are only of importance to the production costs and end with the cost drivers that are only of importance to the tooling costs. That is, the table is not sorted to indicate the relative importance of the cost drivers. As can be observed in the table, most of the identified cost drivers are of importance to both production and tooling cost estimation.

The following subsections are dedicated to elaborations on the presented cost drivers. They will explain each cost driver, discuss why this entity is of importance to estimate the cost of a component, outline its dependencies, and include an evaluation of possible ways to determine the cost driver in an automated manner through analysis of a 3D part model in a CAD native format. The information about all cost drivers and their importance in the cost estimation process is acquired through interviews and discussions with cost engineers at the automotive company where the work is carried out.

#### **4.2.1 Welding information**

Welding is a technique that is used either to join two or more individual components together or to stiffen a singular component by welding it to itself. Whether or not an individual component or small assembly requires welding influences the production costs for this component. This is because the welding contributes an additional process step. One cost is associated with the additional process setup, and subsequent costs are generated the longer the component spends in the additional process.

There are several techniques for welding, such as TIG welding, MIG welding, projection welding, and spot welding [64]. For sheet metal components, spot welding, or resistance spot welding, is the most used [65], and what will be considered in this work. Spot welding is a welding technique in which metal sheets are joined by a number of points, or spots, that are created by applying electric current through the sheets using electrodes that clamp the sheets together from both sides.

Since spot welding is the only technique that will be focused on in this work, and because the costs associated with spot welding relate directly to the number of actual weld spots, or points, identifying this cost driver can be broken down to determining the number of welding points.

Some previous works on this subject aim to reduce the number of weld spots that are used to join metal sheets through optimizing the spot weld layout on a flange using finite element analysis [65][66]. The intention is to reduce the machining times and production costs by including

fewer weld spots without compromising the structural integrity through switching from the more traditional uniform spot weld layout. These works represent one approach to predicting the number of weld spots used to join sheets, namely to perform a finite element analysis of the component. To perform the analysis, the flanges to be welded need to be identified, and subsequent detailed information about these flanges is required. In other words, in order to implement the approach in an automated system, this information would need to be automatically extracted prior to launching the analysis.

Spot welding is dependent on a number of factors, such as mechanical, metallurgical, thermal, and electrical phenomena [67]. Based on these dependencies and the experiences of development engineers, the number of welding spots is assumed to be primarily influenced by the thicknesses, materials, and contact areas of the joined sheets. As a result, an approach that utilizes this information to estimate the number of welding points can also be attempted. Since it is proven that an artificial neural network can approximate any given continuous function arbitrarily well [48], a fully-connected neural network approach can be attempted to approximate the number of welding points from those inputs. Such an approach would be similar to several of those previously outlined in Section 2.3.2, that aim to predict values traditionally obtained with FEA.

In order to automate an approach that employs ANNs to predict the number of welding points, the question is then how the materials, thicknesses, and contact areas of an individual component or the components in a small assembly can be identified, such that they can be used as inputs to the network. Since both the material type and sheet thickness are other cost drivers, these already need to be identified automatically by the overall system. The remaining task that is specific to this cost driver is then to automatically extract the contact areas between sheets within an individual component or between the components in a small assembly. An API program for such an evaluation is readily available, and the task is then merely to implement this program into the finished system.

#### **4.2.2 Blank size**

The blank of a sheet metal component is the initial workpiece from which the component is manufactured. This workpiece is a flat sheet that is cut from a larger coil or plate in preparation for the production process. The 2-dimensional size of this blank is of importance to the production costs because it, together with the sheet thickness, establishes the amount of raw material needed to manufacture the component. The raw material costs are generally considered to constitute the

primary part of the production costs.

Since the blank geometry is determined by the geometry of the desired finished product, the blank size is dependent on both the size of the product and its geometrical shapes. In addition to this, the blank size depends on the production method that is applied to the component because different methods use different tools and feeding mechanisms.

The main method for simulation of sheet metal forming, which includes prediction of the blank geometry, is the finite element method (FEM) [20]. One such approach, which requires short computation time, is the one-step inverse approach [22]. Recent works also explore inverse isogeometric analysis [68][69], which attempts to overcome certain shortcomings in FEM related to time-consuming setups and inaccurate shape descriptions.

Several software solutions that employ these methods to determine this cost driver are available on the commercial market. An example of such a software is AutoForm [70]. However, neither AutoForm nor any other software solutions that are discovered are possible to fully automate through, for example, the use of an API. As such, they can not be directly implemented into the automated system this work aims to develop.

Since the blank size is dependent on the part geometry and size, an approach that attempts to utilize these dependencies directly could potentially estimate this cost driver. For geometrical description, convolutional neural networks have proven efficient in many applications, such as the description of 3D shapes [61] and the identification and classification of 3D part features [38]. As a result, a possible approach to determine the blank size could be to utilize such a network in combination with spatial information.

### **4.2.3 Part mass**

The mass of a part is important to estimate the production costs. Specifically, this cost driver is linked to the calculation of the material costs for the component. The part mass is used to determine a material utility factor, which is calculated as the ratio of the final part mass and the mass of the blank. This material utility factor is then used to evaluate how much scrap material is left from the production process. Since the scrap typically is sold and/or reused, the amount of scrap material is of importance to calculating the net material costs, reducing the total amount by subtraction of the scrap value from the original purchase value.



The mass,  $m$ , of a part can be calculated by multiplying the part volume,  $V$ , and material density,  $\rho$ , as:

$$m = V \times \rho \quad (19)$$

In other words, to automatically determine the part mass, the material density and volume need to be identified in an automated manner. The material of the component is a cost driver and will hence already be automatically identified by the system. The material density can, in turn, be found automatically through, for example, cross-referencing an existing material index. The volume of the component can be extracted using either an API or through analysis of, for example, the STL representation of the component. The STL representation can be generated automatically from the CAD native format. When knowing the material density and volume, an automated solution of Equation 19 is arbitrary.

#### **4.2.4 Material type**

Different components have different load cases, weight requirements, and joining requirements, which in turn lead to multiple different materials being used in different components. The different types of materials have different costs, which means that the choice of material type influences the production costs of the component. Since different materials have different strengths and ductility, different material types pose different force requirements to the tools used for manufacturing them, which in turn influences the tooling costs.

Since the material generally is applied to components in their CAD representation, this cost driver can be identified automatically through the use of an API. If the material is not applied, this cost driver would need to be a manual user input.

#### **4.2.5 Sheet thickness**

For the same reasons that different sheet metal components use different materials, they also have different sheet thicknesses. The sheet thickness is important in combination with the blank size and material density to calculate the mass of the raw material needed to manufacture a single component. Material prices are generally provided per unit mass, and, as such, this raw material mass enables calculation of the raw material cost, which is a part of the production costs.

Additionally, sheet metal coils and plates of different thicknesses have different costs because the difficulty of manufacturing them varies depending on how thin the sheets are.

For tooling costs, the sheet thickness is important because the force required to shape and cut the sheet depends on this entity. As a result, the thickness of a sheet impacts the requirements for the cutting steels and the forces that need to be exerted by a press during production.

An API program is readily available that evaluates a CAD component and returns the sheet thickness. As a result, automatic identification of this cost driver only entails implementation of the existing program in the complete automated system.

#### 4.2.6 Part size

The part size is a cost driver that is relevant for estimating both the production costs and the tooling costs. The part size refers in this setting to the dimensions of the minimum bounding box of a component. The minimum bounding box, or minimum volume bounding box, is the smallest cuboid that can encapsulate a part. The minimum bounding box is not to be confused with the global bounding box, which is the smallest cuboid with sides parallel to the XY-, XZ-, and YZ-planes that can encapsulate a part.

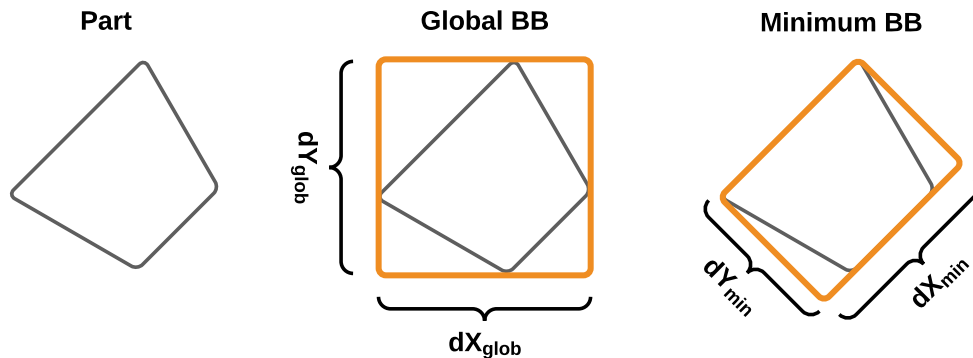


Figure 16 – Illustration of 2D bounding boxes.

An illustrative example of the global and minimum bounding boxes in 2D is provided in Figure 16. The 2D bounding box is chosen as an illustration for simplicity purposes, but the same principle applies in 3D. As can be observed, the global bounding box has sides parallel to the horizontal axis (x-axis) and the vertical axis (z-axis), whereas the minimum bounding box is rotationally invariant and is simply the smallest rectangle that fits around the part. The bounding box is then provided

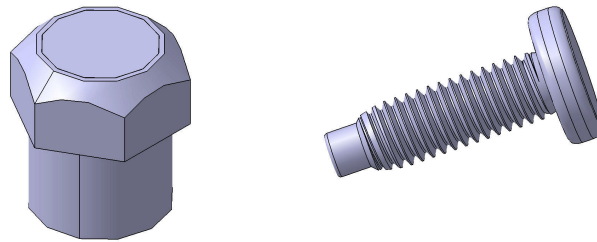
as a measurement on the form  $BB = [dX, dY]$  in 2D, or  $BB = [dX, dY, dZ]$  in 3D, where the indices relate to the lengths of the sides of the bounding box.

The part size is important to the costs of a component because it influences the choice of press that is used for production. Different presses have different tool requirements and purchase costs and impose different hourly operation rates, which, in turn, influences the tooling costs and production costs, respectively.

A mathematical approach to finding the minimum bounding box was presented by O'Rourke in 1985 [71]. This approach can be implemented by considering each triangle vertex in an STL representation of a component.

#### 4.2.7 Fastener information

Information about the fasteners that are used in a component or small assembly is a cost driver for both the production and tooling costs. A fastener can be either a bolt or a nut, and they are used as joining elements inside assemblies and in between assemblies. An example of a nut is provided to the left in Figure 17, whereas an example of a bolt is provided to the right. For the two main categories, namely bolts and nuts, several subcategories exist. For the costs, however, only three distinctions are of importance. These are whether a fastener is an impress element, a welded element, or neither.



**Figure 17** – Illustration of fasteners.

An impress element is a bolt or a nut that is pressed into the sheet metal component in such a way that the two are permanently joined by mechanical force. An impress nut could, for example, be

pressed into a sheet metal component to enable a bolt to be screwed directly into the component.

A welded element is a bolt or a nut that is welded to a sheet metal component in order of permanently joining the two. The end functionality of a welded element and an impress element is generally the same, only the techniques with which they are joined to the sheet metal component differ.

For the production costs, information about the fasteners used in a component or small assembly is of importance because the expense related to the purchase of these parts are included in the calculation of the production costs. As such, to estimate these costs, the specific fasteners in a component or small assembly must be identified. Additionally, subsequent costs are generated if the fastener is a welded element or an impress element since these produce additional process steps and increase the production time.

For the tooling costs, only information as to whether or not the fastener is an impress element is of importance. This is because the impress elements must be inserted into the components inside of the press, requiring the press to have functionality consistent with performing this task. The costs will vary based on the direction from which the impress element must be inserted. This is because the press needs to be equipped with different tools based on whether the fastener is placed from the top side, the bottom side, or in an orientation not perpendicular to the main working plane. This means that also the orientation of the impress element must be found in order to make an accurate tooling cost estimation.

Previous works dedicated to fastener identification employ learning-based methods to solve this task. [72] uses a combination of support vector machines [73] and neural networks for identification of fasteners. The approach requires two input images, one of which contains a reference object to evaluate the size of the part. [39] uses a convolutional neural network (CNN) to recognize a total of 29 different bolts, nuts, and washers from real-life images captured from a constant camera position. Because of the consistent image scale due to the constant camera position, no additional reference object must be included to obtain information about the part size to enable classification. Using a reference image for visual calibration, [40] presents a system that employs a CNN for fastener identification in which the camera position does not need to be constant. The part size is derived from the calibration with the reference image. The approach utilizes CAD-generated synthetic data for real-life recognition of 10 fastener types, namely five diameters each of both bolts and nuts.

The above works present successful approaches to fastener identification through the use of state-of-the-art machine learning methods. The current use case deviates somewhat from these in that the identification will focus on digital models rather than images of physical objects. Nevertheless, a learning-based method combined with some spatial information is considered a promising approach for fastener identification.

#### **4.2.8 Production method**

Different production methods are applied to different components based on their size and the operations that are required to transform a blank into the shape of the final component. The production method that is applied to a component influences both the production and tooling costs. The influence on production costs comes in large from the different hourly rates imposed by the different methods. For the tooling costs, the influence is primarily related to the different tool specifications and purchase costs associated with the different methods.

For sheet metal parts, the three major production methods that are separated between are ProgDie presses, Transfer presses, and Press lines.

The ProgDie, or progressive die, is a press in which a sheet metal coil is fed directly into the press itself. Within the press, several stations perform various operations directly on the coil, shaping a component. In the last station, the finished component is cut from the coil. ProgDie presses are typically used to manufacture small to medium-sized parts that rely on stamping operations. This is usually the cheaper of the three options, but it is limited in that it cannot perform deep drawing operations or manufacture larger components.

A Transfer press has a similar principle to the ProgDie press in the sense that material is moved from station to station until a finished component emerges in the end. In a Transfer press, however, instead of feeding a sheet metal coil directly into the press, blanks for each part are mechanically moved through the line of stations. That is, each blank is suspended between two rails that move it through the press. The benefit of Transfer presses compared to ProgDie presses is that the part is free from the coil as it is moved through the stations. This enables other machining operations, such as deep drawing, to be performed on the part. Transfer presses are typically used for the manufacturing of medium to medium-large parts. Manufacturing parts with a transfer press is usually more expensive than manufacturing them with a ProgDie press (if this is possible) but

cheaper than using a Press line.

Press lines are large presses in which multiple operations are performed on a component blank that is static within the press. This is instead of moving the blank from station to station, as in Transfer presses. Typically, the blank is either not moved at all or moved only once between such large presses before the component is finalized. As opposed to Transfer and ProgDie presses, which are often considered flexible, these presses are generally more customized for the production of a specific component. Press lines are usually used to manufacture medium to large components, and they are typically the more expensive of the three production methods. Several parts exist that can be manufactured using either Transfer presses or Press lines. These parts are sometimes called Deep drawing parts or Press parts, and the choice of press often comes down to logistics relating to which presses that are available at the relevant place of production at the time of production.

From the above, it is clear that the choice of production method is dependent on both the part geometry and its size. The geometrical dependency primarily comes from determining whether the part requires a deep drawing operation or not, as only Transfer presses and Press lines can perform such operations. Additionally, since there are different size limitations to each method, the part size alone is also an important factor in identifying the appropriate production method.

The problem of identifying the appropriate production method to use on a component can be considered a part type identification problem. This is achieved by simply considering parts for which a specific production method is used as members of the same part type. For example, parts that are produced in ProgDie presses can be considered "ProgDie parts".

Recent works on part type identification employ learning-based methods to directly classify parts from images. [41] presents an approach to identify 24 different automotive parts using real-life images. Their system uses a convolutional neural network (CNN) that is trained on synthetically generated CAD data for the classification. [42] presents a sheet metal part identification system for increasing automation during the production process. Their approach employs a CNN to evaluate real-life images of laser cutting components and identify them in comparison to CAD part images.

Since convolutional neural networks have proven efficient in part type identification problems, this is considered to be a promising approach to identify the appropriate production method for a component.

#### 4.2.9 Number of parts per stroke

The number of parts per stroke is a cost driver that refers to how many parts that are in the same station of a press at the same time. That is, how many components that are manufactured simultaneously. These components can be either identical, mirrored, or, in rarer, cases completely different. The number of parts per stroke influences both the machining time and the tool designs, which means it impacts both the production and tooling costs.

The number of parts that are simultaneously manufactured depends on both the part geometry, size, potential sibling components, production method, and logistical considerations. A sibling component here refers to a mirrored part. In vehicles, for example, where there is a lot of symmetry between the left and right sides, such component siblings are common. Since these parts are mirrored images of each other, they generally require the same or similar machining operations, and their blank shapes often fit nicely together. In turn, this enables simultaneous manufacturing. The dependency on the production method relates to the different general specifications relevant to the different methods. The logistical considerations relate at large to what specific presses that are available at the place of production at the time of production. The five dependencies of this cost driver are considered in light of each other to evaluate how many parts to include per stroke in a press. If, for example, a smaller part is to be manufactured, but only larger presses are available, the choice could be to include two of these smaller components inside the press at the same time, provided this is feasible with regards to the geometry. If, however, a smaller press is available, perhaps only a single part would be manufactured per stroke using that press.

Determining this cost driver is an intricate job, consisting primarily of weighting logistical factors with cost impacts and geometrical and spatial considerations. No clear rules are employed by cost engineers when determining the cost driver, who rather consider each case independently. However, such rules could exist, and an investigation into the data linking the available presses and the part geometry and size could, therefore, potentially reveal rules that can be employed to automatically identify this cost driver. Exploring these dependencies with a neural network is also considered a promising approach due to this method's previously discussed abilities in function approximation and geometrical description.

#### **4.2.10 Production volume**

The production volume refers to how many specimens of a component that will be manufactured in total. This influences both the production and tooling costs because some of the expenses related to each of them are one-time costs that need to be distributed across all single components to allow for accurate cost estimates per part. If, for example, a new tool needs to be purchased for the manufacturing of a component, this can constitute a large one-time expense. However, to then estimate the cost impact on each of the single components, the expense must be distributed across all the specimens that will be manufactured over the production lifetime. This will then generate a per-part impact of the expense, which in turn can be added to the other costs estimated for the component to achieve a complete cost estimate.

It is not feasible to determine the production volume through analysis, since this is a value that relates more to the visions and ambitions associated with the product in which the component is used. As a result, such a value would need to be a manual input to a cost estimation system.

#### **4.2.11 Place of production**

The place of production influences both the production and tooling costs. This is due to a number of factors, such as the general price level for various services and products in the country in question, labor costs, logistical considerations, and more.

The place of production is, as the production volume, not feasible to determine through analysis. As such, this would need to be a manual input to a cost estimation system.

#### **4.2.12 Number of operations**

The number of operations is a cost driver that refers to the number of machining operations that need to be performed on a component blank to transform it from its initial state into the finished component. Such operations can include, for example, cutting, piercing, and bending of the blank in order to achieve the final geometry.

The number of operations influences both the production and tooling costs. It affects the production costs because it can influence the choice of press that is used, which, in turn, could influence the hourly rates for production. However, this cost driver is generally found to have little to no importance for the final production costs.



For the tooling costs, the number of operations is of greater importance. This is because it describes the number of different tools that would need to be created to manufacture the specific component, as well as influencing the overall design in the press. However, more information than just the total number of operations is typically required to make full and accurate analytical tooling cost estimates. In fact, a complete process plan (see Section 4.2.13) would be needed to do this.

Previous works related to this subject are found to generally aim at establishing a complete process plan through rule-based approaches, rather than just estimating the total number of operations or processes included within such a plan. Some of this work is summarized in Section 4.2.13. However, estimating the mere number of operations itself is a somewhat different task that is related to the geometrical complexity of a component. That is, a geometrically complex component typically requires more operations than a geometrically simple one. As a result, since convolutional neural networks have proven efficient in tasks relating to geometrical description, using such an approach for estimation of the number of operations could be a possibility. Such an estimate could prove to be of use to a rule-based scheme to determine the complete process plan.

#### **4.2.13 Process plan**

As mentioned above, the tooling costs depend on the complete process plan. This plan is a detailed outline of the specific operations or processes that need to be performed on the component blank to transform it from its initial state into the final geometry. Both the specific processes and their order play a role in this plan. The same type of process may be repeated multiple times if needed. An example of a simplified plan could be:

1. Drawing
2. Cutting
3. Bending
4. Piercing
5. Cutting

Additionally, detailed information about each process is needed to perform accurate analytical tooling cost estimates. This detailed information includes the drawing depth of the component, as

well as information about the flanges, holes, cuts, and potential cams.

A cam is a device that translates vertical force to a different direction than the vertical working direction. This is not to be confused with computer-aided manufacturing, which is often abbreviated CAM. A cam is used when a machining process must be performed in a plane other than the horizontal working plane. Cams are necessary because presses generally only are capable of exerting vertical force. As such, if, for example, a hole needs to be created in the side of a component, this vertical force must be translated to the direction of the hole center axis, such that a tool can create the hole in the desired location. In some cases, such holes can be manufactured before the part is bent, so that the use of a cam is not necessary. In other cases, however, a bending operation could lead to distortion of the hole, which generates the need to create the hole after the bending. Cams are expensive additional devices that need to be included in the press, which impacts the total tooling costs.

The drawing depth of a component is the distance from the component working plane to the lowest point created by a drawing operation. The working plane is the horizontal plane which is coincident with parts of the top surface of the component throughout the manufacturing process. Initially, this will often be the top surface of the blank. In order to determine the drawing depth, both the drawing direction and the working plane must be known. Having established these, the drawing depth can generally be estimated by measuring the distance from the working plane to the lowest point on the part in the drawing direction. The drawing depth influences the drawing tool specifications and the force that needs to be applied by the press on the blank, which both in turn influence the tooling costs.

Information about the different flanges is important to establish the requirements for the bending tools in the press. This information would include the length of the edge that is bent, the area of the bent flange, as well as the direction of the bend. The first two influence the specifications of the bending tool itself, whilst the direction of the bend influences where the bending tool needs to be mounted inside the press. By default, presses usually only have moving components in the top part. As a result, if a flange needs to be bent in an upwards direction, this would generate the need for a subsequent tool to be mounted in the lower parts of the press in order to push the flange upwards. This would generally lead to higher costs. The bending direction also influences the use of cams in the cases where a flange cannot be made directly through the application of only

vertical force.

Information about the sizes and shapes of each individual hole in a component is also of importance to estimating the final tooling costs. The size of the holes determines if standard punches can be used or not. It is generally cheaper to buy standard components than having to manufacture the punches entirely in-house. Since standard punches typically are round, also the shape of the hole has an impact on the costs. This is because a bought standard punch with a round shape would need to be modified in-house before it could be used in the manufacturing of a non-round hole. Even if a hole is of a size for which a standard punch could not be used, it will impact the costs if this hole is round or non-round. This is because different manufacturing techniques would need to be employed to produce the custom punch. From the above, it can be concluded that the hole information generally can be provided in a binary format for the two cases of round/non-round and standard size/non-standard size. This, in turn, means that the information can be concentrated down to four possibilities:

- Round hole that can use a standard punch
- Non-round hole that can use a standard punch
- Round hole that cannot use a standard punch
- Non-round hole that cannot use a standard punch

Additionally, as for the flanges, the direction from which a hole needs to be manufactured influences the tooling costs.

To determine the specifications of the cutting tools used in the manufacturing process, information about the length of each cut to be performed on a component is needed, together with the direction from which the cut is made. The cutting lengths specify requirements for the cutting steels that are used to make the cuts. As for the flanges and holes, the direction from which the cut is made influences if a cam needs to be used and where the cutting tool must be placed in the press (top or bottom), which in turn influences the tooling costs.

Works on the generation of automated computer-aided process plans (ACAPP) for milled components are reviewed and discussed in [74]. The survey presents machining features as the most important input to any computer-aided process planning (CAPP) system. This is also observed

by the detailed information about features, such as holes and flanges, that is required by this cost driver.

Previous works on the generation of process plans for sheet metal components also emphasize the importance of feature information. [75] analyzes STEP files to extract feature information to help determine the process plan in sheet metal bending operations. [76] presents a knowledge-based process planner to be employed in a system for manufacturability evaluation of sheet metal components. This process planner relies on rules relating to component features and general guidelines for process plan generation. [2] proposes a feature-based approach for establishing a process plan for sheet metal components that employs a combination of knowledge-based engineering (KBE) and case-based reasoning (CBR). This approach aims to determine the process plan by comparing the current part to previous parts for which a process plan is known, based on the identified features in the parts.

Since the features in a part are the most important input to automatically generate a process plan, determining this cost driver also entails the identification and classification of such features. STL analysis has previously been employed to recognize the features in sheet metal components using rule-based analysis [15]. More recent works use convolutional neural networks to identify the features in milled parts from both voxel models [36] and images [37] [38]. These approaches are, however, not yet extended to sheet metal components.

There also exists commercial software, such as AutoForm [70], that specializes in deriving detailed process plans for sheet metal components. AutoForm analyzes a component and simulates the production process to deliver good suggestions of operation plans. However, as also mentioned in Section 4.2.2, neither AutoForm nor any other known software solution can be completely automated to deliver a process plan. Nevertheless, the amount of manual inputs required by the software to establish a suggested process plan for a sheet metal component based on its CAD representation is limited. This suggests that full automation could be possible in the near future. To establish a process plan, the primary input to the software relates to which production method that will be used to manufacture the component.

#### 4.2.14 Part surface area

For the tooling cost calculation, also the surface area of the finished component is of importance. This is primarily because the part geometry needs to be milled to create a holder for the part inside of the press. That is, the part surface area is directly related to the area that needs to be milled.

The surface area of a component can easily be extracted from a CAD native format using either an API or through analysis of the component's STL representation.

#### 4.2.15 Cost driver summary

Cost driver	Promising methods for automatic identification		
	API	Rule-based	Learning-based
Welding information		X	X
Blank size		X	X
Part mass	X	X	
Material type	X		
Sheet thickness	X	X	
Part size		X	
Fastener information			X
Production method			X
Number of parts per stroke		X	X
Production volume	-	-	-
Place of production	-	-	-
Number of operations		X	X
Process plan		X	
Part surface area	X	X	

**Table 2** – Overview of methods to identify the various cost drivers.

In the above subsections, the component entities that are required to estimate the production and tooling costs, namely the cost drivers, are evaluated. Table 2 presents a summary overview of which methods that are considered to be the most promising to identify each cost driver automatically from a CAD native format, following the discussions above. In Table 2, rule-based methods include finite element methods, such as the ones that have previously been applied in works relating to

welding information and blank size, in addition to other methods, such as STL analysis, where an evaluation is made based on more heuristic arguments. The term learning-based refers in large to convolutional neural network approaches, which previously have been successfully employed for part type recognition problems where the geometrical shapes of a part are of importance. However, the term also refers to other neural network approaches, such as traditional fully-connected networks, that could potentially be used to link the number of spot welds to its dependencies.

Having established possible and promising methods to automatically identify the various cost drivers, the specific method to pursue for identification of each one can be determined.

### **4.3 Method choices and development**

Table 2 summarizes the knowledge obtained from evaluation of each cost driver. Several of the cost drivers have two methods that are considered promising to automatically identify them from a 3D part representation of a component. As a result, a choice must be made as to which of these methods to pursue. This needs to be done in order to enable further development of the individual methods and the final system architecture.

#### **4.3.1 Choice of methods**

The choice of method is influenced by three primary factors. These are the assumed potential of a method to automatically identify a cost driver, the collaboration between the different methods, and the assumed integration effort of a method. The assumed potential of a method refers to the assumed likelihood of success associated with identifying a cost driver with a specific method. That is, which of the promising methods to identify a cost driver, in the case multiple methods are indicated in Table 2, is more likely to succeed. The collaboration between the different methods refers to how the different methods can complement each other in the final system and in the work to develop it. It is desired to employ similar methods to identify different cost drivers where possible. The reason for this is that employing similar methods would reduce the overall system complexity and enable good communication and interfaces between different system modules. Additionally, employing similar methods would mean that some of the development work is transferable between modules. In turn, this means that the development effort of each individual module is reduced, since unified theories and experiences can be shared. The integration effort is an additional factor of influence in itself because the work that is carried out needs to fit within certain time frames.

The final choice of methods to pursue for automatic identification of the various cost drivers is presented in Table 3. The rest of this subsection is devoted to explaining these choices.

Cost driver	Chosen methods for automatic identification		
	API	Rule-based	Learning-based
Welding information			
Blank size			
Part mass			
Material type			
Sheet thickness			
Part size			
Fastener information			
Production method			
Number of parts per stroke			
Production volume	-	-	-
Place of production	-	-	-
Number of operations			
Process plan			
Part surface area			

**Table 3** – Overview of which methods that are pursued for automatic identification of each cost driver in this work.

The first and most important choice that can be observed in Table 3 is the decision to exclude the process plan from the remainder of this work. This is indicated by the red color on all methods for this cost driver in the table. There are three main reasons for this choice. The first reason is that the implementation effort associated with this cost driver is assumed to be very high and that few other cost drivers can benefit greatly from this effort. In [26], it is, for example, disclosed that previous works on establishing process plans for a sheet metal forming process are highly complex, and that previous success is limited. As discussed in Section 4.2.13, automatically establishing the process plan does not only entail the design of a complex rule-based system that evaluates which operations that need to be performed when and how in order to turn the initial blank into the final component, but also the automatic identification and analysis of all the component features. No state-of-the-art

approach for automatic feature recognition (AFR) is found to be readily applicable to evaluate all necessary sheet metal features. Work relating to AFR also constitutes a significant additional effort in order to collect and structure data from which rules can be generated, or learning-based methods can learn. For learning-based methods, this would entail systematic manual labeling of large amounts of data or an attempt to synthetically generate such data. The effort involved in establishing an exact process plan alongside the effort involved in identifying all other cost drivers is, in other words, not considered feasible within the scope of this work. The second reason for the choice to exclude the process plan from this work is the existence of software that can perform semi-automatic part evaluations to propose such a plan. This means that fully-automated control of these software solutions could be possible in the near future. As a result, the high integration effort may not pay off since such a software solution potentially could be implemented directly into the system in the future. The system will already evaluate part entities that are important for the software in order to establish the process plan, such as the production method, enabling their potential implementation. The third reason for excluding the process plan is that this cost driver is one of only two cost drivers that is only required for the estimation of tooling costs. The other is the part surface area, whose identification is arbitrary and not of importance to the discussion here. Since this cost driver is not of importance to the production costs, it means that the system still can be engaged to determine a full-worthy cost estimate for the component in terms of production costs. It does, however, mean that the following work will be limited to production costs specifically.

Returning to Table 3, it can be observed that a learning-based approach is chosen for the identification of both the welding information and the blank size. For both cost drivers, the other found alternative is finite element analysis. For the welding information, the choice to attempt a learning-based method is influenced by the high integration effort associated with the identification and analysis of the flanges in a component, which would be required by the rule-based method. This circles back to the efforts related to automatic feature recognition in sheet metal components discussed above. For the blank size, the choice is much influenced by the fact that the dependencies of the blank size, namely geometrical and spatial information, are shared with the fastener information and production method cost drivers, both of which will employ learning-based methods. As a result, it is assumed that the blank size can employ a similar method, which reduces the implementation effort significantly in comparison to the rule-based method in this case. If the learning-based method were to fail, a finite element method or a collaboration with a software



provider whose software specializes in blank optimization could be attempted.

For the number of parts per stroke, both rule-based and learning-based approaches are indicated as promising in Table 2. This is because the dependency between part geometry, size, and plant logistics is considered possible to explore with both methods. Unfortunately, however, no complete database containing the required logistical information is available or obtainable within the scope of this work. This means that the logistical considerations, which are assumed to be of great importance to determine the cost driver, cannot be made. Additionally, no data is available on the existence of sibling components, which is another important assumed dependency. Without these, prediction of this cost driver could prove challenging. Nevertheless, it is decided to explore whether there exist similarities related solely to the geometry and size between the components of a certain production method for which the same number of parts are manufactured per stroke. Since convolutional neural networks have proven efficient in describing geometry, a learning-based method is considered to be the most promising and will be pursued. This choice is additionally supported by the fact that the learning-based method that can be applied here, based on the dependencies on geometry and size, is assumed similar to those that will be employed for the fastener information, the production method, and the blank size.

The number of operations is a cost driver that is closely related to the process plan. Since the process plan is excluded from further work, as discussed above, and the number of operations is found to generally have little to no influence on the production costs, a question arises as to how to proceed with this cost driver. Because of its dependency on geometry and size, it is assumed that the number of operations potentially could be identified by means of a similar method to those used for the other cost drivers with this dependency. Since the number of operations could have some influence on the production costs and, more importantly, since it is assumed that knowing the number of operations would benefit future work to establish a full process plan, a learning-based evaluation will be attempted to automatically identify this cost driver.

The part mass and surface area, both of which can be extracted with either an API or a rule-based approach, will employ rule-based STL analysis in this work. This choice is made to make the overall system more generic, since the STL format is a universal format independent of specific CAD software. For the sheet thickness, however, which can also be extracted with STL analysis, an API program that performs geometrical analysis on a CAD component to automatically identify

the thickness is readily available and will be used.

### 4.3.2 Learning-based method development

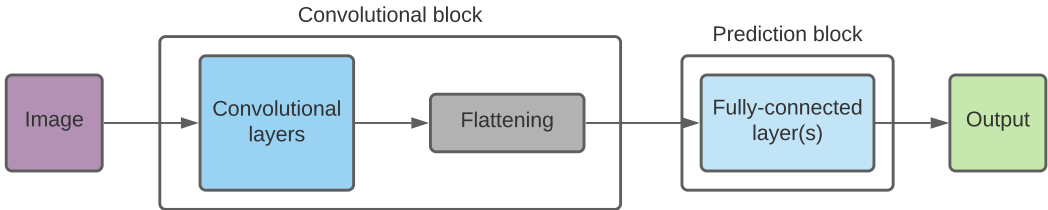
Since the rule-based approaches that will be employed in this work generally are simple and already specified, their methods are not subject to further development. The learning-based methods are, however, thus far, somewhat unspecified and in need of clarification and development.

As mentioned in Section 4.2.15, the term learning-based in Table 2 primarily refers to convolutional neural network (CNN) approaches. In fact, in Table 3, this is true for all cost drivers for which learning-based methods will be used, apart from the welding information. It is argued in the subsection above that the remaining cost drivers, namely the blank size, the fastener information, the production method, the number of parts per stroke, and the number of operations, all can employ similar learning-based methods for automatic identification due to their common dependency on geometry and size. However, the traditional CNN architecture does not directly account for spatial information and is not specifically designed to describe 3D objects. The remainder of this section will be dedicated to developing a suitable convolutional neural network approach for the specific use case encountered in this work. That is, a convolutional neural network aimed at the analysis of 3D objects that evaluates spatial information in addition to geometrical information.

Convolutional neural networks have proven efficient in describing geometry. The traditional CNN takes an image input and evaluates this in terms of geometrical shapes to generate some output. However, as previously discussed for the relevant cost drivers in Section 4.2, spatial information is assumed to be of great importance to accurately identify them. For a stand-alone traditional CNN to evaluate spatial aspects of a part, the ratios of all parts evaluated by the network would need to adhere. Since such a traditional CNN takes a single input image, this means that the spatial dimensions of the image pixels would need to be constant for all images considered. The term spatial dimensions here refers to the physical size associated with a single pixel. For example, a single pixel could represent 1mm x 1mm, which would mean that a part with a width of 100mm imaged from the front would fill 100 pixels horizontally. However, the current work will consider parts ranging from entire automotive side frames with lengths up to 3 meters down to small brackets with lengths of just a couple of centimeters with the same networks. As a result, keeping a constant size ratio would make it very difficult for the network to evaluate the geometry of the smaller parts. Recognizing,

for example, if an animal is a cat or a dog in an image where this animal is appropriately sized and in the center frame is easy. However, making the same distinction when the animal is standing next to the Eiffel tower and the image is scaled such that the entire tower is in the center frame is less arbitrary.

Since a constant size ratio can not be kept in between the images passed as inputs to the CNN, an alternative way to assess spatial information is needed. Observing the traditional stand-alone CNN architecture depicted in Figure 18, this network can be said to contain two main parts. The first and most elaborate part of the network is the convolutional block. Inside of this, input images are broken down into shapes and patterns of shapes, and the output from the block will be some description of the input image in terms of the geometrical patterns present. If the CNN was evaluating images to distinguish photos of cats and dogs, one of the outputs of the convolutional block could, for example, be the number of snouts in the image. The second part of the network, namely the prediction block, would then, for example, evaluate the number of snouts present in the image to make a classification as to whether the image depicts a cat or a dog. That is, if there is a snout in the image, the prediction block would output that the input image depicts a dog, and if not, it would output that it depicts a cat. As opposed to Figure 8, which presents a rough outline of the traditional CNN, Figure 18 illustrates also the *flattening* operation. This operation is performed to convert the output of the convolutional layers into a 1D array that can be input into the subsequent fully-connected layers, but it is rarely included in schematic network illustrations. In this section, however, it is considered to be of importance to highlight this operation, specifically that the output of the convolutional block is a 1D array, and it is therefore included in the figures.

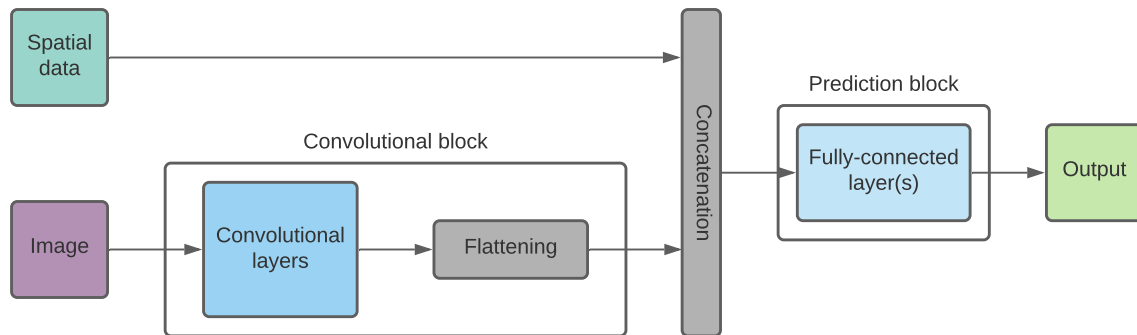


**Figure 18** – Traditional CNN architecture for image classification with the convolutional and prediction blocks indicated.

In the traditional CNN (Figure 18), the output of the convolutional block can be considered as a geometrical description of the input image. The prediction block can, in turn, be thought of as

a tool to process this geometrical information to provide a final output. In problems where there is a spatial dependency in addition to a geometrical one, the prediction block would then also need spatial information to be able to make a solid final prediction. As a result, this traditional architecture is modified so that the flattened output of the convolutional block is concatenated with some spatial information. The concatenated vector is subsequently passed to the prediction block for evaluation. This architecture is presented in Figure 19.

The architecture in Figure 19 could allow the model to evaluate both geometrical and spatial information relating to a part. However, it considers only a single image at a time. In this work, 3D objects are the subjects of evaluation. More specifically, 3D objects that initially could exist in any random orientation. This means that the direction from which the most information about a component can be obtained using a single image is unknown. If, for example, a CNN is trained to recognize people from images of their heads, an image of the face would generally provide more information to the network than an image of the back of the head. In this work, the view that captures an image of the "face" is unknown. In addition, there does not necessarily exist such a "right" or "wrong" side of a 3D sheet metal component. Instead, multiple sides could be of importance to generating a complete impression of the full part geometry. As a result, methods for presenting 3D objects to CNNs must be explored.

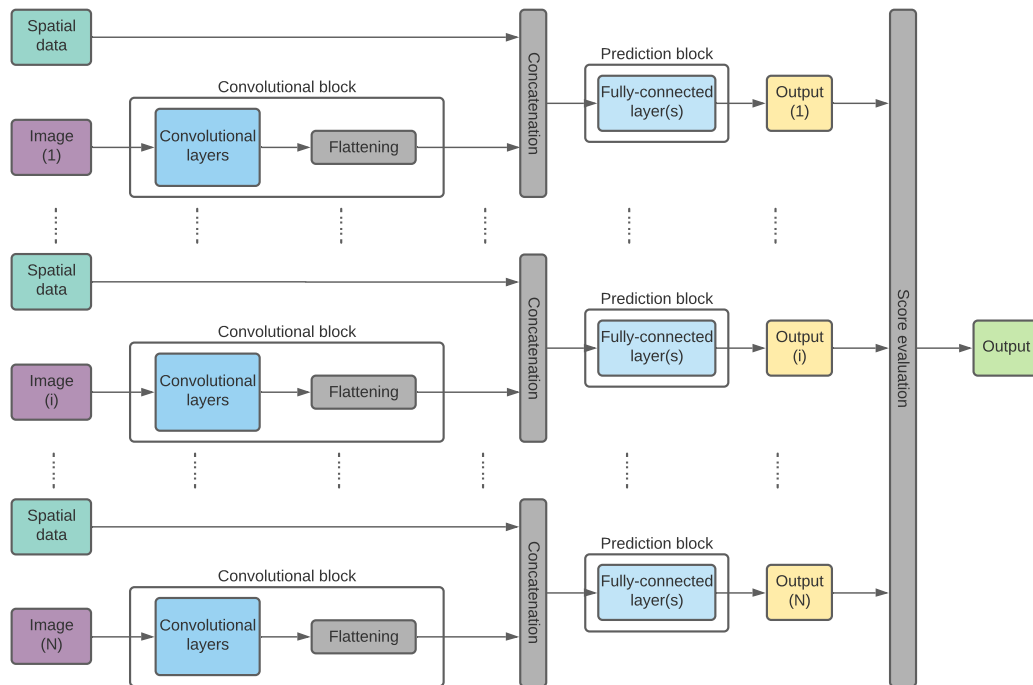


**Figure 19** – The spatial CNN architecture used in this work.

As discussed in Section 3.3.2, the multi-view representation is amongst the best performing and strongest methods for evaluation of 3D objects with CNNs. Additionally, this method allows for an agile architecture that is considered easier to modify and adjust compared to the point cloud representation, which is considered as the other main option. As a result, a multi-view representation is pursued to describe 3D objects in this work.

The subsequent question after deciding to pursue multi-view representation is how to fuse the information obtained from each view into a final description of the 3D object. As discussed in Section 3.3.2, the late fully-connected and score fusion techniques (Figure 15) are considered the most interesting to explore, and are hence also the ones employed in this work.

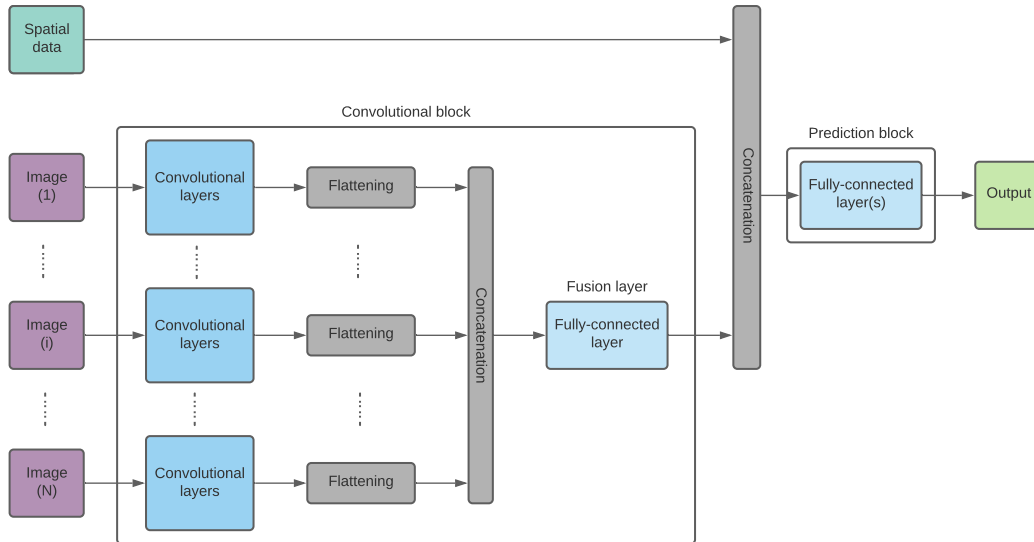
Having established how spatial arguments potentially can be included in traditional CNN architectures (Figure 18) and additionally that this work will employ the multi-view representation with both the late fully-connected and score fusion techniques (Figure 15) to describe 3D objects, the next question is how to combine these approaches. That is, how to provide the spatial input to a multi-view CNN. For a score fusion model, which is just a duplication of multiple single-view models (equalling the number of views) followed by a score evaluation that considers the output from each single-view model, extending the traditional architecture to one that also considers a spatial argument simply entails duplication of the model depicted in Figure 19. Such a multi-view model is depicted in Figure 20.



**Figure 20** – The spatial score multi-view CNN architecture used in this work.

For the late fully-connected fusion technique, a similar idea to that employed in the derivation of the spatial single-view model can be used. By considering the output of the fusion layer

in this architecture as the geometrical description of the component, a spatial argument can be concatenated with this output and fed to the prediction block, following the same argumentation as for the single-view model. This model is depicted in Figure 21.



**Figure 21** – The spatial late fully-connected multi-view CNN architecture used in this work.

All the spatial models presented in this subsection are intended to function in both classification and regression problems. That is, the output of the models can be either some vector where each index indicates the network’s confidence that the input belongs to some class, or it can be some floating-point number.

With the presented method clarifications and model specifications, a system architecture can be developed.

## 5 Implementation

This section elaborates on the work to develop a system for automatic identification of the different cost drivers. First, a system architecture is proposed based on the dependencies of the various cost drivers and the requirements of the methods to identify them. Subsequently, the work relating to data acquisition and preparation is outlined. Finally, the development of the modules in the theorized system is discussed.

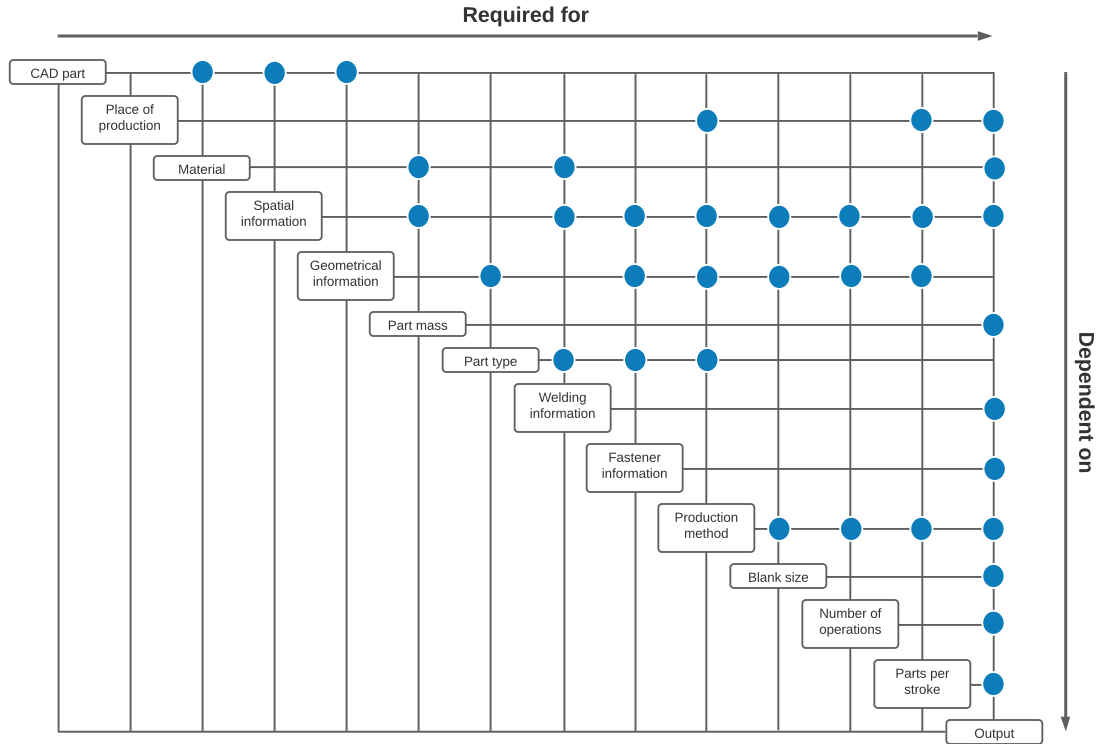
### 5.1 System architecture

The aim of this work is to develop an automated system that evaluates a 3D CAD representation of a sheet metal component to identify its cost drivers. Such a system needs an architecture. The word architecture here refers to a representation of the system that outlines the flow of data through individual evaluation modules to finally deliver a complete set of cost drivers. This subsection is dedicated to the development of such a system architecture. First, the dependencies of the various cost drivers are outlined and discussed. After this, a proposed system architecture is presented and explained.

#### 5.1.1 Dependencies

In order to develop the architecture of an automated system to identify cost drivers, their dependencies must first be evaluated. This needs to be done to ensure that all inputs needed for the identification of a certain cost driver are available to the evaluation module when its analysis is initiated. To determine the number of welding points, for example, the thickness needs to be known, since this is one of the inputs to the network that will be used to attempt estimation of the number of welding points. Hence, the thickness must be identified prior to estimating the number of welding points.

Figure 22 shows an N-squared diagram (NSD) where the dependencies of the various cost drivers are outlined. An NSD is a tool to visualize dependencies, and it is the graphical representation of a design structure matrix (DSM). In an NSD, each horizontal line illustrates an output, and each vertical line illustrates an input [77]. The diagonal of the NSD consists of boxes that represent some entity of interest. An entity can be either a cost driver itself or an input that is important to determine a cost driver, namely a dependency of a cost driver. In the diagram, a blue mark at the intersection of two lines indicates a dependency between the entity at the horizontal end and the



**Figure 22** – N-squared diagram outlining the dependencies of each cost driver.

entity at the vertical end. This dependency is such that the entity on the horizontal line is a required input for that on the vertical line. In other words, to find the inputs needed to determine a certain entity, the horizontal lines must be consulted. For example, to determine the spatial information, which here is used as a collective label for the part area, volume, thickness, and size, only the CAD part, which is the system input and initial state, is required. If, instead, considering the horizontal line starting with the CAD part, it can be observed that this entity is a required input to determine the material, the spatial information, and the geometrical information. That is, the CAD part must be available before evaluating any of those entities. The entity "Geometrical information" refers to data relating to the shapes in the component. A dependency on this indicates that input data to a CNN, which is what will be used for the geometrical description of a component (see Section 4.3.2), must be appropriately preprocessed and available prior to the attempted identification of the relevant entity.

Note that only primary dependencies are outlined in the NSD. This means, for example, that even though the part mass is dependent on spatial information (volume), which in turn is dependent on the CAD part, the part mass itself is not marked with a dependency on the CAD part.



In the NSD, a so-far undefined entity "Part type" is included. The part type is here either a "sheet metal component" or a "fastener". The reason for including this entity is that such a distinction is necessary in order to be able to carry out the appropriate analysis on a part. The input to the system can be either a small assembly or an individual component, both of which can include both sheet metal parts and fasteners. To identify, for example, the sheet metal production method appropriate for a component, the sheet metal part itself, excluding fasteners, must first be identified through the "Part type" classification. Since sheet metal parts and fasteners have very different appearances, it is assumed that such a classification is possible to make based only on geometrical information.

The entity "Output" refers to the actual cost drivers that are needed to perform the subsequent analysis with the analytical cost estimation tool to finally estimate the production costs. It can be observed that not all of the included entities are direct outputs of the system and that some, like, for example, the "Part type", only are required in order to enable the analysis to identify the actual cost drivers.

The NSD is of great assistance when designing a system architecture. This is because it illustrates when a particular entity must be determined relative to the others and because it can highlight entity co-dependencies. The presented NSD is already sorted such that all the blue marks are on the upper side of the diagonal, but it does not always start out like this. Imagine, for example, how the NSD would look if the entities on the diagonal were ordered like in Tables 1 or 2. Sorting the NSD to try and get all marks above the diagonal is the first and most important task when using this tool. If this is successful, it means that the system can be designed without any iterative processes. The alternative is that two or more entities in the system are found to be co-dependent, meaning that one cannot be established without the other and vice versa. This would generate the need to go back and forth between these co-dependent entities in some optimization process. In the presented NSD, however, it is observed that no such co-dependencies, or feedback loops, exist. As such, one result from simply generating and ordering the NSD is the knowledge that the system design can be linear. In other words, each part of the system can be developed and implemented independently, provided that it is ensured that all its dependencies are identified prior to its launch.

The sorted NSD is of use to determine specifically where in the system architecture certain entities should be identified. The horizontal lines can, for example, be considered in order to find entities

that are of particularly high importance to other ones in the system. This would indicate that they should be identified early on, making them available for subsequent parts of the system. When a horizontal line has few or only one mark<sup>6</sup>, it generally means that there exists some flexibility as to exactly where in the system the particular entity must be identified. The welding information, for example, is not required to identify any other entity, and the analysis to determine the number of welding points can therefore be performed at any point after its own dependencies are identified.

### 5.1.2 Proposed system architecture

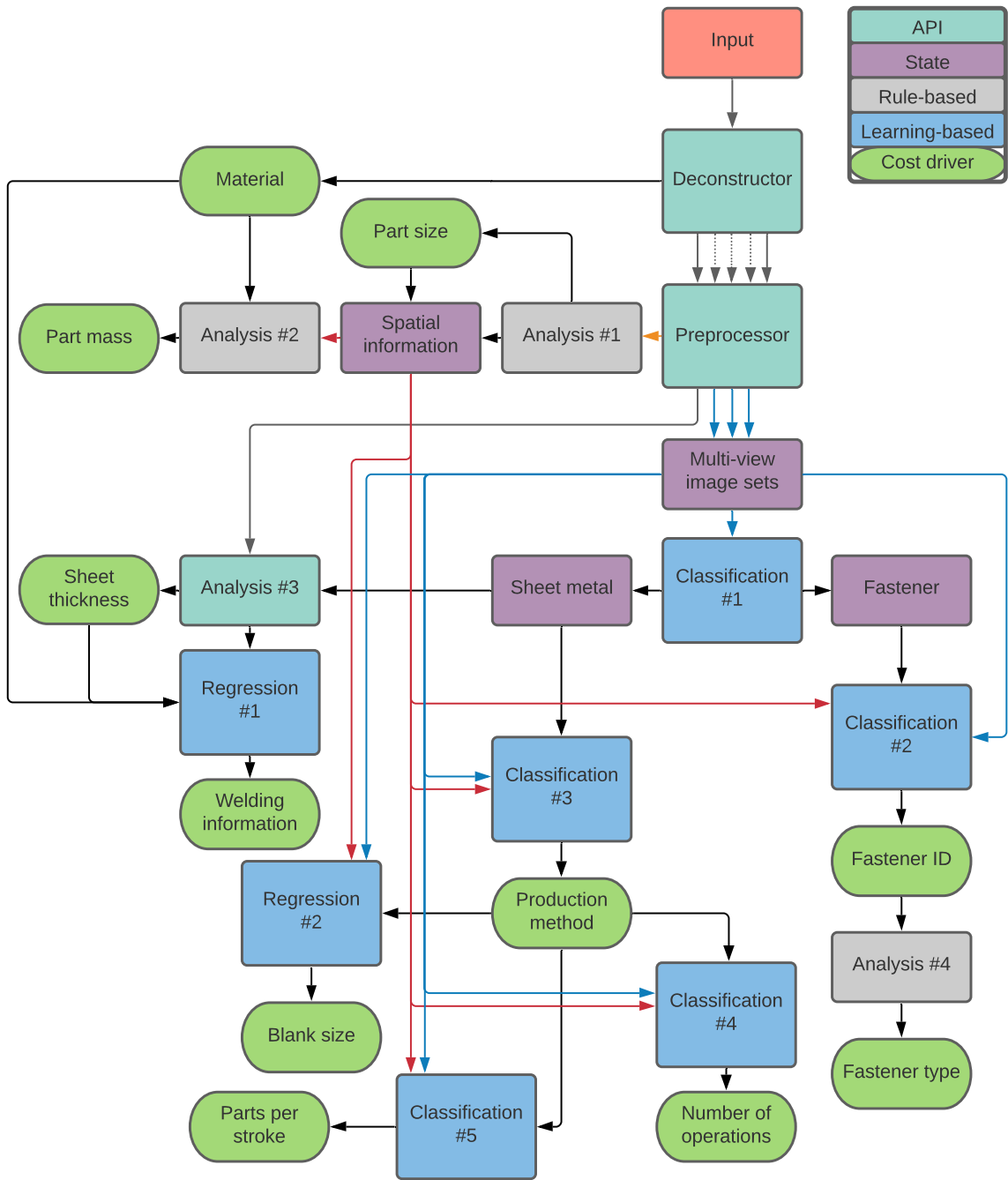
Figure 23 presents the proposed architecture for an automated system to identify cost drivers from 3D part models of sheet metal components. Designing such an architecture prior to the development of each individual module is important both to validate that the system is feasible and to aid in planning a workflow for further development. Validating that the system is feasible means to verify that it is actually logistically possible to implement the envisioned system. In other words, provided all individual modules work, that it is possible with those building blocks to create a system that takes a 3D representation of a sheet metal component or small assembly in a CAD native format as input and outputs the part entities that are necessary to estimate the component costs, namely the cost drivers. The architecture also aids in planning the sequence in which to develop the various modules. Although the data that will be used to train and test the various learning-based modules will be collected from existing databases, meaning that this architecture will not play a direct part in the development of each module, it helps to further visualize the NSD (Figure 22) in terms of which modules should be developed when. This also circles back to validating the feasibility of the system. If, for example, it is unsuccessful to develop a module that determines the production method, having a blank size prediction model with high accuracy is of little use, since no automated way to identify one of the model's dependencies is available.

The remainder of this subsection is dedicated to presenting and explaining the proposed system architecture.

Starting with the input on the top right in the architecture, the analysis process begins. This input is a path leading to a 3D representation of a sheet metal component or small assembly in a CAD native format. The input could also be a unique part ID referencing such a CAD part model, which then subsequently can be downloaded from an available database. Once the CAD representation

---

<sup>6</sup>All horizontal lines will have at least one mark, since the output is included as an entity.



**Figure 23** – The proposed architecture for an automated system to identify cost drivers from 3D representations of sheet metal components.

of the part is obtained, the first module of the system, namely the Deconstructor, is tasked with deconstructing the component or small assembly into all its individual parts. The transfer of CAD data is denoted by grey arrows in the architecture. As previously mentioned, the system input can be either an individual sheet metal component, potentially with some number of fasteners attached to it (1 sheet metal part + X fasteners), or a small assembly consisting of some small number of sheet metal parts and potentially some number of fasteners (Y sheet metal parts + Z fasteners). In order to perform the analysis required to identify the cost drivers associated with the input component, from which a cost estimate will be made, these singular parts must be isolated and analyzed independently. This isolation is the task of the Deconstructor, which generates and stores separate CAD files for each individual part in the original input. To do this, the Deconstructor employs the API associated with the CAD software. The fact that this module uses an API to perform its task can also be determined from its color. In the upper right corner, the legends associated with each module can be found. In addition to deconstructing the original component into individual files for each part, the Deconstructor identifies the materials of each individual part and stores this for future use and final output.

Once a separate CAD file is generated for each individual part, the Preprocessor will access these individual files to convert the 3D part representation from the CAD native format to the formats that will be used by the various evaluation modules of the system. Apart from the module Analysis #3, whose function will be discussed later, all evaluation modules require data in other formats than the CAD native. In the Preprocessor, each individual part is imaged from multiple views, using an API to create a multi-view image set. In this work, such a set will consist of 7 images, and is elaborated on in Sections 3.3.2 and 5.2.2. These multi-view sets for each part are then stored separately, ready to be accessed by the latter parts of the system. Their availability is denoted by the state Multi-view image sets, and the flow of image data can be identified by the blue arrows in the architecture. The Preprocessor also exports the CAD files to the STL format, storing an STL representation of each individual part that will be evaluated by the module Analysis #1. The flow of STL data is denoted by an orange arrow.

Once an STL representation of each individual part is available, these are analyzed in Analysis #1 to retrieve basic spatial information about the component. The basic spatial information considered in this work is the part area, the volume, and the global and minimum bounding boxes. These are chosen because they are assumed to provide a sufficient spatial description of a 3D object, either

alone or in combination. Upon suspicion that this assumption is wrong, further spatial descriptions will be investigated and possibly extracted.

The cost driver Part size (minimum bounding box) is amongst the data that is extracted in Analysis #1. To highlight the identification of this cost driver, it is specified as an individual output of the module that is subsequently passed to the Spatial information state. This is merely done for illustrative purposes to highlight the identification of a cost driver and subsequently to highlight that the part size is amongst the spatial information available in the Spatial information state. The part area is also amongst the information extracted and included in the Spatial information state. However, since this cost driver is only relevant to the tooling costs, which are not the target of the system, this is simply included in the line that passes directly from Analysis #1 to the Spatial information state. The lines now have a black color, which denotes the transfer of knowledge data. This data includes scalar and vector values, such as the volume and bounding box, as well as general knowledge about a part, such as, for example, that a part is a fastener.

The Spatial information state illustrates the existence of base spatial information that can be accessed by the rest of the system. The flow of spatial data is illustrated by red lines in the architecture. From this state, the module Analysis #2 retrieves the part volume. Additionally, the material of the part, which was previously extracted by the Deconstructor, is retrieved. With this information, the part mass is calculated.

At this stage, the base information that is required by the learning-based modules is available, and more complex component analysis can be performed. First, the module Classification #1 evaluates a multi-view representation of a single part to determine whether this part is a sheet metal part or a fastener. Notice that this learning-based module solves a classification problem, as opposed to a regression problem, and is named thereafter. The outcome of this classification decides which path to follow for a full analysis of the part in question, and the part type is therefore included in the system as a state.

If the part is identified as a fastener, the subsequent analysis involves the right branch of the system, where the fastener ID is first determined by the module Classification #2. In Classification #2, both the multi-view representation of the fastener and a spatial input is evaluated to determine the fastener ID. That is, identifying the specific fastener in question. Once the fastener ID is known, its type is identified by the Analysis #4 module.

If the part is found to be a sheet metal part, two individual and independent paths can be followed. One path leads to Classification #3, where the appropriate production method for the sheet metal part is identified. The other path leads to Analysis #3, where the sheet thickness and later the welding information are the cost drivers to be identified.

If following the path through Analysis #3, the sheet metal part is first analyzed using an API to determine the sheet thickness and the contact surface area within the individual part itself. This information is then passed to the module Regression #1, in which a fully-connected neural network estimates how many, if any, welding points are needed in the individual part. Notice that this learning-based module solves a regression problem, as opposed to a classification problem, and is named thereafter. As for the Part size, the Sheet thickness is included as a separate output of Analysis #3 for illustrative purposes.

In the cases where the input is a small assembly, meaning more than one sheet metal part is present in the CAD input, a final assessment by the Analysis #3 module is necessary, where all the parts that are identified as sheet metal parts are included. This analysis will then establish the contact surface areas between the individual parts, so that a subsequent evaluation can be made by Regression #1 to determine the number of welding points that are needed in the assembly process.

If following the path from the Sheet metal state through Classification #3, the production method is first identified. To determine the production method, Classification #3 uses both the multi-view image representation of the part and its spatial information. Once the production method is identified, further analysis can be carried out by Classification #4, Classification #5, and Regression #2 to determine the number of operations, the number of parts per stroke, and the blank size of the part, respectively. Each of these modules additionally receives the multi-view image representation and spatial information of the part as inputs to enable the evaluations.

Having successfully performed all these evaluations for each individual part in the input component, every cost driver required to estimate the production costs is available to be output by the system.

Performing analysis with the proposed system can be done either linearly or partially in parallel. A linear implementation here refers to evaluating each individual part in turn, one after another, until finally having established the full system output. A parallel implementation would involve analyzing each individual part in parallel once the Deconstructor has generated single files for each

of them. The potential final evaluation of assembly welding points, however, must be performed after all individual parts are assessed. This is to ensure that only contact areas between sheet metal parts, not sheet metal parts and fasteners, are considered.

The proposed system architecture demonstrates the logistical feasibility of a system that automatically evaluates a 3D part model to identify the cost drivers associated with an input component. However, each individual module must still be developed, and their performance evaluated.

## **5.2 Data acquisition and preparation**

In order to train neural networks, training examples are required. This section will walk through the approach employed in this work to acquire and prepare data in order to generate such training examples. Specifics on the data that is collected for each cost driver, such as the amount and distribution, will be presented and discussed under the results for that cost driver (Section 6). That is, the aim of this section is to outline the general approach to data acquisition and preparation.

### **5.2.1 Data acquisition**

It is established in Section 4.3 that learning-based methods will constitute the primary part of this work. In order to apply such learning-based methods, data to train, validate, and test the models is required. Having the correct data in sufficient amounts is of great importance. According to [45], a neural network is only as good as its training data. This means that even a simple network trained on a good and large dataset can outperform a complex one that employs all the state-of-the-art methods if that network is trained on an insufficient dataset. In other words, without sufficient training data, the learning-based identification of the various cost drivers will fail, and the more data that is collected, the better results can theoretically be expected. However, data acquisition is a time-consuming process.

A question that arises when starting to collect data is how much data that is "sufficient data". That is, how much data is required to train a network to a satisfactory performance. Knowing such a threshold would be very beneficial, since it would allow to stop the expensive process of collecting data at a certain point. Additionally, it would eliminate the efforts to train and test networks if it was already clear that insufficient data was available. Unfortunately, however, although some general guidelines are available in certain cases, no definite rules exist that explain exactly how

much training data is enough. The reason for this is that different use cases are unique and have different requirements. In cases where the problem has a high complexity, for example, more data would typically be required than in cases where the problem has a lower complexity. This is intuitive by comparison to how humans learn. If, for example, one aims to separate different cat breeds, it would typically require more examples per breed to find the general traits to identify a specific breed, than in a case where one aims to separate different species of animals, such as cats, dogs, and elephants. This is also generally true for networks. Still, however, the exact number of training examples required for each class in either case is difficult to pinpoint. Although previously solved problems can be consulted, it can be difficult to judge the complexity of the current problem compared to this other one. As a result, since finding a threshold with regards to the amount of data is difficult, this work aims to acquire as much data as possible within a reasonable amount of time relative to the project timeline.

In order to train a network through supervised learning, which is the case in this work, the training data must link the network's input to the desired output. In this work, there are two main network approaches, and hence to main input types. The two main network approaches are the fully-connected network to determine the number of welding points and the convolutional neural networks presented in Section 4.3.2.

For the welding information, the input consists of the cost driver's five assumed dependencies. These are the thicknesses, materials, and contact areas of the joined sheets. Unfortunately, no database exists that links these dependencies to the number of spot welds. At the time of data acquisition, the previously mentioned API tool to extract the thicknesses and contact areas of the joined sheets was not available. Because of this, even though databases are available that link part IDs of CAD part models with the number of welding points, those CAD part models could not be automatically evaluated to extract the thicknesses and contact areas of the components. As a result, the collection of this data is done manually by opening a CAD part model, extracting the data of interest, namely the cost driver's dependencies, and referencing these to the number of welding spots used in the part.

For the modified convolutional neural networks described in Section 4.3.2, the input is a set of 7 images depicting a component from different views and some spatial information relating to this component. To acquire these inputs, a CAD part model is needed. In this work, databases



are available that link part IDs that reference such CAD models to the relevant cost drivers for previously manufactured components.

Once the relevant databases are obtained and structured, the next step is to acquire the CAD part models adhering to the part IDs so that the actual network inputs may be prepared. An online database of CAD components is available, from which the CAD part models of a certain part ID can be downloaded. However, manual downloads from this portal entail several minutes of effort per part, and thousands of parts need to be downloaded. As a result, such manual downloads are not feasible to perform within the scope of this work. To rather automate the process of part downloads, a Python package [78] to run the Selenium WebDriver [79] is used. With this web driver, the online database can be automatically accessed, and a loop can be created through all part IDs, downloading the CAD part model associated with each one.

With the CAD part models of all part IDs downloaded, the process to prepare the data for training can be initiated.

### 5.2.2 Data preparation

The manually collected welding data is not subject to further preparation post acquisition. That is, the data is ready to be used in training after it is acquired and will therefore not be discussed in this subsection.

The CAD part models downloaded during the above-described acquisition stage can, however, not be directly input to the convolutional architectures presented in Section 4.3.2. Rather, as discussed above, multi-view image representations of each component must be prepared. In addition to this, spatial information must be extracted from the models.

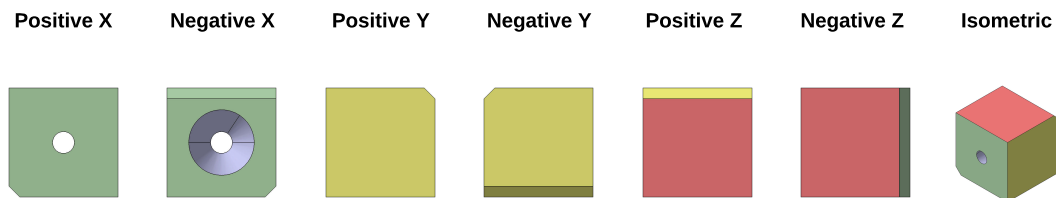
The CAD software utilized in this project is Catia V5. To automate the process of multi-view image capturing, a Python code is written that employs this software’s API. This program uses the Python module pycatia [80] as an interface for automating the Catia V5 COM object [81]. In order to generate the multi-view image representations, the program opens each CAD part model and captures images of the CAD part from the following seven views:

- Positive X-view: The part is imaged in a view parallel to the x-axis from the side of decreasing x-values. The view is hence pointing in the direction of positive x-values. The z-axis is

pointing upwards.

- Negative X-view: The part is imaged in a view parallel to the x-axis from the side of increasing x-values. The view is hence pointing in the direction of negative x-values. The z-axis is pointing upwards.
- Positive Y-view: The part is imaged in a view parallel to the y-axis from the side of decreasing y-values. The view is hence pointing in the direction of positive y-values. The z-axis is pointing upwards.
- Negative Y-view: The part is imaged in a view parallel to the y-axis from the side of increasing y-values. The view is hence pointing in the direction of negative y-values. The z-axis is pointing upwards.
- Positive Z-view: The part is imaged in a view parallel to the z-axis from the side of decreasing z-values. The view is hence pointing in the direction of positive z-values. The y-axis is pointing upwards.
- Negative Z-view: The part is imaged in a view parallel to the z-axis from the side of increasing z-values. The view is hence pointing in the direction of negative z-values. The y-axis is pointing upwards.
- Isometric view: The part is imaged in a view parallel to the vector  $[0.577, 0.577, -0.577]$  from the negative side. The part is oriented such that the vector  $[0.408248, 0.408248, 0.816497]$  is pointing upwards.

An example of these seven views is available in Figure 24.



**Figure 24** – Illustration of the different views employed in this work.

In order to extract spatial information from the component, either an API or rule-based analysis approach can be used. In this work, rule-based STL analysis is chosen, as this is what will be employed in the finished system. As a result, each of the CAD part models needs to be converted to the STL format. This task is also performed by automating Catia with Python to open each CAD part model and export it to the STL format. Once STL representations of all components are available, these can be analyzed to extract spatial information. As discussed in Section 5.1.2, the basic spatial information considered in this work is the part area, the volume, and the global and minimum bounding boxes. The specifics on the rule-based methods employed to identify each of these are discussed in Section 5.3.1.

With both multi-view representations and spatial information at hand, the data is prepared and ready to be used to train the networks. However, a returning concern relating to data is the size of the dataset. As discussed in the section above, no definite rules are found to specify the exact amount of training data that is required to obtain some satisfactory network performance. As a result, the dataset size will usually be amongst the error sources when a network fails to reach satisfactory performance. However, data acquisition is expensive, and it will not always be possible to increase the dataset size at the time when such unsatisfactory results are obtained. As a result, preparation is also done to try and further increase the amounts of data available for training.

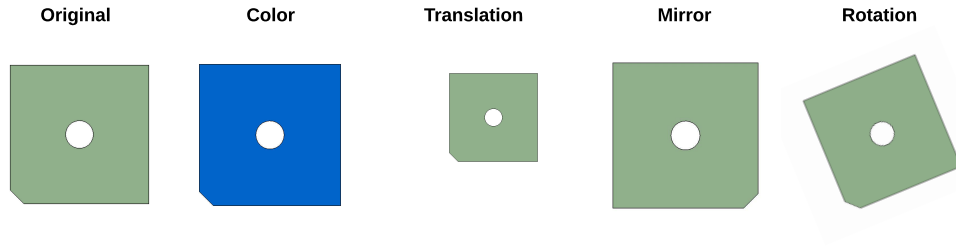
A question that arises when more data is found to be required is how to increase the size of the dataset. That is, how to acquire more data. Since the acquisition of real-life data examples is a very time-consuming process, it is not necessarily possible or feasible to increase the dataset with more such real examples. As a result, techniques are often consulted where data is artificially produced. Such techniques include synthetically generating data from scratch and augmenting already existing data.

Synthetic data generation is a technique where data is produced specifically for network training purposes. For example, if training a network to identify the production method of a sheet metal component, instead of taking an existing component that has been created for functionality to be included in a vehicle and determine this component's production method, a component is specifically produced as an example of a certain production method. Creating such synthetic data can be done manually, but it is usually carried out by means of some automated process. Such an automated process could be to use a CAD software API to either create part models from some set of rules or

to modify some existing parametric models by varying certain parameters. In the paper presenting the FeatureNet, for example, generic part models with different machining features of different sizes are synthetically generated by automating a CAD software [36]. A limitation to synthetically generating data is that some clear rules or specifications must exist and be known that links certain geometrical traits to the desired output. Such rules and specifications are not clear for the cost drivers assessed in this work. As a result, attention is rather turned to data augmentation.

Data augmentation is a technique aimed at increasing the size of the dataset by making slight modifications to the existing data without altering the target output [45]. There are several ways to perform augmentation on image data, such as changing the colors in the image, translating the image, mirroring the image, and rotating the image [82]. An illustrative example of each of these techniques on a 2D image can be found in Figure 25. Notice that both translation and rotation entail a resizing in this case in order to fit the part into frame. Augmentation techniques generally help the network’s ability to become robust and generalize patterns, rather than overfitting to certain peculiarities in the training data. If, for example, a network is trained to distinguish images of cats and dogs, but in all training examples, the dogs are located to the right in the image and the cats are located to the left, the network could think that what identifies an image of a dog simply is that there is an object to the right in the image. If an image came along with a cat standing to the right, the network would then misclassify this image as being of a dog. Another example would be if all the cats were white and all the dogs black. In other words, if the dataset is so small or unvaried that such peculiarities in the training data are true for all, or almost all, examples, the network could overfit to these non-general patterns. Data augmentation is used to counteract this phenomenon by increasing the variation and size of the dataset. This can be done either by actually creating additional modified images, such as rotated copies of an original image, or by, for example, randomly rotating the inputs passed to a network before they are evaluated. In the latter approach, whenever an input image is provided to the network, this image is modified according to some specified augmentation technique or techniques. The dataset size is then effectively also increased by training for more epochs on such continuously modified images.

The above-mentioned techniques for data augmentation are focused on single 2D images. Although such images are also used in this work, it is important to keep in mind that these images are intended to be used in multi-view models to represent 3D objects. As a result, care must be taken to keep consistency between the views of the multi-view representations. Having consistent



**Figure 25** – Illustration of different augmentation techniques in 2D.

multi-view representations is desired in order to have consistent data sets for all parts.

An additional reason to encourage such view consistency is that one of the strengths of a multi-view model could be that it recognizes patterns also in between the different views. That is, it could recognize interrelations between the patterns in the different views and make a classification based on these interrelated patterns. Of the multi-view fusion techniques that are intended to be employed in this work, namely the late fully-connected fusion and the score fusion, this can only happen for the late fully-connected fusion. This is because the score fusion merely performs a rule-based evaluation of the output from each view, not considering which view outputs what. Other similar techniques to the score fusion can, however, also potentially find such patterns in between the views by including more complex view output evaluations.

In order to keep consistency between the views also in the augmented data, the augmentation needs to be performed on the 3D objects themselves before generating augmented multi-view image sets. This is because it often is not possible to maintain the relations between the views if augmenting each view image on its own. If, for example, rotating the positive x-view image of a component, it is not possible to simply modify the positive y-view image to adhere to the same part rotation. The reason for this is that such a rotation would uncover surfaces of the part that are not present in the original y-view image but that should be present in the augmented one.

Since the 3D object itself must be modified to maintain view relations, this somewhat limits the augmentation options. Mirroring of such a 3D object, for example, entails an exhaustive process, and augmentation through mirroring is therefore not pursued in this work. Since all the components are centered in the frame upon imaging, translation is no longer considered a relevant augmentation technique. A further reduction of relevant techniques is found in that all components relevant to

the use case of this work will have an identical uniform color. Although color augmentation still potentially could provide some improvement to the network’s generalization ability, it is decided not to pursue this type of augmentation because there is no natural color variation in the data to be analyzed. Since the 3D object itself, not just the 2D images that are passed as input to the network, needs to be modified, it is no longer possible to simply modify the network input dynamically, and actual augmented datasets must therefore be created.

In order to augment the real-life data that is collected in this work, rotation is the technique that prevails from the previously listed options. As a result, augmented datasets are produced where the components are rotated some random angles about each axis some number of times. Instead of rotating the actual CAD part models, this work makes use of the fact that an identical effect is produced by rotating the views from which the images are taken, ensuring their relations are consistent. The initial size of the augmented datasets is set based on the amounts of data that fits into memory on the server where the training is performed. There are also techniques where data is continuously loaded throughout training, meaning not all data needs to fit into memory at the same time, but these often cause a significant reduction in training efficiency and are hence attempted to avoid. If the initial augmented set proves insufficient, further augmentation can be performed. The augmented datasets are kept aside to be employed when needed. Further details on the number of augmented examples for the specific datasets for each cost driver will be discussed in the results for each cost driver in Section 6.

### **5.3 Development of each module**

Having outlined and verified the feasibility of the complete system with respect to logistics and acquired the relevant data to train the models within, the work to develop each module can start. This section will walk through the development of the different modules, starting with the rule-based and API modules before proceeding to the learning-based ones. All development is done using the programming language Python.

#### **5.3.1 Rule-based and API modules**

Several of the simpler tasks in the proposed system are performed by rule-based and API modules. This subsection will go more into detail on the operations that are performed to turn the module input into the output. For these modules, there will be no results to discuss, and this subsection is

hence devoted to elaborating on all relevant aspects of the rule-based and API modules.

**The Deconstructor** is the first API module. This module will take a CAD model of some component as input and turn this into some number of individual CAD part models, equalling the number of individual parts in this component. In order to perform this task, the Deconstructor engages the Catia V5 COM object [81] through Python, with the help of the Python package pycatia [80]. Using these tools, the Deconstructor loops through the individual solids in the component and exports these to separate files. To additionally make the material information associated with each part available to subsequent modules of the system, the Deconstructor also identifies the material name of each part.

**The Preprocessor** converts the individual CAD part models into formats that can be input into the modules that perform analysis on the parts. The approach employed by this module is the same as discussed in Section 5.2.2 for preparing training data. That is, the Catia V5 API is employed to image a part and convert it to the STL format. This generates a multi-view part representation (Figures 14 and 24) and an STL representation (Figure 3) of each part.

**The Analysis #1 module** will perform rule-based analysis on the STL part representations to identify the part surface area, volume, and global and minimum bounding boxes.

The part surface area,  $A_i$ , is found by calculating the individual area of each triangle in the STL representation as:

$$A_i = \frac{l_i \cdot h_i}{2} \quad (20)$$

where  $l_i$  is the length of the base line of the triangle and  $h_i$  is its height. These individual areas are subsequently summed to find the total part area:

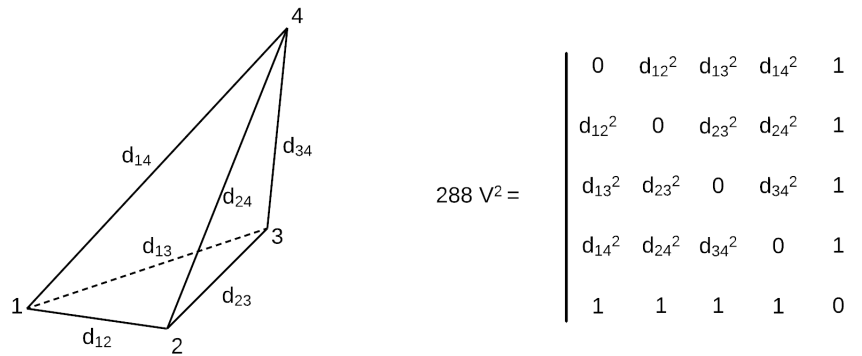
$$A_{part} = \sum_{i=1}^{N_t} A_i \quad (21)$$

where  $N_t$  is the number of triangles in the STL representation.

The volume is calculated by evaluating the signed volume of the tetrahedrons that have the STL triangles as their base and top point in the origin. Figure 26 displays an irregular tetrahedron and the matrix equation used to calculate its volume.

The approach employed to determine the part volume is:

1. Take a triangle in the STL geometry and calculate the volume of the tetrahedron with the triangle as a base and top point in the origin. In Figure 26, "1-2-3" could be the vertices of such a triangle, and "4" would then be in the origin,  $O = (0, 0, 0)$ .
2. Assign a sign (+ or -) to the volume based on the orientation of the normal vector of the triangle in question. If the normal vector points towards the origin, assign a "-", if it points away from the origin, assign a "+".
3. Sum up the signed volumes for all triangles in the geometry to find the part volume.



**Figure 26** – An irregular tetrahedron (left) and the matrix equation to calculate the volume (right).

The global bounding box is the cuboid with the smallest volume that can encapsulate a part whilst having one face parallel to either the XY-, XZ-, or YZ-planes. Since the cuboid has perpendicular angles between all faces, it follows that it will have two faces that are parallel to the XY-plane, two faces that are parallel to the XZ-plane, and two faces that are parallel to the YZ-plane. Because of this, the coordinates of the cuboid's eight vertices will be combinations of the maximum and minimum coordinates of the part it encapsulates in each axis direction. Furthermore, its dimensions will simply be the difference between these maximum and minimum values in each axis direction. In this work, the global bounding box is represented by the vector describing this cuboid's dimensions in each axis direction. That is, the global bounding box is defined as:



$$BB_{global} = [x_{max} - x_{min}, y_{max} - y_{min}, z_{max} - z_{min}] \quad (22)$$

Through STL analysis, the maximum and minimum values in each direction can be found by considering the coordinates of each vertex in the STL part representation, and the global bounding box can subsequently be established.

As described in Section 4.2.6, the minimum bounding box is the smallest cuboid that can encapsulate a part. There are no requirements put forth as to the orientation of the minimum bounding box. That is, as opposed to the global bounding box, where the faces are parallel to the XY-, XZ-, and YZ-planes, the minimum bounding box can have any orientation.

Determining the minimum bounding box is less arbitrary than determining the global one. Although a method for determining the minimum bounding box within a limited amount of time was developed by [71], this method entails a relatively high implementation effort. In this project, the requirements for accuracy on this cost driver are low, meaning only larger deviations from the actual minimum will have an impact on cost results. Because of this, to leave more time for the development of more complex modules where there are higher demands for accuracy, a decision is made to rather find an estimation of the minimum bounding box using a simpler implementation. This implementation is based on rotating the part for which the minimum bounding box is to be identified about each axis in an iterative manner and calculating the global bounding box for each of the orientations, finally settling for the smallest one registered as an approximation of the minimum bounding box.

In order to carry out the above estimation, each triangle vertex in the STL part representation needs to be rotated. This can be done by using the rotation matrices about each axis defined as:

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \quad (23)$$

$$R_y(\beta) = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \quad (24)$$

$$R_z(\gamma) = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (25)$$

A full rotation of some angles,  $\alpha$ ,  $\beta$ , and  $\gamma$ , about each axis, can be calculated by using:

$$R(\alpha, \beta, \gamma) = R_z(\gamma) \times R_y(\beta) \times R_x(\alpha) \quad (26)$$

The rotation of a point, P, defined as:

$$P = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad (27)$$

can then be calculated as:

$$P_{rot} = R(\alpha, \beta, \gamma) \times P \quad (28)$$

With these equations, the following steps are performed to estimate the minimum bounding box of a part:

1. Calculate global bounding box and set this as the default minimum
2. Using the rotation matrix in Equation 23, rotate each of the triangle vertices in the STL part representation about the x-axis, gradually increasing the angle by some amount,  $\delta$ , starting

at  $\alpha = \delta$  and ending at  $\alpha = 90 - \delta$ . Since the bounding box is a cuboid, it is not necessary to rotate the part more than 90 degrees. Calculate the global bounding box and its volume for each rotation angle and reference this against the smallest registered minimum. Update the minimum if the new bounding box is smaller than the reference and store the rotation angle for this minimum registered bounding box.

3. Having checked all rotations about the x-axis, set the default part orientation to the one that gave the smallest bounding box. That is, the default orientation is now the original part rotated some angle about the x-axis. In the case where the global bounding box was the smallest one found in step 2, the default orientation remains unchanged.
4. Repeat 2 for rotations about the y-axis with the new default.
5. Repeat 3 for the rotation angle about the y-axis that provided the smallest bounding box. The new default is then some rotation about both the x-axis and y-axis.
6. Repeat 2 for the z-axis.
7. Repeat 3 for the z-angle.
8. The smallest bounding box found after checking all z-rotations is considered an estimation of the minimum bounding box and is returned. Following a convention relevant to this project, the minimum bounding box is sorted such that the x-dimension is the largest and the z-dimension is the smallest.

**The Analysis #2 module** uses spatial and material information to calculate the part mass. The spatial information that is used is the volume, previously identified in Analysis #1. Using the material name extracted in the Deconstructor, the module can find the density of the relevant material in an existing index. With the volume and material density at hand, the part mass is calculated with Equation 19.

**The Analysis #3 module** uses an API to analyze the CAD model to extract the part thickness and the contact areas between the sheets in the component. As mentioned earlier, this module uses an existing tool to perform the analysis. This tool is written for Catia V5 and is called automatically from Python to implement it into the system.

**The Analysis #4 module** uses the fastener ID to find the type associated with the fastener in question. As discussed in Section 4.2.7, the relevant types are impress elements, welded elements, or neither. The module finds the fastener type by referencing an existing table with information about each fastener.

### 5.3.2 Learning-based modules

The learning-based modules in the system (Figure 23) and their development constitute the core of the technical sides of this work. Two main learning-based methods will be employed, namely the fully-connected network used for the welding information and the convolutional neural networks presented in Section 4.3.2, which are intended to be used for the part type identification, fastener information, production method, number of operations, number of parts per stroke, and blank size. The development of these differs in that the welding information network will employ a more automated development approach. This approach is considered to be better discussed under results (Section 6.6.2), and the following subsection is therefore devoted to the convolutional neural network (CNN) approaches.

A **summary** of the below discussions, outlining the approach and presenting the relevant base information associated with the development, is available at the end of the subsection.

#### General testing approach

The CNN modules will be developed through iterative testing and analysis of various settings and concepts, building towards a final configuration. The final configuration is considered as a configuration with sufficient performance to function as a proof-of-concept for identification of the relevant cost driver with the employed method. The aim of the development is, in other words, not necessarily to maximize the performance of each individual module, but rather to enable an evaluation of all modules on the basis of whether or not their problem can be solved with the intended method or if some other method must be employed in a final system. The reason why this distinction is of importance is that maximization of performance often entails very detailed tuning of all relevant aspects of the network. In turn, this is a time-consuming process that is not feasible within the scope of this work since a large number of modules are considered. The testing approach employed during development is hence designed for the purpose of achieving sufficient proof-of-concepts with reasonable levels of effort, such that all modules can be evaluated.

The results of the development of each module will be presented and discussed in Section 6. Based on different assumptions relating to each cost driver and the experiences from already tested modules, the detailed approaches for building towards a final configuration will vary slightly from module to module. These detailed approaches will also be presented for each module in Section 6. In this subsection, the general testing approach and the common default settings that will be used in the development of the modules will be presented and discussed.

Before any development begins, the data acquired and prepared for each cost driver is split into a training set, a validation set, and a test set. The data is split such that the training set consists of about 70% of the data, the validation set of about 15%, and the test set also of about 15%. This set split is chosen based on the recommendations in [46]. All training is performed solely on the training set, and the validation and test sets are kept aside for evaluating the model performance. Specifically, during development, the performance on the validation data is considered. The final configuration is then validated on the test data. This follows the discussion of the sets in Section 3.2. Once the sets are split, the development can begin.

The general testing approach to develop the CNN modules is as follows:

1. Selection of default convolutional architecture
  - (a) Rough tuning of learning rate
  - (b) Dropout test
2. Validation of spatial model
3. Validation of multi-view model
4. Final tuning

### **Convolutional architectures**

In the first stage, a default convolutional architecture will be selected. A convolutional architecture, or CNN architecture, here refers to the layout and specifics of the convolutional layers within the convolutional block, as well as the adhering fully-connected layers in the prediction block. The models tested in this stage take only a single image input. That is, only the base CNN architecture is tested without any modifications, such as the inclusion of spatial information or a multi-view

implementation. As presented in Section 3.3, several such architectures exist. Since these existing architectures have proven to be highly efficient to describe geometry in image classification tasks [52] [82], this work will aim to employ these rather than attempt to develop a new such architecture from scratch. Which specific architecture to choose for different use-cases is, however, not clear. As a result, this first stage sets out to test and compare certain different architectures to investigate which of them achieves the superior performance on the relevant data for the specific cost driver.

Strengths and gaps of various architectures are presented in [52]. All of these existing architectures are found to have both different strengths and different gaps, meaning there does not exist some universal "right" architecture. The question is then how to narrow down the search for architectures to test, since not all can be tested in this work. Although performance on benchmark datasets, such as the ImageNet [83], can be consulted, different architectures are often found to be better for different use-cases.

To first impose a limitation as to which architectures are available for testing, this work will only be considering architectures that are readily available in the Keras module of the TensorFlow library in Python [84]. TensorFlow is a much-used library for machine learning applications and is recommended in [82] based on user ratings and author experiences. The choice to only consider such readily implemented models is primarily made in order to remove the implementation of the convolutional architecture as an error source in this work. Building a previously developed model architecture does in itself not entail much effort, but it would generate an error source that requires constant attention throughout the testing and development phases. When using an available and thoroughly tested model, this error source is removed. Errors relating to the overall finished architecture, for example after extending the base model to consider also spatial arguments, will still persist, but the complexity of the error search is greatly reduced. TensorFlow Keras offers a variety of state-of-the-art networks, and it is therefore assumed that this limitation will not limit the results of the work. In fact, Keras offers too many models to test all within the scope of this work.

To further narrow down the search for relevant architectures, networks that have previously been successfully employed in applications that are deemed relevant to this work are investigated. Multi-view approaches are considered particularly interesting due to their aim of 3D object recognition with a similar approach to what will be used in this work. That is, papers focused

on such approaches often have both similar data types and network types to those employed here. In addition, some works with similar data types are considered to be of interest. The findings of this investigation can be found in Table 4. Note that the table is ordered by network architecture and year, meaning that the listed order is not intended to imply higher or lower relevance.

Looking at Table 4, it is clear that the VGG architecture [54] is the most frequently used in the considered works. It should be noted that a reason for this simply could be that this is the architecture that was employed in the paper presenting the multi-view approach [61]. Still, it has proven its efficiency in several 3D object classification tasks. Following the VGG, are the ResNet [56] and AlexNet [55] architectures. Both of these are employed in three works each. The AlexNet architecture is, however, not available in TensorFlow Keras. This is also an older architecture than the two others, and some works, such as [88], mention that the choice of this architecture primarily is influenced by efficiency considerations rather than performance. Apart from the three architectures mentioned, the GoogLeNet, also known as Inception v1 [58], and Xception [92] are both employed once.

Based on the findings presented in Table 4, the VGG and ResNet architectures prevail as relevant for testing. Variations of both are also available in TensorFlow Keras. Since no other available architecture is employed more than once in the investigated works, however, further selections are more difficult.

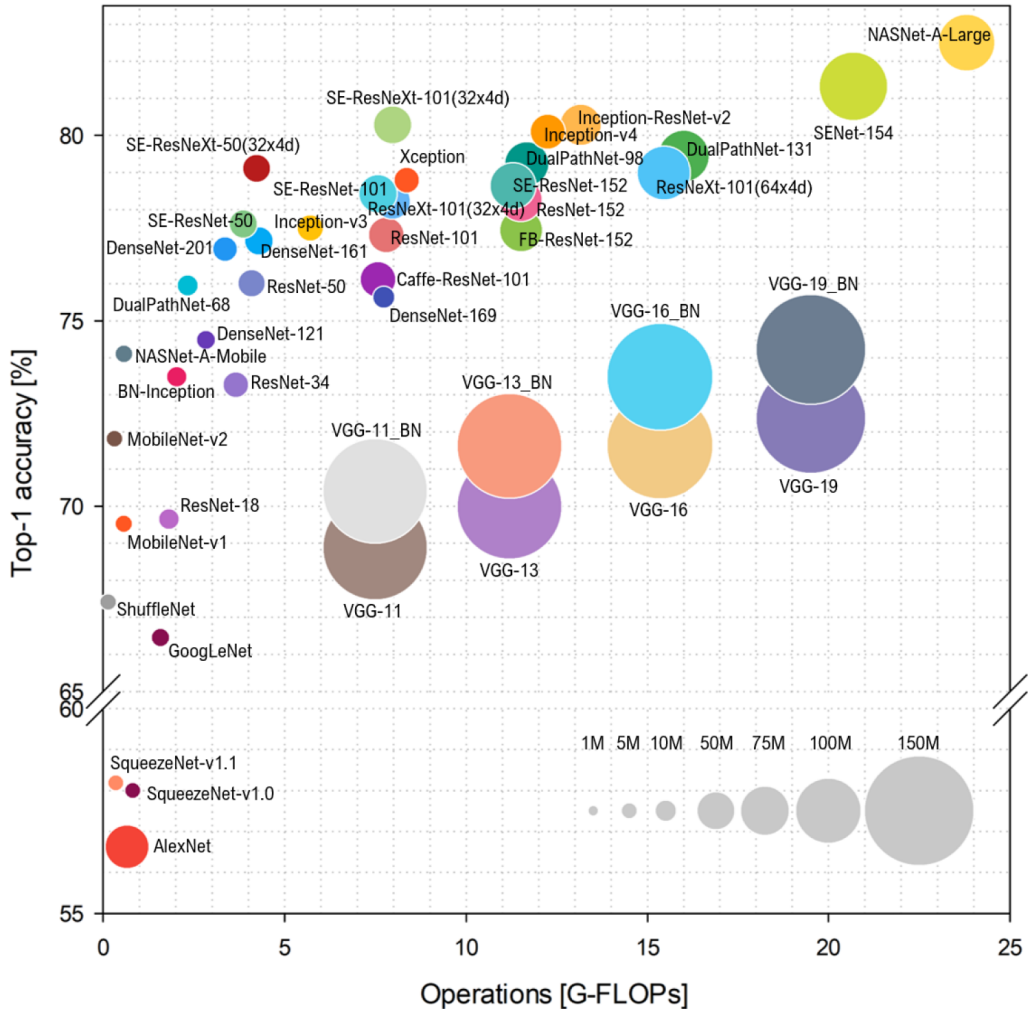
Although frequently used, the VGG and ResNet architectures do not particularly prevail when considering the performance on typical benchmark datasets. Still, as discussed above, performance on benchmark datasets does not necessarily indicate an overall superior architecture. Nevertheless, it is considered to be of interest to investigate also one architecture that has a top performance on such a dataset. As a result, Figure 27, which is Figure 1(a) in [93], is consulted.

In this figure, the Top-1 accuracy on the ImageNet-1k validation dataset for a number of different CNN architectures is presented. The Top-1 accuracy is the accuracy relating to predicting the correct class as the most probable prediction. That is, the class predicted by the network is the actual correct class. This is opposed to, for example, the Top-5 accuracy, which is the accuracy relating to predicting the correct class as one of the five most probable classes. That is, even though the prediction may be wrong, the correct answer is amongst the ones that the network considered to be the most probable. The Top-1 accuracy is considered to be of higher importance and is hence

<b>Paper</b>	<b>Relevance</b>	<b>Architecture</b>
Multi-view Convolutional Neural Networks for 3D Shape Recognition (2015) [61]	Multi-view	VGG-M
Multi-View CNN Feature Aggregation with ELM Auto-Encoder for 3D Shape Recognition (2018) [85]	Multi-view	VGG-M
Dominant set clustering and pooling for multi-view 3d object recognition (2019) [86]	Multi-view	VGG-M
An improved multi-view convolutional neural network for 3D object retrieval (2020) [87]	Multi-view	VGG-M
A novel learning-based feature recognition method using multiple sectional view representation (2020) [37]	Multi-view	VGG-11
Volumetric and Multi-view CNNs for Object Classification on 3D Data (2016) [88]	Multi-view	AlexNet
A Convolution Neural Network for Parts Recognition Using Data Augmentation (2018) [39]	Data type	AlexNet
Recognition of 3D Shapes Based on 3V-DepthPano CNN (2020) [89]	Multi-view	AlexNet
Learning Multi-View Representation With LSTM for 3-D Shape Recognition and Retrieval (2019) [90]	Multi-view	ResNet-18
Automatic Detection and Identification of Fasteners with Simple Visual Calibration using Synthetic Data (2020) [40]	Data type	ResNet-32
Multi-view classification with convolutional neural networks (2021) [63]	Multi-view	ResNet-50
GVCNN: Group-View Convolutional Neural Networks for 3D Shape Recognition (2018) [91]	Multi-view	GoogLeNet (Inception v1)
IDS-DLA: Sheet Metal Part Identification System for Process Automation Using Deep Learning Algorithms (2020) [42]	Data type	Xception

**Table 4** – Overview of CNN architectures used in previous relevant works.





**Figure 27** – Benchmark comparison of different convolutional neural network architectures on the ImageNet-1k validation dataset from [93]. The ball size is the *model complexity*, which is a measurement relating to the number of trainable parameters in the architecture.

the one that is evaluated here.

Considering the top architectures in Figure 27, it is first validated that neither a VGG variation nor a ResNet variation is amongst them. Of the four architectures with the highest Top-1 accuracy, both the NASNet-A-Large [94] and the Inception-ResNet-v2 [57] are available in TensorFlow Keras. The NASNet is the highest-scoring of all tested models. However, the NASNet is a dynamic architecture. This means that, in addition to tuning the weights and biases during training, also the architecture itself is changed to find the optimum for the current dataset. Although this is a very interesting research direction that shows promising results, it is considered to constitute a higher integration effort because these changing architectures would need to be accounted for in the modified spatial

multi-view models presented in Section 4.3.2. As a result, the Inception-ResNet-v2 is chosen as the third CNN architecture to test. This architecture, coincidentally, also uses somewhat similar inception modules to those in the GoogLeNet (Figure 13), which is one of the architectures left in Table 4.

The VGG, ResNet, and Inception-ResNet-v2 architectures are hence, based on related works and the performance on a benchmark dataset, considered to provide a good foundation for testing. As a result, no further architectures are pursued unless the chosen ones are assumed to be insufficient based on the results obtained during development.

Both the VGG and ResNet architectures do, however, have multiple variations, several of which are available in TensorFlow Keras. The difference between these variations is primarily the depth of the networks. That is, they employ the same ideas and have the same base structure, but they have varying numbers of convolutional layers (Figures 11 and 12). Because a deeper variation has more convolutional layers and trainable weights, it is typically assumed able to describe more complex patterns. These deeper variations do, for example, have a higher Top-1 accuracy on the ImageNet-1k benchmark dataset, as presented in Figure 27. However, some problems do not necessarily require such deeper variations in order to obtain the optimum performance for a specific architecture, and the deeper variations constitute a higher computational expense. In order to also test the impact of increased depth, it is decided to evaluate one shallow and one deep variation of both the VGG and ResNet architectures. The chosen variations are the shallowest and deepest variations available in TensorFlow Keras. This means that the CNN architectures that will be evaluated in Stage 1 are:

- VGG-16
- VGG-19
- ResNet-50
- ResNet-152
- Inception-ResNet-v2

Each of these CNN architectures will use transfer learning, adopting pre-trained weights in the convolutional block obtained upon training on the ImageNet benchmark dataset [83]. It is chosen to

employ transfer learning due to the technique’s reported efficiency both specifically in deep learning applications [95] and in general machine learning applications [96]. The ImageNet is chosen as the pre-training dataset both because it is a renowned dataset previously used for this purpose [82] and because the ImageNet weights are readily available in TensorFlow Keras. The latter removes the implementation effort associated with downloading some other dataset, preprocessing it, and subsequently training all architectures on that dataset to obtain transferable weights. In order to limit the efforts of Stages 1-3, fine-tuning of the models will not be performed in the general approach unless the results appear unreasonably poor relative to the expected performance at these stages. That is, only weights in the prediction block will be tuned during training in these stages unless otherwise stated.

### **General network settings**

Apart from the blank size prediction, all CNN modules solve classification problems. This means that they can use the same output activation as is used by default in the tested architectures (Figures 11 and 12), namely softmax activation. This is a natural choice in classification problems, since the softmax activation enables good interpretation of the network output due to its probability distribution nature [45][47]. For the regression problem of blank size prediction, however, softmax activation is not suitable. In this case, linear output activation is used since this is the standard choice in regression problems [46].

In order to train a neural network, both a loss function and an optimizer are required. The loss function will measure how close or far the network prediction is from the actual target, and the optimizer will determine the appropriate changes that should be made to minimize this loss. For classification problems, crossentropy loss functions are considered appropriate [46]. As a result, for the classification problems in this work, categorical crossentropy will be used as a loss function. The categorical crossentropy loss for a single data point is defined in Equation 6 in Section 3.2.

For regression problems, an appropriate and frequently used loss function is the mean squared error (MSE) [46]. Therefore, the regression problems in this work will employ the MSE loss function. The MSE loss for a set of data points is defined in Equation 8 in Section 3.2.

Stage 1 in the general testing approach will aim to determine which of the presented architectures is the most suitable to use in the continued development to evaluate a specific cost driver. However, neural networks have multiple tunable parameters, typically called hyperparameters. In order to

obtain representative results for all architectures, at least some of the hyperparameters must be evaluated independently for each one. The reason for this is that the optimum hyperparameter settings for an architecture can vary, even though the same data is evaluated. As such, some tuning is required to get representative results to compare the performance of each architecture. The different variations of the same CNN architecture are assumed able to use the same hyperparameter settings, and, as a result, only the shallow variations will be subject to tuning, since these demand less computational effort. The deep variation will then inherit the best settings found for the shallow variation in order to perform the final comparison. The assumption that the different variations of the same architecture can employ the same hyperparameter settings is supported by the papers presenting both the VGG [54] and the ResNet [56].

Tuning hyperparameters is a challenging task that can be very time-consuming [45] [97]. As a result, only the hyperparameters that are considered most pivotal to acquiring somewhat representative results for each architecture will be subject to tuning in Stage 1. [97] presents learning rate as the most important hyperparameter to tune. [45], [46], and [47] also stress the importance of this hyperparameter. Because of this, the learning rate is chosen as the primary hyperparameter to be tuned for each architecture. In addition to the learning rate, it is considered particularly interesting to test dropout [49] [50] with each architecture. The reason for this is that dropout is applied in the base configurations of both the VGG and Inception-ResNet-v2. The term base configuration here refers to the settings of the models in the papers presenting each architecture. That is, when testing and comparing the performance of the architecture to other existing architectures, dropout is applied in the prediction blocks of the VGG and the Inception-ResNet-v2.

Since only learning rate and dropout will be tuned in Stage 1, certain default values must be established for other hyperparameters.

The Adam optimizer [98] is chosen as the optimizer in this work. Adam is a gradient descent variation that aims to optimize the learning process by adapting the initial learning rate for each individual parameter through the use of the moments of the gradient. This optimizer is chosen primarily because it generally is considered to be robust with respect to hyperparameter choices, meaning strong results can be obtained even without exhaustive tuning of these [47]. Apart from the learning rate, the Adam optimizer will use the default values suggested in [98]. These are:  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$ , and  $\epsilon = 10^{-8}$ .

A batch size, or mini-batch size, of 32 is used during training. This number is chosen based on the recommendations in [97] and the experimental results in [99].

### **Learning rates**

Having chosen to test different learning rates and dropout configurations, the subsequent question is which values to test. [97] states that typical learning rate values are smaller than 1 and larger than  $1e-6$ . However, as also stated in the paper, no definite rules exist as to which learning rates will perform the best in a certain use case. Nevertheless, the interval  $[1e-1, 1e-5]$  provides a place to start. In the event that the best results for an architecture are obtained for a learning rate on the border of this interval, further learning rates outside the interval on that side will additionally be tested. Although an interval to explore is now set, which learning rates to test within this interval must also be specified. For optimization of numerical hyperparameters, such as the learning rate, [97] suggests searching for an optimum on the logarithmic scale. This is because the impact of changing a learning rate from one number to another generally is found to be dependent on the relative size of the change rather than the absolute size of the change. For example, even though the change from 0.1 to 0.11 has the same absolute size as the change from 0.01 to 0.02, the latter change is generally found to have a larger impact, as this entails doubling the previous learning rate. A logarithmic search is supported by the broad strategy for learning rate tuning presented in [45] for tuning the learning rate when using the stochastic gradient descent (SGD) optimizer. As a result, for rough tuning of the learning rate to achieve somewhat representative results for each architecture in order of comparing them, the following learning rates will be tested:

- 1e-1
- 1e-2
- 1e-3
- 1e-4
- 1e-5

The learning rate that achieves the highest performance with a specific architecture is considered the best for that specific architecture. In the event that two learning rates have indistinguishable performances, the higher learning rate will typically be chosen. This is because a higher learning

rate often is considered superior, both because of reduced overfitting and increased convergence [46] [100]. In many cases, however, when the separation of two or more learning rates is difficult, multiple learning rates are tested with the dropout configuration in order to evaluate if they can be separated based on the results with this.

Learning rate decay is not used in this work. Learning rate decay is a technique where an initial learning is first set and then subsequently dynamically reduced as training progresses. The reductions follow either some set schedule or rules relating to the training and validation data results during training [45]. This differs from the traditional case where the learning rate is constant throughout training. The technique is occasionally used because results sometimes are improved when the early training phase employs a higher learning rate which enables quick convergence through taking large steps towards the optimum, whereas the latter training phases can benefit from lower learning rates to fine-tune the weights and biases to approach the exact optimum. However, [97] reports that the benefits of such techniques are small in many cases. Using learning rate decay also increases the number of tunable parameters to account for in the development stages, thereby entailing an increased testing and development effort. As a result, it is chosen to not pursue this technique.

## **Dropout**

Dropout can be applied to any fully-connected layer in a network. The amount of dropout exists on the predetermined interval of  $[0\%, 100\%)$ , or  $[0, 1)$ . That is, the probability of "dropping" a neuron in a fully-connected layer can be any value ranging from 0%, meaning that dropout is not applied and no neurons are "dropped", to, but not including, 100%, which would mean that all neurons were certainly "dropped". Note that, in this work, the amount of dropout will be referred to by the percentage or ratio probability to "drop" a neuron. 20% dropout, or a dropout of 0.2, then means that there is a 20% probability that a neuron in the respective hidden layer is "dropped". This follows the convention used by TensorFlow Keras but is opposed to some other works where the dropout rate is referred to as the probability that a neuron is "kept".

[50] states that typical amounts of dropout range from 20% to 50%. This is in line with the dropout used in the base configurations of the VGG and Inception-ResNet-v2. The paper also discusses which learning rate that is appropriate for the dropout configuration relative to the optimum learning rate for the no-dropout (regular) configuration. In general, they find that the

dropout configuration should use a learning rate that is 10-100 times higher. However, up-scaling of the activations in the neurons that are not "dropped" during training is presented as equivalent to "*appropriate scaling of the learning rate*". Such an up-scaling is performed by default in the dropout layers in TensorFlow Keras, which are used in this work. The statement that up-scaling removes the need to use a learning rate that is 10-100 times higher is also validated in trial runs for the development of the Classification #2 module. The general testing approach will, therefore, simply use the best performing learning rate found for the no-dropout configuration when testing dropout.

In order to limit the efforts of Stage 1, only a single dropout configuration will be tested for each architecture. For the VGG and Inception-ResNet-v2, the dropout configuration that is tested is the same as the one that is used in their base configurations. For the ResNet architecture, which does not employ dropout in the base configuration, the same dropout configuration as the Inception-ResNet-v2 is tested. This is because the two architectures have the same prediction block, which consists of only a single fully-connected layer with softmax activation. The following dropout configurations will hence be tested for each architecture with the learning rate that performed the best on the no-dropout configuration:

- VGG: 50% dropout applied to each of the fully-connected layers of 4096 neurons in the prediction block.
- ResNet-50: 20% dropout applied to the output of the convolutional block.
- Inception-ResNet-v2: 20% dropout applied to the output of the convolutional block.

### **Stages 2, 3, and 4**

The best-performing architecture and adhering configuration from Stage 1 will be chosen to proceed to Stage 2. In this stage, the aim is to validate the use of spatial information in the model by employing the modified architecture in Figure 19, presented in Section 4.3.2. This model is then compared to the best-performing model in Stage 1.

The default spatial argument that is used is the global bounding box. The reason for this is that the global bounding box is assumed to both be able to provide good general spatial information about the part in question, as well as the fact that it has a direct relation to the spatial information

in a specific view. The direct relation is, for example, that a part imaged in the x-view will have the bounding box dimension in the y-direction as its horizontal length and the bounding box dimension in the z-direction as its vertical height. This could potentially allow the model to consider some more intricate patterns between the spatial and geometrical information. The minimum bounding box was initially also considered as the default spatial argument since it alone arguably provides a better spatial description than the global one. That is, the minimum bounding box is invariant to the part orientation, and it will hence provide a more consistent spatial measurement. However, the minimum bounding box does not have the same direct relation to the different views. The use of the minimum bounding box can also potentially be somewhat problematic when using augmented datasets. This is exactly because it stays constant through the rotations imposed on the base dataset to create the augmented one. This means that, whereas the multi-view image representations of the different components change when the part is rotated, the spatial information would merely be duplicated. If the validation of the spatial model fails and this is assumed to be a result of the spatial information that is used, other information than the global bounding box can be tested.

In Stage 3, multi-view models will be tested to validate that the multi-view method, as assumed, increases the performance. Additionally, the stage aims to determine which fusion technique performs the best on the relevant problem with the relevant dataset. If the use of a spatial model is validated in Stage 2, the modified architectures in Figures 20 and 21 will be evaluated, and their performances compared to the Stage 2 model. If not, multi-view models without spatial information (Figure 15) will be compared to the best model from Stage 1. Following the practice in [63], the fusion layer in all the late fully-connected fusion models consists of 1024 neurons with ReLU activation and 50% dropout applied during training, unless otherwise stated.

Stage 4 aims to increase the performance of the model through further tuning of the best configuration after Stages 1-3. The goal of this stage is to achieve a sufficient model performance if this is not obtained in Stages 1-3. That is, to reach a performance that is considered to provide a proof-of-concept for the employed method to evaluate a specific cost driver. The stage will, in other words, not aim to exhaustively tune all tunable parameters to achieve some optimal performance. The reason not to strive for an optimal model performance is, as previously discussed, that this is outside the scope of this work since such a process is too time-consuming when considering the number of different modules that will be developed.



Generally, Stage 4 entails fine-tuning of the configuration. That is, additionally tuning the weights in the convolutional block, which up to this point typically will be the same weights that are transferred from the pre-trained models. The fine-tuning will normally be tested both with the learning rate that performed the best in Stage 1, as well as a couple of lower learning rates spaced out on the logarithmic scale with distances of 1. The reason for testing also some lower learning rates is that pre-trained weights often only require smaller adjustments and that higher learning rates sometimes can over-adjust the already tuned weights [101]. In addition to fine-tuning, more refined tuning of already tuned hyperparameters or tuning of default valued hyperparameters can be performed in this stage. However, care is taken to not get lost in the model tuning since time will be a factor. That is, if a sufficient performance is achieved, none or only a select few refinements will be done. If a sufficient performance is not achieved within a few refinements, an evaluation is made in the individual case as to how to proceed.

### **Training process**

A question that arises when testing and comparing various architectures and configurations is when to stop the training of a certain configuration. That is, how many epochs the network should be trained for. [97] describes the number of epochs as one of the hyperparameters that can be "*optimized almost for free*" through employing early stopping. When using early stopping, the training is stopped once the network stops improving over a certain amount of epochs. As a result, no predetermined number of training epochs need to be set, since the network will simply train until some saturation is obtained. The use of this technique is deemed preferable in this work in order to obtain representative results for different architectures and configurations, all of which may require a different number of epochs to achieve a representative performance. Using early stopping entails the introduction of two parameters that need to be set. These are the monitoring criterion and the patience. The monitoring criterion is what the early stopping evaluates to see if the performance improves or not, whilst the patience is how many epochs to wait for improved results before ceasing training. Both [47] and [97] suggest using the validation loss as the monitoring criterion. This also makes intuitive sense since the loss is a measurement of how good a network's prediction is and is what the network itself attempts to reduce during training. The second parameter, namely the patience, is somewhat more difficult to set. [45] suggests that the "*no-improvement-in-ten rule*" is a good place to start. As a result, the early stopping patience is initially set to 10 epochs. If it is suspected, based on, for example, the validation loss and accuracy curves, that this is too

strict, the patience will be increased. Such a suspicion could, for example, arise from seeing that the curves still appear to have a steady natural improvement at the time when training is stopped. The weights from the best epoch are restored, meaning that the weights in the stored model that can be used for future validation, testing, or applications, are the ones that generated the minimum validation loss during training.

Three independent training runs are made for each configuration. That is, each configuration will be trained independently three times. The reason for this is that the weights that are not transferred from a pre-trained model, namely the weights in the prediction block, are randomly initialized. This means that the results can differ in between independent runs with identical architectures and configurations because the initial weights differ. To ensure that the optimum performance for a configuration is achieved, more than three independent runs would typically be needed [102]. However, model training demands much time and effort, and a significant amount of models need to be trained throughout all testing stages in this work. As such, one trait of a good model configuration is, in this work, that the configuration can achieve a representative and good performance in few independent runs. Three independent runs are chosen to try and achieve such representative results for each configuration. This also enables investigations of the variation in the performance of the models, which in turn allows for evaluation of the configuration's stability. The run with the highest performance is considered when comparing configurations.

### **Performance indicator**

The above discussion frequently mentions that the performance of a model configuration is used to compare it to other models and configurations. In order to measure the performance, however, a performance indicator is required.

In general, this work will employ the accuracy as the key performance indicator for categorical problems. Specifically, during development, the validation accuracy is the key performance indicator. As previously discussed, the validation data is not used to train the model. The validation accuracy is hence a good measurement of how well the model learns the general patterns associating the input with the output during training. The test data is kept aside for validation of the final configuration.

In classification problems, the accuracy, or classification accuracy, is a measurement that describes how many times the model makes a correct prediction relative to the total amount of predictions

that are made. The simplest calculation of the classification accuracy is:

$$A\% = \frac{N_{correct}}{N_{possible}} \times 100\% \quad (29)$$

Certain individual differences will be applicable to the different modules for the different cost drivers, and these will be discussed for each one in Section 6.

The accuracy is chosen as the key performance indicator because it is a reliable and consistent measurement that exists on a predetermined interval. That is, the meaning of 1% accuracy as  $\frac{1}{100}$ , or one in a hundred, is considered helpful when comparing and evaluating the models. Using the model accuracy as the key performance indicator when using the loss as the early stopping criterion could potentially be perceived as contradictory. However, as previously discussed, using the loss to monitor early stopping is in line with what the model is actually aiming to achieve during training, namely minimizing the loss. The question then arises as to why the accuracy is used instead of the loss as the key performance indicator, and this is answered by the above arguments regarding the accuracy's reliable interpretation. Both the categorical crossentropy and mean squared error losses can take on any positive real value, which makes it more difficult to assess the differences between models without sufficient data to make relative comparisons.

For regression problems, the mean absolute percentage error (MAPE) is used as the key performance indicator in this work, unless otherwise stated. For a set of data points, this is defined as:

$$MAPE = \frac{100\%}{N_b} \sum_{i=1}^{N_b} \frac{|y_i - \hat{y}_i|}{y_i} \quad (30)$$

where  $N_b$  is the number of data points evaluated,  $y_i$  is the target for one of those data points, and  $\hat{y}_i$  is the predicted output for that data point. If only a single data point were evaluated,  $N_b$  would be 1, and the summation would subsequently disappear. The reason for choosing the MAPE as the key performance indicator for regression problems is that this metric provides a reliable measurement of how good a prediction is relative to the target.

An important note on measuring the model performance is that analysis beyond merely considering the key performance indicator will be made to evaluate each configuration. The key performance

indicator can not be blindly trusted to provide a complete assessment of the performance of a model. As a result, additional analysis is performed, such as evaluation of graphs produced during the training, observation of the relation between the training and validation data performances, and evaluation and comparison of the training and validation losses.

## Summary

A summary of the results from the discussions in this subsection is included below.

The general testing approach used to develop each CNN module is as follows:

1. Selection of default convolutional architecture
  - (a) Rough tuning of learning rate
  - (b) Dropout test
2. Validation of spatial model
3. Validation of multi-view model
4. Final tuning

Stage 1 will compare the following base CNN architectures and proceed with the one that achieves the highest performance on the relevant dataset:

- VGG-16
- VGG-19
- ResNet-50
- ResNet-152
- Inception-ResNet-v2

In order to try and acquire somewhat representative results to compare the architectures, the learning rate will be tuned for each architecture individually. The following learning rates will be tested:

- 1e-1
- 1e-2
- 1e-3
- 1e-4
- 1e-5

For the different variations of the same architecture, only the shallow variations are subject to this tuning. The deep variations will inherit the learning rate that achieved the highest performance with the shallow one.

In addition to tuning the learning rate, one default dropout configuration will be tested with each architecture. The learning rate that is tested with the dropout configuration is the same as was found to achieve the highest performance with the no-dropout configuration. The following dropout configurations are tested:

- VGG: 50% dropout applied to each of the fully-connected layers of 4096 neurons in the prediction block.
- ResNet-50: 20% dropout applied to the output of the convolutional block.
- Inception-ResNet-v2: 20% dropout applied to the output of the convolutional block.

All results from Stage 1 are assessed, and the architecture and adhering configuration that achieves the highest performance is chosen to proceed to the subsequent testing stages.

In Stage 2, the inclusion of spatial information is validated by using the model in Figure 19. The default spatial information that is passed to the model will be the global bounding box.

In Stage 3, the extension of the single-view model to a multi-view model is validated.

In Stage 4, final tuning is performed to further improve the model performance.

The default settings and values that are used during the development, unless otherwise stated in Section 6, are:

- Set splits: 70% training, 15% validation, 15% test.
- Key performance indicator: Accuracy for classification problems, mean absolute percentage error for regression problems.
- Transfer learning: Yes. Models are pre-trained on the ImageNet dataset. Unless otherwise stated, model training entails only tuning of the weights in the prediction block in Stages 1-3.
- Output activation: Softmax for classification problems, linear for regression problems.
- Loss function: Categorical crossentropy for classification problems, mean squared error for regression problems.
- Optimizer: Adam. Default settings:  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$ , and  $\epsilon = 1^{-8}$ .
- Learning rate decay: No.
- Batch size: 32
- Early stopping: Yes. Monitors validation loss with 10 epochs patience.
- Number of independent runs: 3.

## 6 Results and discussion

In this section, the results from the development of the various learning-based modules are presented and discussed.

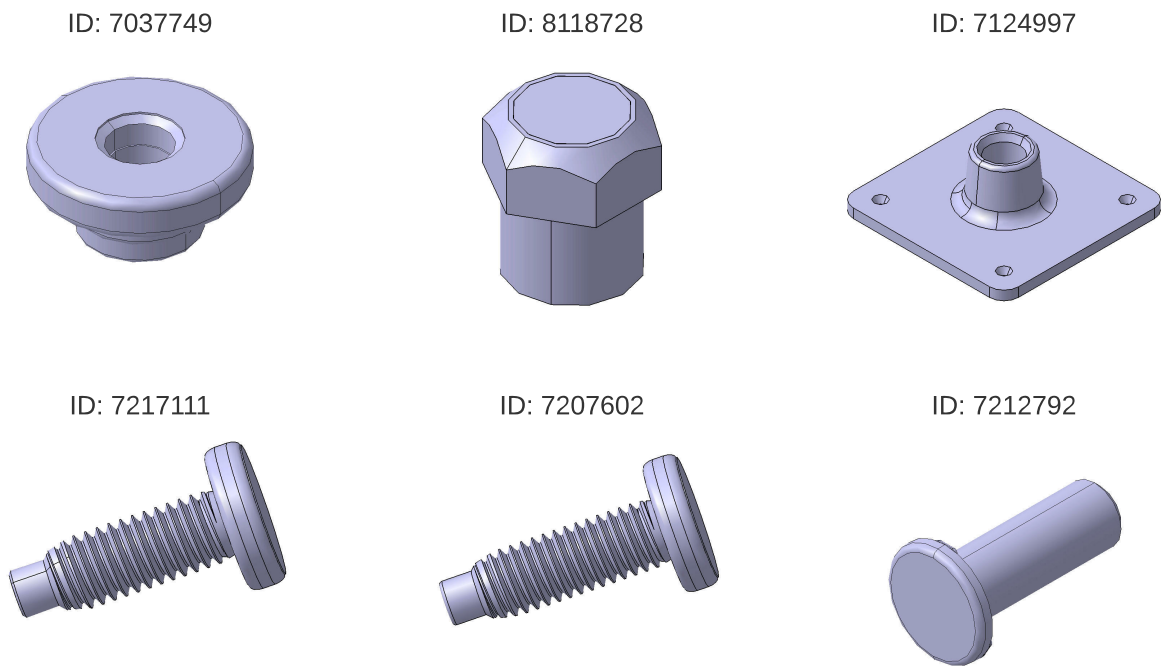
First, the results for each individual module are presented in the order of their development. That is, starting with the first module to be developed and ending with the last. The order of development is inspired by the system architecture in Figure 23. The subsections for each module will have the same general outline, first introducing and analyzing the data related to that module, before presenting the specific testing approach that is employed in its development. After outlining the testing approach of a module, the results from its development are presented and discussed. Each subsection is finalized with a summary of the development of the specific module and some propositions for future works on the specific topic. After presenting and discussing the results for each module, an implemented prototype system is presented together with a case study. Finally, the development is summarized by tabulating all results and discussing some common topics amongst the different modules. For the reader to whom the technical details of the development are not of great interest, the focus could be on the prototype system and summary in Sections 6.8 and 6.9.

### 6.1 Classification #2 - Fastener ID

In Classification #2, the aim is to evaluate a fastener to determine its fastener ID. In total, there are 820 unique fasteners of interest in this work, which means that the problem in Classification #2 is a categorical problem with 820 different target classes.

#### 6.1.1 Data

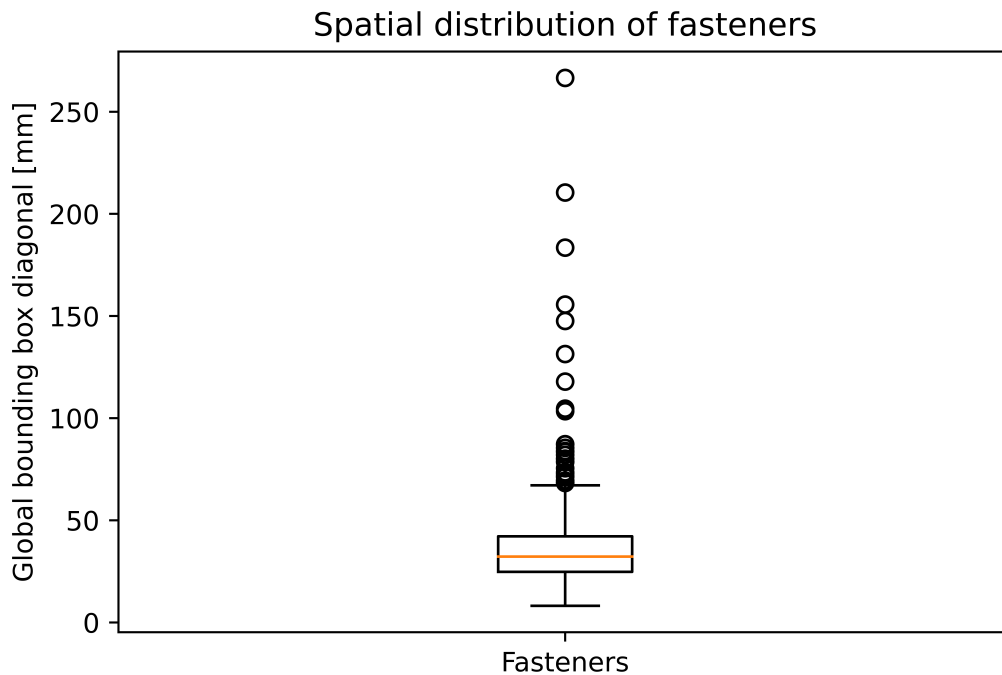
Figure 28 presents examples of fasteners and their IDs. Several of these fasteners have clear distinctions and unique traits, which means they are easily separated by considering only their images. However, as can be observed for IDs 7217111 and 7207602 on the bottom row in the figure, some of the 820 fasteners have a high geometrical similarity. Both these parts have threads and similar bottoms and heads. In turn, this means that they are very difficult to tell apart simply by considering their images. In fact, upon closer investigation, it is found that the only difference between the parts with IDs 7217111 and 7207602 is that the part with ID 7207602 is slightly longer. The spatial information relating to each fastener is therefore also considered to be of great importance to the final classification.



**Figure 28** – Examples of fasteners and their IDs.



In order to analyze the spatial distribution of the fasteners, a boxplot is created. This boxplot can be found in Figure 29. The diagonal of the global bounding box, calculated as  $BB_{diag} = \sqrt{BB_x^2 + BB_y^2 + BB_z^2}$ , is used as the spatial measurement. A boxplot is a frequently used graphical tool to illustrate a data distribution. In this plot, the median of the dataset is indicated by the orange line, whereas the lines above and below the median indicate the limit for the 75th and 25th percentile, which together create the "box". In Figure 29, it can, for example, be observed that the median bounding box diagonal for the fasteners is approximately 35mm and that 75% of the fasteners have bounding box diagonals smaller than 50mm. Outside the box itself, illustrations of the "maximum" and "minimum" are provided by horizontal lines. Outliers, meaning data points outside of the illustrated "maximum" and "minimum", are indicated by circles. In the figure, it can be observed that the largest bounding box diagonal of a fastener is about 270mm, whilst the smallest is just around 10mm.

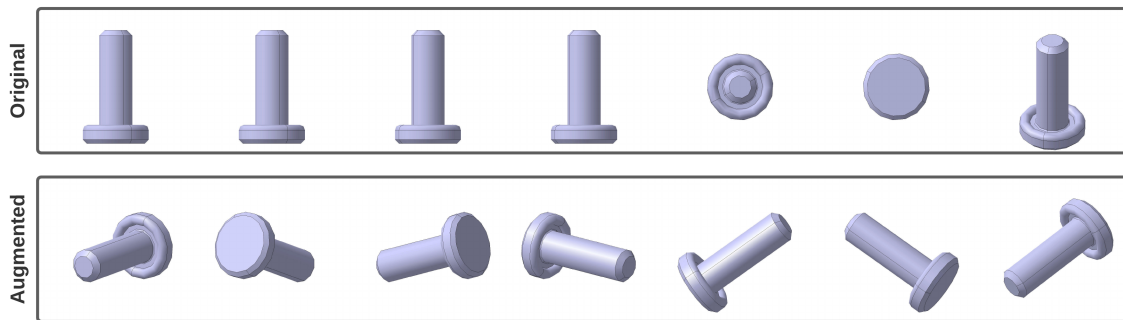


**Figure 29** – Spatial distribution of fasteners.

The fasteners are a special case with regards to data. Since a certain fastener ID adheres to one specific geometry, only a single CAD part model of each ID exists. This means that the base dataset, which is not subject to any augmentation and hence made up solely of the multi-view representation of all unique CAD models, contains very few examples of each fastener ID. In fact, it means that only 7 image examples, 1 for each view, exist for each ID in the base dataset. However,

in real use-cases, the fasteners can have any orientation, which means that not all real images will be identical to these 7 examples, although the CAD geometry itself is. A mere 7 examples is very little in terms of teaching a network to recognize the general patterns of a class.

Since the only natural variation of the fasteners in real use-cases is the part orientation, however, this also means that any augmentation through rotation can be considered a brand new and real data point. That is, for example, both the original and augmented part in Figure 30 can be considered full-worthy independent examples of the adhering fastener ID. As such, augmentation of the fasteners through rotation can be considered equivalent to actually increasing the number of real training examples. This is opposed to the normal case for augmentation, where this technique is found to typically aid generalization, but an augmented example can not be considered equal to a brand new one.



**Figure 30** – Example of augmented fastener.

Since there are very few examples of each fastener in the base dataset, an augmented dataset is generated for this problem, where each fastener is subject to 15 random rotations. Hence, an augmented dataset with 16 times the size of the base dataset is available if the base dataset proves insufficient. This dataset then contains  $16 \times 7 = 112$  image examples of each fastener. The size of this and other augmented datasets in this work is, as previously discussed in Section 5.2.2, determined based on the capacity of the server where the training is performed. If, however, this augmented dataset proves insufficient, more augmentation is possible.

The training, validation, and test sets are split according to the default split discussed in Section 5.3.2. That is, the complete dataset is randomly split such that about 70% of the data is contained in the training set and about 15% is contained in each of the validation and test sets. However, some logic is also included upon splitting the sets in this problem. This logic ensures that at least

one example of each fastener is available in each set. For the base dataset, this means that at least one image of each fastener exists in all sets. For the augmented dataset, it means that at least one multi-view example (7 images) of each fastener exists in each set. In multi-view, the augmented dataset needs to be used since there only exists a single multi-view example of each part in the base dataset. This makes it impossible to evaluate the network performance since no training examples would exist for any of the parts in the validation or test sets.

The reason for including some logic when splitting the datasets for this problem is to try and ensure that all sets are representative of the full dataset. If, for example, one ID were to be excluded from the training set, there is no way for the model to learn to recognize that ID. If another ID were excluded from the validation set, there is no way to validate the network's performance at recognizing that ID.

### 6.1.2 Testing approach

The testing approach that will be employed to develop the Classification #2 module to determine the ID of a particular fastener is as follows:

1. Selection of default convolutional architecture - **type target**
  - (a) Rough tuning of learning rate
  - (b) Dropout test
2. Validation of spatial model
3. Validation of multi-view model
4. Final tuning

The outline itself is identical to the one presented in Section 5.3.2. There is, however, one noteworthy difference. Since spatial information is assumed necessary to determine the exact fastener ID in many cases, Stage 1, where spatial information will not be available to the models, will not use the fastener ID as a target. This is in order of not confusing the models by including multiple training examples that have identical geometry but different targets. This would generate an error source that potentially could influence the convolutional architectures differently and hence

be a factor in their performance when compared to each other. As a result, the target output in Stage 1 is chosen to be the fastener type. As opposed to the coarse categorization that is required for the cost estimation, only separating impress elements, welded elements, and neither, this stage will use more specific fastener types as a target. In Figure 28, for example, IDs 7037749 and 8118728 are of type "press-in nuts", ID 7124997 is a "weld nut", IDs 7217111 and 7207602 are "riveted threaded studs", and ID 7212792 is a "welded unthreaded stud". In total, there are 18 different fastener types that are distinguished, making the problem in Stage 1 a categorical problem with 18 target classes. This is then opposed to the 820 target classes intended in the final network. Since the identical dataset is used only with coarser output classes, however, results on the coarse problem are assumed to still function as a basis for comparison of the performance of the different convolutional architectures on the current dataset.

An additional benefit to using the type as the target in Stage 1 is that many more examples are available for each class. This means that strong results can likely be obtained without the use of the augmented dataset. In turn, this leads to a much lower computational expense.

The key performance indicator for the models in this module is the regular validation accuracy, as described in Section 5.3.2. However, attention is also directed to the validation loss and the curves of the validation and training accuracy and loss generated during training.

### **6.1.3 Results and discussion**

Table 5 shows the characteristics of the final network developed for the Classification #2 module. The final configuration, targeting the fastener ID, is observed to achieve a validation accuracy of 96.81% for a configuration of the VGG-16. The validation accuracy is further supported by the accuracy on the test set, which was kept out of consideration by both the network and designer throughout the development. The test accuracy of 96.90% hence validates the accuracy of the final configuration.

The remainder of this chapter is dedicated to presenting and discussing results from the development process.

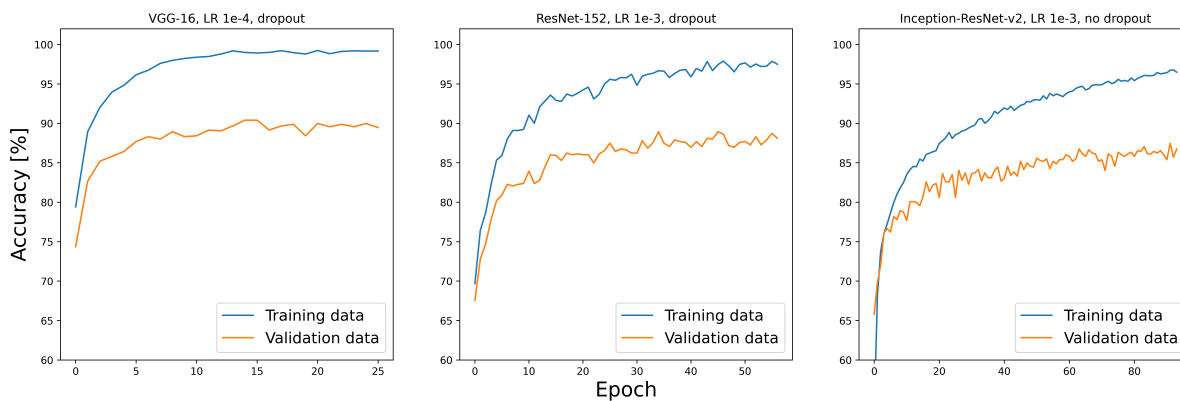
#### **Choice of convolutional architecture**

The choice of the VGG-16 as the convolutional architecture follows the results from Stage 1. These

Classification #2	
Target	Fastener ID
Validation accuracy	96.81%
Test accuracy	96.90%
Convolutional architecture	VGG-16
Learning rate	1e-4
Fine-tuning learning rate	1e-4
Dropout	50% on both FC4096
Spatial input	Global bounding box
Multi-view fusion technique	Product-score

**Table 5** – The final configuration of the network in Classification #2.

results are presented in Figure 31 by the training histories of the best configurations for each convolutional architecture after the rough learning rate tuning and dropout test. Only the best of the shallow and deep variation for the VGG and ResNet is included.



**Figure 31** – The best models with each architecture in Classification #2 - Stage 1.

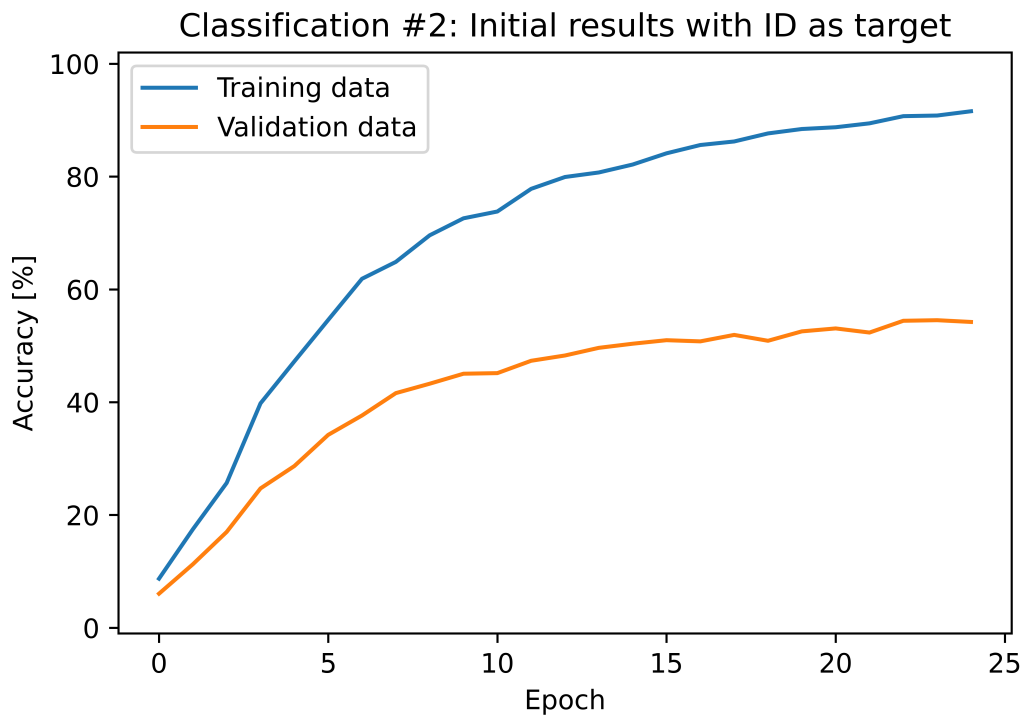
The training histories in Figure 31 display the evolution of the validation and training accuracies on the y-axis over the course of the epochs for which the models trained on the x-axis. From the graphs, it can be observed that the highest validation accuracy for all models is around 90%, with the VGG-16 having the highest validation accuracy of 90.41% around epoch 15.

Although the results in Stage 1 are close, the VGG-16 is observed to have superior accuracy to the other architectures. This is true for all three independent runs made with the presented

configuration. This accuracy is additionally achieved through training over fewer epochs, and the VGG-16 is the most efficient of the tested architectures in this stage. Following this, the VGG-16 is chosen as the base architecture to use in Classification #2.

### Impact of augmentation

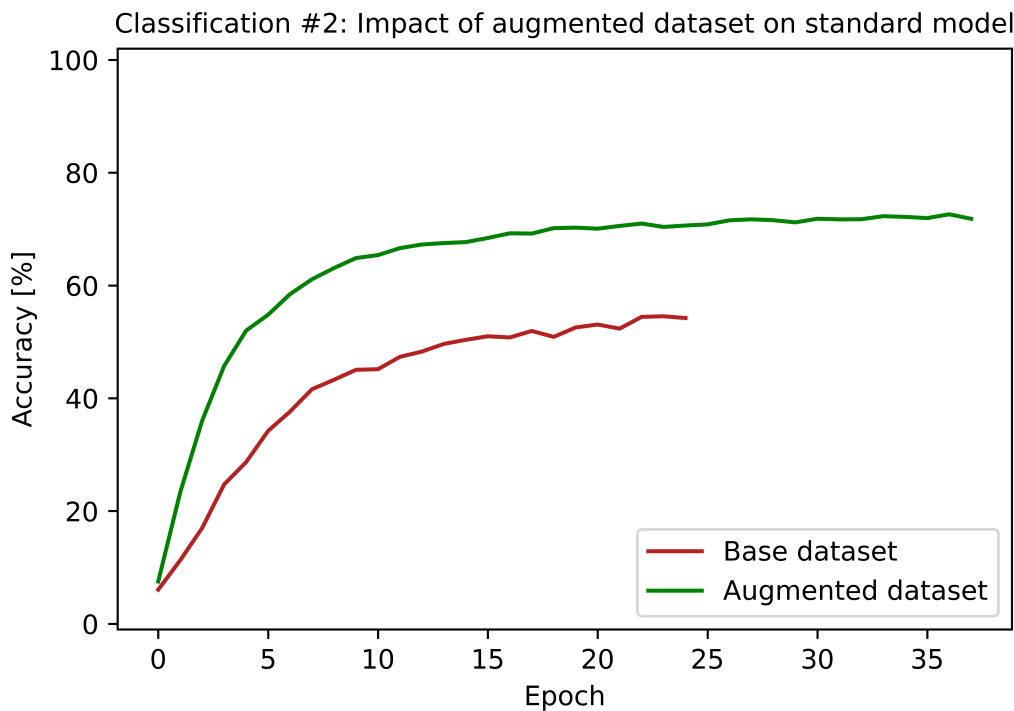
The accuracies of the models in Figure 31 must not be compared to the ones reported in Table 5. The reason for this is that Stage 1 targets the fastener type rather than the ID, as discussed in Section 6.1.2. When changing the target to the ID, the accuracy of the standard VGG-16 model from Stage 1 suffers a drastic decrease. The training history of this model when trained on the base dataset to predict the fastener ID, is presented in Figure 32. In this figure, it can be observed that the validation accuracy is below 60%, meaning it would not even be visible in the graphs in Figure 31, which start at 60%.



**Figure 32** – The initial results when changing the target from fastener type to ID in Classification #2.

The decrease in accuracy that is experienced when changing targets is, however, expected. The reason for this is that the presented model only considers a single image input and is trained on the base dataset. In the base dataset, only 7 examples in total exist for each ID, which means that at most 5 examples are contained in the training set. When instead training on the augmented

dataset described in Section 6.1.1, the performance is observed to drastically increase. Figure 33 shows a comparison of the validation accuracy for the standard VGG model when trained on the base and augmented datasets. It can be observed that the change in dataset results in an increase in accuracy of more than 15%, from around 55% to above 70%. Because of this, the augmented dataset is employed in all subsequent training.

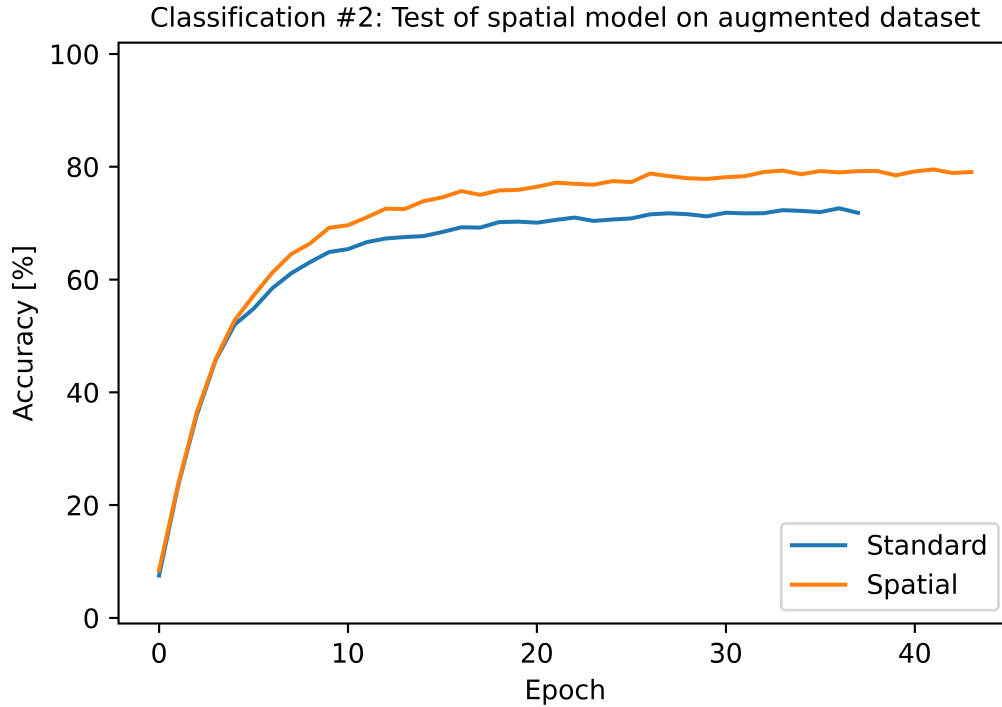


**Figure 33** – The impact of an augmented dataset on the validation accuracy when training the standard VGG-16 to target the fastener ID in Classification #2.

### Inclusion of spatial information

When extending the standard VGG-16 model to a spatial one (Figure 19) and including the global bounding box as spatial information, the results are further increased. A comparison of the validation accuracies of the standard and spatial model is presented in Figure 34. Including this spatial information increases the maximum validation accuracy by almost 7%, from 72.62% to 79.51%.

A question that naturally arises when passing new information to a neural network is if this information should be preprocessed in any way. Normalization is, for example, a common way to preprocess network inputs to ensure correlation between the magnitudes of the inputs themselves



**Figure 34** – The validation accuracies of the standard and spatial VGG-16 models trained on the augmented dataset in Classification #2 - Stage 2.

and between the inputs and the initial weights in the network [46]. Since the spatial information is concatenated with the output of the convolutional block, the interval on which the convolutional output exists for the standard VGG-16 model needs to be considered. Upon investigation, this interval is found to be  $[0, 373]$  for the current dataset. These sizes are comparable to the sizes of the global bounding boxes for the fasteners, as presented in Figure 29. As a result, it is decided not to pursue normalization of the spatial information in the development of this module unless the results prove insufficient. The spatial input used in the spatial model in Figure 34 is, in other words, the actual global bounding box and is not subjected to any modification.

### Error analysis

Another question that arises when passing new information to a network is how the network processes the information. This is especially important to investigate for the fasteners, since the employment of the spatial model in Stage 2 of the development of Classification #2 constitutes the first use of this modified architecture. As a result, it must be validated that the inclusion of spatial information does not generate strange network behavior, which could suggest overfitting to the



spatial information. However, neural networks are often considered to be black boxes. This means that analyzing and evaluating exactly what happens inside the network to determine how it turns the input into the output is difficult and abstract. That is compared to more traditional rule-based methods where a specific mathematical function is used to turn the input into the output. Much information about how the network "thinks" can, however, be obtained by analysis of the errors it makes. As a result, the errors made by the spatial model in Figure 34 are analyzed to try and validate that the increase in accuracy is achieved because the network uses the spatial information to complement the geometrical information from the images.


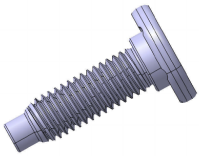
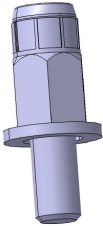
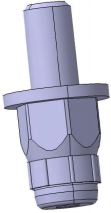
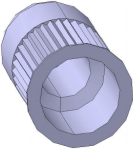
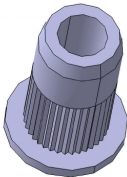
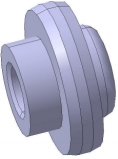
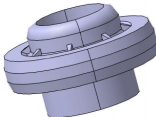
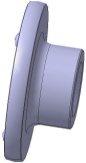
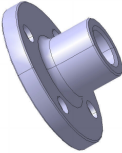
Figure 35 presents 5 representative examples of errors made by the spatial model. In the figure, the input image of a fastener is displayed to the left, whereas an image of the fastener adhering to the falsely predicted fastener ID is displayed to the right. That is, the image to the left is what the model receives as an input, and the image to the right depicts the fastener that the model wrongly thinks it is. By observation of these errors, it is evident that the geometrical similarity is high between the actual and predicted fasteners. In the first row, for example, both fasteners have threads, similar heads, and similar ends. In fact, upon closer investigation of the fastener data, it is found that the only difference between the two is that one is slightly longer than the other. In the third row, both parts have the same pattern around the base. The remaining three examples also display such clear similarities in geometrical patterns between the actual and predicted part. This is a general trait in the errors that are analyzed.

The analysis of the errors made by the spatial model suggests that the network makes reasonable predictions. That is, the network behaves as desired, weighting both the spatial and geometrical information it is provided to improve the classification accuracy. Because of this, the global bounding box is included as a spatial input when multi-view architectures are tested.

### **Extension to multi-view model**

As presented in Table 5, the best multi-view fusion technique is found to be the product-score fusion. This is evident when comparing the multi-view results presented in Table 6. Looking at these results, it can be observed that the score fusion techniques, namely Multi-view #2 and Multi-view #3, drastically outperform the late fully-connected fusion. In fact, even the single-view reference model, which is the spatial model in Figure 34, outperforms Multi-view #1.

The fact that Multi-view #1 is outperformed this drastically is unexpected. In [63], this was the

Actual	Predicted
	
	
	
	
	

**Figure 35** – Examples of errors made by the spatial model tested in Classification #2 - Stage 2.

fusion technique that was found to be the most efficient. However, when comparing this model’s training history, presented in Figure 36, to the one presented in Figure 32 for the initial training with the ID as the target, similarity can be observed. The core issue with the model in Figure 32 was found to be an insufficient training set, and a drastic increase in performance is observed when increasing the dataset size, as presented in Figure 33.

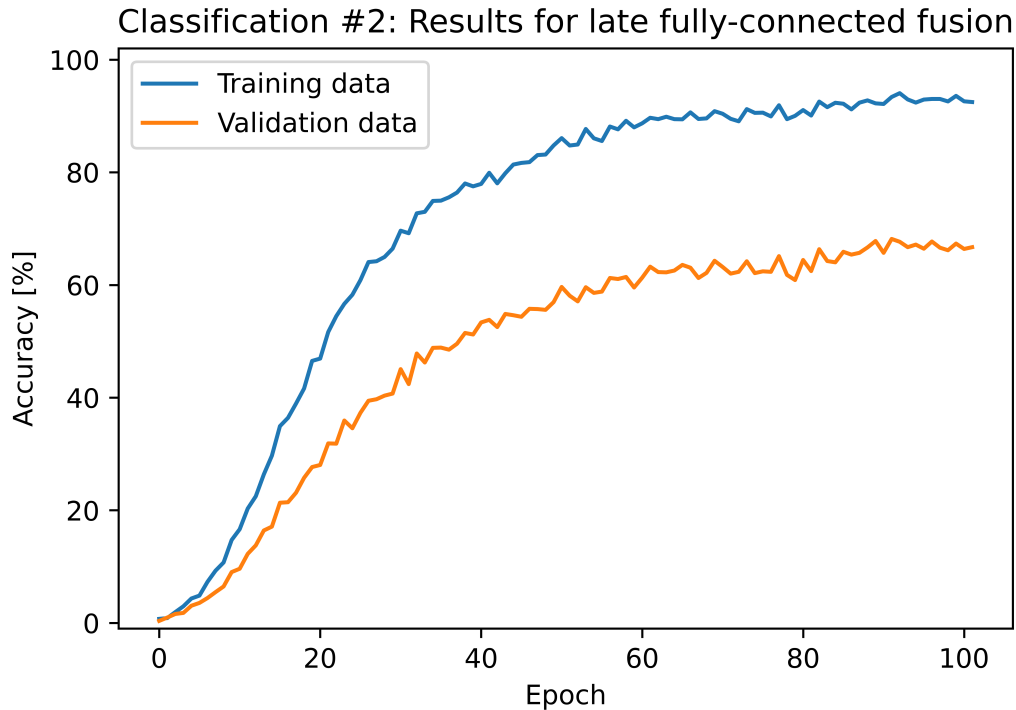
Version	Fusion technique	Validation accuracy
Single-view (reference)	None	79.51%
Multi-view #1	Late fully-connected	68.17%
Multi-view #2	Product-score	91.79%
Multi-view #3	Democratic-score	89.97%

**Table 6** – Comparison of the results for different multi-view fusion techniques in Classification #2.

The graphs in Figures 32 and 36 have striking similarities, with both of the validation accuracy curves displaying a much less steep initial increase when compared to other training history graphs presented in this section. As such, an explanation as to why the Multi-view #1 model performs so poorly can perhaps be that the training set is insufficient to train the model. This theory is supported when considering the fact that this multi-view model effectively receives 7 times fewer training examples per class when compared to the single-view and score fusion models. Although the training set used for all models is identical, the single-view model evaluates single images, whilst the late fusion multi-view model evaluates 7 images at a time. As a result, even though the examples per part might be better for this multi-view model since it receives multiple images from different views, there are 7 times fewer examples. Since the score fusion models simply duplicate a single-view model and no further training is performed with the multi-view architecture, training of these models is equivalent to training the single-view model.

A question hence arises as to whether or not further augmentation should be performed to investigate the effects of this on the late fully-connected multi-view model. However, training with the existing augmented dataset, which consists of over 100 000 images, is found to be expensive. In fact, the average training time over three runs when training the model in Figure 32 on the augmented dataset is almost 24 times higher than the training time when training on the base dataset. Generation and preparation of an additional augmented dataset also entails a substantial effort. Since the results for the score fusion multi-view models are considered to be strong, it is

therefore not prioritized in this work to pursue the creation of and training with a further increased dataset.



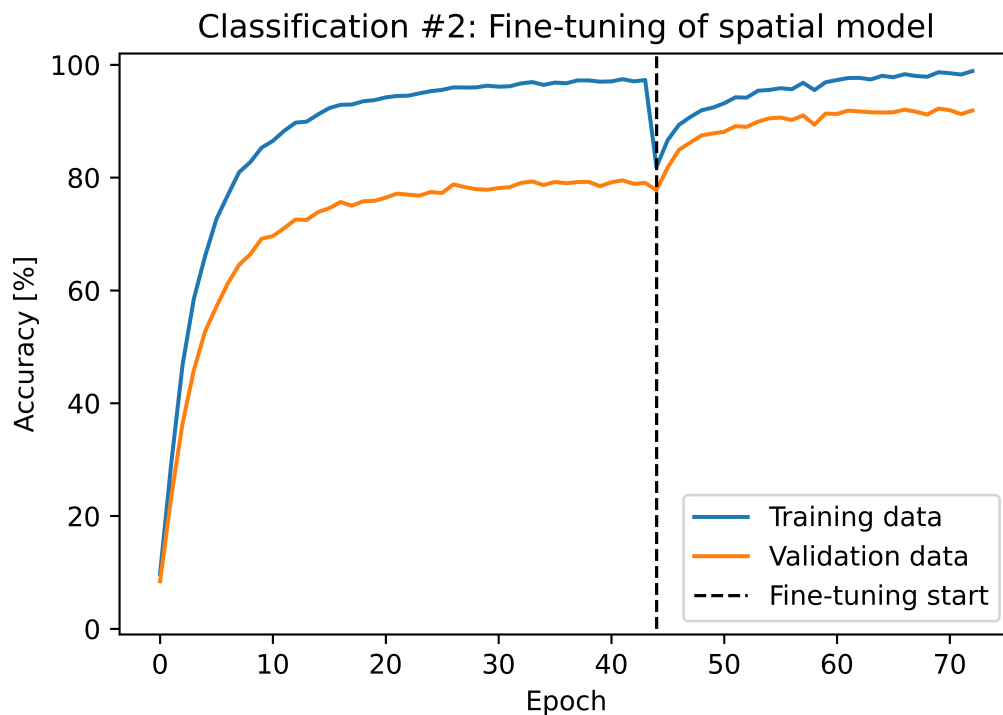
**Figure 36** – Results for the late fully-connected multi-view fusion technique in Classification #2 - Stage 3.

Table 5 also reveals that the product-score fusion model achieves superior results to the democratic-score model. This could potentially be related to the fact that the product-score model also translates the confidence of the model predictions for each view, and not only the actually predicted class. For example, if the model is 99% confident in one view that the fastener in question belongs to a specific class, and only 51% confident in another view that it belongs to another class, these two predictions are weighted the same by the democratic-score fusion model. Still, however, the democratic-score fusion model could be less sensitive to drastic errors in single views. That is, for example, if the 99% prediction turns out to be wrong. In this extreme case, that would perhaps suggest that the model itself is not particularly good, since it displays high confidence in an erroneous prediction. Still, in less extreme cases, there could exist nuances. It is therefore considered interesting to also investigate the democratic-score fusion performance.

## Fine-tuning

The results for the score fusion models, in particular the product-score fusion model, presented in Table 6, are considered to go a long way as a proof-of-concept for the current approach to automatic fastener identification. As a result, little further tuning and development is prioritized for this module. However, all previous development has been performed without training the weights in the convolutional block. It is therefore considered to be of interest to perform a fine-tuning step where the training of the best single-view model, namely the spatial model in Figure 34, is continued, also allowing tuning of the weights in the convolutional block.

The results from the fine-tuning of the spatial single-view model are presented in Figure 37. The initial training, namely the training history presented in Figure 34, makes up the first part of this graph, up to the dashed vertical line. After this line, the training history when continuing training and also allowing the convolutional block weights to be tuned is plotted. It can be observed in the graph that the accuracies after the first fine-tuning epoch are lower than the final accuracies of the initial training stage. This is especially true for the training data, which experiences a drop of almost 10%. Such a drop is a phenomenon that can sometimes occur when initiating fine-tuning, and it is likely related primarily to reduced overfitting due to the increased variation in the dataset.



**Figure 37** – Fine-tuning of the spatial single-view model in Classification #2 - Stage 4.

Fine-tuning is observed to drastically increase the validation accuracy. From around 80%, the validation accuracy increases by more than 12%, reaching a final maximum of 92.23%. Such a drastic increase in results is somewhat surprising, since this continued training often is expected to improve the performance by merely a couple of percent. This could suggest that the ImageNet might not be the most appropriate pre-training dataset for this problem. That is, there might not be a high degree of similarity between the patterns found in the ImageNet data and the patterns found in the fastener data. Such a theory is supported by the results for fine-tuning of the standard model in Figure 37, which also experiences a drastic increase in validation accuracy, reaching a maximum of 90.01%.

The ImageNet dataset contains millions of real images of, amongst other things, animals and everyday objects. The patterns present in these images are many and complex. As such, it is possible that the patterns found by a network in this dataset are too complex for the simpler computer images of fasteners. Considering the geometries of the fasteners (Figures 28 and 35), they are observed to generally contain rather simple shapes and patterns. An interesting observation that could further support the theory that the ImageNet might require identification of more complex patterns as compared to the fasteners, is the change in the interval on which the convolutional output exists. From  $[0, 373]$  in the original models, the output of the convolutional block changes to exist on the interval  $[0, 34]$  in both fine-tuned models. Such a change in the convolutional output could suggest a significant change in the identified patterns.

Even though drastic improvements are experienced upon fine-tuning of both the standard and spatial models, the results still validate the use of spatial information. The gap between the fine-tuned models is smaller than the original, but a difference of 2.22% is still considered significant.

Converting this fine-tuned spatial model into a product-score multi-view model results in the model presented in Table 5 as the final network developed for the Classification #2 module.

#### **6.1.4 Summary and future directions**

As presented in Table 5, the final model achieves an accuracy of 96.90% on the unseen test set when identifying fasteners among 820 different possibilities. This result is considered to be strong and to function as a proof-of-concept for the presented approach for automatic fastener identification.

Further improvement of the performance is still, however, considered to be possible before

implementing the module into a finished system. For one thing, more and finer tuning to discover optimum hyperparameters could result in increased accuracy. More augmentation could also be performed to increase the size of the dataset and potentially also the accuracy. Especially the late fully-connected multi-view model could benefit from such a dataset increase. Additionally, normalization of the spatial information to the interval of the convolutional output, or normalization of both the spatial information and the convolutional output to the same suitable interval, could be tested to enhance the collaboration between the spatial and geometrical information. It can also be tested to increase the number of views in the multi-view representation. This could perhaps improve the performance, as well as potentially make the model more rigid and independent of the part orientation.

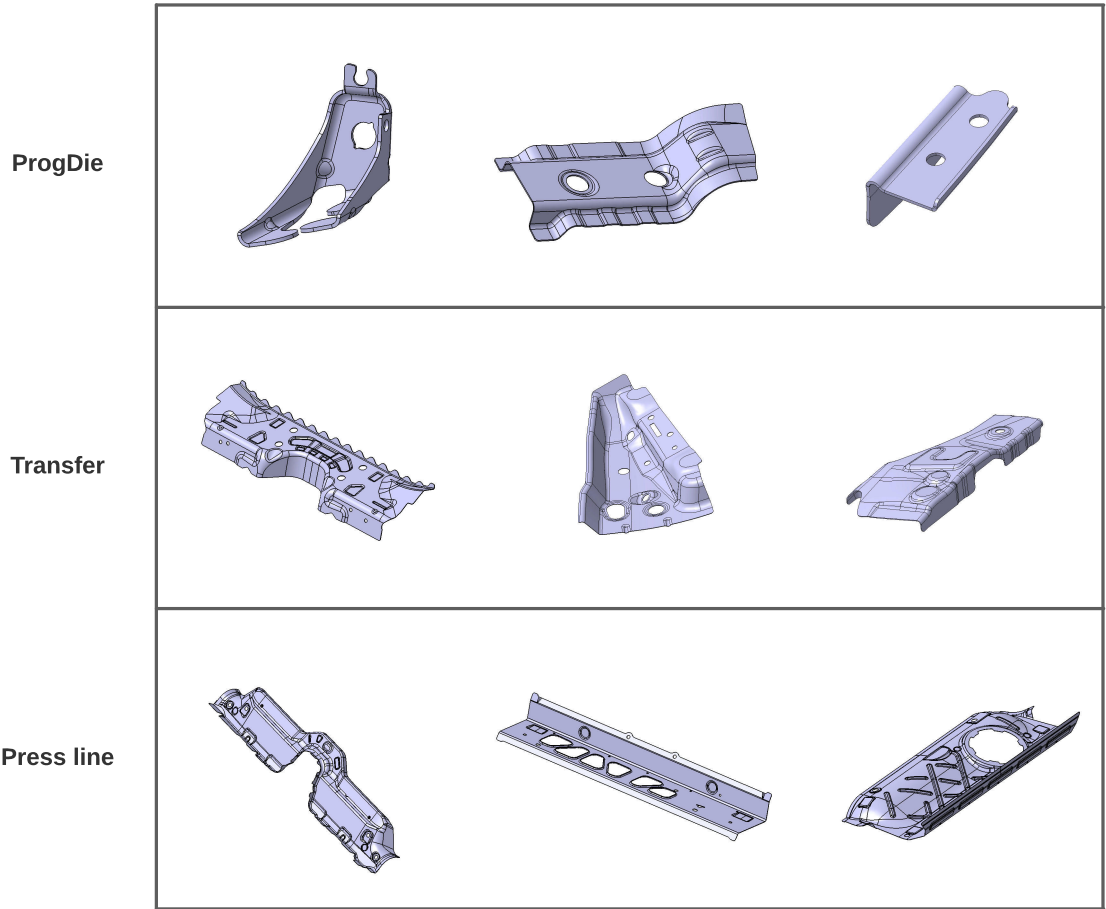
Further development of the employed method itself could also be possible. An interesting direction of future work on this module would, for example, be to experiment with employing the current spatial multi-view model in a Siamese network architecture [103]. Such networks are frequently used in tasks related to comparing an unknown sample with existing data. These applications include signature verification [103], facial recognition [104], and fingerprint recognition [105]. Since the fasteners have constant geometry, meaning a fastener adhering to a certain ID has identical geometry in every real case, the problem of fastener identification has many similarities to such applications. The presented method is assumed possible to employ in a Siamese manner, and it could hence be investigated if this could further increase the accuracy of the module.

## **6.2 Classification #3 - Production method**

In Classification #3, the aim is to evaluate a sheet metal part to determine its production method. There are three different production methods distinguished in this work, and the problem in Classification #3 can hence be considered a categorical problem with three target classes.

### **6.2.1 Data**

Figure 38 presents 3 examples of sheet metal parts manufactured with each production method. From these images, it can be observed that the parts produced in Transfer presses and Press lines have higher similarity to each other than to the parts produced in ProgDie presses. The Transfer and Press line parts typically have more complex geometries, as can be observed, for example, by considering the first example part for each production method. In fact, as previously



**Figure 38** – Examples of sheet metal parts manufactured with the different production methods distinguished in this work.



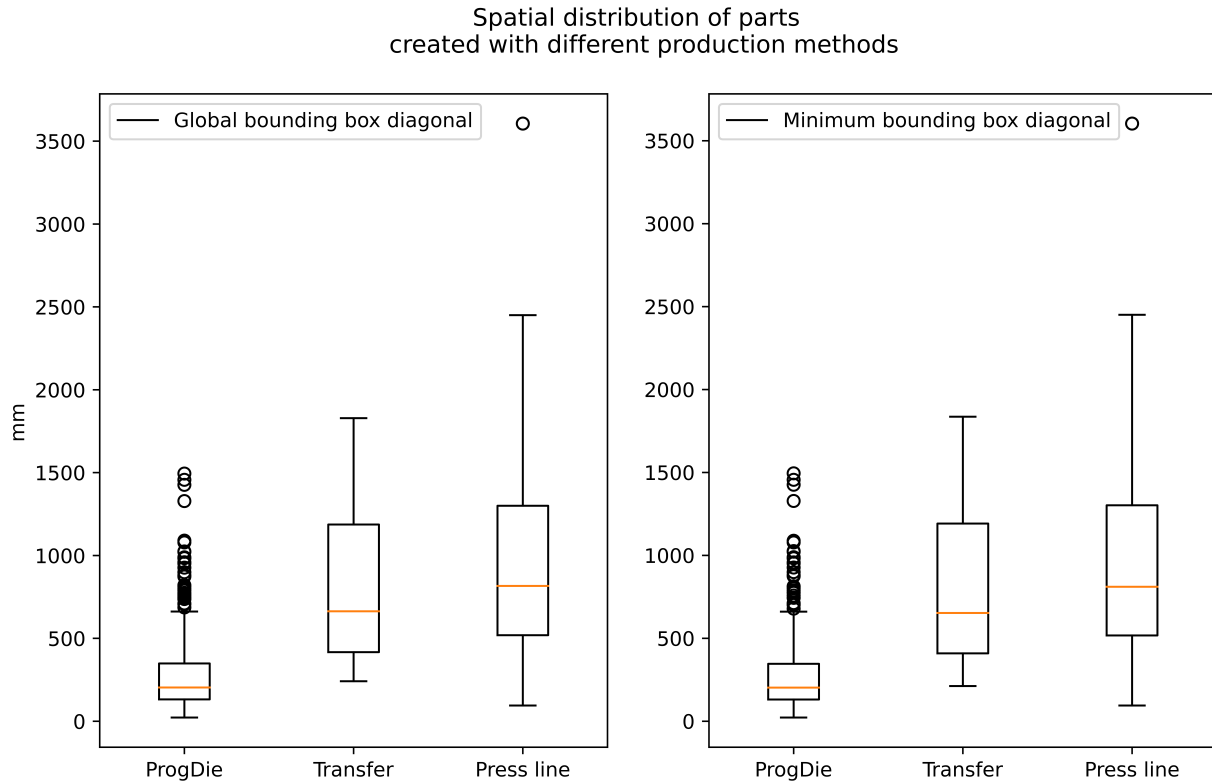
mentioned, do several parts exist that can be manufactured with either of these production methods. That is, several parts that theoretically could be manufactured in a Transfer press are in practice manufactured in Press lines. Some overlap also exists between ProgDie and the two others, although this overlap is significantly smaller. When a part can be manufactured with multiple production methods, the specific choice of production method is related to logistical considerations based on which presses that are available at the relevant place of production at the time of production. Such logistical data is not available in this work. There are, however, many parts that can only be produced in Press lines due to their size and shape, and, because it is usually cheaper, it is, for example, generally preferable to produce parts that can be produced with Transfer presses, but not in ProgDie presses, in Transfer presses, if such a press is available. As a result, if the training data consist primarily of *best-practice* examples, where parts that can only be produced in Press lines are labeled "Press line parts" and parts that can be produced in Transfer presses are labeled "Transfer parts", a distinction between the two could be possible with the intended network approach. At the start of the development of the Classification #3 module, it is, however, unclear if this is the case with the data available in this work. Nevertheless, a categorical problem with three target classes is pursued.

An observation that can be made upon comparing the sheet metal parts in Figure 38 and the fasteners in Figure 28 is that the sheet metal parts have much more complex geometries. Whereas the fasteners often have distinct edges and curves, the sheet metal parts are freeform surface components with rounded edges and multiple different machining features, such as holes, flanges, and beads. As a result, even though the ImageNet dataset was found to perhaps be somewhat too complex for the fasteners, pre-training with this dataset is continued in the development of Classification #3.

As discussed in Section 4.2.8, is the part size assumed to be an important factor to determine the appropriate production method. This assumption is supported by the spatial distributions of the sheet metal parts, presented in Figure 39. In this figure, boxplots of both the global and minimum bounding box diagonals for parts manufactured with the different production methods are presented. The minimum bounding box diagonals are included since the minimum bounding box can be considered a better spatial descriptor on its own because of its invariance to rotation. In the boxplots, however, it can be observed high similarity between the distributions of the global and minimum bounding box diagonals, which could suggest that most CAD part models in the

dataset are created and exist in the global coordinate system.

Considering both the medians and maximums of the bounding box diagonals, it can be observed that the ProgDie parts generally are the smallest, followed by the Transfer parts and finally the Press line parts. Much overlap is, however, observed between the sizes, especially for the Transfer and Press line parts. This overlap could potentially suggest that the dataset does not contain primarily best-practice examples, as discussed above.

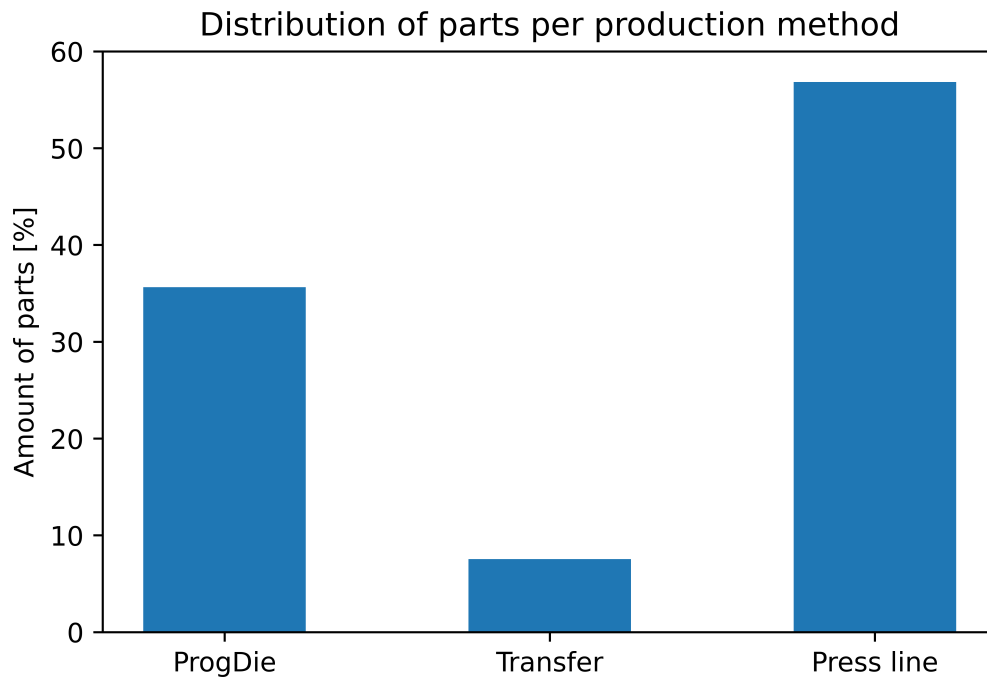


**Figure 39** – Spatial distribution of parts manufactured with the different production methods.

Augmentation proved to cause a significant improvement in the results for the fasteners (Figure 33). However, the special case that was experienced for the fasteners, namely that any part rotation could be considered a completely new data point, is not the case for the production method data. The reason for this is that the natural variation in the current dataset relates to more factors than only rotation. That is, whilst rotation of one of the parts in Figure 38 creates a new training example and increases the variation in the dataset, it is not equal to introducing, for example, a completely new ProgDie part into the dataset. Nevertheless, since augmentation does increase the variation in the dataset and often is found to improve a network’s ability to generalize the data,

an augmented dataset is generated and made available also for Classification #3. This augmented dataset contains 8 random rotations of each of the parts in the original dataset.

Considering the data for the production method classification problem, there is one particularly notable difference to that of the fasteners. This difference is that the distribution of available data on each production method suffers from imbalance. Figure 40 shows the number of parts that are available for each production method in percent of the total dataset size. Whereas one single part was available per fastener, making their distribution uniform, this is evidently not the case for the production method. Instead, it can be observed that there are much fewer Transfer parts than ProgDie parts, and even more Press line parts. In fact, there are over 7 times more Press line parts than Transfer parts.



**Figure 40** – Number of parts in the dataset for each production method in percent.

The imbalance observed in the dataset persists in the training, validation, and test sets, which are split randomly to make up about 70%, 15%, and 15% of the data, respectively. The distribution within the sets is presented for the base dataset in Figure 41. It can be observed that the distribution of image examples in each set is similar to that of the parts, also after the random split. The split is made per part, which ensures that all 7 view images of a certain part are contained within the same set. This split is maintained in the augmented dataset, where the random rotations of each

part are included in the set in which the part exists for the base dataset.

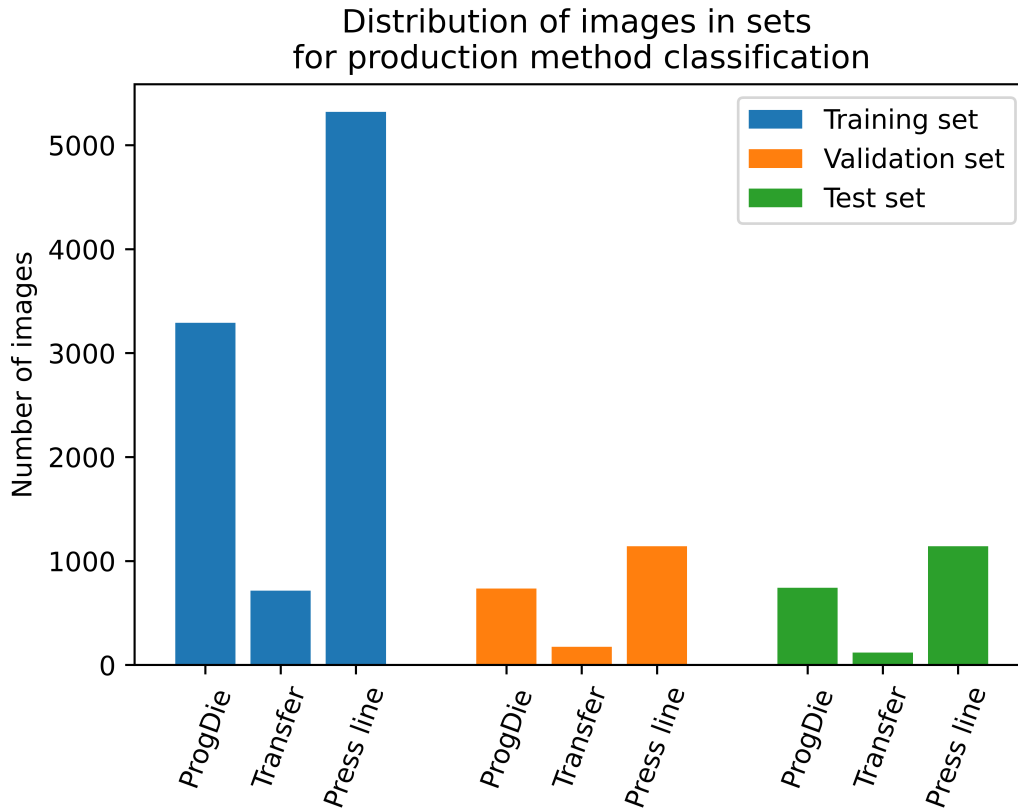


Figure 41 – Number of image examples for each production method in the base dataset.

### 6.2.2 Testing approach

An imbalance like the one experienced in the production method data raises the question as to how this affects the network training. Since the imbalance will cause the network to "see" many more examples of some classes, it could end up preferring these majority classes at the expense of the minority classes. Here, that could mean, for example, that the network is more likely to predict that a part is a Press line part than a Transfer part, because it "thinks" that Press line parts are much more common. This behavior is undesired, and Stage 1 in the development of Classification #3 will therefore additionally experiment with a technique to combat the class imbalance. The technique that is tested is to weigh the losses during training. This is a common technique to tackle class imbalance, where the losses on each class are assigned a weighting factor based on the number of examples of that class relative to the others [106]. This means that the network is more sensitive to errors made on the minority class, which, in turn, results in the network making larger

adjustments towards accurately predicting the minority class whenever it is provided a training example of this class, as compared to the adjustment it makes when provided with a training example of the majority class. In this work, the weighting factors for each class are calculated as:

$$W_i = \frac{1}{N_i} \times \frac{N}{N_c} \quad (31)$$

where  $N_i$  is the number of training examples of class  $i$ ,  $N$  is the total number of training examples, and  $N_c$  is the number of classes. The weights are implemented using the *class\_weights* argument in Tensorflow Keras during training.

The additional testing of the imbalance technique in Stage 1 means that all architectures are tested both with and without class weights for all learning rates. This is the only change in the testing approach compared to the general one. The testing approach employed in the development of the Classification #3 module is then:

1. Selection of default convolutional architecture
  - (a) Rough tuning of learning rate and test of imbalance technique
  - (b) Dropout test
2. Validation of spatial model
3. Validation of multi-view model
4. Final tuning

The class imbalance in the production method data also has an impact on how the results should be analyzed, and hence on the choice of the key performance indicator. The reason for this is that merely considering the regular classification accuracy (Equation 29) on imbalanced data could result in a significant loss of information. If, for example, simply guessing that all parts are Press line parts, this would result in a regular classification accuracy of almost 57%. This is opposed to the accuracy of about 33% that could be natural to expect in a 3-class problem. As a result, the weighted accuracy is used as the key performance indicator in the development of this module. The weighted accuracy is calculated as the mean of the individual accuracies on each class:

$$A_w \% = \frac{100\%}{N_c} \times \sum_{i=1}^{N_c} \frac{N_{i,correct}}{N_{i,possible}} \quad (32)$$

where  $N_c$  is the number of classes,  $N_{i,correct}$  is the number of correct predictions on class  $i$ , and  $N_{i,possible}$  is the number of possible correct predictions on class  $i$ , or, in other words, the total number of evaluated examples of class  $i$ .

The weighted accuracy can, however, also sometimes be misleading. As a result, much attention is additionally put on the individual accuracies of each class, in particular, the minimum individual accuracy, when analyzing the results during the development of Classification #3.

### 6.2.3 Results and discussion

Table 7 presents the characteristics of the final network developed for the Classification #3 module. The final model achieves a weighted validation accuracy of 90.78% when predicting the production part type. This accuracy is supported by the accuracy of 91.45% on the unseen test set.

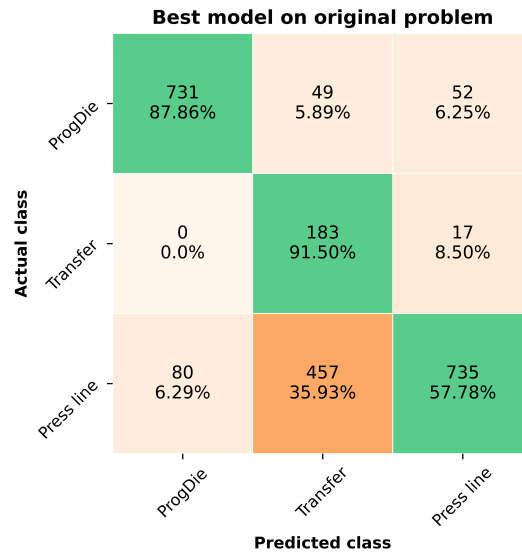
Classification #3	
Target	Production part type
Weighted validation accuracy	90.78%
Weighted test accuracy	91.45%
Convolutional architecture	VGG-16
Learning rate	1e-5
Fine-tuning learning rate	1e-6
Imbalance technique	Weighted losses
Dropout	50% on both FC4096
Spatial input	Global bounding box
Multi-view fusion technique	Product-score

**Table 7** – The final configuration of the network in Classification #3.

The first and most important thing that stands out in Table 7 is that the target of the model is the production part type, not the production method. The change of target follows a problem redefinition that was made due to the observation of a strong interdependency between the Transfer and Press line parts.

### Problem redefinition

The strong interdependency in the original problem can be observed in Figure 42. This figure contains a plot of the validation data confusion matrix of the best model developed on the original problem. This model is a multi-view model with similar parameters to those presented in Table 7. A confusion matrix is a way of presenting results in classification problems. The confusion matrix is particularly useful in cases where there is a class imbalance, because it allows investigation of the individual accuracies on each class. Each row in a confusion matrix represents the predictions obtained on parts from each specific class. Looking at the first row, for example, it can be observed that 731 parts, or 87.86% of the ProgDie parts in the validation set, are correctly classified as ProgDie parts. This is what is referred to as the individual accuracy, or precision, of this class, and it is colored green in all confusion matrices in this work. It can also be observed in the figure that 49 parts, or 5.89% of the ProgDie parts in the validation set, are incorrectly classified as Transfer parts, whereas 52 parts, or 6.25% of the ProgDie parts in the validation set, are incorrectly classified as Press line parts. The sum of all parts in the first row is then equal to the total number of ProgDie examples in the validation set. Note that the sum will vary based on whether the model is a single-view or a multi-view model, since the number of examples is 7 times lower in multi-view.



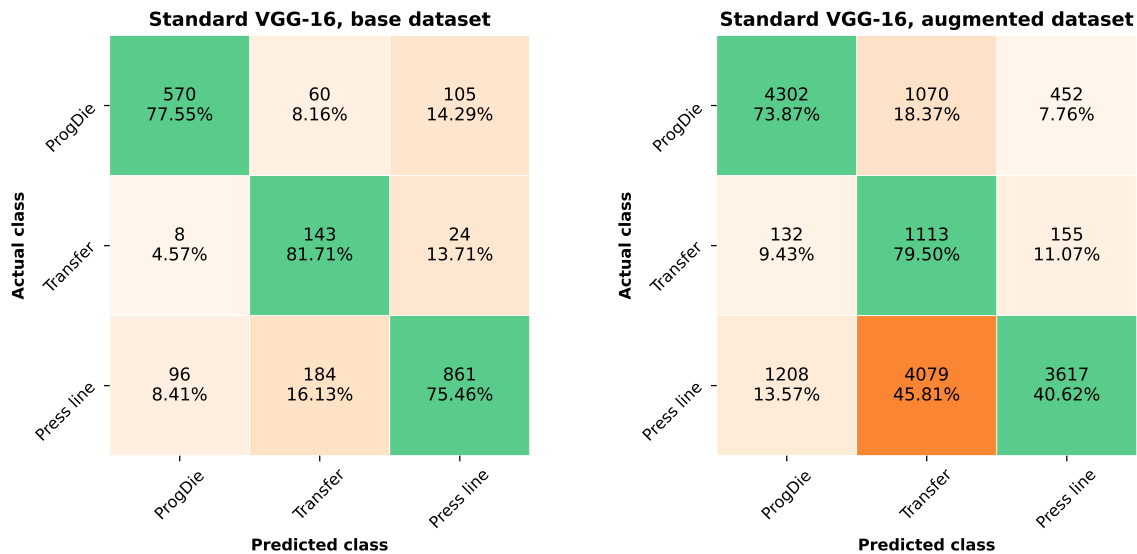
**Figure 42** – Validation results for the best model on the original problem in Classification #3.

Looking at Figure 42, the strong interdependency between the Transfer and Press line classes is evident. In the orange field on the bottom row, it can be observed that almost 36% of all Press line parts are incorrectly predicted as Transfer parts. The individual accuracy on the Press line class

is below 60%, with an accuracy above 90% on the Transfer class. These results suggest that the model is experiencing significant problems in separating the two classes.

The interdependency observed in Figure 42 is, however, somewhat expected. As previously discussed in Section 6.2.1, several parts exist that can be manufactured using either Transfer presses or Press lines. Many of the parts that are labeled as "Press line parts" could, therefore, potentially be possible to manufacture also with Transfer presses. Using best-practice labeling, they should then rather be labeled as "Transfer parts". The degree to which this overlap existed in the current dataset was not clear, and initial results suggested that the interdependency was not so strong.

The initial validation results from the development of the Classification #3 module are presented to the left in Figure 43. This VGG-16 model, which was the strongest after Stage 1, is trained on the base dataset. To the right are the results when this model is trained on the augmented dataset described in Section 6.2.1. The strong interdependency between the Transfer and Press line parts is, as can be observed, not present in the model trained on the base dataset. Instead, this model displays promising results when considering the fact that it neither includes spatial information nor is a multi-view model. However, upon testing the inclusion of spatial information and extension to multi-view, as well as fine-tuning the model in all configurations, the results are not improved. Because of this, the model is trained on the augmented dataset to investigate the effects of this.



**Figure 43** – Initial validation results on original problem in Classification #3. To the left are the results from training on the base dataset, to the right from training on the augmented dataset.



When training on the augmented dataset, the strong interdependency between the Transfer and Press line parts emerges. In fact, as can be observed in the right plot in Figure 43, the accuracy on Press line parts is reduced by almost 35%. These results are surprising, since augmentation generally is expected to improve the network performance through increasing its generalization abilities. However, they are assumed to reveal a significant issue with the base dataset, namely lack of variation. The results on the base dataset are, in other words, assumed to be tainted due to overfitting to non-general patterns that exist in the training set, which also happen to exist in the validation set. Especially because the total number of Transfer parts is low, such non-general patterns can be present. The non-general patterns could, for example, be that most Transfer parts, completely by chance, have a certain rotation or some particular lighting or shade in the images. In the augmented dataset, however, where all parts are randomly rotated, these patterns are no longer present. The classification must hence be based on actual general patterns that identify Transfer parts. As can be observed by the interdependency, the model is not able to find sufficiently strong general patterns to separate the Transfer and Press line parts. The theory that the difference in results between the base and augmented datasets is due to overfitting to non-general patterns is also supported by the fact that the weighted accuracy on the training set drops from 95.65% when training on the base dataset to 75.67% when training on the augmented one.

Following these results, the augmented dataset is employed in all subsequent development of Classification #3.

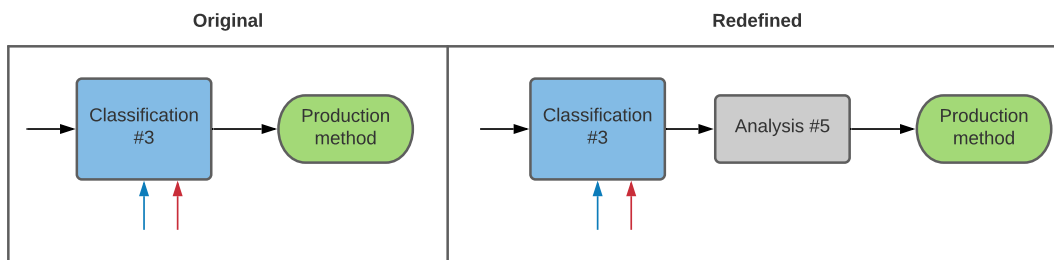
Because it is unsuccessful to remove the interdependency between Transfer and Press line parts through further development, as is evident by the results of the best developed model presented in Figure 42, the problem in Classification #3 is redefined. This redefinition entails combining the Transfer and Press line parts into a new class, called "Press parts". That is, the original 3-class problem is redefined as a 2-class problem, separating "ProgDie parts" and "Press parts".

Separation of the new Press parts class into Transfer and Press line parts, in order to solve the original problem, is assumed possible based on some heuristic argument. The most important factor when determining whether a part can be manufactured in a Transfer press or not, provided it can not be manufactured in a ProgDie press, is, as previously discussed in Section 4.2.8, the part size. As a result, this work will simply consider a heuristic threshold argument relating to the size of a Press part to identify Transfer parts. This heuristic classifies a part as a Transfer part if its

minimum bounding box diagonal is smaller than, or equal to, the largest minimum bounding box diagonal of any Transfer part in the available dataset.

From observation of Figure 39, which presents the spatial distributions of the parts in the current dataset, it follows that many of the parts that are labeled as Press line parts in the dataset will be identified as Transfer parts by the presented heuristic. This is a conscious choice influenced by an assumption that it is desired to perform the cost estimation based on the cheapest possible option, which in most cases is production with a Transfer press.

With the problem redefinition, the system architecture is hence modified according to the illustration in Figure 44. The Analysis #5 module performs the rule-based separation of Press parts into Transfer and Press line parts in order to finally determine the appropriate production method.



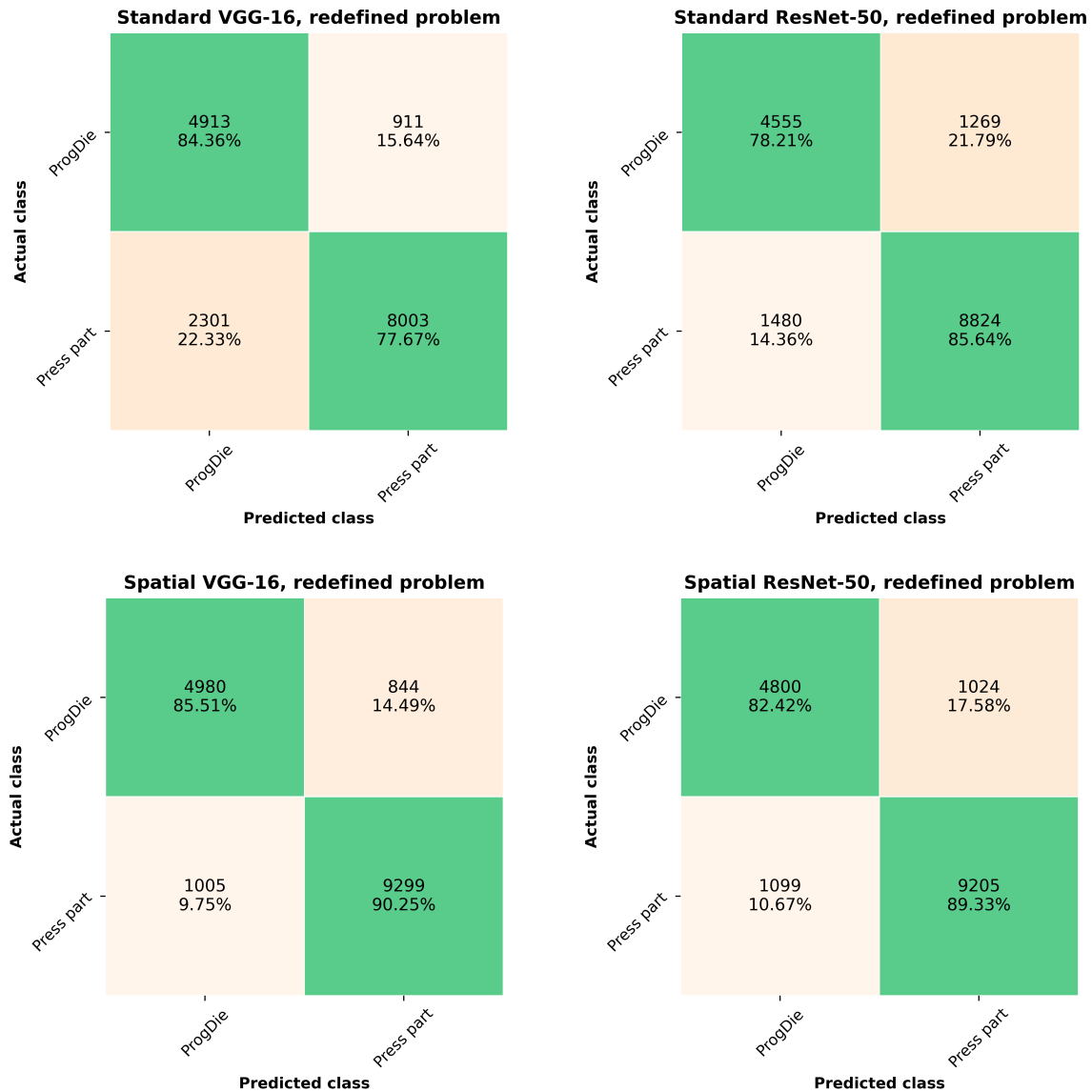
**Figure 44** – The change in system architecture following the problem redefinition in Classification #3.

After the problem redefinition, the development of the new Classification #3 module can begin. In order to reduce the development efforts, only the VGG-16 and ResNet-50, which are the architectures that performed the best in the initial Stage 1, are considered. The configurations that performed the best in the initial Stage 1 are inherited by the redefined problem, and subsequent testing is started with validation of a spatial model for both architectures (Stage 2). These configurations both employ weighted training and dropout, with the VGG-16 using a learning rate of  $1e-5$  and the ResNet-50 a learning rate of  $1e-4$ .

### Effects of spatial information

The initial validation results for the VGG-16 and ResNet-50 on the redefined problem are presented on the top row in Figure 45. On the bottom row are the results with the inclusion of spatial information. As for the fasteners, the spatial information used in these models is the regular global

bounding box.



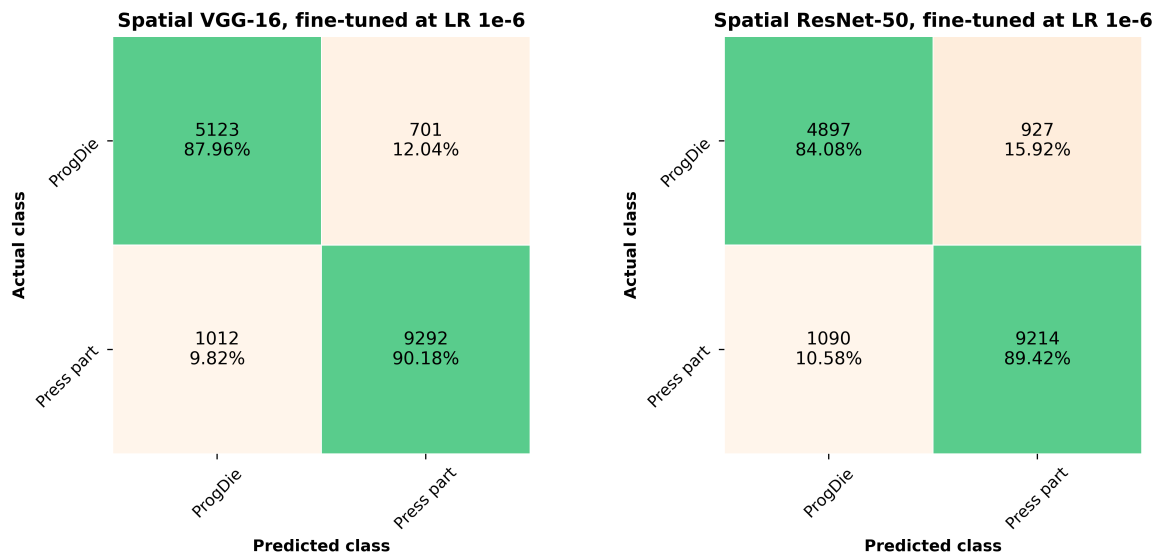
**Figure 45** – Initial validation results after problem redefinition in Classification #3. On the top row are the results for the standard VGG-16 and ResNet-50 configurations, on the bottom row are the results for the spatial models.

For both models, the inclusion of spatial information is observed to improve the results. In fact, both spatial models have increased accuracies on both classes. The most significant increase observed between the standard and spatial models, is the increase of more than 12% on the Press part accuracy for the VGG. Whereas the weighted accuracy of the ResNet-50 is slightly higher than that of the VGG-16 without the spatial information, this improvement puts the VGG ahead on

both classes.

From the results presented in Figure 45, the VGG appears to have stronger abilities than the ResNet to process additional spatial information. One explanation for this could perhaps relate to the different prediction blocks in the models. Whereas the ResNet only has a single fully-connected layer with softmax activation as its prediction block, the VGG has two additional fully-connected layers before its final output layer. These additional layers could enable better processing of the combined spatial and geometrical information.

Even though the VGG achieves superior performance and appears to better handle the spatial information, both spatial models are fine-tuned. The reason for this is to investigate if fine-tuning could enable the ResNet, which has fewer tunable weights when only training the prediction block, to better process the spatial data and surpass the VGG on performance. This is of special interest when considering the drastic improvement experienced by the fastener models upon fine-tuning.



**Figure 46** – The best validation results obtained upon fine-tuning of the spatial VGG and ResNet on the redefined problem in Classification #3.

### Fine-tuning

Figure 46 shows the best validation results obtained upon fine-tuning the spatial VGG and ResNet. Comparing these to the results in Figure 45, both models are observed to achieve higher performance after fine-tuning. The increase in accuracy is, however, much smaller than for the fasteners. This

supports the initial assumption that the ImageNet is a more suitable pre-training dataset for the sheet metal parts due to their high complexity relative to the fasteners.

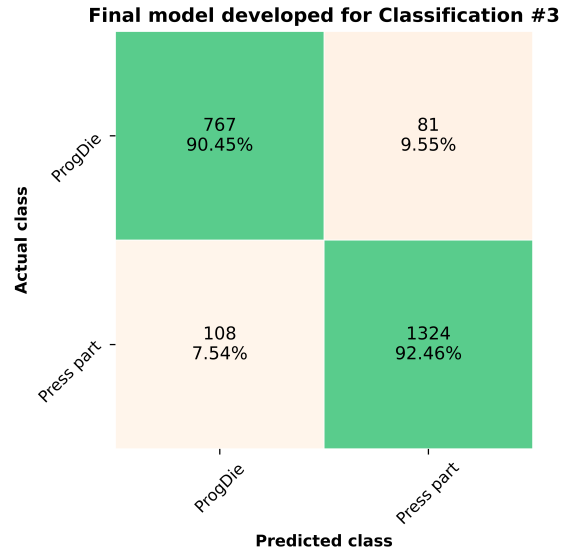
The changes in both models are observed to be similar, with a slight increase in accuracy on ProgDie parts and virtually identical results on Press parts. The VGG does, however, experience a slightly greater increase in weighted validation accuracy than the ResNet and is hence still the strongest of the two models after fine-tuning.

The VGG is also stronger when both architectures are extended to multi-view with the various fusion techniques. The results presented in Table 7 as the characteristics of the final network are, in fact, the fine-tuned VGG presented in Figure 46 extended to a product-score multi-view model.

#### **6.2.4 Summary and future directions**

Figure 47 presents the test data confusion matrix of the final network developed in Classification #3, as described in Table 7. The final network separates the two classes "ProgDie parts" and "Press parts", where the latter is assumed possible to separate with a heuristic argument in order to obtain the actual production method. In the confusion matrix, it can be observed that the final network achieves an accuracy of above 90% on each individual class on the unseen test data. This result is considered to be strong and to function as a proof-of-concept for the proposed approach for automatic identification of the production part type.

As for the fasteners, it is assumed that further improvement in performance possibly can be achieved through more and finer hyperparameter tuning, an increased dataset size, normalization of the spatial information, and by increasing the number of views. Future works could also focus on improving the solution to distinguish between Transfer and Press line parts. In this work, the distinction is simply made by a heuristic argument comparing the minimum bounding box diagonal to some threshold, as discussed above. Future works can develop more advanced rules to determine whether a Press part should be classified as a Transfer or Press line part. These rules could, for example, include logistical considerations relating to which presses that are available for production and more complex spatial evaluations. The rules can additionally be modified with regards to the relevant problem, such that the most accurate cost estimate can be achieved in the end. This work assumes that it is desired to label any part that can be manufactured in a Transfer press as a Transfer part, since this generally would lead to lower costs. Other works may have other preferences



**Figure 47** – The test results of the final model developed in Classification #3. The characteristics of this model are presented in Table 7

and can adjust their rules accordingly. To achieve a pure network solution to this module, as was originally intended, a database with best-practice labels could be created and tested.

### 6.3 Classification #1 - Part type

In Classification #1, the aim is to distinguish if a part is a fastener or a sheet metal part. This module is required in order to identify which subsequent analysis to perform on the part. The problem in Classification #1 is then a categorical problem with two target classes.

#### 6.3.1 Data

Figure 48 displays examples of the parts in each class in the Classification #1 module. These are the same parts that have previously been presented for Classifications #2 and #3, but the task is now merely to separate between the main part types "Fastener" and "Sheet metal part", rather than to perform some refined classification within one type. The problem in this module is, as such, assumed to be a simplified combination of those in the two previously developed modules. The reason for developing this module after the two that succeed it is an assumption that the development effort of this module can be reduced by inheriting experiences from the development of the two previous modules.

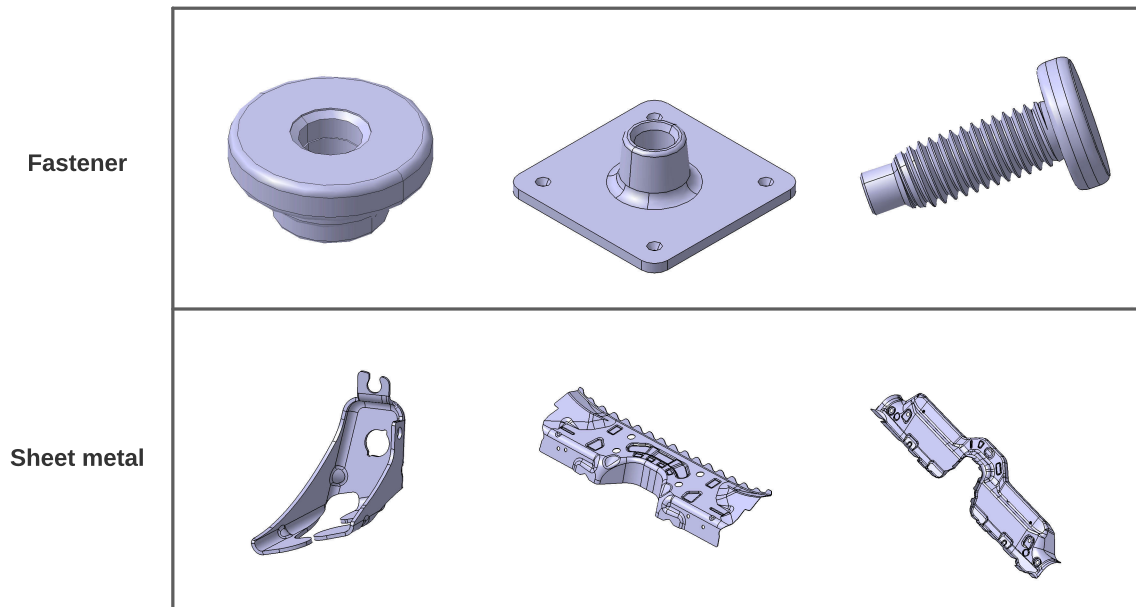


Figure 48 – Examples of each class in Classification #1.

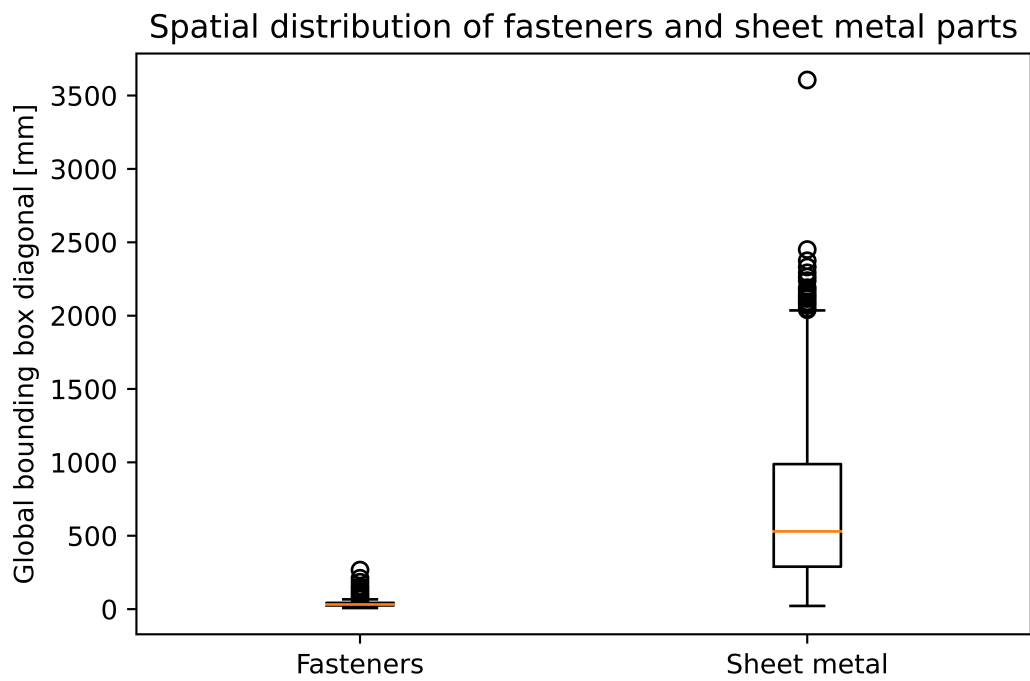
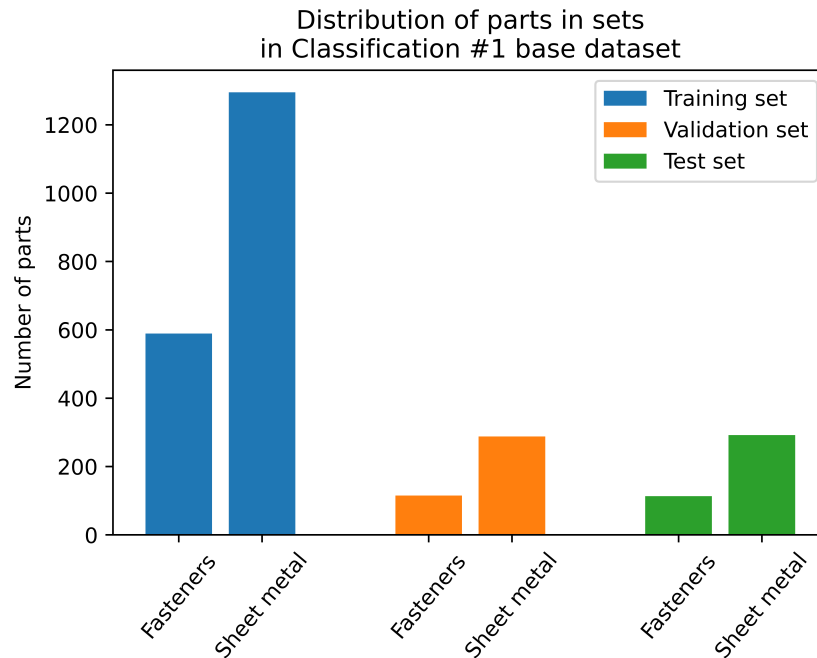


Figure 49 – Spatial distribution of fasteners and sheet metal parts.

In Figure 48, it can be observed that there generally is a clear geometrical distinction between the two classes. However, it is not only with regards to geometry that the classes differ. Considering also their spatial distributions, presented in Figure 49, it is evident that there generally also is a large difference in size between the classes. In this figure, it is observed that a high classification accuracy seems possible to achieve simply by considering the spatial dimensions of a part. However, there exist no general rules as to how small or big a fastener or sheet metal part can be. Whereas the Press parts class in Classification #3 is assumed possible to separate by spatial rules due to the different limitations and specifications of various presses, the size of a sheet metal part is, for example, based on the design requirements. As such, making a pure spatial distinction between the classes is assumed to result in a non-rigid classification.

The base dataset in this module is comprised of all unique parts that have been evaluated in Classifications #2 and #3. Because of the issues encountered with the base dataset in Classification #3, which are assumed to be related to lack of variation, all parts in the base dataset of this module are subjected to a random rotation prior to the generation of their multi-view image representation.



**Figure 50** – Number of parts for each class in the base dataset of Classification #1.

The number of parts of each class in each of the training, validation, and test sets is presented in Figure 50. The dataset is split randomly with 70% training data and 15% validation and test data.



The only logic involved in the split is that it is ensured that all multi-view images of a single part are contained within the same set.

It can be observed in Figure 50 that the base dataset is imbalanced, as was also the case in Classification #3. The imbalance between the majority (sheet metal) and minority (fastener) classes is, however, much smaller in this case than in the original problem in Classification #3. Whereas the ratio between the Press line and Transfer parts is over 7, the number of sheet metal parts is about twice that of the fasteners in the base dataset of Classification #1.

### 6.3.2 Testing approach

The testing approach employed in the development of Classification #1 is as follows:

1. Test of imbalance techniques with the VGG-16
  - (a) Rough tuning of learning rate
  - (b) Dropout test
2. Validation of multi-view model
3. Final tuning

This approach deviates from the general one presented in Section 5.3.2 in several points. First of all, only the VGG-16 is tested in the development of this module. The reason for this is that the VGG-16 proved efficient on the relevant data in both previous problems, and that this problem is considered a simplified combination of these.

Because of the imbalance in the dataset, Stage 1 in the development of Classification #1 will test imbalance techniques, in addition to the rough tuning of learning rate and dropout test. Whereas only weighting of the losses is tested in the development of Classification #3, Stage 1 will here additionally test random under-sampling of the majority class. This is another common technique for tackling the class imbalance problem, where data examples of the majority class are randomly removed in order to even out the part distribution in the dataset [106]. Under-sampling was not tested for the problem in Classification #3 because of the few available examples of the minority class.

In order to test the under-sampling imbalance technique, an additional dataset with uniform distribution is generated for this problem. In this dataset, sheet metal parts are randomly removed so that it finally contains the same amount of data examples of each class. The uniform dataset is randomly split in the same manner as the base dataset.

Each imbalance technique is tested with all learning rates, before the best results achieved with each technique are compared to determine which to employ in the subsequent testing.

An additional difference between the presented testing approach and the general one is that the best model from Stage 1 is directly extended to multi-view. That is, spatial information will not be considered in the current problem. The reason for this is the previously discussed large spatial difference between the classes (Figure 49), and that it is undesirable to make a classification that is strongly influenced by size. Although the error analysis and results from the development of Classification #2 indicate that the spatial model evaluates the spatial and geometrical information together to achieve increased performance, the spatial difference between the classes is much larger in this problem. Since the general geometrical difference between the classes is significant, it is assumed that a strong classification is possible without the inclusion of spatial information.

Because of the class imbalance in this problem, the weighted accuracy is used as the key performance indicator for this module. The individual accuracies are additionally, as in the development of Classification #3, monitored and analyzed.

### **6.3.3 Results and discussion**

Table 8 presents the characteristics of the final network developed for the classification #1 module. This model achieves an accuracy of 99.57% on the validation data when separating fasteners and sheet metal parts. The final configuration is validated by the accuracy of 100% on the test data.

As for the network in Classification #3, the weighted imbalance technique is observed to prevail as the strongest in Table 8. The different techniques are, however, difficult to separate.

#### **Imbalance techniques**

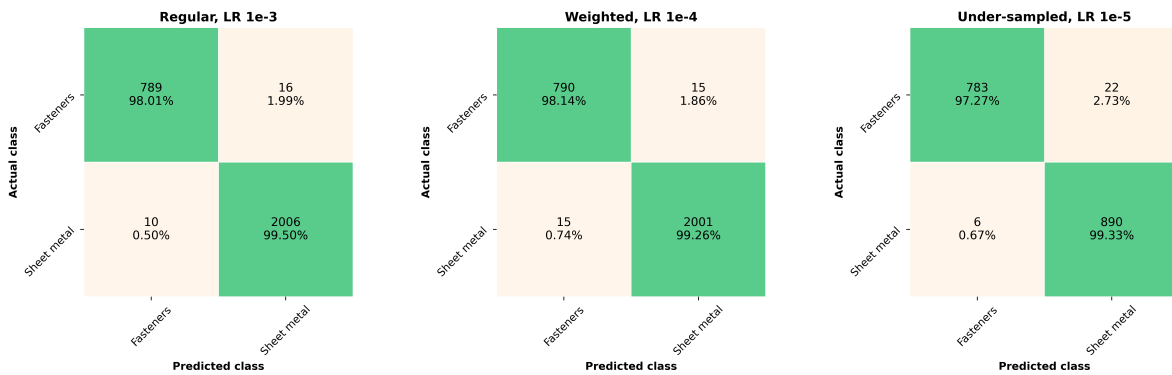
The strongest validation results for each imbalance technique are presented in Figure 51. To the left are the results for the regular model, trained on the base dataset without any imbalance technique, in the middle are the results for the model trained on the base dataset using weighted losses during

Classification #1	
Target	Part type
Weighted validation accuracy	99.57%
Weighted test accuracy	100%
Convolutional architecture	VGG-16
Learning rate	1e-4
Fine-tuning learning rate	1e-5
Imbalance technique	Weighted losses
Dropout	50% on both FC4096
Spatial input	None
Multi-view fusion technique	Product-score

**Table 8** – The final configuration of the network in Classification #1.

training, and to the right are the results for the model trained on the under-sampled dataset.

The first thing that is evident upon observation of the plots in Figure 51 is that the results for all models are strong. In fact, the lowest individual accuracy obtained by any model is the 97.27% accuracy achieved on the Fastener class by the model trained on the under-sampled dataset. These strong initial results support the assumption that the current problem is less complex than the previous ones.



**Figure 51** – Best validation results achieved with the different imbalance techniques in Classification #1 - Stage 1.

With such strong results as observed in Figure 51, making a decision as to which imbalance

technique to pursue is difficult. Although the under-sampled technique achieves the lowest weighted validation accuracy of the three with 98.30%, the gap of less than 0.5% to the accuracy of 98.76% achieved by the regular model is considered to be too small to carry any definite weight. In fact, one of the three independent runs with both the regular and weighted models achieves a lower weighted accuracy than the under-sampled one. This could suggest that the difference simply is the result of a non-preferable weight initialization. Nevertheless, there is another factor that plays in the disfavor of the under-sampling technique. Since the under-sampling relies on randomly removing data examples from the Sheet metal class, it follows that the model receives fewer training examples of this class. This could, in turn, result in reduced model rigidity and poorer generalization ability as compared to the models that are provided over twice the number of training examples of sheet metal parts. When the results additionally are neither stronger nor more balanced than those achieved for the other techniques, it is decided to not pursue under-sampling in the further development.

Since no strong arguments are found to separate the regular and weighted techniques, further testing is pursued with both.

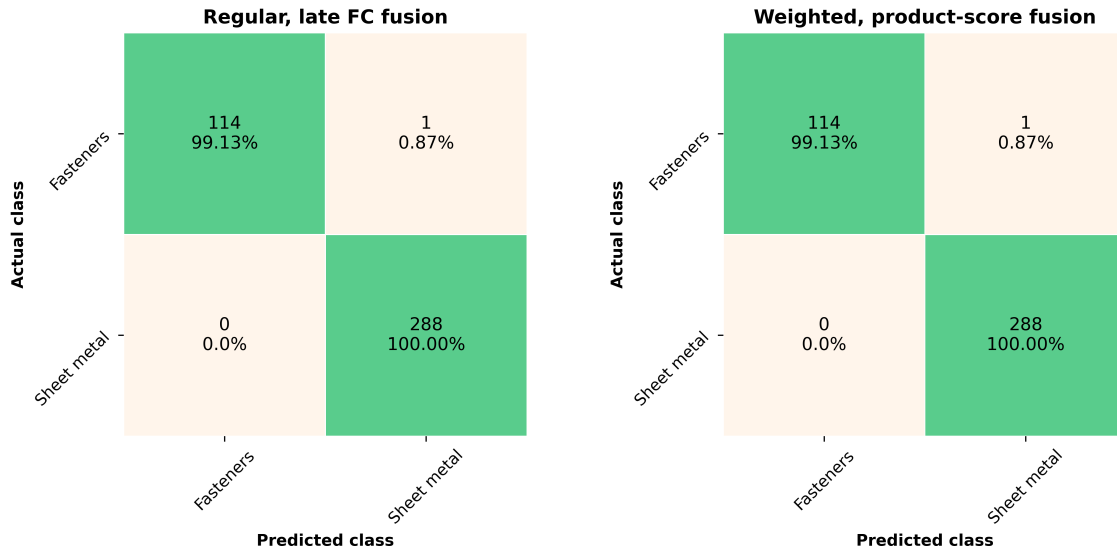
### **Extension to multi-view**

In Table 8, the weighted product-score multi-view fusion model is presented as the strongest model developed for Classification #1. The selection of this configuration, however, does not follow directly from the initial validation results obtained upon extension to multi-view in Stage 2. In fact, the results are so close between the regular and weighted techniques of the late fully-connected and score fusion multi-view models that fine-tuning is performed on all the relevant configurations in order to try and separate them.

Still after fine-tuning, the regular fully-connected fusion model and the weighted product-score fusion model are inseparable. Their validation confusion matrices are presented in Figure 52. The confusion matrices of the two models are observed to be identical, with both models making only a single error on the fasteners.

### **Error analysis**

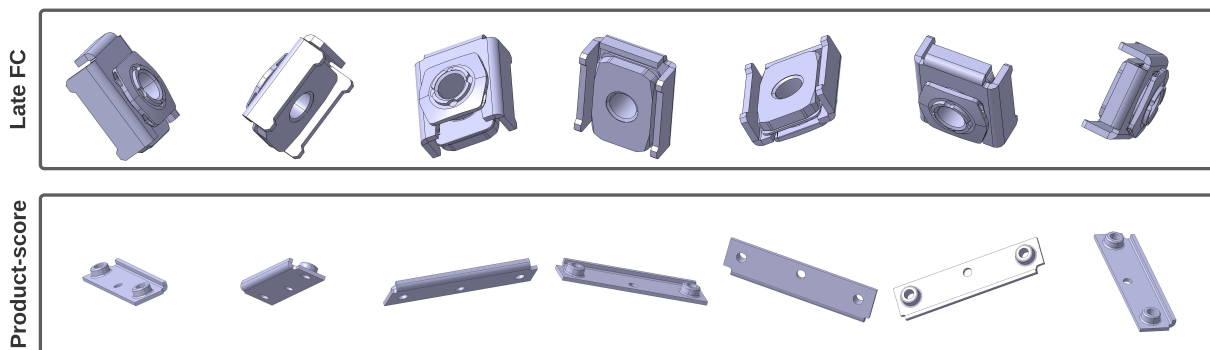
To attempt a separation between the two models presented in Figure 52, the error they make is evaluated. Figure 53 presents the multi-view representations of these errors, with the error made by the late fully-connected model on the top row and the error made by the product-score model



**Figure 52** – Validation results of the two strongest models developed in Classification #1.

on the bottom row.

Interestingly, the error made by each model is different. This suggests that the models rely on somewhat different patterns to classify the parts. Of the two errors, the one made by the product-score model perhaps appears to be the most reasonable. This fastener is thin and long and does, to some extent, resemble the ProgDie part presented to the right on the top row in Figure 38. However, also the fastener wrongly classified by the late fully-connected model is thin-walled and has some resemblance to sheet metal parts. It is, as such, difficult to make a final distinction between the models based solely on these errors.



**Figure 53** – The errors made by the regular fully-connected and weighted product-score fusion multi-view models in Classification #1.

In order to make the final separation between the two models in Figure 52, their performance on the test data is consulted. On this unseen data, the late fully-connected model also makes one error, incorrectly classifying a sheet metal part as a fastener, whilst the product-score model achieves a 100% accuracy. Following these results, the weighted product-score multi-view model is considered as the strongest configuration for Classification #1.

#### **6.3.4 Summary and future directions**

The final model that is developed for Classification #1 makes only a single error (Figure 52) on all evaluated parts in the training, validation, and test sets. This result is considered to be strong and to provide a proof-of-concept for the employed approach to classify whether a part is a fastener or a sheet metal part.

In order to remove also this one error, further hyperparameter tuning and experimentation with other architectures can be performed. If this proves unsuccessful, an analysis block could potentially be included to evaluate the parts that are predicted to be sheet metal. The fact that the model error is made on a fastener is considered to be a positive trait, since this fastener will have identical geometry in every case. As a result, a rule-based analysis of parts predicted to be sheet metal can aim solely to evaluate if the part is this one fastener or not.

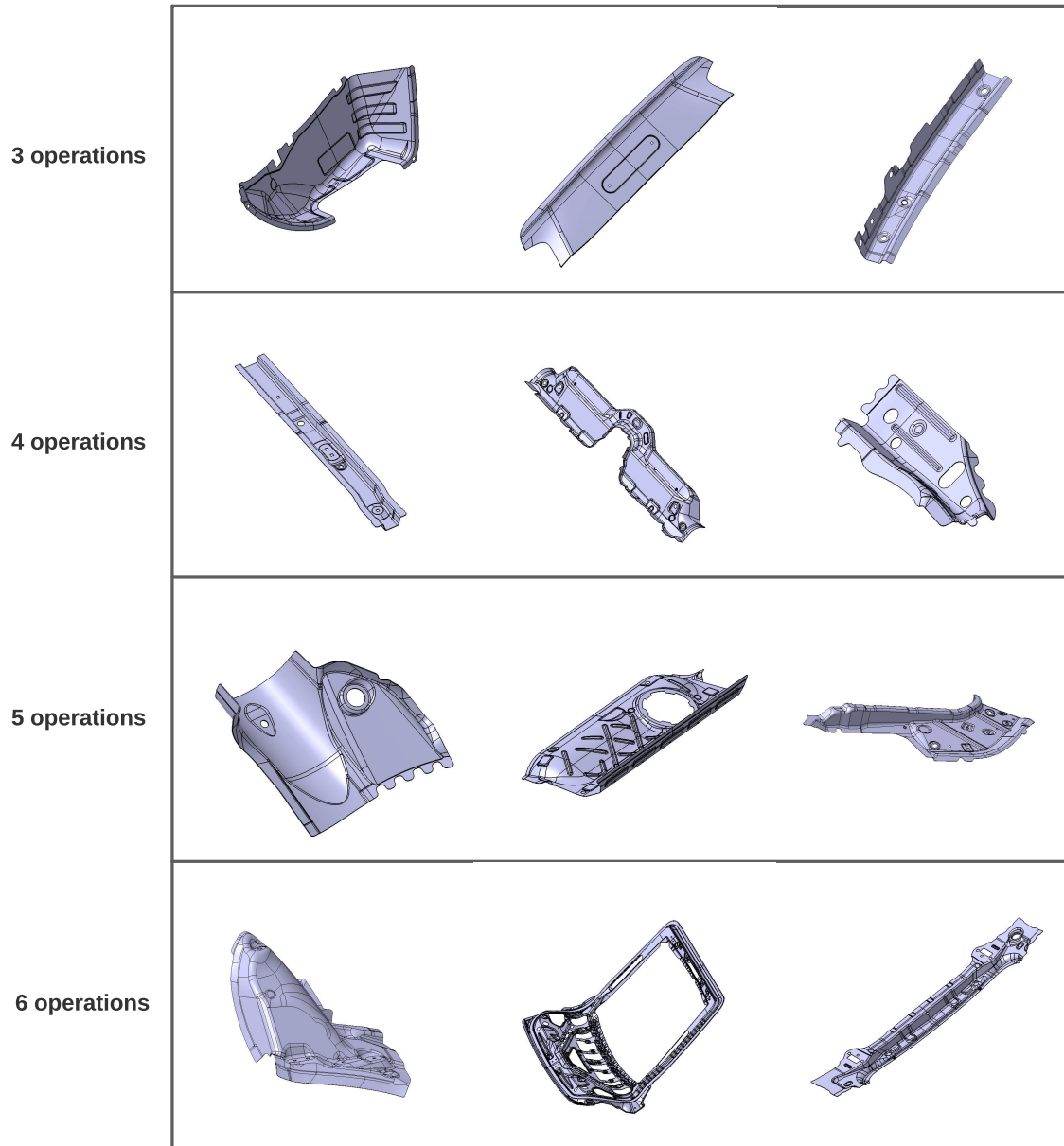
### **6.4 Classification #4 - Number of operations**

In Classification #4, the aim is to determine the number of operations required to transform the blank of a sheet metal part into the final geometry. The number of operations has a dependency on the production method, since different methods employ different operations during manufacturing. Classification #4 hence succeeds Classification #3.

#### **6.4.1 Data**

In this work, data is, unfortunately, only available on the number of operations for Press line parts. The development of this module will, as a result, only focus on Press line parts and assume that the final results are transferable as a proof-of-concept also for Transfer and ProgDie parts if data is available for training.

For Press lines, the number of operations is typically either 3, 4, 5, or 6. That is, the problem of identifying the number of operations required to manufacture a Press line part can be considered



**Figure 54** – Examples of Press line parts manufactured with the different numbers of operations.

a categorical problem with four target classes. Examples of parts manufactured with each number of operations are available in Figure 54.

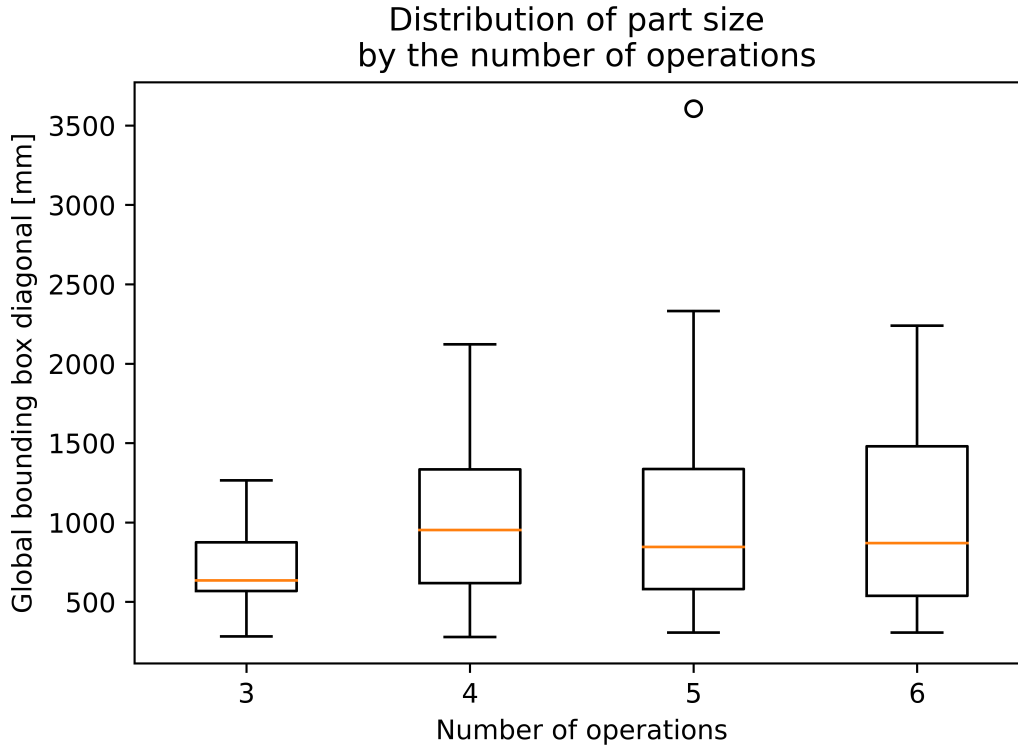
From the examples in Figure 54, it is evident that the difference between the classes in this problem is smaller than for the previous ones. At least after the problem redefinition in Classification #3, the classes have some apparent distinctions in the general case. In Figure 54, however, finding such general tendencies for each class is difficult. Parts in each class have several similarities, such as the rails to the right on the top row, left on the second row, and right on the bottom row. Nevertheless, some tendencies exist when considering the geometrical complexity of the parts on each row. In general, the further down, meaning the more operations that are required, the higher complexity. This can be observed, for example, in the center examples of each row. Whereas the top part is relatively plain with few features, the complexity can be observed to increase gradually in the second and third rows, until the final part that requires 6 operations, which has multiple features and complex shapes.

The number of operations is assumed to be less dependent on size than the previous problems. Since the production method is already determined, its dependency on size is more abstract and relates primarily to providing a complete part description. That is, size in itself is not assumed to provide any specific grounds for separating the classes, unlike, for example, separating two fasteners with almost identical geometry but different lengths. Spatial information is still, however, assumed to be of importance in the evaluation of a part. There are, for example, general rules as to whether or not a hole can be created before a bending operation, depending on its distance from the bent edge.

The assumption that there are no direct spatial distinctions between the classes is supported by the spatial distributions of each class presented in Figure 55. Although it can be observed that the parts manufactured with 3 operations seem to generally be smaller than the rest, it should be noted that there are very few data examples of this class (Figure 56). This could potentially cause an incomplete picture of the actual spatial distribution of the class. Apart from this observation relating to the parts manufactured with 3 operations, no clear tendencies are noted in the spatial distributions.

The dataset employed in this problem includes the same Press line parts as in Classification #3, provided data is available on the number of operations used to manufacture them. Because of the





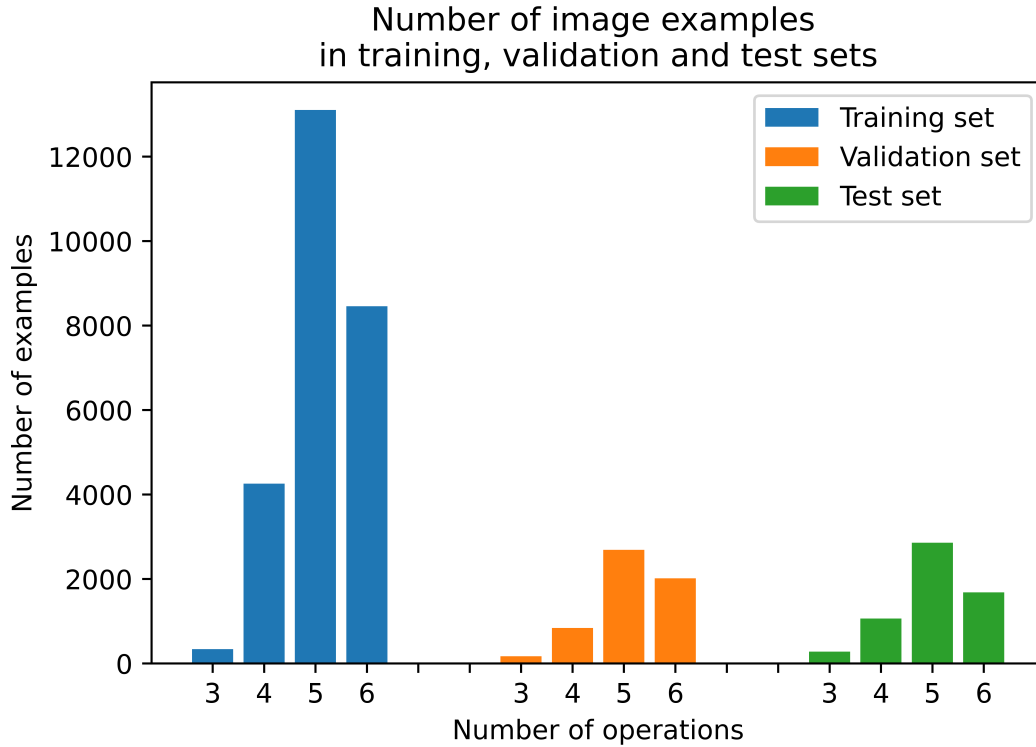
**Figure 55** – Spatial distribution of parts manufactured with different number of operations.

issues experienced with the base dataset in Classification #3 and because there are many fewer examples of each class in this problem, the development of this module employs the augmented dataset from Classification #3. That is, each unique CAD part is subjected to 8 random rotations, and the multi-view image representations of these rotated parts make up the training, validation, and test datasets. The sets are split in the same manner as in Classification #3.

The distribution of image examples in each set is presented in Figure 56. In this plot, it is clear that the problem in this module suffers from great imbalance. Very few examples exist of parts manufactured with 3 operations compared to the others.

#### 6.4.2 Testing approach

Because of the large imbalance between the classes in this problem, as observed in Figure 56, weighting of the losses will be performed during training. The reason for directly employing weighting rather than testing multiple techniques, as in other developments, is that this technique has proven efficient on previous imbalanced problems. The development efforts are hence reduced by assuming the same is true here.



**Figure 56** – The distribution of image examples in each set in Classification #4.

The testing approach employed in the development of Classification #4 is as follows:

1. Selection of default convolutional architecture, weighting losses during training
  - (a) Rough tuning of learning rate
  - (b) Dropout test
  - (c) Fine-tuning of best configurations
2. Validation of spatial model
3. Validation of multi-view model
4. Final tuning

Apart from employing weighted losses, the approach presented above differs from the general approach on one point, namely the 3rd step of Stage 1. Since fine-tuning previously has caused both significant and more moderate improvements in performance, it is considered to be of interest

to investigate if fine-tuning in Stage 1 results in a different conclusion as to which architecture to pursue in the development. The best configurations of each architecture after the learning rate tuning and dropout test in Stage 1 are therefore fine-tuned at four learning rates, namely the same learning rate as is used in the original training, 10 times smaller than this, 100 times smaller than this, and 1000 times smaller than this.

Like in previous problems that have a class imbalance, the weighted validation accuracy is used as the key performance indicator.

### 6.4.3 Results and discussion

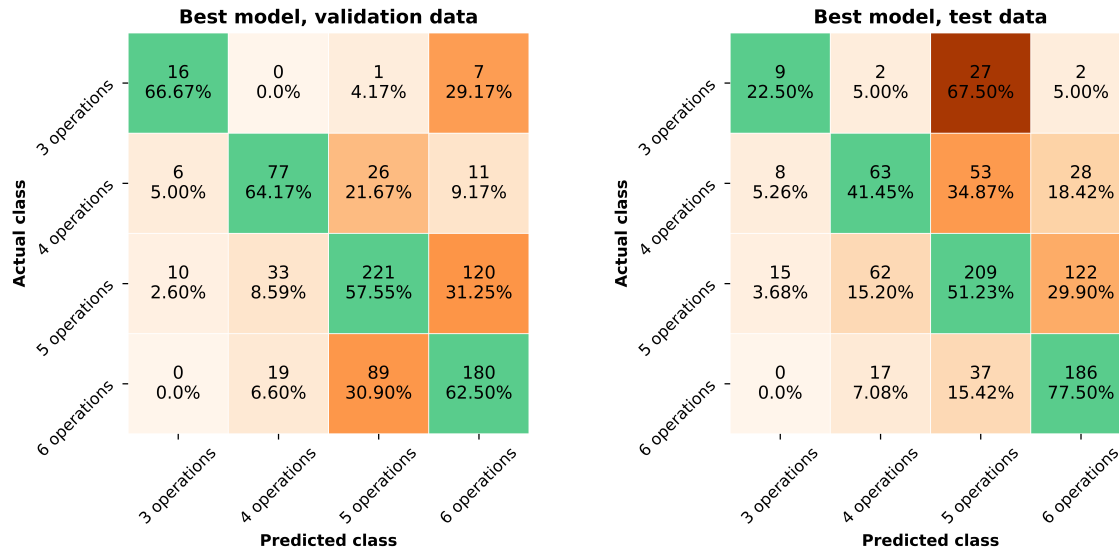
Table 9 presents the characteristics of the strongest model developed in Classification #4. From observation of the final achieved performance, it is immediately clear that it is unsuccessful to develop a sufficiently strong model to solve this problem with the employed approach on the current dataset. The final model achieves a weighted validation accuracy of 62.72%, which is not validated by the final accuracy of 48.16% on the unseen test set.

Classification #4	
Target	Number of operations
Weighted validation accuracy	62.72%
Weighted test accuracy	48.16%
Convolutional architecture	Inception-ResNet-v2
Learning rate	1e-3
Fine-tuning learning rate	1e-5
Imbalance technique	Weighted losses
Dropout	50% on both FC4096
Spatial input	None
Multi-view fusion technique	Product-score

**Table 9** – The final configuration of the network in Classification #4.

The poor performance is particularly evident upon investigation and comparison of the validation and test data confusion matrices of the final model. These are presented in Figure 57, with the validation data to the left and the test data to the right. The large deviations that can be observed between the individual accuracies on the same classes suggest that the model is not able to find

general patterns to recognize each class. This is especially clear when considering the accuracy on the "3 operations" class. Whereas a validation accuracy of 66.67% is achieved on this class, the test performance is a mere 22.50%. On the test data, as much as 67.50% of the parts in this class are predicted to require 5 operations.



**Figure 57** – Performance of the best model in Classification #4 on the validation data (left) and test data (right).

Rather than finding general patterns, the model is likely overfitting to non-general patterns in the training data, some of which are present in the validation data and others in the test data. Although the accuracy on most classes is above the random accuracy of 25% that can be expected in a 4-class problem, this is not something that can be much weighted when the performance is observed to be this different on the validation and test data. Convolutional neural networks are very strong tools with millions of tunable parameters that receive a large amount of input data, namely a 3-channel image. Because of this, they can find highly complex non-general patterns that could result in above-random accuracies. However, a slightly positive result is that the errors, apart from the ones made on the "3 operations" class, typically are made on neighboring classes. That is, for example, the errors made on the "5 operations" class are made by predicting either 4 or 6 operations. Nevertheless, since the tendency that the validation and test data performances strongly deviate is observed to be a common theme in all models in the development, the conclusion in this module is that the classification to determine the number of operations fails with the proposed approach on the current dataset.

Further results from the development of this module are not presented and discussed. The reason for this is that analysis of results that are assumed to be strongly influenced by overfitting to non-general patterns can lead to wrong conclusions. That is, when comparing different configurations of models that contain multiple "wrong" patterns, it can not be determined beyond a reasonable doubt if their results are representative or not. One configuration could, for example, overfit to a non-general pattern that happens to also exist in parts of the validation set, causing the results to indicate that this configuration is superior to another one that overfits to patterns that are not present in the validation set.

Although further results are not presented, one interesting notion from the development is that the fine-tuning is observed to particularly benefit the Inception-ResNet-v2. This could suggest that this architecture is given somewhat unfair grounds for comparison in the general Stage 1 because of the exclusion of fine-tuning. An explanation as to why the Inception-ResNet-v2 especially benefits from the fine-tuning could relate to the inception blocks in the model. These allow the network to "choose" which convolutional approach it prefers to solve the current problem. As a result, the architecture could perhaps be somewhat more sensitive to the similarity between the pre-training dataset and the target dataset than the other architectures. As discussed above, however, these results could be strongly influenced by overfitting to non-general patterns and should therefore not be weighted beyond a notion.

Instead of presenting and discussing the results from the development, the remainder of this section is devoted to a discussion as to why the classification failed.

## **Dependencies**

There can be several explanations as to why this classification fails. For one thing, the initial assumptions about the dependencies of the cost driver might be wrong. It is assumed that the geometry and spatial dimensions of a part are the most crucial dependencies and hence that the employed method is the most suitable. If, however, other factors are equally or more important to identify this cost driver, the problem would not be solvable based on just these.

Consultations with cost and tooling engineers, however, suggest that the geometry and spatial dimensions of a part are, in fact, the most important factors to determine the number of operations. An additional factor of importance is the tolerance with which a flange, for example, needs to be created. A high positional tolerance could generate the need for an additional "restrike" of the

flange, increasing the number of operations by 1. Tolerance data is not available in this work, and it is hence not tested if the inclusion of such information could improve the results. Since there are some tendencies in Figure 57 that the models miss by only 1 operation when they make errors, the exclusion of tolerance information could potentially explain some of these. The tendencies are, however, not sufficiently strong so that it is assumed that the inclusion of this information would solve all problems, and the geometrical and spatial information is still assumed the most important dependencies.

The assumed primary dependencies are additionally supported by previous works on the subject of establishing process plans. These suggest that the machining features, which are geometrical shapes of a certain size and position, are the most important dependency of the process plan [74]. Since the number of operations is a count of the steps in a process plan, it follows that the machining features also are suggested as the primary dependency of the number of operations.

### **Abstract problem**

The above discussion indicates that the assumed dependencies of the number of operations are correct and that these are not the primary reason for the errors experienced. A more probable explanation as to why the classification in this module fails, could be that the problem is too abstract. That is, even though the number of operations depends on the geometry and size, there are too many factors in play for the employed network to find general patterns in the current dataset. There are two main solutions that are considered promising to resolve this issue.

The first solution is to increase the size of the dataset. As can be observed in Figure 56, there are, for example, very few parts in the current dataset that are manufactured with 3 operations. The number of examples of the other classes is also low when considering the high complexity of the problem at hand and that the number of unique parts is  $7 \times 8 = 56$  times less than the presented number of image examples due to the augmentation and multi-view image representation. In machine learning, the amount and quality of the training data is of crucial importance to solve a problem. As previously mentioned, the data is, in fact, often considered even more important than the model itself. Increasing the amount of training data could, as such, perhaps allow the model to find the necessary general patterns to identify the correct number of operations.

A subsequent question relates to the quality of the data. There is not sufficient time within the scope of this work to engage production engineers to manually validate the dataset. That is,

manually going through all, or most, data points to validate that the label matches the part. It can, as a result, not be excluded as a possibility that the insufficient model performance in this module could be caused by a poor dataset with multiple inaccurate labels.

A second solution to solve the problem in this module is to attempt an abstraction of the problem itself. For example, since the process plan, and hence also the number of operations, depends on the machining features, a potential solution in order to solve the problem could be to attempt identification of the machining features in the part prior to extracting the number of operations. In such an approach, the number, size, and position of the various features could be analyzed either with a network or a rule-based scheme to finally determine the number of operations. A network approach could potentially combine this concrete information about the features with a direct geometrical input in the form of an image to attempt a classification. The spatial model architecture could, as such, potentially be employed to solve the problem by including the feature information as the "spatial" information.

#### **6.4.4 Summary and future directions**

The results presented in Table 9 are considered to be poor, and they do not function as a proof-of-concept for the employed approach to determine the number of operations required to manufacture a part. The primary causes of the poor results are assumed to be related to the complexity of the problem and the available data.

Since the number of operations generally are found to have a small impact on the production costs, which are the main aim of this work, it is not prioritized to pursue further solutions to this problem. The above discussion is, however, intended to function as a basis for future works to develop a working module to determine the number of operations.

#### **6.5 Regression #2 - Blank size**

In Regression #2, the aim is to determine the blank size of a sheet metal part. The blank geometry is dependent on the production method that is employed, and this module hence follows Classification #3. In this work, it is assumed that the Press parts can be considered as one with regards to blank geometry. Regression #2 will hence consist of two network models, where one is trained to predict the blank size of the Press parts and the other the blank size of the ProgDie parts.

As opposed to the previous problems, this is a regression problem. That is, instead of belonging to some class, the blank size can take on any real positive value. This results in some general changes to the settings of the network, which are discussed in Section 5.3.2.

### 6.5.1 Data

The blank of a sheet metal part is the initial workpiece from which the part is manufactured. The blank size refers in general to the length and width of this workpiece, prior to it being trimmed to any particular shape and subsequently formed into the final part geometry. That is, the blank size is the 2-dimensional size of the material that is required to manufacture the component. For the production costs, this 2-dimensional size is used to find the volume and later the mass of the required material in order to calculate the cost of the raw material, whose price typically is provided per unit mass. Since the final target with respect to the production costs is the blank mass, meaning the geometrical target is the volume, the actual rectangular dimensions of the blank are of less importance. This work will therefore consider the blank size as the area of the blank. That is, the problem in this module is a regression problem with a single floating-point output value, namely the blank area. The reason for desiring to use the blank area rather than the length and width as a target, is an assumption that it is easier for the network to predict a single floating-point output, rather than the two required for the rectangular dimensions.

Figure 58 displays the distribution of blank areas in the dataset. On the top row, the blank area is plotted against the part area for both Press parts (left) and ProgDie parts (right). On the bottom row, the blank area is plotted against the global bounding box diagonal. Together with the data points in these plots are linear regression lines. These lines are plotted in green and are the fitted linear curves relating the blank area,  $A_{Blank}$ , to either the part area,  $A_{Part}$ , or the global bounding box diagonal,  $BB_{Diag}$ . The linear regression lines are defined as:

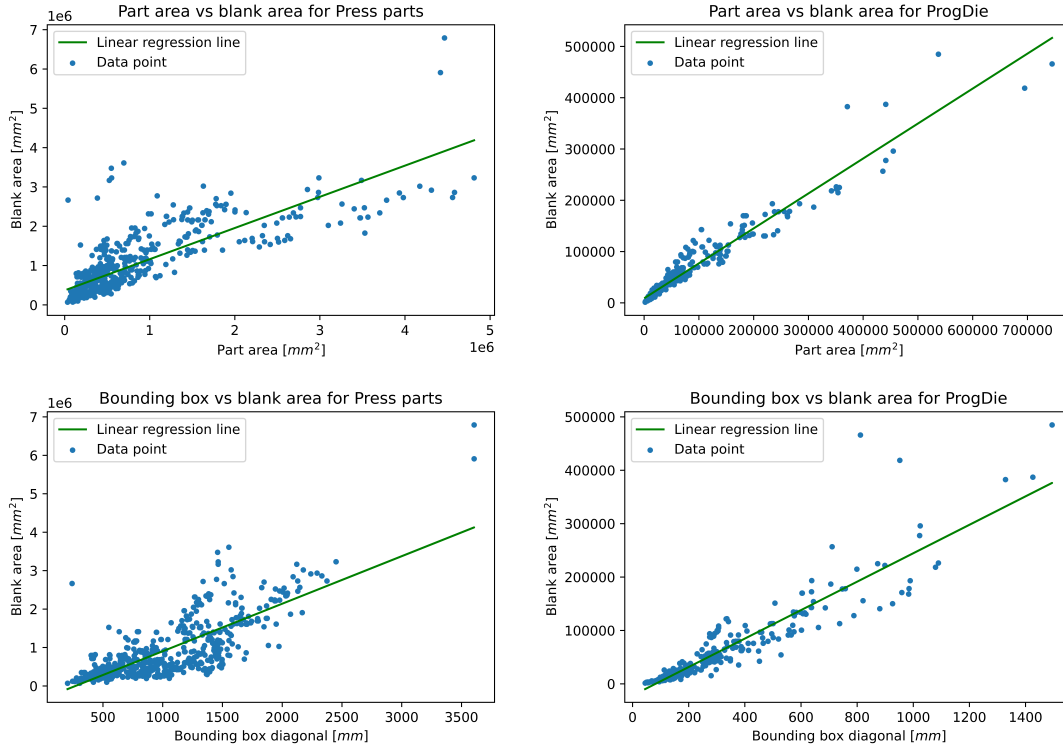
$$A_{Blank} = a_0 \times A_{Part} + b_0 \tag{33}$$

for the part area, and:

$$A_{Blank} = a_1 \times BB_{Diag} + b_1 \tag{34}$$



for the bounding box.  $a_0$  and  $b_0$  are the parameters of the linear regression for the part area, and  $a_1$  and  $b_1$  are the parameters of the linear regression for the global bounding box diagonal. These parameters are found using the first-degree *polyfit* function in the Numpy library in Python [107].



**Figure 58** – Distribution of blank areas by part area (top) and global bounding box diagonal (bottom) in Regression #2.

The linear regressions are performed to provide a reference for the development and to enable further analysis of the data. The results of the regressions are presented in Table 10. Here, since this is a regression problem, the mean absolute percentage error (MAPE) and mean absolute error (MAE) are used as metrics to describe the model performance. The MAPE is presented and discussed in Section 5.3.2. The MAE is simply the mean of the absolute errors on all evaluated data points,  $N_b$ , calculated as:

$$MAE = \frac{1}{N_b} \sum_{i=1}^{N_b} |y_i - \hat{y}_i| \quad (35)$$

where  $y_i$  is the target value and  $\hat{y}_i$  is the predicted value. Note that, as opposed to the case for accuracy, a superior model is identified by a smaller error.

Metric	Part area	Global bounding box
MAPE Press parts	52.29%	51.18%
MAE Press parts	306850.65mm <sup>2</sup>	321221.00mm <sup>2</sup>
MAPE ProgDie	38.42%	38.61%
MAE ProgDie	10439.46mm <sup>2</sup>	13536.01mm <sup>2</sup>

**Table 10** – The results from the linear regressions in Regression #2.

In Table 10, the results are observed to be similar for both regressions, although the part area appears slightly better than the global bounding box diagonal at estimating the blank area by itself. This is also somewhat visible in the plots in Figure 58. For ProgDie, for example, some more outliers are observed far from the linear regression line for the bounding box diagonal than for the part area. Still, the results are fairly close, and the two are hence considered to have similar performance alone. For both linear regressions, however, the results are poor, with above 50% MAPE on estimating the blank area of Press parts and almost 40% MAPE on ProgDie.

The fact that the performance is better on ProgDie than on Press parts is interesting. This is also clearly visible in the plots in Figure 58. That is, the data points are much closer to the regression line for the ProgDie parts than for the Press parts. A reason for this could potentially be that the ProgDie parts typically have somewhat less complex geometry than the Press parts.

The observations made upon analysis of the linear regressions are kept in mind when developing the convolutional neural networks to estimate the blank size.

The dataset used for the development of this module contains the same parts as the one in Classification #3, provided data is available on their blank sizes. Because of the issues encountered with the base dataset in that module, the augmented dataset with 8 random rotations of each part is employed in this module as the base. Additionally, a larger augmented dataset with 16 random rotations is prepared for this problem. This dataset is possible to employ within reasonable computational efforts due to the separation of Press parts and ProgDie.

### 6.5.2 Testing approach

The testing approach employed in the development of this module is as follows:

1. Rough tuning of spatial VGG-16
  - (a) Rough tuning of learning rate
  - (b) Dropout test
2. Validation of spatial model
3. Augmentation test and fine-tuning
4. Validation of multi-view model

The development is performed on the Press part dataset. That is, the above testing approach is applied only on the Press part dataset, and the final configuration developed for that dataset is subsequently inherited to train a model on the ProgDie dataset. This is done in order to reduce the development efforts, based on the assumption that these problems are sufficiently similar to employ the same model configurations. The reason for using the Press part dataset for development is that this dataset is the larger one.

As can be observed in the above testing approach, only the VGG-16 is used in the development. The reason for this is the strong results obtained with this architecture in other modules. Although Classification #4 found the Inception-ResNet-v2 to be the best, it is unclear the extent to which this result is influenced by the overfitting observed in the models. Additionally, the Inception-ResNet-v2 only prevailed after fine-tuning. Requiring fine-tuning of all configurations proved an exhaustive task in the development of Classification #4, and the VGG-16 is hence rather employed directly in this module to limit the development efforts.

An additional change in the testing approach compared to previous modules is that Stage 1 in this development employs the spatial model. That is, rather than performing the rough model tuning with the regular model and subsequently extending this to a spatial model in Stage 2, the spatial model is the one subjected to rough tuning. This is done because the blank size is assumed to be so dependent on the spatial information that the regular model is unable to achieve sufficient results during tuning. Comparing configurations of models that are not learning well is undesired, because the results in such cases can be strongly influenced by overfitting to non-general patterns or other random factors. The spatial model is hence employed during tuning, and the assumption that this model is superior to the regular one is validated in Stage 2 by comparing the performance of this

model to that of the regular one, both employing the roughly tuned configuration.

Both the part area and global bounding box are considered particularly interesting to use as spatial information in this problem. Additionally, combinations of the two, perhaps also together with the sheet thickness, could be of interest. However, to limit the development efforts, only the global bounding box is prioritized to test as a spatial input in this module. The reasons for this are that the global bounding box has proven efficient as a spatial input in previous modules, that the initial results from the linear regression indicate that the bounding box diagonal is similarly efficient to the part area at predicting the blank size alone (Table 10 and Figure 58), and that it is assumed that more information can be obtained about the spatial dimensions of the part from the bounding box. The latter assumption is based on the fact that the bounding box has 3 measurements, namely length, width, and height, compared to the 1 measurement that is the part area.

In order to further reduce the development efforts, the 16x augmented dataset and fine-tuning of the single-view model are tested as Stage 3. These techniques have previously improved the results, although to differing degrees. The available development time for this module is limited, and these techniques are hence validated prior to extending the models to multi-view rather than afterward in the previous "Final tuning" stage.

The extension to multi-view bids an important change in this problem compared to the categorical ones that have previously been discussed. This change is that the score fusion no longer can be performed according to the presented democratic and product evaluations. That is, the output from each individual view in a regression problem is the estimated target value of that view, rather than a vector of predicted probabilities that the input belongs to the different classes. As a result, the outputs from these views must be evaluated differently from those in the previous modules, that either multiply the individual probabilities of each class for each view (product-score) or select the class most frequently predicted across the views (democratic-score). Three main scores are monitored for the regression problem in this work, namely the mean of the estimations from each view, the mean of the center 3 estimations, and the median of the estimations. The mean and median are common metrics used to evaluate a collection of numbers and are hence natural choices to monitor. The center 3 mean, which is the mean of the three predictions in the middle of the sorted vector of estimations from each view, is included as a combination of the two. This metric could be less sensitive to single-view performances than the median and less sensitive to large

variations in view performances than the mean. In addition to these three metrics, the outputs for each individual view are monitored for further analysis.

Since this is a regression problem, the mean absolute percentage error (MAPE), defined in Equation 30, is chosen as the key performance indicator. Much attention is, however, also put on the mean absolute error (MAE), defined in Equation 35, since this describes how large the actual errors are. These two metrics typically adhere relatively well, meaning that a superior configuration often will have both a lower MAPE and MAE than an inferior one, but they can, in some cases, deviate. In these cases, prediction graphs and other relevant aspects are analyzed to attempt a separation of the models. If the separation is still not successful, the model with the better MAPE is chosen.

Note that the performance indicators in this problem relate to errors rather than accuracy, which is the performance indicator in previous modules. The aim is, in other words, to minimize the metrics rather than maximize them.

### 6.5.3 Results and discussion

Regression #2		
	Press parts	ProgDie
Target	Blank area	Blank area
MAPE validation	22.79%	18.66%
MAE validation	166484.94mm <sup>2</sup>	12190.82mm <sup>2</sup>
MAPE test	18.14%	22.57%
MAE test	112122.39 mm <sup>2</sup>	8424.14mm <sup>2</sup>
Convolutional architecture	VGG-16	VGG-16
Learning rate	1e-3	1e-3
Fine-tuning learning rate	1e-5	1e-5
Dropout	50% on both FC4096	50% on both FC4096
Spatial input	Global bounding box	Global bounding box
Multi-view fusion technique	Score (median)	Score (median)

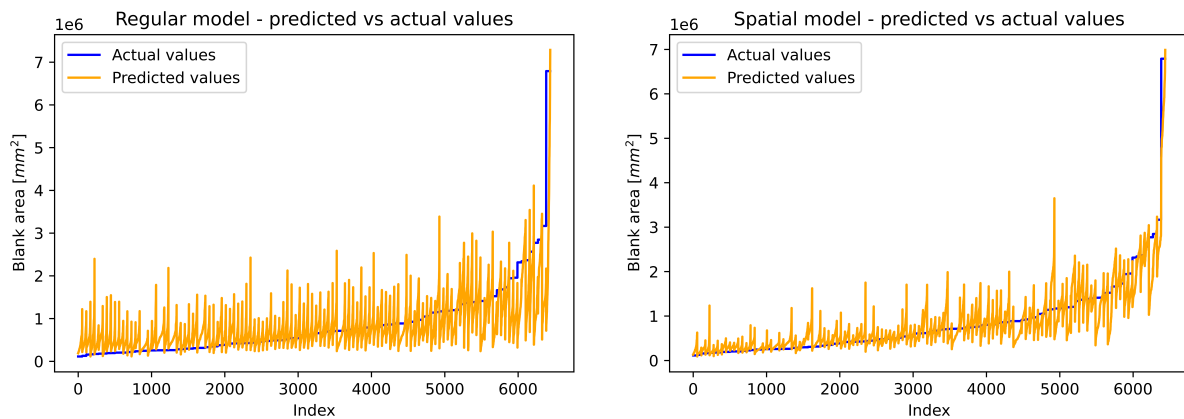
**Table 11** – The final configuration of the networks in Regression #2.

Table 11 presents the characteristics of the final networks developed for Regression #2. On the Press parts, a validation MAPE of 22.79% is achieved, supported by the test MAPE of 18.14%.

For ProgDie, a validation MAPE of 18.66% is achieved. This is considered fairly supported by the test MAPE of 22.57%.

### Impact of spatial information

As can be observed in Table 11, the assumption relating to the superiority of the spatial model on this problem is validated. This is clear from observation of the plots in Figure 59, the left of which displays the validation results of the regular model and the right one the results of the spatial model. In these plots, the predicted blank area values of the parts in the validation set are displayed in orange together with the actual blank areas of these parts in blue. The y-axis measures the blank area, with the index on the x-axis. The index is merely a count of the examples in the validation set, and it can hence be observed in the plots that there are around 6500 examples in the validation set. The values are sorted according to the actual blank areas, which means that index 0 has the smallest blank area. The graph of the actual values can be observed to contain "steps", where multiple indices have the same blank area. This is a combined result of the augmentation and multi-view representation, which results in  $7 \times 8 = 56$  images of equal blank areas. Looking closely at the graph of the predicted values, it can be observed that these increase in size over the indices of a step. This is merely a result of the sorting and is not related to any view relations.

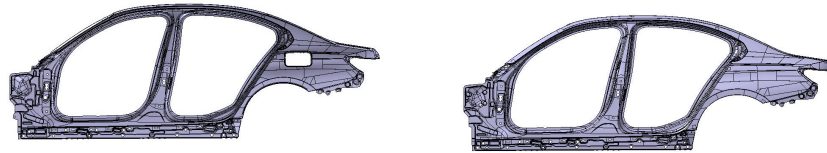


**Figure 59** – The results of the regular (left) and spatial (right) models after the initial rough tuning in Regression #2.

From the plots in Figure 59, it can be clearly observed that the spatial model outperforms the regular one. Whereas the predictions in the regular model tend to merely oscillate around some mean value of a little below  $1e6mm^2$ , the predictions in the spatial model follow the actual values much closer, particularly for the smaller blank sizes. Once the actual values start increasing more,

around index 4000, the predictions of the spatial model are somewhat worse, but there is still a tendency that they increase for higher indices.

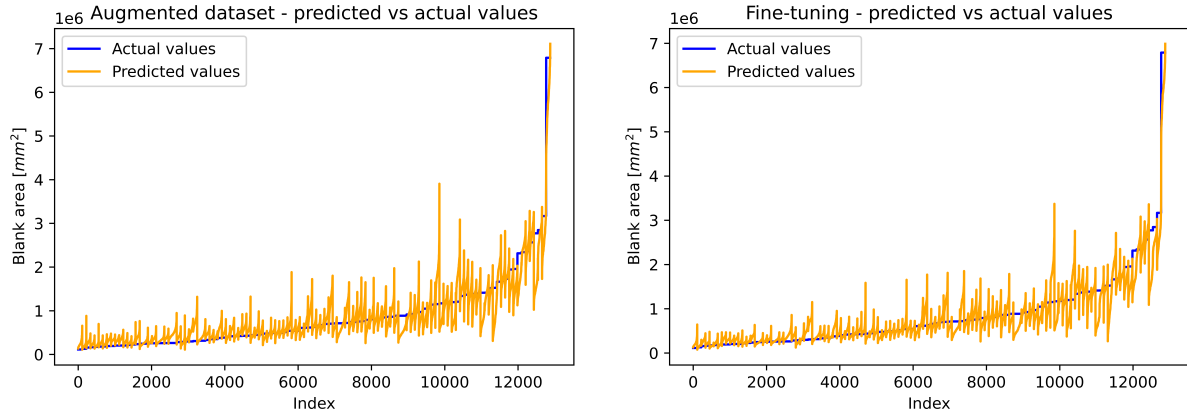
An interesting observation that can be made in Figure 59 is that both models make relatively good estimations of the largest blank size. That is, the orange graph in both plots is observed to peak close to the blue one with a blank size of approximately  $7e6mm^2$  around index 6500. Upon closer investigation of the dataset, it is found that only two parts have blank sizes in this region. This is also visible in the plots of the Press part blank sizes in the left column in Figure 58, where two data points exist alone in the upper right regions. These two parts are presented in Figure 60. The left one of these parts exists in the training set, whilst the right one exists in the validation set. These two parts are observed to have very similar geometries, both being complete car side frames. As such, the fact that the regular model also is able to make a decent prediction for this part, seeing as it has trained on a similar part with a similar blank size, is not that surprising.



**Figure 60** – The two parts with the largest blank sizes in the dataset in Regression #2.

### Improving the single-view model

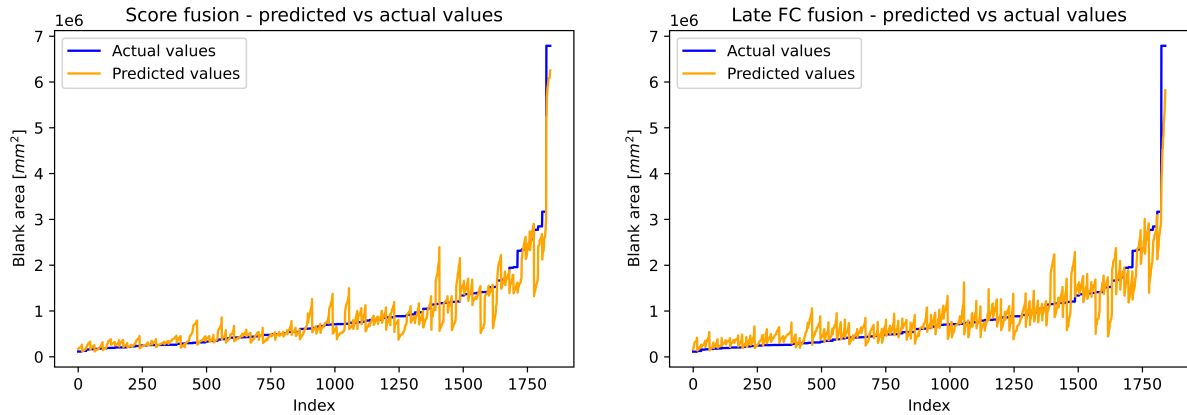
To further improve the results of the spatial model in Figure 59, it is trained on the 16x augmented dataset and subsequently fine-tuned on the same dataset. The results from these tests are presented in Figure 61. In these plots, both the increased augmentation and fine-tuning can be observed to slightly improve the performance of the model. From the spatial model in Figure 59 to the same model trained on the augmented dataset to the left in Figure 61, and finally to the fine-tuning of this model to the right in the same figure, the orange graph is getting closer to the blue target line, with fewer and lower error peaks. In total, the MAPE is reduced from 30.81% in the initial spatial model to 26.44% in the model trained and fine-tuned on the augmented data.



**Figure 61** – The validation results when training on the 16x augmented dataset (left) and subsequently fine-tuning the model on the same augmented dataset (right) in Regression #2.

### Extension to multi-view

As presented in Table 11, score fusion is found to be the superior multi-view fusion technique for this problem. Figure 62 presents the two strongest models obtained with the score (left) and late FC (right) fusion techniques. The score fusion performs visibly better than both the late FC model and the previous single-view model. As presented in Table 11, this model achieves the final validation MAPE of 22.79%.



**Figure 62** – The best validation results for the different fusion techniques in Regression #2.

The score evaluation that is used to obtain the presented results is the median of the predictions from each view. The median prediction proved slightly better than the center 3 mean, both performing around 2% better than the mean. This could suggest that the predictions are quite view-dependent, with some views performing much poorer than the rest.



That the predictions are view-dependent is also evident upon analysis of the results from each individual view. In fact, the most interesting and promising result from the development of this module is found when investigating what performance is achieved if the best single-view prediction always is selected as the final output. This investigation shows that always selecting the best prediction across the views reduces the MAPE to 12.81%. That is, if finding a better way than the median to evaluate the score fusion, the MAPE of this model can be reduced by around 10%. The view that provides the best prediction, however, differs from part to part.

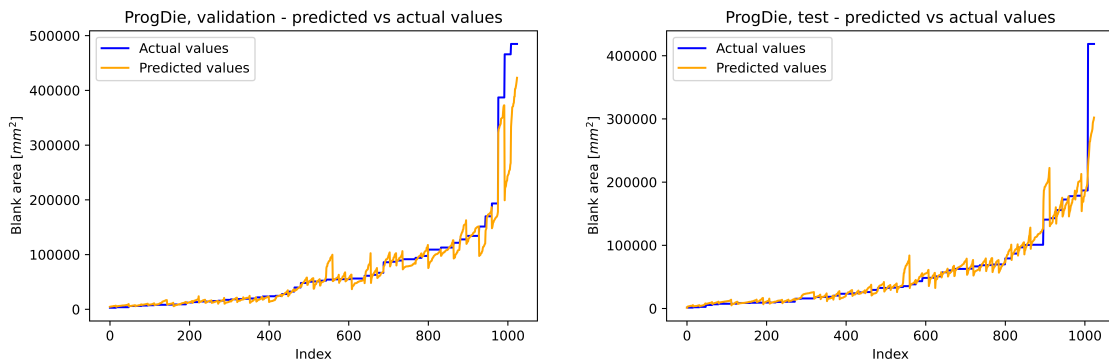
Unfortunately, there is not sufficient time within the scope of this work to dive into a deeper analysis of the views that provide the best predictions.

### ProgDie results

The best performing configuration from the development of the Press parts network is trained on the ProgDie data in order to evaluate also these parts. The dataset that is employed is the 16x augmented dataset.

The validation and test results for the ProgDie parts are presented in Figure 63. In both these plots, the predicted values are observed solid on the smaller parts, with some larger deviations from the actual blank size for the higher indices.

These results are also significantly increased if the best view prediction is chosen each time, rather than merely evaluating the median. The validation MAPE, for example, is then reduced from 18.66% to 7.75%.



**Figure 63** – The validation and test results on the ProgDie parts in Regression #2.

#### 6.5.4 Summary and future directions

The final results of the networks developed for Regression #2, as presented in Table 11, are considered to be decent. On both the Press parts and ProgDie parts, the predictions of the smaller blank areas are better than those of the larger ones. This can potentially be explained by the smaller amounts of data available on the larger blank sizes. As can be observed in the distribution of the blank sizes in Figure 58, most parts have blank sizes in the lower region of the plots.

The results obtained in the development of this module are considered to function as a proof-of-concept for the employed method to estimate the blank size. However, further work is necessary to achieve a higher performance before the networks can be implemented into a finished system.

Future works are encouraged to pursue analysis of the views that achieve the highest performance in the score fusion multi-view models. The images of each view can be investigated to see if any patterns emerge that can be employed as a score evaluation to ensure the selection of the view that provides the best prediction each time. These patterns could, for example, relate to the 2D bounding box of the part in the images of the different views. A network approach for the selection of the best view could also be possible. In such an approach, a late fully-connected multi-view model could, for example, be trained to predict which of the 7 input images is most likely to provide the best estimate.

Additional improvements are assumed possible here, as in other modules, through further hyperparameter tuning and by increasing the dataset size. Since only moderate improvements are observed when increasing the augmentation from the 8x augmented dataset to the 16x augmented dataset, the primary gains in dataset increase are assumed to lie within the labeling of further new parts. Normalization of the spatial information and an increased number of views, as discussed for the fastener identification in Section 6.1.4, could also potentially increase the performance here. The impact of separating the Press parts into Transfer and Press line parts can also be investigated.

Future works can also consider implementing a FEM solution rather than the employed learning-based method. As discussed in Section 4.2.2, FEM is the main method for simulation of sheet metal forming, and FEA is able to make strong estimations of the blank geometry in a relatively short amount of time. Implementation of such a method in the proposed system

would entail extending the Preprocessor to additionally prepare a finite element model that can be analyzed.

## 6.6 Regression #1 - Welding points

In Regression #1, the aim is to determine the number of welding points inside of a single sheet metal part or between the different sheet metal parts in a small assembly. The number of welding points is assumed to be primarily dependent on the thicknesses, materials, and contact areas of the sheets that are welded together. These five entities are hence the input provided to the fully-connected network in this problem.

As the estimation of the blank size, this is a regression problem. The reason for this is that there is no discrete interval on which the number of welding points exists. Unlike the problem in Regression #2, however, the number of welding points can only take on integer values, as compared to the floating-point blank areas. For example, it is not possible to have 3.5 welding points. The linear output neuron in a regression network does, however, still output a floating-point value. The integer output of the networks in this module will therefore be determined by rounding the floating-point output to the nearest integer value.

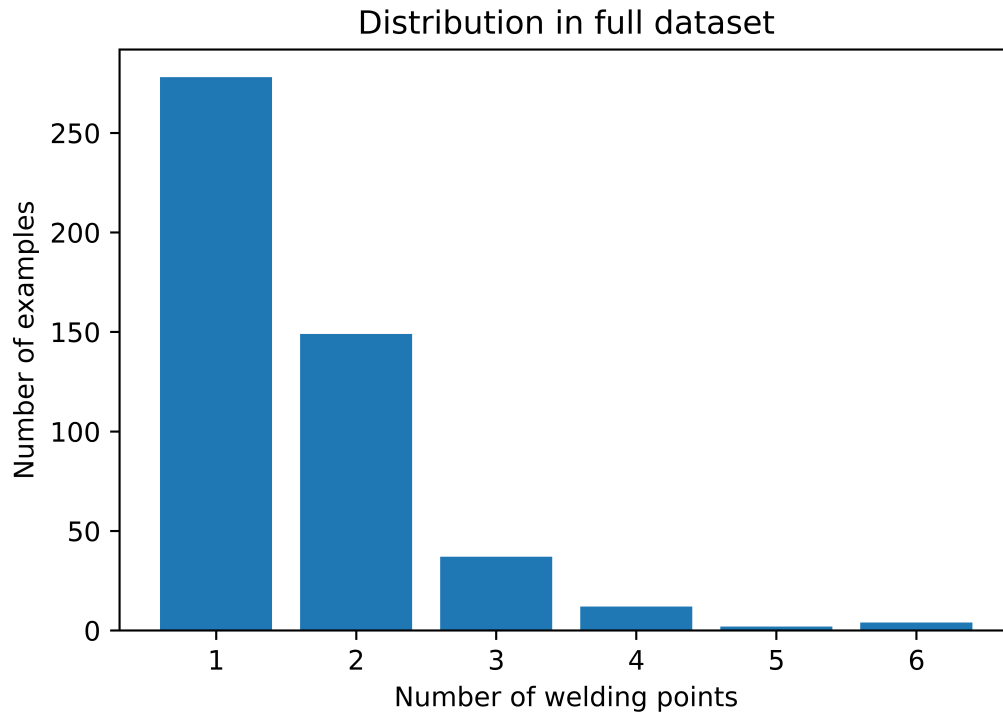
### 6.6.1 Data

The data used in Regression #1 relates the materials, thicknesses, and contact areas of welded sheets together with the number of welding points that are used to join them. In order to include the material type as an input, an integer is assigned to each specific material type. The reason for this is that the network requires numerical inputs. 1 could then, for example, refer to some aluminum alloy.

Unfortunately, there is little data available that relates the number of welding points to its assumed pivotal dependencies. This is because the tool employed in Analysis #3 was unavailable during the data acquisition, meaning manual labor was required in order to acquire data for this module.

Figure 64 displays the distribution of welding points in the available dataset. In this plot, it can be observed that there are very few examples of 4 welding points and above. In fact, only two examples are available of 5 welding points. There are also no data points with more than 6 welding points, without 6 welding points being some threshold for the maximum number of welding points

possible. Still, the data is assumed sufficient to make an evaluation of the intended approach to determine the number of welding points.



**Figure 64** – Distribution of the number of welding points in the full dataset in Regression #1.

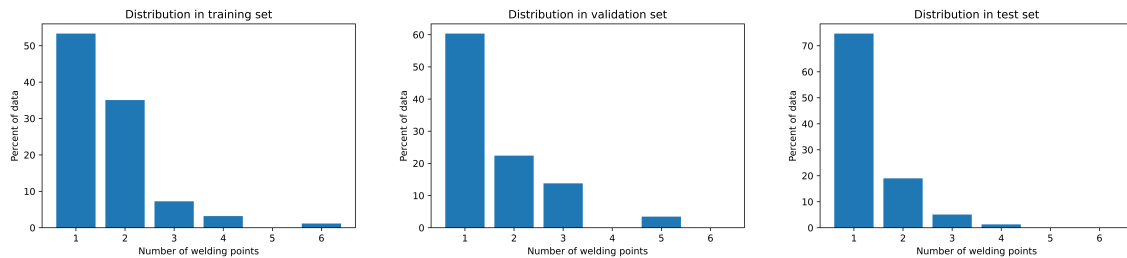
Because so little data is available on 4 welding points and above, it is considered problematic to demand certain amounts of examples of each number of welding points in each of the training, validation, and test sets in order to keep their split random. Since this is a regression problem, where the output takes on a continuous floating-point value which subsequently is rounded to the nearest integer, a strong model should be able to learn general patterns that allow it to predict a number of welding points of which it has not seen any examples. In other words, the network attempts to learn a function that relates the input to the output. This is different from the categorical case, where the model needs to see examples of each class in order to learn the general patterns associated with all of them.

Since it is considered problematic to include any logic in the dataset split, the training, validation, and test sets are hence split completely randomly. About 70% of the data is included in the training set, and about 15% is included in each of the validation and test sets. The distribution in the sets is evaluated after the random split to ensure satisfaction with regard to the output space covered

by each set. If, for example, no examples exist of more than 3 welding points in one of the sets, the dataset is re-split. This is common practice to try and ensure representative sets [46].

Figure 65 presents the final distributions in each set. To the left, the distribution in the training set is plotted, in the center the distribution in the validation set, and to the right the distribution in the test set. The y-axis is here measuring the percent of data that each number of welding points make up in the given set. For example, it can be observed that about 50% of the data in the training set are examples of 1 welding point.

In the plots in Figure 65, it can be observed that there are no examples of 5 welding points in the training set, no examples of 4 or 6 welding points in the validation set, and no examples of 5 and 6 welding points in the test set. This split is considered acceptable with the available data because it, amongst other things, enables both an evaluation of the model performance at predicting 5 welding points, of which it has not been provided any training examples, and a final validation of the performance on predicting the most common numbers of welding points, namely 1 through 4.



**Figure 65** – Distribution in percent in each set in Regression #1.

The base dataset is additionally subjected to normalization, generating a normalized dataset. The reason for this is that normalization, as briefly mentioned in Section 6.1.3, often is found to improve the network performance because it ensures correlation between the sizes of the different inputs and the initial randomly generated weights in the network [46]. The weights are initially drawn from a normal distribution with mean 0, and the inputs are hence normalized on the interval [-1, 1] according to:

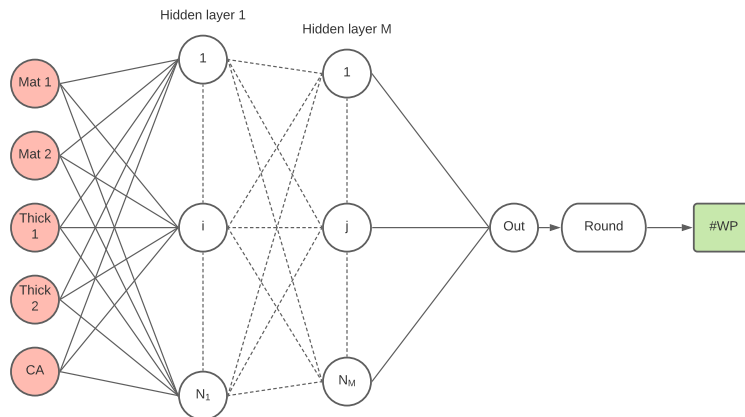
$$\bar{x}_i = 2 \times \frac{x_i - \min(\vec{x})}{\max(\vec{x}) - \min(\vec{x})} - 1 \quad (36)$$

where  $\bar{x}_i$  is the normalized value,  $x_i$  is the original, and  $\vec{x}$  is the vector of all original values.

The thicknesses of the different sheets are normalized together, meaning that the overall maximum thickness is 1 and the overall minimum is -1. The same is done for the materials.

### 6.6.2 Testing approach

The testing approach employed to develop the Regression #1 network is unique in this work. This is a result of the different network type that is tested to solve this problem. Whereas previous modules employ complex convolutional networks that take image inputs, testing different existing architectures and extensions of these with different configurations, this module will test a simpler fully-connected network, taking 5 positive real number inputs and predicting a single floating-point value that is rounded to the nearest integer. An illustration of the network that will be developed in the module is provided in Figure 66. This fully-connected network will take the material of sheet 1, the material of sheet 2, the thickness of sheet 1, the thickness of sheet 2, and the contact area between the sheets as inputs to finally predict the number of welding points. As can be observed in Figure 66, the architecture itself is yet to be determined. Determining the architecture is one aim of the testing approach for this module.



**Figure 66** – Illustration of the network to be developed in Regression #1.

Instead of an iterative testing approach where each configuration is manually analyzed to determine the next steps, the development of Regression #1 will employ an optimization loop. This loop will perform rough grid searches to determine a network architecture, a suitable learning rate, and a suitable batch size. As in previous modules, early stopping that monitors the loss is used to cease training once no improvement is achieved in 10 epochs. The loss function employed in this problem is the same as in Regression #2, namely the mean squared error (MSE).

The following parameters are tested in the loop:

- **Architecture:** 1 hidden layer with the following numbers of neurons: [1, 2, 3, 4, 5, 6, 7, 8, 9]
- **Learning rates:** [1e-1, 1e-2, 1e-3, 1e-4]
- **Batch sizes:** [1, 2, 4, 8, 16, 32, 64]

The reason for testing only a single hidden layer is that this generally is found to be sufficient in fully-connected networks [46]. Basic "rules of thumb" for the number of hidden neurons are discussed in [108] as:

- *"The number of hidden neurons should be between the size of the input layer and the size of the output layer."*
- *"The number of hidden neurons should be: (number of inputs + outputs) \* (2/3)"*
- *"The number of hidden neurons should be less than twice the input layer size."*

The first rule of thumb suggests the number of hidden neurons for this problem is one of 4, 3, or 2. The second one suggests the number of hidden neurons should be 4. The last one states that the number of hidden neurons should be less than 10. Multiple other equations from other works to determine the number of hidden neurons are also presented, none of which result in a higher suggestion than 9.

In [97], it is stated that the most important consideration when selecting the number of hidden neurons whilst using early stopping, which is the case here, is to select this number sufficiently high. [46] supports this by discussing that the typical approach for selection of the number of hidden neurons is to begin exploration with more neurons than strictly necessary. In general, the downside to using "too many" neurons appears to be the computational effort. As a result, the above-presented numbers of neurons are tested, using the maximum suggestion of 9 as an upper threshold. The neurons in the hidden layer use ReLU activation, whilst the output neuron uses linear activation, which is the standard for regression networks.

The learning rates that are tested are the same as for previous modules, apart from the exclusion of

1e-5. The reason for removing this learning rate is that initial trial runs with the optimization loop in this module reveal that configurations employing this learning rate train for several thousand epochs, without ever reaching the performance achieved by other learning rates. For reference, the best performing configurations in the initial trial typically train for less than 100 epochs. As a result, 1e-5 is excluded from testing to increase the efficiency of the loop, with the assumption that this will not influence the results.

In addition to the architecture and learning rate, the batch size is tuned in this loop. Previous modules have successfully used a batch size of 32, but these modules have much larger datasets than this one. As a result, it is considered to be of interest to also investigate different batch sizes in order to achieve strong results on the current problem with the current dataset. The above-presented batch sizes are tested following the findings in [99] that batch sizes up to and including 32 perform the best. This work explores these and includes 64 as an upper limit to validate that no trends exist where higher batch sizes beyond the selected interval increase performance. The batch sizes are distributed on the interval [1, 64] on the binary logarithmic scale following the practice in [99].

In the optimization loop, all combinations of the different parameters are tested, resulting in a total of  $9 \times 4 \times 7 = 252$  tested configurations. The optimization loop is additionally run on both the regular and normalized datasets to investigate the impact of normalization. Each configuration is run 20 times, following the recommendation in [102]. That is, each configuration is trained 20 times with randomly initiated weights (20 random starts).

The highest validation accuracy achieved over the 20 runs for each configuration is stored in a tensor. The same is true for the adhering training and test accuracies for this run. The accuracy, calculated according to Equation 29, is hence employed as the key performance indicator in this development. Although this equation is aimed at categorical problems, it can be employed for this regression problem because of the binary nature of the evaluation of the integer output. That is, a certain prediction of the number of welding points is either right or wrong. Once the loops are finished, the stored tensor is subject to analysis to identify the strongest network configuration.

The reason for not blindly trusting the calculated accuracy by simply returning the final configuration as the one that achieves the highest accuracy in the optimization loops, is primarily the imbalance observed in the distribution of the number of welding points (Figure 64). Because of this imbalance, there could be some differences in the individual accuracies between the different



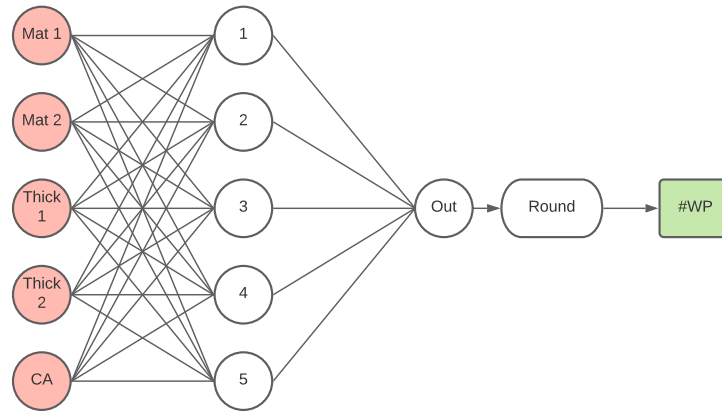
numbers of welding points that suggest the best configuration is not the one that achieves the highest accuracy. A natural question is hence whether the weighted accuracy should be employed rather than the regular one when analyzing the results. This could be achieved by considering the problem as a categorical problem where each number of welding points on the interval  $[1, 6]$  is a "class". However, since there would then exist different "classes" in each set where no examples are available, the direct interpretation of the weighted accuracy is somewhat more difficult to define. In previous categorical problems that employ the weighted accuracy as the key performance indicator, such as Classification #3, a general correlation is observed between the regular accuracy and the weighted one in strong models. That is, a model with a higher weighted accuracy generally also has a higher regular accuracy than an inferior model. As a result, the regular accuracy will be used as the key performance indicator that is registered in the results tensor returned by the optimization loops. The 10 configurations with the highest validation accuracy on each of the non-normalized and normalized datasets are then further analyzed to determine the strongest overall configuration. This analysis involves investigations of the performance also on the training and test sets, as well as evaluation of the confusion matrices that are generated when considering the problem as a categorical one.

### 6.6.3 Results and discussion

Table 12 presents the characteristics of the final network configuration developed for the Regression #1 module. As can be observed in the table, the final network is a fully-connected network with 5 hidden neurons that achieves a validation accuracy of 87.93%. This network is depicted in Figure 67.

Regression #1	
Target	Number of welding points
Validation accuracy	87.93%
Test accuracy	93.67%
Architecture	1 hidden layer, 5 neurons
Learning rate	1e-2
Batch size	1
Dataset	Normalized

**Table 12** – The final configuration of the network in Regression #1.



**Figure 67** – Illustration of the final network developed for Regression #1.

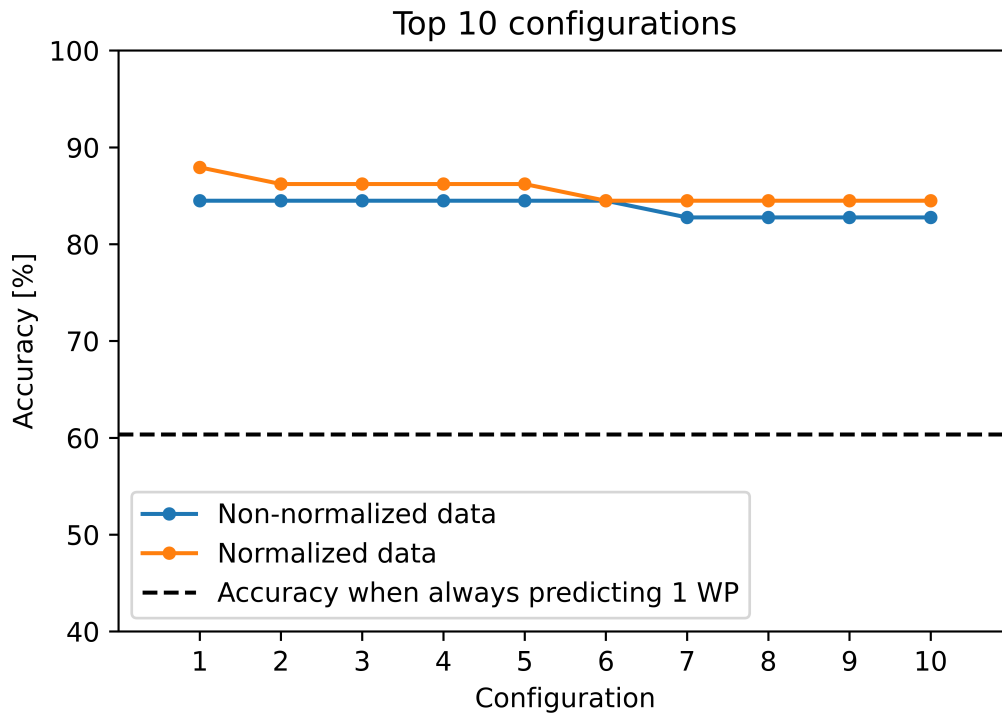
### Optimization loops

The validation results of the 10 best configurations for each of the regular and normalized datasets are presented in Figure 68. In this figure, the top 10 configurations for the normalized data are plotted in orange, with the top 10 configurations for the regular data in blue. The plot additionally includes a dashed line, which indicates the accuracy achieved if always predicting that the answer is 1 welding point, for reference. All configurations on both datasets are observed to achieve far superior results to this "random" one, with the smallest gap being around 20%.

As can be observed in the plot in Figure 68, all configurations trained on the normalized data perform better than or equal to the non-normalized data on validation accuracy. The normalized data is also observed to outperform the regular data upon further analysis of these results. Normalization is hence found to improve the performance of this module.

The specifics of the 20 configurations plotted in Figure 68 are presented in Table 13. Note that the sorting amongst the configurations that have the same validation accuracy (see Figure 68) is random. This means, for example, that the table does not suggest that configuration 1 for the non-normalized data is better than configuration 2.

In general, few strong tendencies are observed amongst the configurations. That is, relatively few common traits are found that seem to be superior in the general case. Some tendencies are, however, perhaps worth noting. The best learning rate for the non-normalized data is, for example, usually found to be 1e-1. Although the most frequent learning rate for the normalized data also is 1e-1, the best configurations are observed to employ a lower one, often 1e-2. Another observation



**Figure 68** – Validation results for the top 10 configurations after the optimization loops in Regression #1.

Configuration	Non-normalized data			Normalized data		
	Neurons	Learning rate	Batch size	Neurons	Learning rate	Batch size
1	9	1e-1	16	5	1e-2	1
2	7	1e-1	16	9	1e-2	1
3	6	1e-1	2	8	1e-2	1
4	6	1e-2	1	4	1e-4	8
5	5	1e-1	2	3	1e-3	4
6	3	1e-2	1	9	1e-1	4
7	9	1e-1	4	9	1e-1	8
8	9	1e-2	1	9	1e-1	32
9	7	1e-1	1	7	1e-1	8
10	6	1e-1	1	7	1e-4	2

**Table 13** – The best performing configurations from the optimization loop in Regression #1.

is that a batch size of 1 generally seems to work well for both datasets. Using a batch size of 1 is sometimes referred to as *online learning*, since each weight update made in the network during training is made based on the gradient calculated from a single training example, rather than the mean over all gradients in a larger batch. One reason why online learning performs well for this problem could relate to the fact that there are few data points, and hence that very few weight updates are made per epoch for larger batch sizes that often prove superior in other cases, such as 32.

The number of neurons in the hidden layer in the best performing configurations is generally observed to vary between the configurations. Although there is a tendency that the configurations have more hidden neurons than suggested by the two first "rules of thumb" presented in the section above, it should be noted that several of the configurations perform the same with respect to the validation accuracy. For example, configuration 5 on the normalized data, which employs only 3 hidden layer neurons, achieves the same validation accuracy as, for example, configuration 2, which employs 9 neurons. Still, however, a higher number of neurons typically seems to perform well.

### **Best configuration**

Upon further analysis of the available data, the model with the overall highest validation accuracy is considered the best. This configuration, namely number 1 for the normalized data in Table 13, is hence selected as the strongest configuration. The characteristics of this model are hence what is presented in Table 12, and its architecture is depicted in Figure 67. The validation confusion matrix of this model is presented in Figure 69.

The results in the confusion matrix are considered positive. It can, for example, be observed that when an error is made, this error is generally only a single welding point away from the actual number. Additionally, it is considered to be particularly positive that the network is able to evaluate the two examples of 5 welding points relatively well. The model has never seen any examples with 5 welding points during training. In fact, it has been provided with very few examples of more than 3 welding points. As a result, it is considered a positive sign that the network is able to evaluate the examples with 5 welding points to make one correct prediction and one prediction only one welding point away. The confusion matrix in Figure 69 does, in other words, suggest that the model is learning general patterns in the data that relate the assumed dependencies to the number of welding points.

**Final configuration**

<b>Actual number of welding points</b>	1	33 94.29%	2 5.71%	0 0.0%	0 0.0%	0 0.0%	0 0.0%
	2	0 0.0%	11 84.62%	1 7.69%	0 0.0%	1 7.69%	0 0.0%
	3	0 0.0%	2 25.00%	6 75.00%	0 0.0%	0 0.0%	0 0.0%
	4	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%
	5	0 0.0%	0 0.0%	0 0.0%	1 50.00%	1 50.00%	0 0.0%
	6	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%
		1	2	3	4	5	6
		<b>Predicted number of welding points</b>					

**Figure 69** – Validation results for the final network developed for Regression #1.

## Sensitivity analysis

To investigate how the model processes the different inputs, a simple sensitivity analysis is performed on the final network. This analysis aims to reveal which inputs the network has found to be the strongest dependencies of the target, as well as to ensure that the network is behaving as expected.

In order to perform the sensitivity analysis, distorted datasets with differing degrees of noise in the inputs are generated. Since there are two inputs that relate to the material and two inputs that relate to the thickness, this results in three primary datasets:

- **Dataset 1:** Distortion in contact area
- **Dataset 2:** Distortion in thickness
- **Dataset 3:** Distortion in material

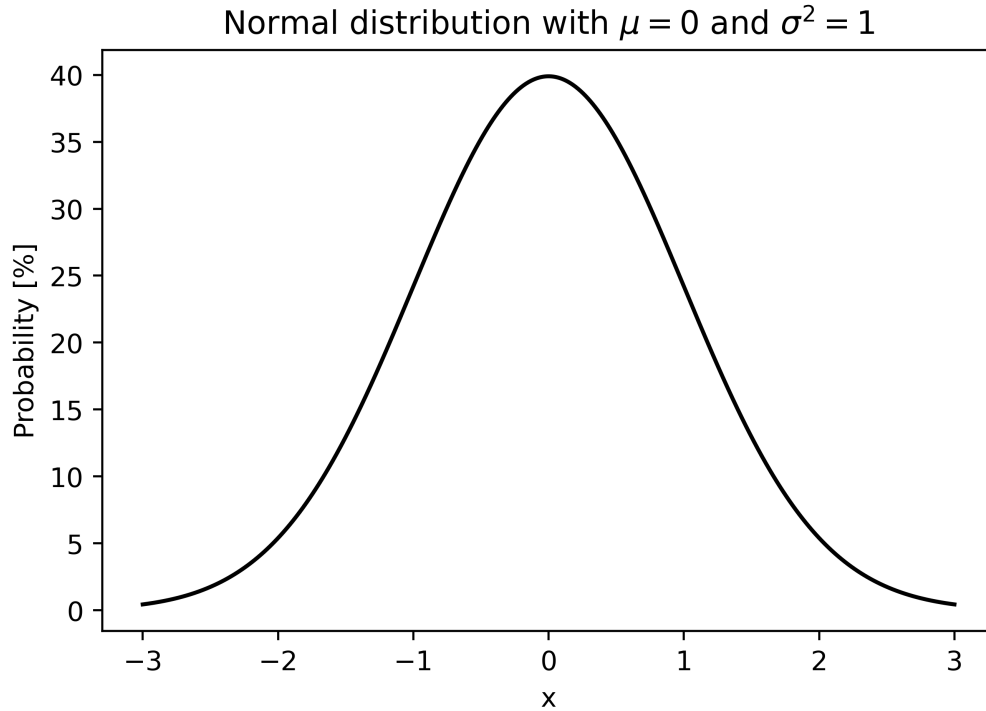
In each of these datasets, the relevant input is subjected to manipulation in order to generate noise. The inputs that are not subjected to distortion remain the same as in the original data.

The 3 primary datasets contain 20 variations for differing degrees of distortion. These variations begin at 5% distortion and increase the distortion in intervals of 5% up to and including 100%. The distorted data points are calculated as:

$$\hat{x} = x + \mathcal{N}(0, 1) \times (D \times x) \quad (37)$$

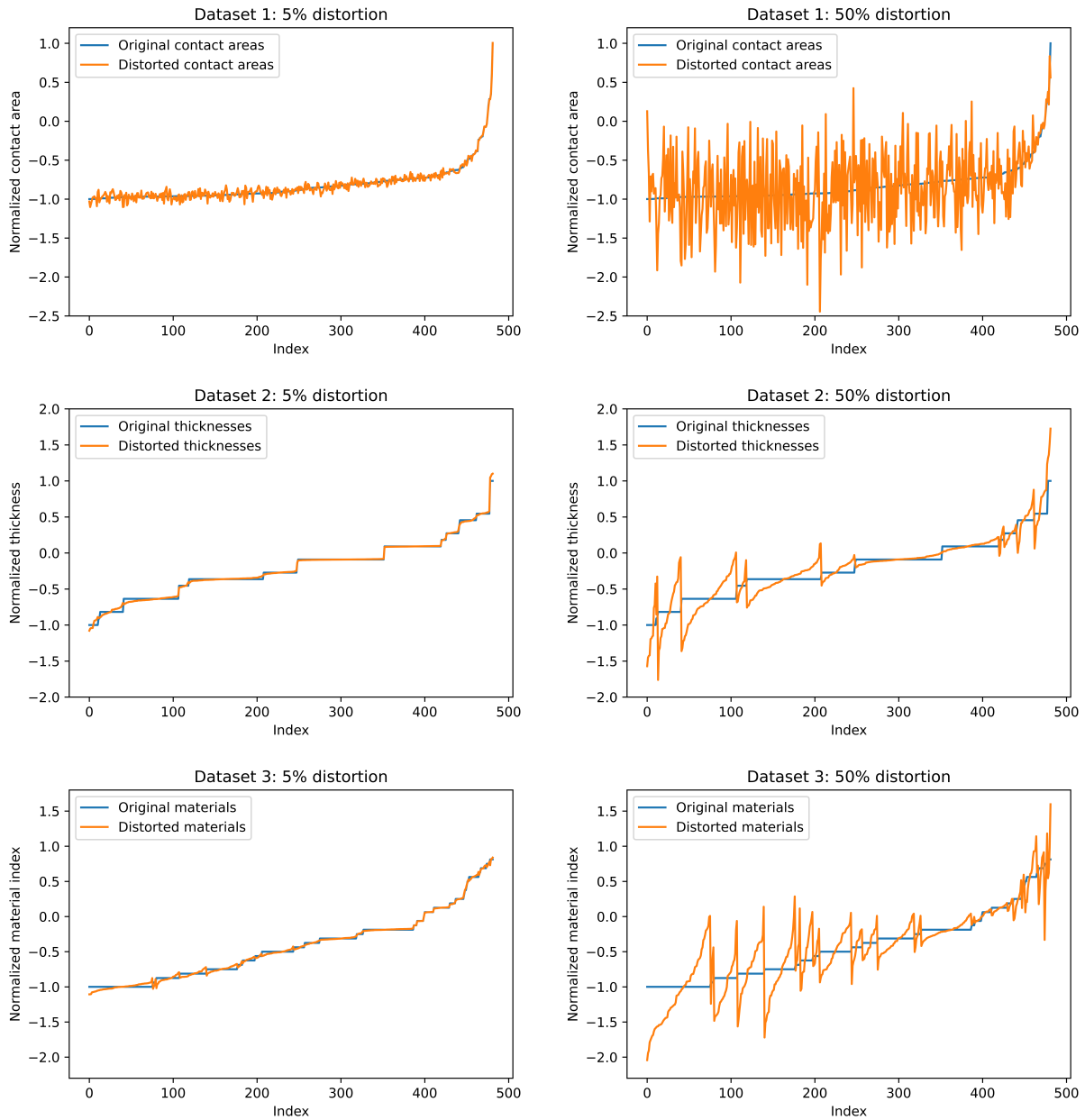
where  $\hat{x}$  is the distorted data point,  $x$  is the original data point,  $\mathcal{N}(0, 1)$  is a number drawn from the normal distribution with mean 0 and variance 1, and  $D$  is the decimal distortion. 5% distortion, for example, means  $D = 0.05$ .

The distorted data point is hence the original data point with some change added that relates to a percentage of the size of the original data point multiplied by a number drawn from the probability distribution plotted in Figure 70. The most probable outcome, at around 40%, is that the original value remains, with probabilities of around 25% each that the percentage distortion is added or removed from the original.



**Figure 70** – The normal distribution with a mean of 0 and a variance of 1.

Examples of the distorted datasets are available in Figure 71. On the top row, the distorted normalized contact areas are plotted together with the original ones for 5% and 50% distortion. In the plot with 5% to the left, it can be observed that the distorted values, plotted in orange, remain relatively close to the blue graph that depicts the original normalized contact areas. For 50% distortion, the values can be observed to oscillate far from this line, meaning the distorted values generally deviate much from the original ones. The second and third rows display the same distortions for the thickness and material, respectively. In these plots, it can be observed that the original values have "steps", meaning multiple indices have the exact same material or thickness. This follows from the fact that the thicknesses and materials are discrete parameters. That is, they take on only specific values. The different material types are, as previously discussed, assigned an integer identifier in order to use them as network inputs. Before the normalization, they are hence spaced out in intervals of 1. Although metal sheets theoretically can have almost any real number as thickness, raw material sheets are mass-produced with certain thicknesses. In turn, this means that also the thicknesses of sheet metal parts typically only take on specific standard values, such as  $1.0\text{mm}$  or  $1.2\text{mm}$ .



**Figure 71** – Examples of the distorted datasets used for the sensitivity analysis in Regression #1.



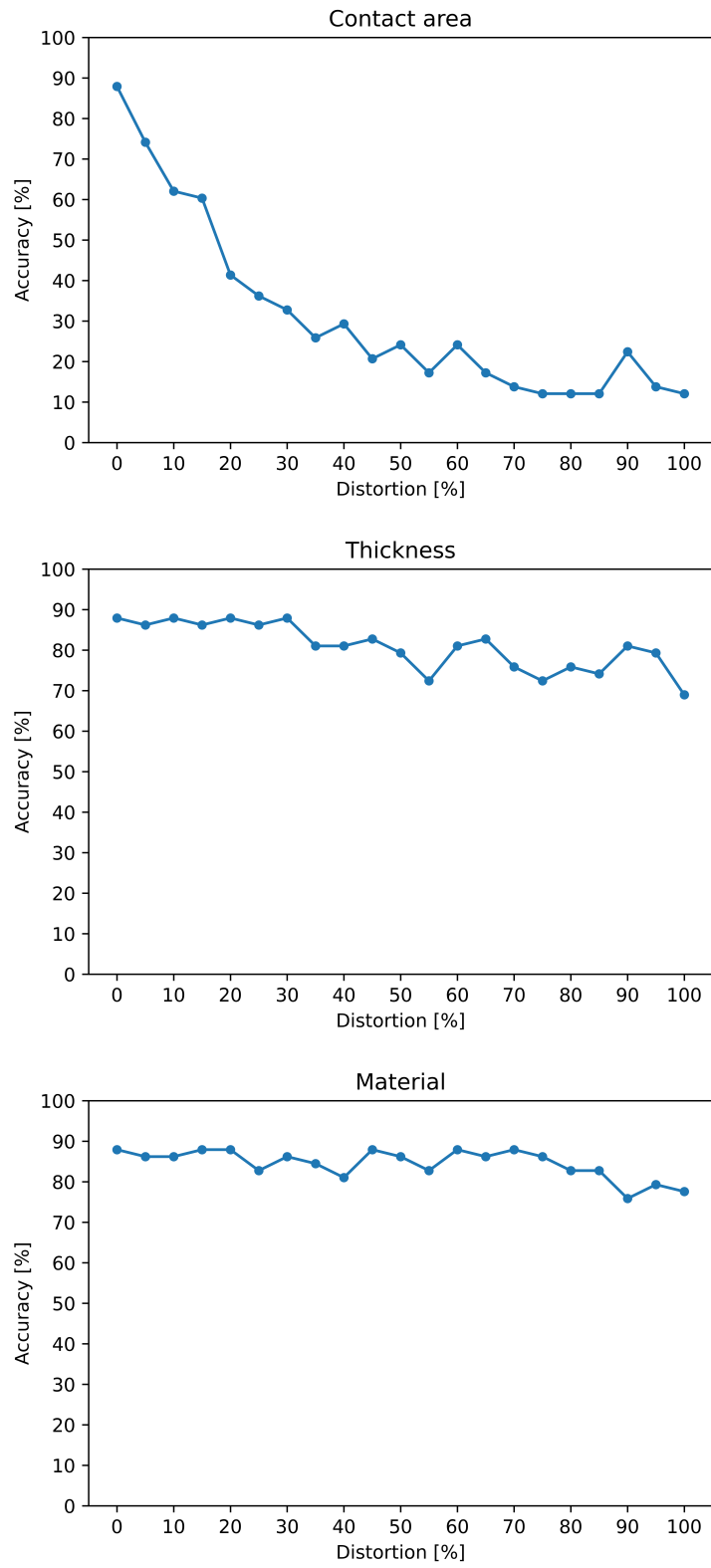
As can be observed in the plots of the distorted thicknesses and materials on the second and third rows in Figure 71, the distorted data takes on continuous values rather than discrete ones. This is done because the aim of the sensitivity analysis is to investigate how the performance of the network reacts to differing degrees of noise in the different inputs, which is more complex to evaluate if the values must be shifted in a discrete space. As is the case for the plots of predicted and actual values in Regression #2, where the actual values also have "steps", the patterns observed in the continuous distorted values originate from the sorting of the lists prior to plotting. That is, the orange curve has its lowest value for a step at the beginning of the step and its highest at the end.

The sensitivity analysis is performed by evaluating the performance of the trained network on the distorted validation sets for all variations. The results of this analysis are presented in Figure 72. In these plots, it is immediately clear that the network is most sensitive to changes to the contact area. From around 90% on the original dataset, the validation accuracy drops more or less steadily down to around 10% for 100% distorted variation. For the thickness and material, the drop between the original dataset and 100% distortion is smaller at around 20% and 10%, respectively. This suggests that the network finds the contact area as the most important influence to the number of welding points, and hence their strongest dependency. Still, however, it is observed that there is a definite dependency on both the thickness and material as well. It should additionally be noted that the results for the thickness and material could be influenced by the discrete nature of their original values. That is, a higher degree of distortion is required to shift the distorted values so that they are closer to another "step" than the original. As such, it is natural that the network is somewhat more robust to smaller distortions of these inputs.

The sensitivity analysis reveals that the network behaves as expected when increasing amounts of noise are included in its inputs. That is, more noise returns poorer results. Additionally, it shows that all the inputs, to some extent, are considered together by the network to perform the evaluation to determine the number of welding points. This supports the initial assumption regarding the pivotal dependencies of this cost driver.

#### **6.6.4 Summary and future directions**

As presented in Table 12, the final network configuration achieves a validation accuracy of 87.93%, with a test accuracy of 93.67%. The confusion matrix in Figure 69 details the validation results and reveals that the network makes good predictions, also on higher numbers of welding points.



**Figure 72** – Results from the sensitivity analysis in Regression #1.

In general, these results are considered to be sufficient to function as a proof-of-concept for the employed method to determine the number of welding points. However, due to the limited data that is available for both training and validation in this development, there is too much uncertainty to support an argument that these results constitute a solid proof-of-concept.

The results that are obtained in the development of this module are nevertheless considered positive. The network is observed to generally behave as expected, and the sensitivity analysis suggests that the assumptions regarding the dependencies of the cost driver are correct. As such, future works are encouraged to further pursue the presented approach. The primary focus could be data acquisition, in order to build a database with sufficient training and validation examples. Furthermore, extensions of the optimization loop can be made to perform more refined searches on more hyperparameters.

## **6.7 Classification #5 - Parts per stroke**

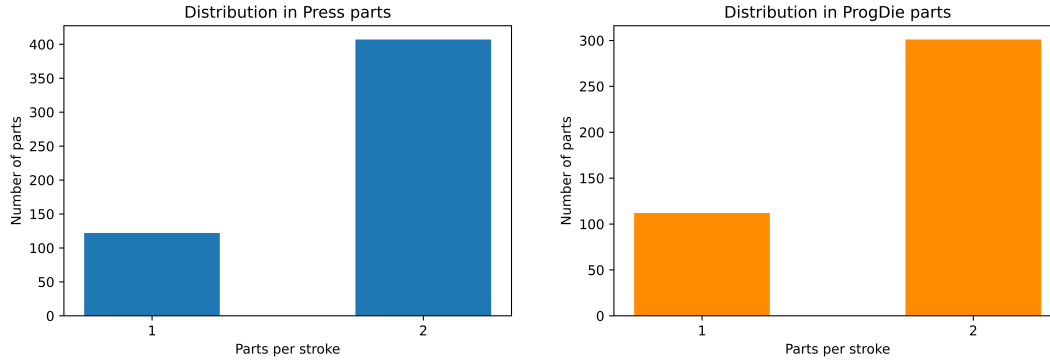
In Classification #5, the aim is to determine the number of parts that are manufactured per stroke in a press. This cost driver depends on the production method, which means the module succeeds Classification #3. It is assumed in this work that the Press parts can be considered as one, which means that two separate networks are developed for this module, one that evaluates Press parts and one that evaluates ProgDie parts.

The number of parts per stroke is assumed to additionally depend on the existence of sibling components and plant logistics, as discussed in Section 4.2.9. This data is, however, unfortunately not available in this work. Still, it is explored if some clear patterns exist amongst parts for which the same number of parts are manufactured per stroke, based solely on the geometry and size.

### **6.7.1 Data**

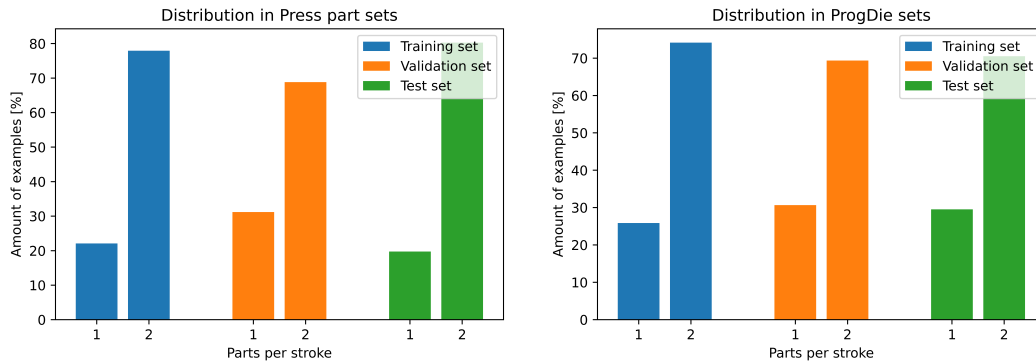
Cost engineers suggest that 1 and 2 parts per stroke are the most common choices, and therefore that these are the relevant cases to this work. The problem in Classification #5 is hence a categorical one with two target classes, namely "1 part per stroke" and "2 parts per stroke".

The distributions of the different numbers of parts per stroke for the Press parts and ProgDie parts are presented in Figure 73. These plots are assumed representative of the general distributions encountered in real applications. It can hence be observed that there is a clear tendency that 2



**Figure 73** – Distribution of the number of parts per stroke for the Press parts (left) and the ProgDie parts (right) in Classification #5.

parts per stroke is the most common case for both the Press parts to the left and the ProgDie parts to the right. In fact, if always selecting 2 as the answer, it would achieve an accuracy of 76.94% for the Press parts and 72.88% for the ProgDie parts. As such, 2 appears to be a good default value for the number of parts per stroke in the general case.



**Figure 74** – Distribution of the number of parts per stroke in the Press part sets (left) and ProgDie sets (right) in Classification #5.

The dataset employed in this module is the same as the one employed for the production method, provided information exists on the number of parts per stroke for the various parts. The distribution in each set is presented in Figure 74. The columns indicate the percentage amount of examples of 1 and 2 parts per stroke in each set. In the Press part sets, it can be observed that there is a slightly more even distribution in the validation set, which is represented by the orange columns in the left plot, than in each of the training and test sets. In the ProgDie sets, presented to the right, the distribution within the validation and test sets are almost identical, whereas a slightly larger

imbalance exists in the training set in favor of 2 parts per stroke. As in previous modules, the sets are split such that approximately 70% of the data is in the training set, and 15% is in each of the validation and test sets.

### 6.7.2 Testing approach

Unfortunately, very little time is available for the development of this module. Because of this, the testing approach that is employed takes advantage of several experiences from previous modules. The approach employed to develop a network for Classification #5 is as follows:

1. Rough tuning of spatial VGG-16 with default dropout and weighted training
  - (a) Testing learning rates: [1e-3, 1e-4, 1e-5]
2. Validate multi-view model
3. Fine-tuning

In Stage 1, the spatial VGG-16 with dropout, which has performed well for multiple previous modules on the same data, is tested with different learning rates. Fewer learning rates are tested than in the general approach in order to reduce the development efforts. Previous developments show that the relevant network typically performs poorly for the higher learning rates, and it is hence assumed that these can be excluded without an impact on the final results.

Having determined a learning rate to proceed with, the model is extended to multi-view. The best performing model after this stage is finally subjected to fine-tuning to achieve the final results.

The development is performed on the Press parts dataset because this is larger than the ProgDie one. The final configuration is subsequently trained on the ProgDie dataset to prepare a model to evaluate also these parts. It is assumed that the problem is sufficiently similar for both production part types to employ the same network configuration. All development will employ the 16x augmented dataset that is used for the blank size estimation in Regression #2, since this proved efficient for that problem.

### 6.7.3 Results and discussion

Table 14 presents the characteristics of the final network developed for Classification #5. The final Press part network achieves a weighted validation accuracy of 78.87%, with a weighted test accuracy of 84.62%. For the ProgDie parts, a weighted validation accuracy of 85.05% is achieved, supported by a weighted test accuracy of 84.23%.

Classification #5		
	Press parts	ProgDie
Target	Parts per stroke	Parts per stroke
Weighted validation accuracy	78.87%	84.62%
Weighted test accuracy	85.05%	84.23%
Convolutional architecture	VGG-16	VGG-16
Learning rate	1e-5	1e-5
Fine-tuning learning rate	1e-6	1e-7
Imbalance technique	Weighted losses	Weighted losses
Dropout	50% on both FC4096	50% on both FC4096
Spatial input	Global bounding box	Global bounding box
Multi-view fusion technique	Product-score	Product-score

**Table 14** – The final configuration of the networks in Classification #5.

#### Detailed results

Figure 75 presents the validation and test data confusion matrices for the final networks for both production part types. In all plots, it can be observed that the accuracy on the "2 parts per stroke" class is higher than the one on "1 part per stroke". This is assumed to be related to the imbalance in the dataset, which perhaps is somewhat insufficiently managed by weighting the losses in this case. In previous developments, such larger imbalances in the results have been observed in certain cases for configurations that are not optimal. Generally, the imbalance is observed to even out once an optimum is approached. The difference in accuracy between the classes is observed largest for the Press parts network on the validation data to the upper left. This could potentially be a result of the slightly different distribution that is observed in the validation set for the Press parts in Figure 74, compared to the training and test sets. Since the network appears to prefer predicting the "2 parts per stroke" class and the validation set in this case has more examples of the "1 part

per stroke” class relative to the other sets, it is likely to make more errors. That is, there are more parts that are likely to be predicted wrong. Although this perhaps intuitively should not affect the accuracy, since the accuracy is a relative measurement, there is the factor that more parts are available on which errors are likely to be made. This can hence potentially result in a lower individual accuracy on this class.



**Figure 75** – Confusion matrices of the final networks developed for Classification #5.

In general, observation of the results in Figure 75 indicates that the model is able to find some general geometrical and spatial patterns to separate the parts for which the same number of parts are manufactured per stroke. Still, however, both the imbalance between the accuracy on the

different classes and the observed performance suggest that these patterns are not sufficient to make a robust classification. As such, the initial assumption that there are additional dependencies of high importance to this cost driver could be supported by the results.

#### **6.7.4 Summary and future directions**

The results presented in Table 14 and Figure 75 are considered to be promising, but not to function as a proof-of-concept for the employed method to determine the number of parts per stroke. General geometrical and spatial patterns appear to exist and are identified by the model for each class. However, it is assumed that additional dependencies must be evaluated in order to extend the module to achieve a performance that is sufficient for a finished system.

Future works can attempt the inclusion of both sibling component information and plant logistics information to further improve the module performance. This could potentially be done either by rule-based heuristics or extensions of the proposed learning-based method. Sibling information could, for example, potentially be included as a simple Boolean parameter, indicating whether a sibling exists or not for the relevant component, that is concatenated with the spatial information.

### **6.8 Prototype**

The developed modules are assembled into a prototype system. This simplified system is intended as a final proof-of-concept for the complete theorized system detailed in Section 5.1.2. The prototype is implemented with Python.

The developed prototype system is depicted in Figure 76. As opposed to the system architecture in Figure 23, this graph shows the flow of data over time in the implemented prototype. In order to simplify the system, all parts are evaluated by all included modules in a linear fashion. The final output is then filtered by the identified part type and production part type to generate a profile for the specific part, including only the relevant evaluations. A sheet metal part is, for example, evaluated by Classification #2 to determine its "fastener ID". This information is subsequently removed from the output by crosschecking the part type.

The prototype requires that the part to be analyzed is placed in a folder named "CAD" in a CAD native format. As previously mentioned, this work utilizes Catia V5 as the CAD software. From this folder, the Deconstructor retrieves the component, deconstructs it, and saves each identified



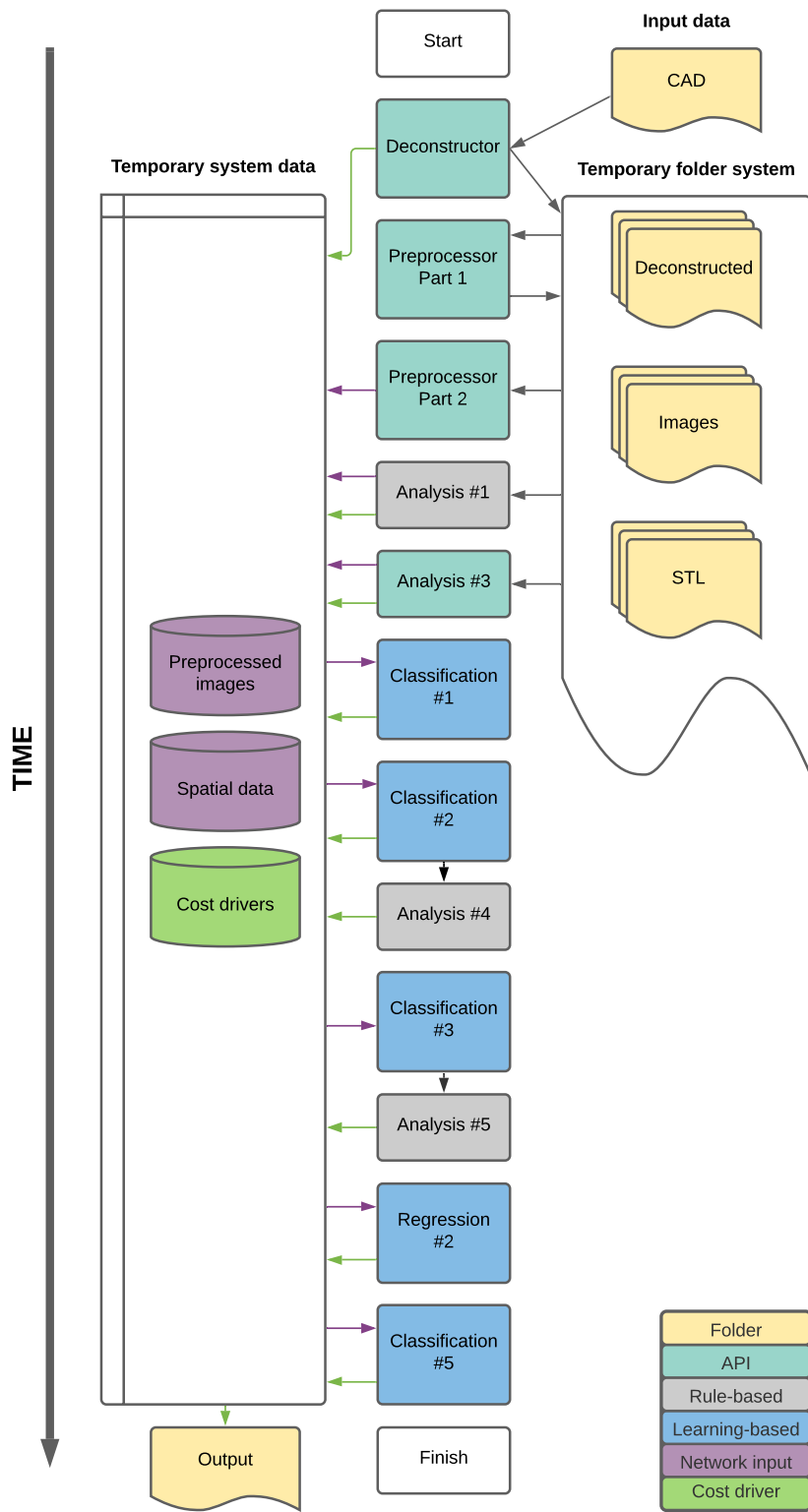


Figure 76 – Prototype system.

individual solid body to the folder "Deconstructed" in the temporary folder system. This temporary folder system is used by the prototype to store and retrieve information from the permanent computer memory throughout the analysis, and it can subsequently be wiped once the analysis is finished. The flow of file data, such as CAD files and image files, is denoted in Figure 76 by grey arrows. The Deconstructor also identifies the material information associated with each solid, as stored in the CAD native format, and keeps this in the temporary system data, where it can be accessed by other modules and finally output from the system. The flow of cost driver data is denoted by green arrows.

The preprocessor is split into two parts in Figure 76 to enhance the readability of the graph. Part 1 illustrates the part that retrieves the CAD representation of each solid body from the "Deconstructed" folder, images them, and saves the images to the temporary folder "Images". Additionally, the STL part representation of each solid is exported to the "STL" folder. Part 2 of the Preprocessor illustrates the part that reads each of the image files stored in "Images" and prepares them to later be input into the learning-based modules. Network inputs are denoted by purple lines. The preprocessed image tensors are stored in the temporary system memory, and they can subsequently be retrieved by all latter parts of the system.

Analysis #1 retrieves the STL files from the "STL" folder and performs the necessary analysis to extract both the spatial data needed as input to the networks and the "Part size" cost driver. These are all stored in the temporary system memory.

A complete material index is not available at the time of implementation, and Analysis #2 is therefore left out of the prototype. That is, the system outputs the volume and material type, but not the part mass. Inclusion of Analysis #2 is arbitrary when an index relating all relevant materials and their densities is available.

Analysis #3 completes the initial preprocessing and rule-based analysis by evaluating the CAD input part to extract the thicknesses and contact areas. After this, the subsequent modules all retrieve their inputs from the temporary system memory and return their respective cost drivers.

There are a few changes to the latter parts of the system when compared to the complete proposed architecture in Figure 23. For one thing, Analysis #5 is included to separate the Press parts into Transfer and Press line parts. As discussed in Section 6.2, the separation is in this work achieved

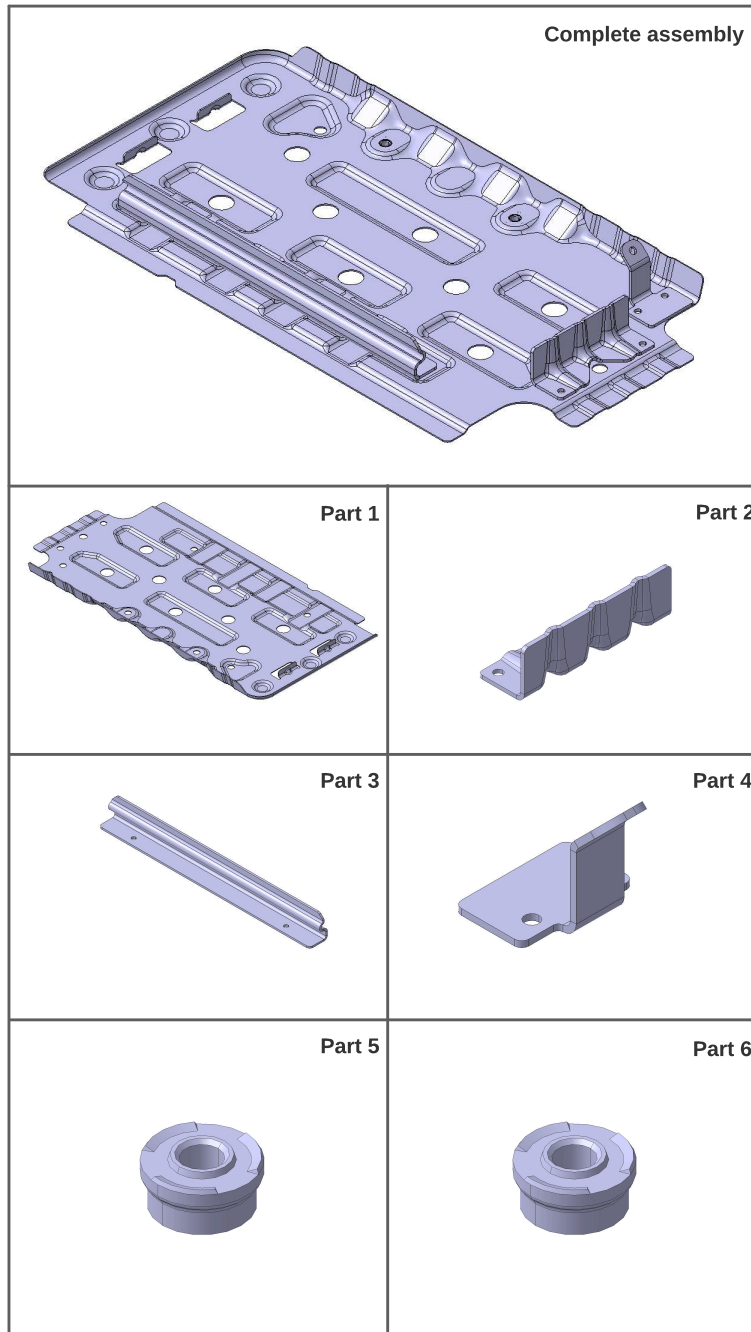
by simply considering a spatial threshold value related to the minimum bounding box diagonal of the part identified as a Press part. Another change to the system is that it is not prioritized to implement the Regression #1 module that estimates the number of welding points. This is because the implementation effort of this module is higher relative to the others due to additional required logic relating to the identification of relevant contact areas. Classification #4 is not included both because of its insufficient performance and the small impact the number of operations has on the production costs.

### 6.8.1 Case study

To demonstrate the functionality of the prototype system, a small assembly is evaluated as a case study. The assembly consists of a total of 6 parts, four of which are sheet metal and two fasteners. These six parts are depicted in Figure 77, together with the complete assembly on the top. This is the same part included as an example of a small assembly in Figure 2 in Section 2.1. As can be observed in Figure 77, the assembly consists of one main part, Part 1, on which three brackets are mounted, namely Parts 2-4. Additionally, two identical weld nuts, Parts 5 and 6, are mounted on the underside of the main part.

The results from the prototype system evaluation of the case study part are presented together with the actual target values in Table 15. As can be observed, the system accurately predicts the part type of all parts in the assembly. The material, thickness, volume, and part size are also predicted to satisfaction. The latter 2 show some deviations from the actual values, which partially relate to the approximation done when converting the CAD native geometry into the STL format. For increased accuracy, a higher STL "resolution" can be chosen, but this would, in turn, somewhat impact the efficiency of the evaluation. The current Catia V5 settings are a fixed "3D accuracy" of 0.2 and a "Curves' accuracy ratio" of 0.5.

Note that the "actual" part size is calculated with a tool that is used by development engineers today to determine the minimum bounding box of a part. This tool does, in fact, also perform a mere estimation of the minimum bounding box, which means that the "actual" part size simply is another estimation than the one made by the automated system. For most parts, the values are observed to generally correlate well, although some larger deviation is observed for Part 4. The minimum bounding box identified by the automated system is, in this case, smaller than the one estimated by the existing tool.



**Figure 77** – The part that is included as a case study.

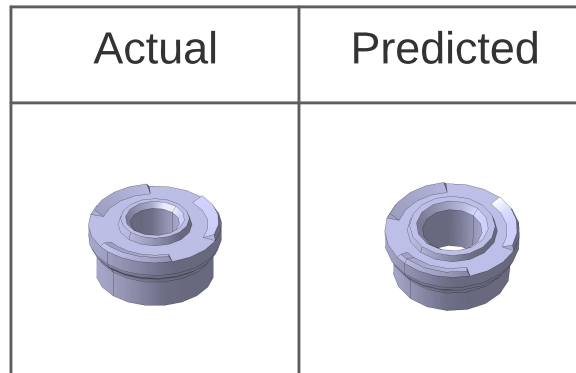
	Part 1		Part 2		Part 3		Part 4	
	Actual	Predicted	Actual	Predicted	Actual	Predicted	Actual	Predicted
Part type	Sheet metal	Sheet metal	Sheet metal	Sheet metal	Sheet metal	Sheet metal	Sheet metal	Sheet metal
Material	CR210LA	CR210LA	CR2404BH	CR2404BH	CR2404BH	CR2404BH	CR2404BH	CR2404BH
Thickness	1.2mm	1.2mm	2.5mm	2.5mm	2.5mm	2.5mm	2.5mm	2.5mm
Volume	164588.62mm <sup>3</sup>	164004.61mm <sup>3</sup>	52155.64mm <sup>3</sup>	51901.00mm <sup>3</sup>	14643.60mm <sup>3</sup>	14562.57mm <sup>3</sup>	5737.84mm <sup>3</sup>	5737.99mm <sup>3</sup>
Size	[516.61mm, 270.19mm, 26.56mm]	[516.61mm, 270.88mm, 25.83mm]	[320.00mm, 34.45mm, 26.95mm]	[320.00mm, 41.23mm, 22.10mm]	[103mm, 33.37mm, 32.99mm]	[103.00mm, 33.36mm, 32.99mm]	[48.88mm, 47.51mm, 46.72mm]	[66.94mm, 46.72mm, 24.44mm]
Production method	ProgDie	ProgDie	ProgDie	ProgDie	ProgDie	ProgDie	ProgDie	ProgDie
Blank size	168720mm <sup>2</sup>	138148.02mm <sup>2</sup>	25848mm <sup>2</sup>	51941.02mm <sup>2</sup>	18493.51mm <sup>2</sup>	11506.61mm <sup>2</sup>	5191.67mm <sup>2</sup>	5760.63mm <sup>2</sup>
Parts per stroke	1	1	1	1	2	2	1	2
	Part 5		Part 6					
	Actual	Predicted	Actual	Predicted				
Part type	Fastener	Fastener	Fastener	Fastener				
Volume	1492.64mm <sup>3</sup>	1475.67mm <sup>3</sup>	1492.64mm <sup>3</sup>	1475.67mm <sup>3</sup>				
Size	[18.00mm, 18.00mm, 8.80mm]	[18.00mm, 17.86mm, 8.80mm]	[18.00mm, 18.00mm, 8.80mm]	[18.00mm, 17.86mm, 8.80mm]				
Fastener ID	1938957	1129428	1938957	1129428				
Fastener type	Weld nut	Weld nut	Weld nut	Weld nut				

**Table 15** – Case study results.

The production methods of all sheet metal parts are correctly predicted as ProgDie. However, a mistake is made when determining the number of parts per stroke for Part 4. This is expected since the Classification #5 module, as discussed in Section 6.7, is considered to require further development to achieve satisfactory performance. The estimated blank sizes are also observed to deviate to some differing extents from the actual values. Whereas the estimations for Parts 1 and 4 are fairly reasonable, the deviations are rather large for Parts 2 and 3. This is in line with the expectations, based on the discussions in Section 6.5.

The fasteners are identified as identical and correctly found to be weld nuts. However, the actual fastener ID is incorrectly identified. Upon comparison of the predicted and actual parts adhering to these fastener IDs, presented in Figure 78, a high degree of similarity between the two is observed. In fact, upon closer investigation of the data, the only difference between them is found to be the size of the bore through their center. In turn, this suggests that the wrong prediction is of minor importance to the costs. Still, it demonstrates that more tuning of the network perhaps should be performed to potentially eliminate such errors.

Although the case study reveals certain areas of improvement, it is considered to function as a



**Figure 78** – The actual and predicted fastener in case study.

proof-of-concept for the theorized system for automatic part evaluation. In just around 2 minutes, without the need for any expert knowledge or inputs, the developed rule- and learning-based modules are able to automatically evaluate the input 3D CAD sheet metal component to identify key cost drivers.

## 6.9 Summary

Table 16 summarizes the developed learning-based modules in this work. Note that individual summaries for each also is provided in their respective development sections. All in all, proof-of-concepts or promising results are obtained for most modules. Above 90% accuracy is achieved in Classifications #1, #2, and #3. For the blank size prediction in Regression #2, the results are promising enough to be considered as a proof-of-concept, although more work is required in order to achieve a system-ready performance. For the welding points prediction in Regression #1 and parts-per-stroke prediction in Classification #5, the results are also considered promising. However, a lack of data in both developments does not allow for them to be considered as full proof-of-concepts for the employed method. Data is also assumed to be a contributor to the insufficient results achieved in Classification #4, although they could suggest that the problem at hand is too complex to be solved without a redefinition. Note that the "N.A." for Regression #1 is an abbreviation for "not applicable", which is necessary since this module has fundamental differences from the others.

There are several commonalities amongst the different modules in Table 2. For one thing, the VGG is employed in all modules except for Classification #4. Considering the ImageNet performances

Summary					
	Classification #1	Classification #2	Classification #3	Classification #4	
Target	Part type	Fastener ID	Production part type	Number of operations	
Key performance indicator	Weighted accuracy	Accuracy	Weighted accuracy	Weighted accuracy	
Validation performance	99.57%	96.81%	90.78%	62.72%	
Test performance	100%	96.90%	91.45%	48.16%	
Sufficient proof-of-concept	Yes	Yes	Yes	No	
Base architecture	VGG-16	VGG-16	VGG-16	Inception-ResNet-v2	
Learning rate	1e-4	1e-4	1e-5	1e-3	
Fine-tuning learning rate	1e-5	1e-4	1e-6	1e-5	
Imbalance technique	Weighted losses	None	Weighted losses	Weighted losses	
Dropout	Yes	Yes	Yes	Yes	
Spatial input	None	Global BB	Global BB	None	
Multi-view fusion technique	Product-score	Product-score	Product-score	Product-score	
	Classification #5		Regression #1	Regression #2	
	Press parts	ProgDie		Press parts	ProgDie
Target	PPS	PPS	#Welding points	Blank area	Blank area
Key performance indicator	Weighted accuracy	Weighted accuracy	Accuracy	MAPE	MAPE
Validation performance	78.87%	84.62%	87.93%	22.79%	18.66%
Test performance	85.05%	84.23%	93.67%	18.14%	22.57%
Sufficient proof-of-concept	No	No	No	Yes	Yes
Base architecture	VGG-16	VGG-16	FC5	VGG-16	VGG-16
Learning rate	1e-5	1e-5	1e-2	1e-3	1e-3
Fine-tuning learning rate	1e-6	1e-7	N.A.	1e-5	1e-5
Imbalance technique	Weighted losses	Weighted losses	None	None	None
Dropout	Yes	Yes	No	Yes	Yes
Spatial input	Global BB	Global BB	N.A.	Global BB	Global BB
Multi-view fusion technique	Product-score	Product-score	N.A.	Score (median)	Score (median)

**Table 16** – Summary of the developed learning-based modules.

presented in Figure 27 in Section 5.3.2, this is somewhat surprising since both of the other architectures that are tested in this work achieve higher performances on this benchmark dataset. However, the VGG performed the best in the comparison tests made in this work, both on the fastener dataset in Classification #2 and the sheet metal dataset in Classification #3. The VGG-16 and VGG-19 have inseparable performance on this data, and the VGG-16 is therefore used since it is more efficient to train and because it generally is considered preferable to employ simpler architectures where possible [46]. Still, it is not the intention of this work to claim that the VGG-16, or the VGG in general, is superior to the other architectures tested, nor other existing architectures for that matter. The aim of the development of each module, and hence also the employed testing approach, is to produce results that suffice as proof-of-concepts for the proposed identification methods to be implemented into the theorized system. As such, the results reflect that the VGG is found to be the best architecture for this purpose. However, that is not to say that the other architectures could not potentially achieve stronger final results than the VGG under different testing conditions. The Inception-ResNet-v2, for example, is observed to achieve superior results when fine-tuning is performed in Stage 1 in Classification #4. Although these results could be influenced by other factors, as discussed in Section 6.4, they might also suggest that the Inception-ResNet-v2 would have shown superior performance in other modules as well if fine-tuning was performed prior to the selection of a convolutional architecture. Still, however, fine-tuning throughout all steps in the development demands a much higher effort and is not prioritized in this work since sufficient results are achieved by the VGG architecture without fine-tuning. This could perhaps be related to the higher number of trainable weights in the prediction block of the VGG compared to the others, although it also could be related to the general patterns found through pre-training on the ImageNet.

Fine-tuning in itself is generally found to improve the results, especially on the fastener dataset in Classification #2. This could suggest that the ImageNet is not the most suitable pre-training dataset for this problem. For the sheet metal parts, improvement is also generally observed, although this improvement is much more subtle at around a couple percent, suggesting perhaps that the ImageNet is a more suitable pre-training dataset for this data.

The inclusion of spatial information is generally observed to improve the results when included. In all cases where the spatial model is used, the global bounding box is included as the spatial argument. The reason for choosing the global bounding box is, as previously discussed, for one



thing, that its 3 measurements are assumed to provide more spatial information than arguments consisting of only a single number, such as the area or volume. Secondly, it is considered potentially useful that the global bounding box is directly related to the different views. Lastly, a reason for choosing the global bounding box is that it is affected by rotation. This may be considered a negative trait, and, in many cases, it is, but because of the augmentation performed through rotation, it is considered positive that the bounding boxes also change between the augmented examples. In cases where sufficient data is available without rotational augmentation, the better choice could potentially be the minimum bounding box, especially if score fusion is implemented. Since the score-fusion multi-view models perform a "stupid" evaluation of the results on each view, it is not assumed that these models benefit from the spatial argument being directly correlated with the view. As such, the minimum bounding box, which is considered a better spatial descriptor in itself exactly because of its rotational invariance, could be the better choice.

The score multi-view fusion techniques proved the strongest in all image recognition modules. Specifically, the product-score fusion models performed the best in all classification modules. The reason why the product-score fusion outperforms the democratic-score fusion in these modules is, as discussed in Section 6.1, assumed to be that the product-score evaluation also translates the confidence of the predictions from each view into the final results. The reason why both score fusion techniques generally are observed to outperform the late fully-connected fusion technique, is assumed to be related to insufficient amounts of data to train the models employing late FC fusion. Since the score fusion models can be trained to completion in single-view, this effectively means that they have 7 times more training data available compared to the late fully-connected models, as also discussed in Section 6.1. It should be mentioned that more work could have been invested into reducing this handicap for the late FC models, for example by attempting to transfer more tuned weights from the single-view models into the late FC models before training in multi-view. It could also have been experimented with different numbers of neurons in the fusion layer, as well as increasing the sizes of the augmented datasets. However, these measures are not prioritized in this work since sufficiently strong results are achieved by the score fusion models. Still, as also pointed out for the convolutional architectures, the presented results should not be read to suggest that score fusion is a superior fusion technique to late FC fusion, but that for the purposes of this work, product-score fusion proved the most efficient. In fact, it is assumed that the late FC fusion technique would outperform the score fusion techniques if provided with sufficient training data.

This is because of the patterns that potentially can be found in between the different views with this technique due to the more intelligent fusion performed. Additionally, this technique could perhaps gain from the direct relation between the views and the dimensions of the global bounding box.

## 7 Conclusion and future work

In this work, a system for automatic evaluation of 3D part models of sheet metal components to identify key cost drivers is proposed and developed.

First, cost drivers that are important to estimate the costs of sheet metal components are outlined and evaluated. Based on the evaluations, methods to pursue for automatic identification of the cost drivers relevant to the production costs are established and developed. Following the method clarifications, an architecture for the automatic evaluation system is proposed. The proposed architecture accounts for the dependencies of the cost drivers and the specific methods intended to identify them, outlining the flow of data in different formats through the system. The system modules rely on an API, rule-based analysis, and learning-based analysis to describe the input sheet metal component in terms of its cost drivers. The subsequent parts of this work aim to develop proof-of-concepts for all the system modules to finally provide a proof-of-concept for the theorized system. The API and rule-based modules are directly implemented, whereas the work to develop the neural networks in the learning-based modules is performed with an iterative approach that tests different techniques and concepts in order to build the final model.

The development shows promising results for most system modules. Above 90% accuracy is achieved on separating fasteners and sheet metal parts, determining the specific ID of a fastener amongst 820 possibilities, and identifying the production part type of the individual sheet metal parts in the input component. Promising results are also achieved when estimating the blank size of the sheet metal parts, predicting the number of welding points needed to join the sheets, and determining the number of parts that are simultaneously manufactured together with the relevant part. However, some further work is required before these modules achieve a performance sufficient for a finished system.

The successfully developed modules are assembled into a prototype system. This prototype is intended as a final proof-of-concept for the proposed system, and a case study is included to demonstrate its functionality. In about 2 minutes, without needing any expert input or handling, the developed rule- and learning-based modules in the prototype automatically evaluate the input component to identify key cost drivers. The results from the case study are solid but still also highlight the need for certain future efforts to finalize the system, in line with the expectations based on the individual module performances.

Propositions for future works on the specific learning-based modules are made for each one in the sections presenting their development. For the system as a whole, the most relevant future direction is considered to be related to extending the proposed system to one that also evaluates tooling cost drivers. This would entail developing and implementing a module that automatically establishes a process plan for the input component. Such a module could either be developed entirely from scratch or aim to incorporate a commercial software solution. Future works can also seek to evaluate the input component in terms of assembly cost drivers, enabling analysis of larger assemblies.

Some future efforts are necessary to increase the performance of individual modules before the proposed system can be readily implemented in the industry. However, the presented work is considered to constitute a proof-of-concept for a system that automatically evaluates a 3D part model of a sheet metal component to identify key production cost drivers.

## 8 References

- [1] V. Naranje and S. Kumar, "AI Applications to Metal Stamping Die Design", World Academy of Science, Engineering, and Technology, vol. 68, 2010, pp. 1516-1522
- [2] S. Karadgi, U. Müller, D. Metz, W. Schäfer, and M. Grauer, "Cost estimation of automotive sheet metal components using knowledge-based engineering and case-based reasoning", 2009 IEEE International Conference on Industrial Engineering and Engineering Management, 2009, pp. 1518-1522, doi: 10.1109/IEEM.2009.5373084.
- [3] B. Verlinden, J. R. Duflou, P. Collin, and D. Cattrysse, "Cost estimation for sheet metal parts using multiple regression and artificial neural networks: A case study", International Journal of Production Economics, vol. 111, issue 2, 2008, pp. 484-492, doi: 10.1016/j.ijpe.2007.02.004
- [4] S. Kashid and S. Kumar, " Applications of Artificial Neural Network to Sheet Metal Work - A Review", American Journal of intelligent systems, vol. 2, no. 7, 2012, pp. 168-176, doi: 10.5923/j.ajis.20120207.03
- [5] R. Roy, P. Souchoroukov, and E. Shehab, "Detailed cost estimating in the automotive industry: Data and information requirements", International Journal of Production Economics, vol. 133, 2011, pp. 694-707, doi: 10.1016/j.ijpe.2011.05.018
- [6] Dunbing Tang, Walter Eversheim, and Günther Schuh, "Qualitative and quantitative cost analysis for sheet metal stamping", International Journal of Computer Integrated Manufacturing, vol. 17, no. 5, 2004, pp. 394-412, doi: 10.1080/09511920410001662120
- [7] M. Geiger, J. Knoblach, and F. Backes, "Cost estimation for large scale production of sheet metal parts using artificial neural networks", Production Engineering, vol. 2, 1998, pp. 81-84
- [8] F. Ning, Y. Shi, M. Cai, W. Xu, and X. Zhang, "Manufacturing cost estimation based on the machining process and deep-learning method", Journal of Manufacturing Systems, vol. 56, 2020, pp. 11-22, doi: 10.1016/j.jmsy.2020.04.011
- [9] F. Ning, Y. Shi, M. Cai, W. Xu, and X. Zhang, "Manufacturing cost estimation based on a deep-learning method", Journal of Manufacturing Systems, vol. 54, 2020, pp. 186-195, doi: 10.1016/j.jmsy.2019.12.005

- [10] S. Yoo and N. Kang, "Explainable artificial intelligence for manufacturing cost estimation and machining feature visualization", *Expert Systems with Applications*, vol. 183, 2021, doi: 10.1016/j.eswa.2021.115430
- [11] W. Greska, V. Franke, and M. Geiger, "Classification problems in manufacturing of sheet metal parts", *Computers in Industry*, vol. 33, issue 1, 1997, pp. 17-30, doi: 10.1016/S0166-3615(97)00008-0
- [12] K. V. Ramana and P. V. M. Rao, "Automated manufacturability evaluation system for sheet metal components in mass production", *International Journal of Production Research*, vol. 43, no. 18, 2005, pp. 3889-3913, doi: 10.1080/00207540500066853
- [13] T. R. Kannan and M. S. Shunmugam, "Processing of 3D sheet metal components in STEP AP-203 format. Part I: feature recognition system", *International Journal of Production Research*, vol. 47, no. 4, 2009, pp. 941-964, doi: 10.1080/00207540701510055
- [14] R. K. Gupta and B. Gurumoorthy, "Classification, representation, and automatic extraction of deformation features in sheet metal parts", *Computer-Aided Design*, vol. 45, issue 11, 2013, pp. 1469-1484, doi: 10.1016/j.cad.2013.06.010
- [15] V. B. Sunil and S. S. Pande, "Automatic recognition of features from freeform surface CAD models", *Computer-Aided Design*, 2008, vol. 40, no. 4, pp. 502-517, doi: 10.1016/j.cad.2008.01.006
- [16] J. E. Makem, H. J. Fogg, and N. Mukherjee, "Automatic Feature Recognition Using the Medial Axis for Structured Meshing of Automotive Body Panels", *Computer-Aided Design*, vol. 124, doi: 10.1016/j.cad.2020.102845
- [17] V. Cicirello and W. C. Regli, "Machining feature-based comparisons of mechanical parts", *Proceedings International Conference on Shape Modeling and Applications*, 2001, pp. 176-185, doi: 10.1109/SMA.2001.923388
- [18] S. Ghaffarishahri and L. Rivest, "Feature-Based Model Difference Identification for Aerospace Sheet Metal Parts", *Computer-Aided Design and Applications*, vol. 18, no. 3, 2020, pp. 443-467, doi: 10.14733/cadaps.2021.443-467

- [19] B. Babic, N. Nesic, and Z. Miljkovic, "A review of automated feature recognition with rule-based pattern recognition", *Computers in Industry*, vol. 59, issue 4, 2008, pp. 321-337, doi: 10.1016/j.compind.2007.09.001
- [20] M.A. Ablat and A. Qattawi, "Numerical simulation of sheet metal forming: a review", *The International Journal of Advanced Manufacturing Technology*, vol. 89, 2017, pp. 1235-1250, doi: 10.1007/s00170-016-9103-5
- [21] I. El Mrabti, A. Touache, A. El Hakimi, and A. Chamat, "Springback optimization of deep drawing process based on FEM-ANN-PSO strategy", *Structural and Multidisciplinary Optimization*, vol. 64, 2021, pp. 321-333, doi: 10.1007/s00158-021-02861-y
- [22] J. Lan, X. Dong, Z. Li, "Inverse finite element approach and its application in sheet metal forming", *Journal of Materials Processing Technology*, vol. 170, issue 3, 2005, pp. 624-631, doi: 10.1016/j.jmatprotec.2005.06.043
- [23] A. Lobov and T. A. Tran, "Object-oriented approach to product design using extended NX Open API", *Procedia Manufacturing*, Vol. 51, 2020, pp. 1014-1020, doi: 10.1016/j.promfg.2020.10.142
- [24] V. A. Tikhomirov, "The Method of Automatic Determination of the Types of Spatial Angles in 3D Models of CAD Systems", 2019 International Science and Technology Conference "EastConf", 2019, pp. 1-4, doi: 10.1109/EastConf.2019.8725312
- [25] R. Hambli and F. Guerin, "Application of a neural network for optimum clearance prediction in sheet metal blanking processes", *Finite Elements in Analysis and Design*, vol. 39, issue 11, pp. 1039-1052, doi: 10.1016/S0168-874X(02)00155-5
- [26] S. Kumar and H. M. A. Hussein, "AI Applications in Sheet Metal Forming", Springer, 2017, doi: 10.1007/978-981-10-2251-7
- [27] M.R. Jamli and N.M. Farid, "The sustainability of neural network applications within finite element analysis in sheet metal forming: A review", *Measurement*, vol. 138, 2019, pp. 446-460, doi: 10.1016/j.measurement.2019.02.034
- [28] R. Kazan, M. Fırat, and A. E. Tiryaki, "Prediction of springback in wipe-bending process of

- sheet metal using neural network”, *Materials & Design*, vol. 30, issue 2, 2009, pp. 418-423, doi: 10.1016/j.matdes.2008.05.033
- [29] W. Liu, Q. Liu, F. Ruan, Z. Liang, and H. Qiu, ”Springback prediction for sheet metal forming based on GA-ANN technology”, *Journal of Materials Processing Technology*, vols. 187-188, 2007, pp. 227-231, doi: 10.1016/j.jmatprotec.2006.11.087
- [30] J. Wang, X. Wu, P. F. Thomson, and A. Flitman, ”A neural networks approach to investigating the geometrical influence on wrinkling in sheet metal forming”, *Journal of Materials Processing Technology*, vol. 105, issue 3, 2000, pp. 215-220, doi: 10.1016/S0924-0136(00)00534-3
- [31] M. B. Gorji and D. Mohr, ”Towards neural network models for describing the large deformation behavior of sheet metal”, *IOP Conference Series: Materials Science and Engineering*, vol. 651, no. 1, 2019, pp. 012102, doi: 10.1088/1757-899X/651/1/012102
- [32] Z. Fu, J. Mo, L. Chen, and W. Chen, ”Using genetic algorithm-back propagation neural network prediction and finite-element model simulation to optimize the process of multiple-step incremental air-bending forming of sheet metal”, *Materials & Design*, vol. 31, issue 1, 2010, pp. 267-277, doi: 10.1016/j.matdes.2009.06.019
- [33] D. Opritescu and W. Volk, ”Automated driving for individualized sheet metal part production — A neural network approach”, *Robotics and Computer-Integrated Manufacturing*, vol. 35, 2015, pp. 144-150, doi: 10.1016/j.rcim.2015.03.006
- [34] C. Hartmann, D. Opritescu, and W. Volk, ”An artificial neural network approach for tool path generation in incremental sheet metal free-forming”, *Journal of Intelligent Manufacturing*, vol. 30, 2019, pp. 757–770, doi: 10.1007/s10845-016-1279-x
- [35] B. Babic, N. Nesic, and Z. Miljković, ”Automatic feature recognition using artificial neural networks to integrate design and manufacturing: Review of automatic feature recognition systems”, *Artificial Intelligence for Engineering Design Analysis and Manufacturing*, vol. 25, 2011, pp. 289-304, doi: 10.1017/S0890060410000545
- [36] Z. Zhang, P. Jaiswal, and R. Rai, ”FeatureNet: Machining feature recognition based on 3D Convolution Neural Network”, *Computer-Aided Design*, vol. 101, 2018, pp. 12-22, doi: 10.1016/j.cad.2018.03.006



- [37] P. Shi, Q. Qi, Y. Qin, P. J. Scott, and X. Jiang, "A novel learning-based feature recognition method using multiple sectional view representation", *Journal of intelligent manufacturing*, vol. 31, 2020, pp. 1291-1309, doi: 10.1007/s10845-020-01533-w
- [38] P. Shi, Q. Qi, Y. Qin, P. J. Scott, and X. Jiang, "Intersecting Machining Feature Localization and Recognition via Single Shot Multibox Detector", *IEEE Transactions on Industrial Informatics*, vol. 17, no. 5, 2021, pp. 3292-3302, doi: 10.1109/TII.2020.3030620
- [39] Q. Dong, A. Wu, N. Dong, W. Feng, and S. Wu, "A Convolution Neural Network for Parts Recognition Using Data Augmentation", 2018 13th World Congress on Intelligent Control and Automation (WCICA), 2018, pp. 773-777, doi: 10.1109/WCICA.2018.8630451
- [40] S. Noh, S. Back, R. Kang, S. Shin, and K. Lee, "Automatic Detection and Identification of Fasteners with Simple Visual Calibration using Synthetic Data", 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2020, pp. 38-44, doi: 10.1109/ETFA46521.2020.9212155
- [41] A. Börold, M. Teucke, J. Rust, and M. Freitag, "Recognition of car parts in automotive supply chains by combining synthetically generated training data with classical and deep learning based image processing", *Procedia CIRP*, vol. 93, 2020, pp. 377-382, doi: 10.1016/j.procir.2020.03.142
- [42] R. Sheu, Y. Lin, C. Huang, L. Chen, M. S. Pardeshi, and H. Tseng, "IDS-DLA: Sheet Metal Part Identification System for Process Automation Using Deep Learning Algorithms", in *IEEE Access*, vol. 8, pp. 127329-127342, 2020, doi: 10.1109/ACCESS.2020.3007257
- [43] M. Szilvsi-Nagy and Gy. Mátyási, "Analysis of STL files", *Mathematical and Computer Modelling*, vol. 38, 2003, issues 7-9, pp. 945-960, doi: 10.1016/S0895-7177(03)90079-3
- [44] I. Baturynska, "Statistical analysis of dimensional accuracy in additive manufacturing considering STL model properties", *The International Journal of Advanced Manufacturing Technology*, vol. 97, 2018, pp. 2835-2849, doi: 10.1007/s00170-018-2117-4
- [45] M. A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015, url: <http://www.neuralnetworksanddeeplearning.com>
- [46] M. T. Hagan, H. B. Demuth, M. H. Beale, and O. De Jesús, "Neural Network Design", 2nd

Edition, Martin Hagan, 2014

- [47] I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning", MIT Press, 2016, url: <http://www.deeplearningbook.org>
- [48] K. Hornik, M. Stinchcombe, H. White, "Multilayer feedforward networks are universal approximators", *Neural Networks*, vol. 2, issue 5, 1989, pp. 359-366, doi: 10.1016/0893-6080(89)90020-8
- [49] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors", arXiv e-prints, 2014, arXiv:1207.0580
- [50] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *Journal of Machine Learning Research*, vol. 15, no. 56, 2014, pp. 1929-1958, url: <http://jmlr.org/papers/v15/srivastava14a.html>
- [51] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, vol. 86, no. 11, 1998, pp. 2278-2324, doi: 10.1109/5.726791
- [52] A. Kahn, A. Sohail, U. Zahoor, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks", *Artificial Intelligence Review*, vol. 53, 2020, pp. 5455-5516, doi: 10.1007/s10462-020-09825-6
- [53] I. S. Mohamed, Ihab, "Detection and Tracking of Pallets using a Laser Rangefinder and Machine Learning Techniques", Master's thesis, University of Genova, Genova, Italy, 2017, doi: 10.13140/RG.2.2.30795.69926
- [54] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", arXiv e-prints, 2015, arXiv: 1409.1556
- [55] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", *Advances in Neural Information Processing Systems*, vol. 25, 2012, url: <https://proceedings.neurips.cc/paper/2012/file/>

- [56] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition", 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90
- [57] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning", Thirty-first AAAI conference on artificial intelligence, 2017, url: <https://www.aaai.org/ocs/index.php/AAAI/AAAI17/paper/viewPaper/14806>
- [58] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions", 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 1-9, doi: 10.1109/CVPR.2015.7298594
- [59] X. Liu, Z. Han, Y.-S. Liu, and M. Zwicker, "Point2Sequence: Learning the Shape Representation of 3D Point Clouds with an Attention-Based Sequence to Sequence Network", Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, no. 1, 2019
- [60] R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, and D. Cohen-Or, "MeshCNN: a network with an edge", ACM Transactions on Graphics, vol. 38, issue 4, 2019, pp. 1-12, doi: 10.1145/3306346.3322959
- [61] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view Convolutional Neural Networks for 3D Shape Recognition", Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2015, pp. 945-953, doi: 10.1109/ICCV.2015.114
- [62] Y.-P. Xiao, Y.-K. Lai, F.-L. Zhang, C. Li, and L. Gao, "A survey on deep geometry learning: From a representation perspective", Computational Visual Media, vol. 6, 2020, pp. 113-133, doi: 10.1007/s41095-020-0174-8
- [63] M. Seeland and P. Mäder, "Multi-view classification with convolutional neural networks", PLoS ONE, vol. 16, no. 1, 2021, doi: 10.1371/journal.pone.0245230
- [64] M. P. Groover, "Groover's Principles of Modern Manufacturing: Materials, Processes, and Systems, SI Version", 6th edition, Wiley, 2016

- [65] A.-B. Ryberg and L. Nilsson, "Spot weld reduction methods for automotive structures", *Structural and Multidisciplinary Optimization*, vol. 53, 2015, pp. 923-934, doi: 10.1007/s00158-015-1355-4
- [66] L. Yan, Q.-T. Guo, S. Yang, X.-W. Liao, and C. Qi, "A size optimization procedure for irregularly spaced spot weld design of automotive structures", *Thin-Walled Structures*, vol. 166, 2021, doi: 10.1016/j.tws.2021.108015
- [67] R.S. Florea, K.N. Solanki, D.J. Bammann, J.C. Baird, J.B. Jordon, M.P. Castanier, "Resistance spot welding of 6061-T6 aluminum: Failure loads and deformation", *Materials & Design*, vol. 34, 2012, pp. 624-630, doi: 10.1016/j.matdes.2011.05.017
- [68] M. Shamloofard and A. Assempour, "Development of an inverse isogeometric methodology and its application in sheet metal forming process", *Applied Mathematical Modelling*, vol. 73, 2019, pp. 266-284, doi: 10.1016/j.apm.2019.03.042
- [69] C. Wang, X. Zhang, G. Shen, and Y. Wang, "One-step inverse isogeometric analysis for the simulation of sheet metal forming", *Computer Methods in Applied Mechanics and Engineering*, vol. 349, 2019, pp. 458-476, doi: 10.1016/j.cma.2019.03.004
- [70] AutoForm: Homepage
- <https://www.autoform.com/en/>
- Entered: 03.11.2021
- [71] J. O'Rourke, "Finding minimal enclosing boxes", *International Journal of Computer and Information Sciences*, vol. 14, 1985, pp. 183-199, doi: 10.1007/BF00991005
- [72] N. Sephus, S. Bhagavatula, P. Shastri, and E. Gabriel, "An Industrial-Strength Pipeline for Recognizing Fasteners", 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA), 2015, pp. 781-786, doi: 10.1109/ICMLA.2015.191
- [73] S. Suthaharan, "Machine Learning Models and Algorithms for Big Data Classification", Springer, 2016, doi: 10.1007/978-1-4899-7641-3
- [74] M. Al-wswasi, A. Ivanov, and H. Makatsoris, "A survey on smart automated computer-aided

process planning (ACAPP) techniques”, The International Journal of Advanced Manufacturing Technology, vol. 97, 2018, pp. 809–832, doi: 10.1007/s00170-018-1966-1

[75] E. Murena, K. Mpofu, A. T. Ncube, O. Makinde, J. A. Trimble, and X. V. Wang, ”Development and performance evaluation of a web-based feature extraction and recognition system for sheet metal bending process planning operations”, International Journal of Computer Integrated Manufacturing, vol. 34, no. 6, 2021, pp. 598-620, doi: 10.1080/0951192X.2021.1891570

[76] K. V. Ramana and P. V. M. Rao, ”A System Level Modeling for Sheet Metal Process Planning”, Proceedings of the ASME 2004 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, volume 3d: 8th Design for Manufacturing Conference, 2004, pp. 971-983, doi: 10.1115/DETC2004-57781

[77] J. S. Sobieszczanski-Sobieski, A. Morris, and M. J. L. van Tooren, ”Multidisciplinary Design Optimization Supported by Knowledge Based Engineering”, 1st edition, Wiley, 2015

[78] PyPi: selenium

<https://pypi.org/project/selenium/>

Entered: 10.11.2021

[79] Selenium: WebDriver

<https://www.selenium.dev/documentation/webdriver/>

Entered: 10.11.2021

[80] Read the Docs: pycatia

<https://pycatia.readthedocs.io/en/latest/>

Entered: 11.11.2021

[81] CATIADOC: Object Architecture Overview

<http://catiadoc.free.fr/online/CAAScdInfTechArticles/CAAIInfArchitectureOverview.htm>

Entered: 11.11.2021

[82] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions", *Journal of Big Data*, vol. 8, 2021, article no. 53, doi: 10.1186/s40537-021-00444-8

[83] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database", 2009 IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 248-255, doi: 10.1109/CVPR.2009.5206848

[84] TensorFlow: Module: tf.keras

[https://www.tensorflow.org/api\\_docs/python/tf/keras](https://www.tensorflow.org/api_docs/python/tf/keras)

Entered: 12.11.2021

[85] Z.-X. Yang, L. Tang, K. Zhang, and P. K. Wong, "Multi-View CNN Feature Aggregation with ELM Auto-Encoder for 3D Shape Recognition", *Cognitive Computation*, vol. 10, 2018, pp. 908–921, doi: 10.1007/s12559-018-9598-1

[86] C. Wang, M. Pelillo, and K. Siddiqi, "Dominant Set Clustering and Pooling for Multi-View 3D Object Recognition", arXiv e-prints, 2019, arXiv: 1906.01592

[87] X. He, S. Bai, J. Chu, and X. Bai, "An Improved Multi-View Convolutional Neural Network for 3D Object Retrieval", in *IEEE Transactions on Image Processing*, vol. 29, 2020, pp. 7917-7930, doi: 10.1109/TIP.2020.3008970

[88] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. J. Guibas, "Volumetric and Multi-view CNNs for Object Classification on 3D Data", 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 5648-5656, doi: 10.1109/CVPR.2016.609

[89] J. Yin, N. Huang, J. Tang, and M. Fang, "Recognition of 3D Shapes Based on 3V-DepthPano CNN", *Mathematical Problems in Engineering*, vol. 2020, Article ID 7584576, 11 pages, 2020, doi: 10.1155/2020/7584576

[90] C. Ma, Y. Guo, J. Yang, and W. An, "Learning Multi-View Representation With LSTM for

- 3-D Shape Recognition and Retrieval”, IEEE Transactions on Multimedia, vol. 21, no. 5, 2019, pp. 1169-1182, doi: 10.1109/TMM.2018.2875512
- [91] Y. Feng, Z. Zhang, X. Zhao, R. Ji, and Y. Gao, ”GVCNN: Group-View Convolutional Neural Networks for 3D Shape Recognition”, 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018, pp. 264-272, doi: 10.1109/CVPR.2018.00035
- [92] F. Chollet, ”Xception: Deep Learning with Depthwise Separable Convolutions,” 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 1800-1807, doi: 10.1109/CVPR.2017.195
- [93] S. Bianco, R. Cadene, L. Celona, and P. Napoletano, ”Benchmark Analysis of Representative Deep Neural Network Architectures”, IEEE Access, vol. 6, pp. 64270-64277, 2018, doi: 10.1109/ACCESS.2018.2877890
- [94] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, ”Learning Transferable Architectures for Scalable Image Recognition”, 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018, pp. 8697-8710, doi: 10.1109/CVPR.2018.00907
- [95] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, ”A survey on deep transfer learning”, International conference on artificial neural networks, Springer, 2018, pp. 270-279, doi: 978-3-030-01424-7\_27
- [96] K. Weiss, T. M. Khoshgoftaar, and D. Wang, ”A survey of transfer learning”, Journal of Big Data, vol. 3, 2016, article no. 9, doi: 10.1186/s40537-016-0043-6
- [97] Y. Bengio, ”Practical recommendations for gradient-based training of deep architectures”, Neural networks: Tricks of the trade, Springer, 2012, pp. 437-478
- [98] D. P. Kingma and J. Ba, ”Adam: A method for stochastic optimization”, arXiv e-prints, 2014, arXiv: 1412.6980
- [99] D. Masters and C. Luschi, ”Revisiting small batch training for deep neural networks”, arXiv e-prints, 2018, arXiv: 1804.07612
- [100] L. N. Smith, ”A disciplined approach to neural network hyper-parameters: Part 1–learning

rate, batch size, momentum, and weight decay”, arXiv e-prints, 2018, arXiv: 1803.09820

- [101] R. Girshick, J. Donahue, T. Darrell, and J. Malik, ”Region-Based Convolutional Networks for Accurate Object Detection and Segmentation”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 38, no. 1, 2016, pp. 142-158, doi: 10.1109/TPAMI.2015.2437384
- [102] M. S. Iyer and R. R. Rhinehart, ”A method to determine the required number of neural-network training repetitions”, IEEE Transactions on Neural Networks, vol. 10, no. 2, 1999, pp. 427-432, doi: 10.1109/72.750573
- [103] J. Bromley, J. W. Bentz, L. Bottou, I. Guyon, Y. LeCun, C. Moore, E. Säckinger, and R. Shah, ”Signature verification using a “siamese” time delay neural network”, International Journal of Pattern Recognition and Artificial Intelligence, vol. 7, no. 4, 1993, pp. 669-688, doi: 10.1142/S0218001493000339
- [104] M. Heidari and K. Fouladi-Ghaleh, ”Using Siamese Networks with Transfer Learning for Face Recognition on Small-Samples Datasets”, 2020 International Conference on Machine Vision and Image Processing (MVIP), 2020, pp. 1-4, doi: 10.1109/MVIP49855.2020.9116915
- [105] C. Lin and A. Kumar, ”Contactless and partial 3D fingerprint recognition using multi-view deep representation”, Pattern Recognition, vol. 83, 2018, pp. 314-327, doi: 10.1016/j.patcog.2018.05.004
- [106] J. M. Johnson and T. M. Khoshgoftaar, ”Survey on deep learning with class imbalance”, Journal of Big Data, vol. 6, 2019, article no. 27, doi: 10.1186/s40537-019-0192-5
- [107] Numpy: numpy.polyfit
- <https://numpy.org/doc/stable/reference/generated/numpy.polyfit.html>
- Entered: 30.11.2021
- [108] T. Vujcic, T. Matijevic, J. Ljuovic, A. Balota, and Z. Sevarac, ”Comparative analysis of methods for determining number of hidden neurons in artificial neural network”, Central European Conference on Information and Intelligent systems, 2016, pp. 219-223



