



Norwegian University of  
Science and Technology

# Multiple Instance Learning for Car Brand Classification in the Leboncoin Online Marketplace

**Martin Hallen**

Master of Science in Computer Science

Submission date: June 2017

Supervisor: Helge Langseth, IDI

Co-supervisor: Ning Zhou, Schibsted ASA

Norwegian University of Science and Technology  
Department of Computer Science



---

*Human accuracy is not a point.  
It lives on a tradeoff curve.*

- Andrej Karpathy

---

---

---

# Abstract

This research project applies machine learning to a large-scale dataset of car images from the marketplace website Leboncoin. The project develops a model that classifies images from sales ads of cars with the brand of the car. The model combines the predictions of all the images within an ad to achieve an ad-wise classification.

The motivation behind automatic image classification is to increase accuracy, identify mislabeled items and guide the user through the registration process of a sales ad. The methods developed in this project can be applied to a variety of visual classification tasks, such as other e-commerce sites, medical imagery and video classification.

The project uses *transfer learning* to obtain faster training and higher accuracy due to a limited size of the dataset. The combination of predictions draws inspiration from *multiple instance learning* and shows that averaging of the predictions is a powerful bag-of-instance classifier.

The classifier developed in this project gives an accuracy of 0.823 and 0.945 for image-wise and ad-wise respectively.

---

---

# Sammendrag

Dette forskningsprosjektet bruker maskinl ring p  et stor-skala datasett av bilannonser fra markeds plass-nettsiden Leboncoin. Prosjektet utvikler en modell som klassifiserer bilder fra bilannonser med riktig bilmerke. De individuelle prediksjonene blir s  sl tt sammen for   danne en prediksjon for hele annonsen.

Motivasjonen bak automatisk bildeklassifisering er    kt n yaktigheten, identifisere feilklassifiserte annonser og hjelpe brukeren gjennom registreringsprosessen av en annonse. Metodene som er brukt i dette prosjektet kan overf res til andre visuelle klassifiseringsproblemer, slik som andre markeds plass-nettsider, medisinsk fotografi og video klassifisering.

Prosjektet bruker *overf ringsl ring* for   oppn  kortere treningstid og h yere n yaktighet. Dette er begrunnet i en begrenset datasettst rrelse. Sammensl ingen av prediksjoner henter inspirasjon fra multiinstans-l ring og viser at man kan oppn  en kraftig modell ved   ta gjennomsnittet av prediksjonene.

Klassifiseringsmodellen som er utviklet i dette prosjektet gir en treffprosent p  82.3% og 94.5% p  henholdsvis bilder og annonser.

---



# Preface

This Master's Thesis is the final delivery of the *Computer Science* program at Department of Computer Science (IDI) at Norwegian University of Technology and Science (NTNU) in Trondheim, Norway. The research started in January 2017 and was due in June the same year. The research is a collaboration between NTNU and Schibsted ASA. The research is executed by *Martin Hallén*, supervised by *Helge Langseth* at NTNU and *Ning Zhou* at Schibsted.

## Acknowledgments

First of all, I would like to thank Schibsted for proposing an interesting project. It has been valuable to be introduced to workplace where data science is valued as important. The project would not be possible without your arrangements, and I am grateful for being welcomed to work at your offices whenever I wanted. In addition, this project would not be possible without the provided dataset and computing platform.

*Håkon Åmdal* deserves acknowledgment for introducing me to Schibsted and organizing the initial meetings, as well as being a discussion partner in both scientific topics and life in general.

I would like to thank *Ning Zhou* for facilitating the project and make it possible for me to focus on the research instead of formalities. I would like to express my gratitude to *Audun Mathias Øygard* for providing me with academic and technical support as well as being my co-supervisor at Schibsted, together with the rest of the data-science team which kindly accepted stupid questions.

I would like to express my gratitude to my supervisor at NTNU, *Helge Langseth*, for expecting the very best from me and encourage me to expect the same of myself, as well as helping with academic discussion.

The five years in Trondheim would not be the same without the student organization *Abakus*. I would like to thank everyone who made these years unforgettable, and made me learn way more than the education can do alone.

Last but not least, I would express my gratitude to my parents for endless support.

---

# Table of Contents

<b>Abstract</b>	<b>i</b>
<b>Sammendrag</b>	<b>iii</b>
<b>Preface</b>	<b>v</b>
<b>Table of Contents</b>	<b>viii</b>
<b>List of Tables</b>	<b>ix</b>
<b>List of Figures</b>	<b>xi</b>
<b>Abbreviations</b>	<b>xiii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background and Motivation . . . . .	1
1.2 Research Goals . . . . .	2
1.3 Research Scope . . . . .	2
1.4 Research Method . . . . .	3
1.5 Contributions . . . . .	4
1.6 Thesis Structure . . . . .	4
<b>2 Problem Dataset</b>	<b>7</b>
2.1 Dataset Description . . . . .	7
2.2 Dataset Discussion . . . . .	9
<b>3 Background Theory</b>	<b>11</b>
3.1 Machine Learning . . . . .	11
3.2 Artificial Neural networks . . . . .	12
3.2.1 Feedforward Neural Networks . . . . .	12
3.2.2 Optimizers . . . . .	14
3.2.3 Regularization . . . . .	15

---

3.2.4	Dropout . . . . .	16
3.2.5	Convolutional Neural Networks . . . . .	17
3.2.6	Inception and Inception-v3 . . . . .	18
3.3	Transfer Learning . . . . .	21
3.4	Multiple Instance Learning . . . . .	22
3.4.1	Ensemble methods . . . . .	23
<b>4</b>	<b>Software Architecture and Learning Model</b>	<b>25</b>
4.1	Software Pipeline . . . . .	25
4.1.1	Software Dependencies . . . . .	25
4.1.2	Data Preparation and Analysis . . . . .	28
4.1.3	Learning Software . . . . .	29
4.1.4	Evaluation . . . . .	31
4.2	Neural Network Model Structure . . . . .	32
<b>5</b>	<b>Experiments and Results</b>	<b>33</b>
5.1	Experimental Plan . . . . .	33
5.1.1	Single Instance Classifier . . . . .	33
5.1.2	Multiple Instance Classifier . . . . .	34
5.1.3	An Improved Model for Classification . . . . .	35
5.2	Experimental Setup . . . . .	35
5.2.1	Hardware . . . . .	36
5.2.2	Dataset . . . . .	36
5.2.3	Single Instance Classifier . . . . .	36
5.2.4	Multiple Instance Classifier . . . . .	37
5.2.5	An Improved Model for Classification . . . . .	37
5.3	Experimental Results . . . . .	39
5.3.1	Single Instance Classifier . . . . .	40
5.3.2	Multiple Instance Classifier . . . . .	43
5.3.3	An Improved Model for Classification . . . . .	46
<b>6</b>	<b>Discussion</b>	<b>51</b>
6.1	Performance . . . . .	51
6.1.1	Single Instance Learning . . . . .	51
6.1.2	Multiple Instance Learning . . . . .	52
<b>7</b>	<b>Conclusion and Future Work</b>	<b>53</b>
7.1	Conclusion . . . . .	53
7.2	Contributions . . . . .	53
7.3	Future Work . . . . .	54
	<b>Bibliography</b>	<b>55</b>
<b>A</b>	<b>Example of Leboncoin car ad</b>	<b>59</b>

# List of Tables

2.1	The diversity of viewpoints in dataset . . . . .	9
3.1	Inception-v3 architecture . . . . .	21
4.1	car_ads.csv . . . . .	28
4.2	car_images.csv . . . . .	28
5.1	Accuracy on the different learning tasks. . . . .	39
5.2	Noisy-or predictions on a sales ad. . . . .	44
5.3	Probability of getting 50% of the votes . . . . .	45
5.4	The viewpoint's bias on accuracy . . . . .	47

---

# List of Figures

2.1	The structure of the dataset . . . . .	7
2.2	Images per sales ad in the Leboncoin image set . . . . .	8
2.3	Distribution of images per car brand in the Leboncoin dataset. . . . .	8
2.4	Some of the detail-oriented images in the dataset . . . . .	10
3.1	Architecture of a vanilla fully connected feed-forward network . . . . .	13
3.2	Comparison of a normal network and a dropout network. Adapted from [29]	16
3.3	Convolution neural network. . . . .	17
3.4	Naïve version of inception module. . . . .	18
3.5	Max pooling with receptive field of $2 \times 2$ . . . . .	18
3.6	Overview of inception architecture. . . . .	19
3.7	The five Inception modules building up Inception-v3 . . . . .	20
4.1	An overview of the main architecture. . . . .	26
4.2	Example of resizing image . . . . .	30
4.3	Top on neural network . . . . .	32
5.1	Experiment 1 training accuracy. . . . .	40
5.2	Experiment 2 and 7 training accuracy. . . . .	41
5.3	Visualization of image predictions . . . . .	41
5.4	Combining predictions on 6 randomly selected sales ads. . . . .	42
5.5	Average accuracy versus index in ad. . . . .	44
5.6	Confusion matrices for the single instance classification task. . . . .	48
5.7	Histograms of the fraction of images in an ad that were correctly classified.	49
5.8	Accuracy versus model year. . . . .	49
A.1	Example of car ad . . . . .	61

---



# Abbreviations

NN	=	Neural Network
ANN	=	Artificial Neural Network
CNN	=	Convolutional Neural Network
AWS	=	Amazon Web Services
SGD	=	Stochastic Gradient Descent
RMSProp	=	Root Mean Square Propagation
ReLU	=	Rectified Linear Unit
MLP	=	MultiLayer Perceptron
MIL	=	Multiple Instance Learning
CUDA	=	Compute Unified Device Architecture

---

# Introduction

## 1.1 Background and Motivation

The Leboncoin website is a consumer-to-consumer marketplace website located in France. At the moment of writing, it contains 881 thousands car ads [12]. The site monetizes on sales between its users. A satisfied user is a prerequisite to increase the number of sales and is therefore a main priority.

For a user to buy an item, the user has to find the item. Most e-commerce sites have options to search for an item as well as categories to place the different sales ads in. On Leboncoin, one of these categories is cars. The categories are hierarchical, where the brand of a car is a subcategory of the car category. Advanced search options makes it possible to filter ads on queries, year of making, brand, model and mileage. It is important to keep ads correctly categorized. Failing to do this can result in dissatisfied users and users that do not find the item they are looking for.

The users are responsible for correctly categorizing the items they are selling. Consumer-to-consumer marketplace websites have users that might not have the knowledge to correctly categorize the ads themselves. Automatic classification tools can guide the user through this process. However, automatic classification is not trivial and is currently an active field of research.

When users are looking at a sales ad, they might find themselves wanting a similar item, but not quite the current one. Sites like Amazon proposes a set of other items frequently bought by users buying the current items. Other sites display items in the same subcategory. The similar ads are often selected based on category of the current item or purchases done by other users looking at the same ad. The images in the sales ad often provide a lot of information, although this information might be difficult to automatically process.

E-commerce sites are big business. As an example, finn.no was the 6th biggest website for Norwegian users in 2012. Finn.no was only surpassed by vg.no regarding most visits by a Norwegian based websites. Finn.no had almost 800.000 daily users in 2012 [17]. The large scale usage of e-commerce sites does not only generate revenue, but also a lot of

information.

Big datasets can be difficult and time consuming to manually analyze. Machine learning has proved useful to extract relevant information from such datasets. Recent advances in image classification provide a platform for learning features in images. These features can be used to correctly classify ads based on the images. An image classification system can generate valuable insight as well as help users.

## 1.2 Research Goals

Based on our motivation we define the following goal:

**Goal: Create an accurate classification model to classify car sales ads based on images from the Leboncoin e-commerce website.**

The categories we are interested in are the brand, model and year model of a car. An accurate classification model to classify sales ads will make it possible to improve the current user interface and guide the user through the sales ad registration and sales process. In addition, the results can give important knowledge to the Schibsted team which can be generalized and applied to other e-commerce sites in the Schibsted Media Group.

From a scientific standpoint, we are also interested in the methods used to create such a classification method. We therefore define some research questions that can give a performance measure of the model.

We would like to investigate the challenges with a dataset created by the users of the website. Can machine learning be used if the dataset is not cleaned. We expect the dataset to contain noise in form of images which are hard to generalize on. How does this affect the model? We define this question as **RQ1**

**Research question 1:** What are the main challenges with a training dataset created by a user base?

In addition, we hypothesize that multiple images will improve the accuracy of the model. We want to utilize the dataset structure in a way to use the full potential. Answering **RQ2** is therefor necessary to achieve the goal of the project.

**Research question 2:** Each sales ad consist of one or more images. How can we utilize multiple images to improve accuracy of the model?

## 1.3 Research Scope

Even though image classification seems like a narrow field of research, there are a wide variety of methods and models which are actively used. Classification tasks span from video, object detection, object classification and similarity between images. Although the domains share similarities as well as learning methods, we have to frame the project and define a scope to restrict the comprehensiveness of the research.

The thesis will give a proof of concept for one specific machine learning method. It will show the effectiveness of machine learning methods on image classification and how it can be applied on a dataset without much cleaning.

## 1.4 Research Method

The research of this project is highly data-driven. First of all, it is important to understand the dataset properly. It is only when we have understood the dataset structure that we can do research into making a as good classifier as possible.

A research like this one is iterative and open for changes. In the start of the research, we lacked the knowledge to make a detailed schedule or plan. The research contained multiple main aspects as listed below.

1. **Obtain the dataset:** A prerequisite for a data-driven research project is the dataset. Not all datasets are open and free to use. Therefore, a number of precautions had to be taken to obtain the dataset. When the dataset is obtained, it also has to be stored in a secure place, only accessible for people that need access. However, it has to be easily accessible to the model and analysis platform.
2. **Setup of computing platform:** A large dataset require a lot of computing power. A personal computer might therefor lack the ability to run the model and the analysis of the dataset. This step is therefor crucial to bootstrap the project. The computing platform should support all necessary software libraries and should be able to store the dataset.
3. **Dataset analysis:** The next step is to understand the dataset. This step is a prerequisite for most of the following steps. We are interested in the properties of the dataset, the distributions and the similarity to known datasets.
4. **Background research:** The background research is the backbone of every research project. This is the step where we can stand on the shoulders of giants and create a basis for further investigation.

The first part of the background research was to find the state-of-the art classifier methods on similar datasets as the one in this project. Then we expanded our knowledge by reading papers that were referenced and relevant.

5. **Develop classifier model:** The goal of the project is to develop an accurate classification model. This step is therefore central in the research. The background research will provide us with a platform which we can expand on.
6. **Run benchmarks and analyze the results:** The classifier has to be tested to evaluate the model. Analysis of the results are necessary to improve the model in further iterations.

As the project developed, it was clear that many of the tasks was interleaved. The *dataset analysis* and *background research* was an iterative process. Discovering new features in the dataset required more literature to be read.

The development of the classifier is highly connected to analysis of the results. New results leads to new questions, as well as problems with the current solution. A thorough analysis helps us to discover weaknesses in the model.

## 1.5 Contributions

This thesis performs multiple instance image classification on data created by users. The research shows that combining predictions on multiple input improves the classification accuracy. It also shows that a naïve averaging of the predications outperforms more complicated approaches. The results presented in this thesis can be applied to other datasets and improve accuracy in fields such as video classification, image detection in medicine and e-commerce.

## 1.6 Thesis Structure

The thesis is divided into six chapters which contributes to the thesis in different ways. The first three chapters will introduce the problem and the relevant related work to the thesis.

- **Chapter 1** introduces the project. It includes the motivation for the project and the goals we are set to solve. The chapter explains how the research contributes in the research field of image classification. Finally it lists what the different chapters contains.
- **Chapter 2** introduces the dataset used in the research. The dataset is introduced early in the thesis to give an understanding of the challenges and the possibilities in the research. A good introduction is necessary to understand the relevance of the background theory as well as the decisions made during the research.
- **Chapter 3** contains relevant background information and refers to state-of-the art research in the domain of image classification. The chapter includes basic theory about *Artificial Neural Networks* as well as an into-depth explanation *Convolutional Neural Networks* and the building stones of todays state-of-the-art deep learning libraries. In addition, the chapter explains the concepts of *Transfer Learning* and *Multiple Instance Learning*.

The next two chapters contains the information which are necessary to reconstruct the project. They describe the complete setup including architecture, constants, research plan and results.

- **Chapter 4** describes the implementation of the model. It includes the architecture of the neural network used in the research as well as the general structure of the program.
- **Chapter 5** is the chapter describing the conducted experiments. First, it introduces the experimental plan. It then follows with a complete explanation of the experimental setup, including the hardware and the software libraries used to conduct the experiments. The last part of the chapter includes the results of the experiments.

The final two chapters discusses the results achieved in the research compared to the research goals, and concludes if the project was successful and what which could be done to improve if we were to do the research again.

- **Chapter 6** compares the results from the experiments and discusses how well the results are. It draws lines to the research goals as well as state-of-the-art solutions. It analyses the results and discusses possible shortcomings.
- **Chapter 7** is the final chapter. It tries to wrap up the research project and conclude how it went. Research is not only about results, but also discovering new research questions. This chapter will describe the aspects that should be investigated further.





# Problem Dataset

This chapter will help the reader to understand the problem as well as the relevance of the background theory in Chapter 3. It will focus on the dataset of the project as the project is highly data-driven. While the dataset looks similar to other large image datasets, such as ImageNet[7], it has some properties that are worth investigating.

Section 2.1 will define the important properties of the dataset. The challenges and opportunities of these properties will be discussed in section 2.2.

## 2.1 Dataset Description

The dataset in this project is based on sales ads of cars from the Leboncoin website. The dataset consists of 226735 sales ads with a total of 1161748 images. Images are taken from all angles of the cars, including the interior. All images are taken by users of the Leboncoin website. Each sales ad is categorized in a hierarchical structure. The categories available in this project is *brand*, *model* and *year*. The structure is show in Figure 2.1.

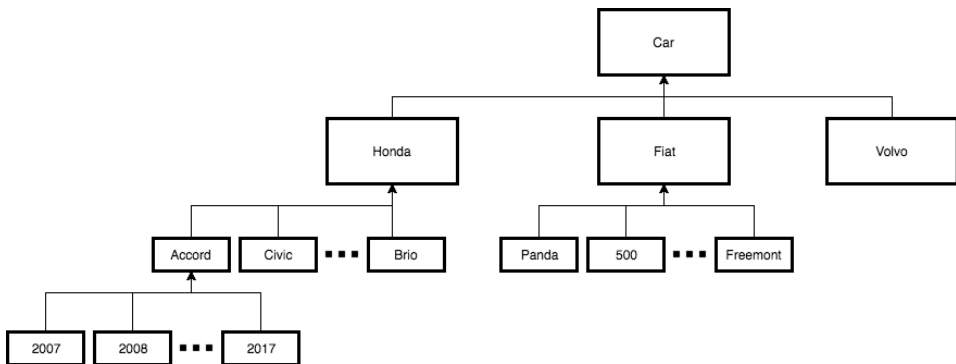


Figure 2.1: The structure of the dataset



Viewpoint	Count
Front	19
Back	10
Side	31
Front seats	7
Back seats	8
Trunk	4
Dashboard whole	11
Dashboard detail	4
Outside detail	3
Engine	2
No car in image	1
<b>Total</b>	<b>100</b>

**Table 2.1:** The diversity of viewpoints in dataset. Made by manually labeling 100 images.

describing the dataset. We can define the dataset,  $X$ , as a disjunct set-of-sequences shown in equation 2.1.  $x_{nm}$  represent the  $m$ 'th instance in ad number  $n$ . Note that the sequences are not of the same length. That is, the condition  $a = b = m$  does not have to be true.

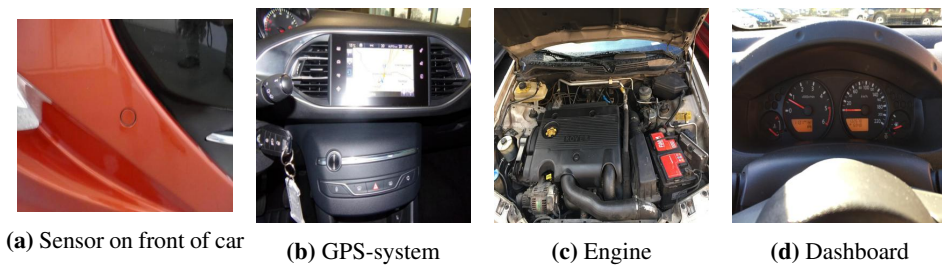
$$X = \{[x_{11}, x_{12}, \dots, x_{1a}], [x_{21}, x_{22}, \dots, x_{1b}], \dots, [x_{n1}, x_{12}, \dots, x_{nm}]\}, \quad (2.1)$$

## 2.2 Dataset Discussion

The hierarchical structure of the dataset makes it possible to classify on different granularities. The optimal result would be to successfully classify the brand, model and year of images in an ad. However, the further we move down the hierarchy, the less examples we have per class. The difference between bottom-level classes are also small. For instance, a Volvo V70 2010 model look almost identical to a Volvo V70 2011 model. However, in some cases, classifying on different on fine-grained levels in the hierarchy makes sense. Two cars of the type station wagon might look similar, although they originate from two different brands. Even though we classify on the highest granularity, brands, the classifier has to learn what separates a Volvo from a Renault. The shape of the vehicle will in many examples not be representative of the brand.

The hierarchical structure of the dataset also gives the opportunity to combine results from different granularities. We might not need to decide the precision of classification up front. A more complex model will decide how confident it is of the prediction. If it is not confident, it might lessen the precision and tell us the brand instead of the exact model.

The variety is big when it comes to images per ad. We hypothesize that some of these spikes are originated in the website interface, however there are no clear indications on why exactly 4, 6 and 11 images per ad is over-represented. One possible explanation could be that there are 4 images displayed at the same time when you look at an ad at the Leboncoin website. This can encourage users to add at least 4 images. However, this hypothesis does not explain the spikes at 6 and 11 images.



**Figure 2.4:** Some of the detail-oriented images in the dataset

Even though we do not know the origin of the distribution in figure 2.2, it is valuable knowledge. Multiple images per sales ad gives us more examples which we can classify per ad. We know that all images from the same sales ad are from the same vehicle. This segregates our classification task from the classical image recognition task.

The fact that multiple images come from the same sales ad does not mean that each individual image correctly corresponds to the label of the ad. Instead we can say that the images from the same ad share a label. Normally, such a dataset would be called a *bag of instances* [33]. However, the term bag of instances usually refers to unordered data. Since our ads contain ordered instances, we will refer to this as a *sequence of instances* [38]. Note that all the images in the dataset are not independent. When dividing the dataset into a training and testing set, there should not exist pictures from the same ad in both sets. In other words, the training set and test set should be disjoint in regards of sales ads. This may seem obvious for the experienced reader, but this is not an unusual error made in machine learning research.

The fact that some of the images show only details of a car is a worth some thoughts. A model might wrongly learn that a GPS system belongs to one car brand, although the same GPS-system can be used in a wide range of different brands and models. We hypothesize that the first images of in the sales ad gives a more general representation of the vehicle than the successive images. For instance is the first image is often an image from the outside of the car. The images gets more detail-oriented the further you scroll through the images. Figure 2.4 shows some examples of very detail-oriented pictures. We would not expect the model to correctly learn such details. This shows the importance of combining predictions on images from the same sales ad, explained in section 3.4 and 4.1.4.

The distribution of images per brand as seen in figure 2.3 adds some considerations. As the classes are not balanced, the model will see many more example of a common class than an uncommon class. Classes with few examples are also prone to overfitting. The solution might be to select a balanced subset of the dataset or augment the dataset with extra examples for the under-represented classes.

# Background Theory

This chapter will introduce the relevant background theory as well of the state-of-the-art research within relevance. It first introduces general concepts of *machine learning* before it goes more in detail on the specific methods used in this project.

## 3.1 Machine Learning

The increase of generated data creates the need to do automatic analysis of datasets. One field of data analysis is machine learning. It is used in applications such as page rank, collaborative filtering, image classification and language translation. Common for these applications are big datasets which makes it possible to see patterns that disappear for the human analyst.

There are two main branches of machine learning, classification and regression. Classification tries to assign a discrete value to a data sample while regression finds a continuous value. However, they are often closely related, and regression techniques can be used for classification [28]. This section will focus on the task of classification, more specifically multi-class classification.

Multi-class classification takes an input  $x$  and tries to assign a label  $y \in \{1, 2, 3 \dots, n\}$  to  $x$ . In other words, it tries to learn the function  $f : x \rightarrow y$ . The challenge is to find the function mapping  $f$  such that  $f(x) = y$  for all instances of  $x$ .

Supervised learning is a field within machine learning. Assume we have a model which maps from the domain of  $x$  to the domain of  $y$ .  $x$  can be a multi dimensional input, while  $y$  is a discrete value in the case of classification. The model is given pairs of example inputs and outputs,  $(x, y)$ , called training examples. The model will try to generalize and estimate the true function  $f$ . All the training examples form what is known as the *training set*.

We can also feed new examples to the model, this time without the correct output. The model will predict the output from learned experience. This set of samples are known as the *test set*.

In addition to the training and test set, we have the validation set. The validation set is used during training to verify the progress of the model. The validation set is not used to train the model, in the sense that the model does not try to learn from the samples in the validation set. However, we can use the results on the validation set to make decisions such as to stop the training. The complete dataset is usually split into these three disjoint datasets.

Accuracy is defined as the percentage of the samples which are correctly classified. Similarly, the error rate is the percentage of misclassified samples. When presenting the results, we should calculate the accuracy on the test set. The training set and validation set is used during training, and can therefore not tell us how good the model is to predict unseen data.

## 3.2 Artificial Neural networks

Artificial neural networks (ANNs) is a term used to describe distributed and parallel processing inspired by the structure of the biological neural network in our brain. An ANN is a structured network of simple computing units, often called neurons. Together they form a complex network able to do more advanced computing. ANN's can be seen as a universal function approximator. They can be trained to solve both supervised and unsupervised learning problems. This project will focus on the supervised training for classification, and all further description assumes this [37].

### 3.2.1 Feedforward Neural Networks

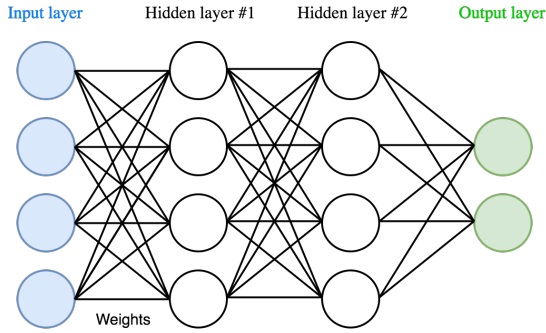
Feedforward neural networks are the most basic kind of ANN's. It is sometimes described as the "vanilla" neural network [10]. The units are organized in a layered structure. The first layer is the input layer and the last layer is the output layer. In between we have one or multiple "hidden" layers. The number of units in the input layer corresponds to the number of dimensions in the input. For instance will a network which works with 200x200 pixel color images have  $200 \cdot 200 \cdot 3 = 120000$  input units, one for each RGB value of each pixel.

The output  $y_i$  of an unit  $x_i$  in a layer is defined in equation 3.1, where  $\sigma$  is the activation function,  $w_{ij}$  is the weight to unit  $x_i$  from unit  $x_j$  in the previous layer [34].

$$y_i = \sigma \left( \sum_{j=1}^N w_{ij} x_j \right) \quad (3.1)$$

The activation function,  $\sigma$ , is often a sigmoid (equation 3.2) or ReLU (equation 3.3) function. The activation function introduces non-linearity to the network. The sigmoid function has an advantage where the output is bounded so it does not diverge. However, the gradient tend to vanish when the activations grow large. ReLU does not have this problem, but as a result the gradients can blow up. Regularization techniques can be used to minimize this problem.

$$S(x) = \frac{1}{1 + e^{-x}} \quad (3.2)$$



**Figure 3.1:** Architecture of a "vanilla" fully connected feed-forward network. Adapted from [34]

$$R(x) = \max(0, x) \quad (3.3)$$

Each layer in the neural network feeds to the next one, hence the name feedforward network. The last layer contains the same number of units as we have target classes. The outputs can be seen as the values  $y = \{y_1, y_2, \dots, y_n\}$  in the case of  $n$  target classes. Instead of using those values directly, we often apply a softmax function to this output. The softmax has multiple advantages. First of all, it bounds the values in the range  $(0, 1]$ , where all the outputs sum up to 1. Additionally, these values can be interpreted probabilistic. The softmax function is defined in equation 3.4 [18].

$$h_i = \frac{e^{y_i}}{\sum_j (e^{y_j})} \quad (3.4)$$

The output corresponds to the predictions of the model. Usually, we assign the label  $y_j$  with the maximum value to sample  $x_i$ ,  $y_j = \max_j(y_j)$ .

We evaluate the output of the network with a cross-entropy loss defined in equation 3.5. The softmax classifier therefore minimizes the cross entropy loss between the estimated class probabilities and the true distribution. The true distribution is defined as a distribution where all the mass is placed at the correct class, such as  $p = [0, 0, \dots, 0, 1, 0, \dots, 0]$ . The cross-entropy between the predicted  $h$  distribution and the true distribution  $p$  is then defined in equation 3.6.

$$L_i = -\log h_i \quad (3.5)$$

$$H(p, h) = -\sum_x p(x) \log h(x) \quad (3.6)$$

The loss function is also called the objective function. The challenge is the optimize the weights of the network as to make the sum of objective function as small as possible.

### 3.2.2 Optimizers

Gradient decent optimization algorithms is a method to minimize an objective function. There are many different versions of gradient descent, and most of them are provided in machine learning libraries without a proper definition. The basic variant of gradient decent computes the gradient of the loss with respect to the parameters and updates the parameters in the model in the opposite direction of the gradient of the objective function. This will minimize the objective function. The magnitude of the update is defined by a hyper-parameter we call *learning rate*. The learning rate of the model decides how fast we approach the supposed solution. However, in the most basic variant described here, we risk getting stuck in a local minimum. This means that the gradient increases in all directions with respect to the parameters. However, it might not be the global minimum. If the learning rate is too big, we might overshoot the global minimum and oscillate around the optimal solution. If we select a too small learning rate, the training might be very slow. We can design a learning rate schedule to overcome these problems. The scheduler changes the learning rate as the loss stops decreasing. However, it is difficult to select the correct schedule without excessive testing. In addition, we can use *early stopping*. Early stopping monitors the logs of the training and stops it when the training hits some condition.

There are developed a wide variety of gradient decent methods to deal with the difficulties of selecting hyper-parameters. Some of those will be presented in this section [26].

#### Mini-batch Gradient Descent

Mini-batch gradient descent is a combination of batch gradient descent and stochastic gradient descent (SGD). The former calculates the gradient with respect to the parameters for the whole training set. While this method is guaranteed to converge to the global minimum, the training is slow and problematic if the whole dataset does not fit in memory. SGD works by performing an update for each training sample. The training set is shuffled between each iteration of the dataset. SGD is fast, but the variety of training examples makes it difficult to reach the global minimum for the whole dataset.

Mini-batch gradient descent tries to combine batch descent and SGD. It works on batches of examples from the dataset. Instead of using the whole training set as each batch, it divides the training set into disjunct mini-batches. It updates the gradient after each such mini-batch. One complete iteration of the training set is called an epoch.

Mini-batch gradient descent is stochastic in the sense that it shuffles the training data between each epoch. If we try to minimize the objective function  $J(\theta)$  with respect to the parameters  $\theta$ , we calculate the opposite direction of the gradient  $\nabla_{\theta} J(\theta)$ . We also select a learning rate  $\eta$ . The parameters are then updated by equation 3.7, where  $i$  is the index of a training example and  $n$  is the mini-batch size.

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \quad (3.7)$$

The simplicity of stochastic gradient descent is elegant, but it has a few shortcomings. As the mentioned gradient descent methods, it also suffers from a sensitive selection of learning rate. A too high learning rate, and the network is going to diverge or oscillate around the solution. A too small learning rate, and the learning is going to be slow.



A common addition to mini-batch gradient descent is to add momentum. It does this by adding a fraction of the last update to the current update. The idea is the same as pushing a ball down a hill. As the speed increases, it gains momentum. Even if it hits a flat area or a small uphill, it will continue moving in the same direction for a while.

### RMSProp

Root Mean Square Propagation (RMSProp) is a further development of mini-batch gradient descent with momentum. It is unpublished, but has gained wide popularity due to its simplicity to use and good results. It is now included in most machine learning libraries. It was introduced by Geoff Hinton in his Coursera Class on machine learning [16]. RMSProp has advantages such as adapting the learning rate to how frequent the parameters are. The update function can be seen in equation 3.8.  $g_{it}$  is the gradient of the parameter  $\theta_i$  at time step  $t$  and  $E[g^2]$  is a running average of the past squared gradients.  $\eta$  is the learning rate and  $\gamma$  the momentum.

$$\begin{aligned} E[g^2]_t &= \gamma E[g^2]_{t-1} + (1 - \gamma)g_{it}^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_{it} \end{aligned} \quad (3.8)$$

In effect RMSProp will decrease the learning rate as the training goes on. The suggested values for learning rate and momentum is  $\eta = 0.001$  and  $\gamma = 0.9$  respectively.

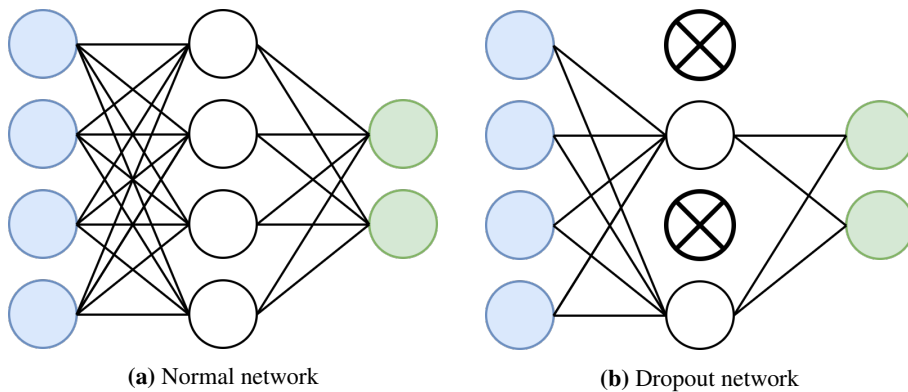
### 3.2.3 Regularization

Large neural networks can have millions of parameters. Overfitting is a major problem in such networks. One method to solve the problem with overfitting is to penalize the complexity of the model.

*L1* and *L2* regularization adds a penalty to the objective function for large weights. *L2* regularization adds the squared magnitude of all the weights to the objective function. In effect, this encourages the network to use all its input a little rather than that some of the inputs does all the work.

*L1* regularization is similar, but it does not square the weight magnitudes. The effect is that the weights gets sparse during training. Both *L1* and *L2* defines a parameter  $\lambda$  to decide the strength of the regularization. *L1* and *L2* regularization can also be used together. The penalty is then typically  $\lambda_1|w| + \lambda_2w^2$ .

A third regularization method is *max-norm*. It enforces a maximum magnitude of the weight vector of each unit. This constraints the weights and make sure that they do not explode, even when using a high learning rate.



**Figure 3.2:** Comparison of a normal network and a dropout network. Adapted from [29]

### 3.2.4 Dropout

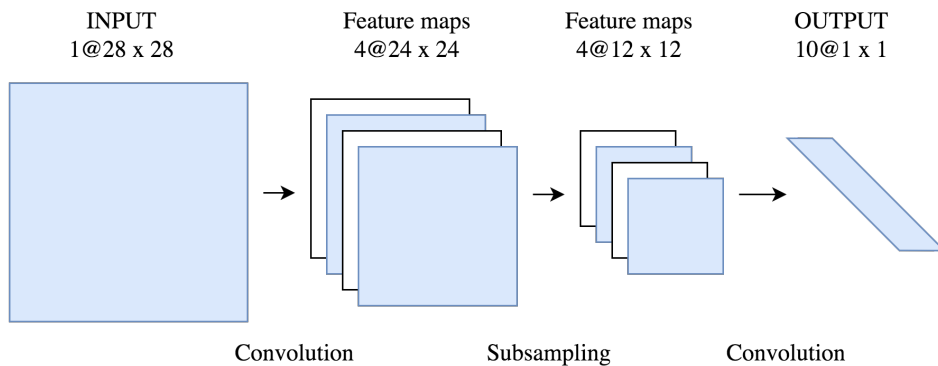
Dropout addresses the problem with overfitting in a different way than the regularization techniques described above. Dropout prevents units from co-adapting. It works by randomly setting the output of units to zero. This is conceptually the same as temporarily removing the connection of the node in the network. By making the presence of other units uncertain, each unit is forced to perform well in many different contexts, it cannot rely on other units. Because of this, it is hypothesized that each unit better adapts to detects a single feature at a time, as opposed to only a part of a feature or multiple features. Dropout has proved to significantly reduce the error rate, and has been added to a wide variety of ANN's [29].

In effect, dropout is similar to generating a large set of different sparse networks. The output of these sparse networks can then be averaged to produce the final output. However, each of the networks has to be trained. This makes the training slow. In addition, each of the networks has to be tested at test time.

Since the dropout network is a single network, we have to use a different approach. We do not drop units at test time. However, we scale the weights by a factor to make up for the missing weights at training time. Assume we set the dropout parameter to  $p$ . We set individual weights to zero with the probability of  $1 - p$  during training of each mini-batch. At test time, we run the input through the full network, but each weight is scaled by multiplying it with  $p$ . Opposed to the averaging of different sparse networks, we simply use the full network, scaling the weights to take into account that we have more weights contributing between each layer.

We can have different dropout rate in different parts of the same network. Actually, different dropout rate at different layers has proved effective [29]. At test time, each layer has to be scaled corresponding to the dropout rate of the layer to get the correct output.

Dropout can substitute normal regularization methods in many networks. However, we can also use a combination of dropout and regularization methods. [29] got the best result using a combination of dropout and max-norm regularization.



**Figure 3.3:** Convolution neural network. Adapted from [14]

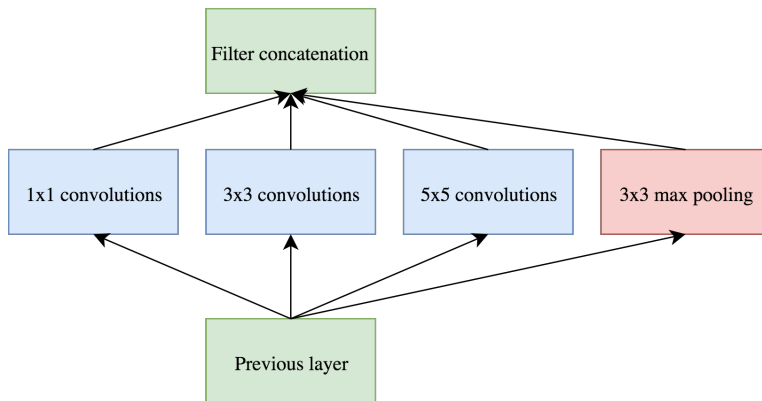
### 3.2.5 Convolutional Neural Networks

Conventional feedforward neural networks performs well in a wide variety of pattern recognition, including images. However, there are some shortcomings. The fully connected layers suffer from the *curse of dimensionality*. They also do not factor in the spatial structure of the data.

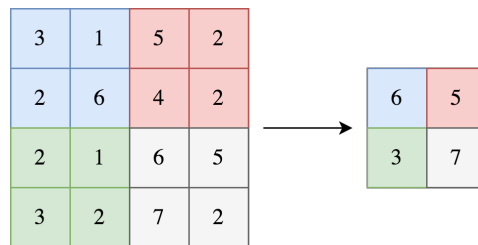
Convolutional neural networks introduces some new ideas to solve these problems in neural networks. They were motivated by neurobiological studies of the receptive field of animals [14]. The network is specifically designed to tackle visual tasks, but may also be used in other domains. A restricted connection scheme helps detecting and combining local features in an image. Each unit corresponds to a local receptive field, but performs the same operations on different locations by sharing weights. The same idea can be applied to subsequent layers as well. Each layer will then combine small features into more complex and abstract features [20, 21]. A convolutional layer is often followed by contrast normalization or max-pooling to reduce the number of dimensions [30].

A simplified example of a convolutional neural net is seen in figure 3.3. The input of the net is a 48x48 pixel gray-scale image of a handwritten number. We want to predict the number in the image. The first hidden layer is a convolutional layer. In the figure, a kernel of size 5 x 5 was used to create 4 feature maps. The next layer performs sub-sampling and local averaging. The receptive field of each neuron is 2 x 2. It is important to note that this layer also has trainable weights and bias. This helps to reduce the resolution of the feature maps. The sensitivity of features are hence reduced. This helps with distortion or small shifts of the kernel [14]. The last convolution reduces the feature maps to 10 different outputs, corresponding to the 10 single digit numbers. The kernel has the same resolution/size as the feature maps in the last hidden layer. In this example is the size 12 x 12. However, this size is usually smaller as the network is deeper and the size is reduced gradually.

We can also use max pooling instead of sub-sampling and local averaging. Figure 3.5 shows how this works with a receptive field of 2 x 2, sometimes called stride. Note that the strides do not overlay each other. Max-pooling works by taking the maximum value in each non-overlapping stride. While average/sum-pooling uses the same general idea,



**Figure 3.4:** Naïve version of inception module. It presents the main idea of the Inception module. Adapted from [30].



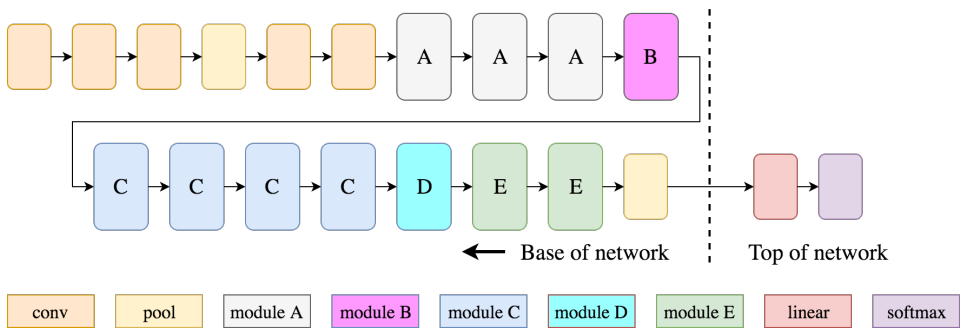
**Figure 3.5:** Max pooling with receptive field of 2 x 2. Adapted from [18]

max-pooling has historically been preferred as it works better in practice [18].

### 3.2.6 Inception and Inception-v3

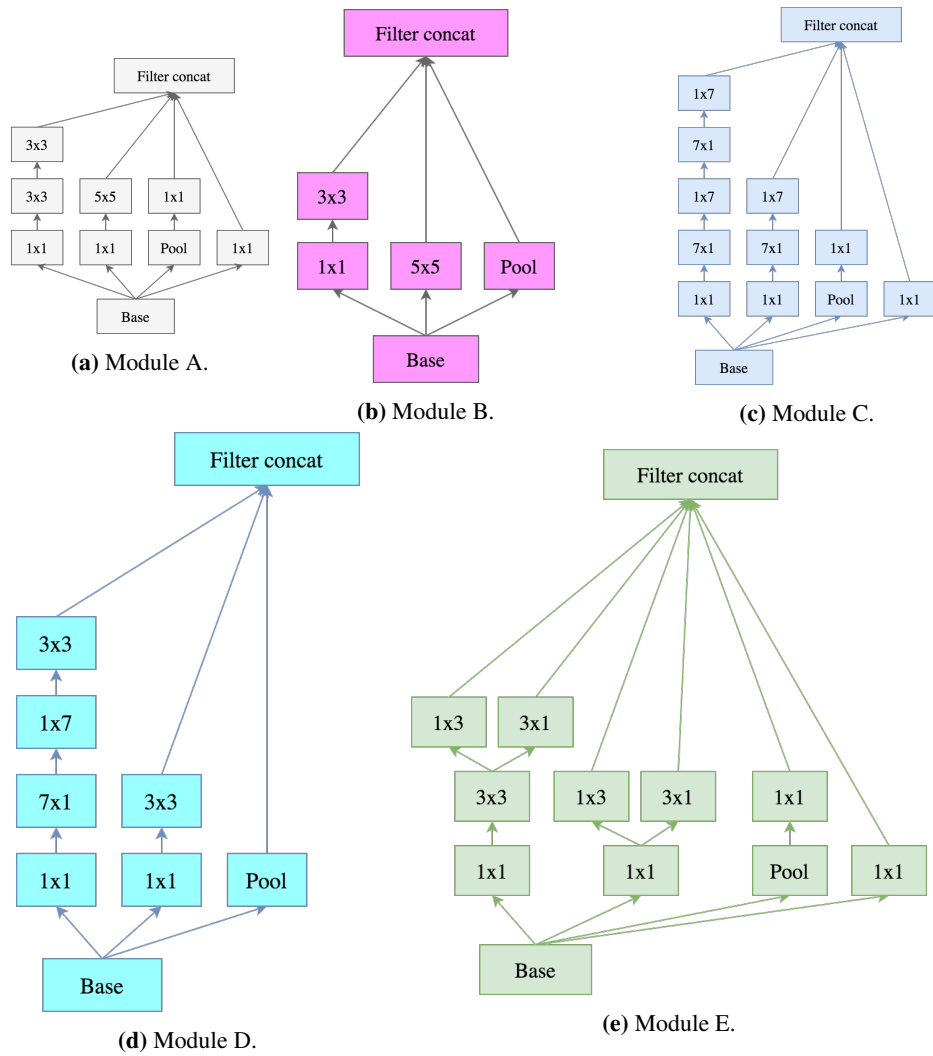
The Inception architecture was first introduced by Google in their GoogLeNet neural network [30]. The main difference from conventional CNN's is the *Inception* module. The Inception module works by combining different-sized convolutions as well as max-pooling to form a more complex computing unit. A simplified model of the Inception module is shown in figure 3.4. The implementation differ from this visualization by refactoring the convolutions to reduce computational complexity. However the concept remains the same. Multiple Inception-modules is stacked in a similar fashion as convolutional layers in a normal convolutional network with occasional max-pooling layers between. GoogLeNet is 22 layers deep, not counting the max-pooling layers.

An modification to the original Inception architecture was presented in [31]. The main principle of the new architecture is the same, but it focuses more on reducing the computational bottlenecks. They introduced massive refactoring of the Inception unit. The refactoring made it possible to scale the network to 42 layers deep, only increasing the computational cost by 2.5 percent. The resulting network is referred to as Inception-v2. In addition, they added batch normalization auxiliary to form what is known as Inception-v3.



**Figure 3.6:** The complete inception architecture. A visual representation of the Inception-v3 architecture described in [31]

The Inception-v3 model is build up of 5 different Inception modules, in addition to the standard convolutional layers. The overview of the Inception-v3 model can be seen in figure 3.6 and the detail of each Inception module in figure 3.7.



**Figure 3.7:** The five Inception modules building up Inception-v3. All figures adapted from [30] and changed to represent the implemented Inception-v3 architecture in Tensorflow v1.1.0.

Type	Patch size/stride or remarks	Input size
conv	3x3=2	299x299x3
conv	3x3=1	149x149x32
conv	3x3=1	147x147x32
pool	3x3=2	147x147x64
conv	1x1=1	73x73x64
conv	3x3=1	73x73x80
conv	3x3=1	35x35x192
pool	3x3=2	71x71x192
3xInception	As in figure 3.7a	35x35x192
1xInception	As in figure 3.7b	35x35x288
4xInception	As in figure 3.7c	17x17x768
1xInception	As in figure 3.7d	17x17x768
2xInception	As in figure 3.7e	8x8x1280
pool	8x8=1	8x8x2048
linear	logits	1x1x2048
softmax	classifier	1x1x1000

**Table 3.1:** Inception-v3 architecture. Adapted from [31] and changed to represent the implemented Inception-v3 architecture in Tensorflow v1.1.0. The two last layers represent the "Top of network" in figure 3.6.

### 3.3 Transfer Learning

Transfer learning is an useful technique to overcome some common problems in training a CNN, such as deficit of training samples or limited time frame for training [24, 25]. It aims to transfer knowledge between related source and target domains. Transfer learning is often used in image domains [24], but is also used in other tasks such as Natural Language Processing [6]. We will refer to this concept as *transfer learning* or *knowledge transfer*.

Transfer learning is not limited to machine learning. Transfer learning is around us every day. Learning to read Norwegian text will definitely help you with understanding English. Even though the words are different, the letters are mostly the same. Additionally, the sentences are built using words with spaces in between. Similarly, learning to play guitar is indeed going to help you learn the base. It is probably helping you to learn the piano as well for that matter. The human brain excels in knowledge transfer. It is therefore not a surprise that knowledge transfer can be used in a model like CNN's.

Historically, low-level descriptors have been created or selected manually [4], such as Gabor filters and SIFT descriptors. These features are global and do not try to find the overall structure of the input. The resulting features are often called low-level features. Modern CNN's have proven successful at finding such low-level descriptors automatically through training [24]. The key idea behind transfer learning is that low- and mid-level representation of samples from two different domains can be similar [25]. Imagine a large CNN trained on the ImageNet dataset. The first layers of the CNN will learn representations of colors, edges, etc.. These low-level features are often identical to features present

in other image datasets. This is the knowledge we want to transfer to our other domain of interest.

There are also unsupervised approaches that solves similar issues with a small amount of training samples. If a large unlabeled dataset is available, it is possible to pre-train a model using a sparse auto-encoder [22].

### 3.4 Multiple Instance Learning

When multiple images are known to belong to the same class, we ought to make advantage of this. Multiple Instance Learning (MIL) tackle this problem. There are two main solutions to do this. One is to make use of the grouping information during training. The other method omits this step trains a single instance classifier using individual images. At test time, the model simply combines the predictions for images belonging to the same group.

There are some considerations to take before deciding on the approach. For some datasets, the individual images cannot be classified individually, simply because they do not represent the complete object the class label tells us. For instance will an image of a foot not correctly represent the concept "person", although a group of images including the image of a foot can. For learning problems as this, we say that that a group of images share a label. Such groups are often called *bags of instances* in machine learning [33]. Note that individual images does not have a label, the bags have.

Common for all the types of MLP investigated in this project is to connect predictions of the individual instances of the bag. The most naïve solution is to average the predictions of all the instances. Xu & Frank [35] assume that all instances in a bag contributes equally to the prediction. They can then take the average of all the predictions. The results are promising, but they use a constructed 2D dataset. Their statistical approach to the problem can be difficult to transfer to more complex domains. They also run some benchmark tests on their classification algorithm. Not surprisingly, the more instances there are in a bag, the greater the advantage of the bag approach is. However, it is close to impossible to infer how many examples which are needed for other domains or algorithms. Although there are problems with the generality of their approach, it definitely shows that MIL can help in both classification and regression tasks.

Another approach is to develop a custom loss function which are used during training. Singh and Garzon [27] develops a loss function that takes the loss over a whole bag of instances. Because of this, they group the instances that share bag together into its own mini-batches. The model is then trained on the mini-batches of variable size.

Singh and Garzon uses a dataset where each bag of instances have 0, 1 or multiple attributes. An attribute is denoted by  $a$ . The loss over a bag of instances is defined in equation 3.9.  $y_{ija}$  is the loss of training sample  $j$  in bag  $i$  with the respect of attribute  $a$ .

$$L_{\text{total},i} = \sum_{a=1}^n \begin{cases} \sum_j (y_{ija})^2, & \text{if } a \text{ is negative for bag } i \\ \sum_j (y_{ija} - 1)^2, & \text{if } a \text{ is positive for bag } i \end{cases} \quad (3.9)$$

Singh and Garzon also proposes two different methods for combining the predictions of the samples in a bag of instances. Each instance in the bag is classified using the classifier. The first method is to iterate through all the target classes. Assume a target



class  $a$ , instance  $x_i$  in bag  $X_j$  and prediction  $y_{ia}$  on instance  $x_i$  for each target class. If one of the instances  $x_i$  in bag  $X_j$  produced a prediction  $y_{ia}$  which is greater than a given threshold, then assign class  $a$  to  $y_i$ . The final bag prediction  $Y_{ia}$  is given by equation 3.10.

$$Y_{ja} = \max_i (y_{ia}), \forall y_i \in Y_j \quad (3.10)$$

The other approach is to assign 1 to a prediction if it is above a threshold and 0 if it's not. They then assign label  $a$  to bag  $X_j$  if a certain percentage of the examples in the bag were positive. If none of the labels were selected, then the sample is classified with none of the attributes.

There are other MIL models which are worth mentioning. Babenko et al. [3] and Viola et al. [33] both use a boosting approach called MILBoost. Their approach is a slight modification of the AdaBoost algorithm and is out of scope for this project. However, both use a prediction combination which is interesting. Instead of averaging the predictions as in Xu & Frank [35], they adapt the Noisy-OR (NOR) model as seen in equation 3.11. It weights large positive results more than negative results. This way they ensure that if one example is very positive of a class, then the bag is probably this class.

$$p(Y_i|X_i) = 1 - \prod_j (1 - p(y_i|x_{ij})) \quad (3.11)$$

### 3.4.1 Ensemble methods

Ensemble methods are learning algorithms that combine predictions from a set of machine learning classifiers. Instead of getting the result from a single classifier, the predictions from many classifiers are combined. Usually this is done by averaging or voting (possibly weighted) to produce the final prediction. For ensemble methods to work, the classifiers have to be accurate and diverse [13]. In this example, an accurate classifier means that it is better than random guessing at unseen data. The diversity of the classifiers means that they have different errors on new data points. Both weak and strong learners can be used in ensemble methods.



# Software Architecture and Learning Model

This chapter will present the architecture of the experiment. This includes the architecture of the learning program as well as the neural network model.

## 4.1 Software Pipeline

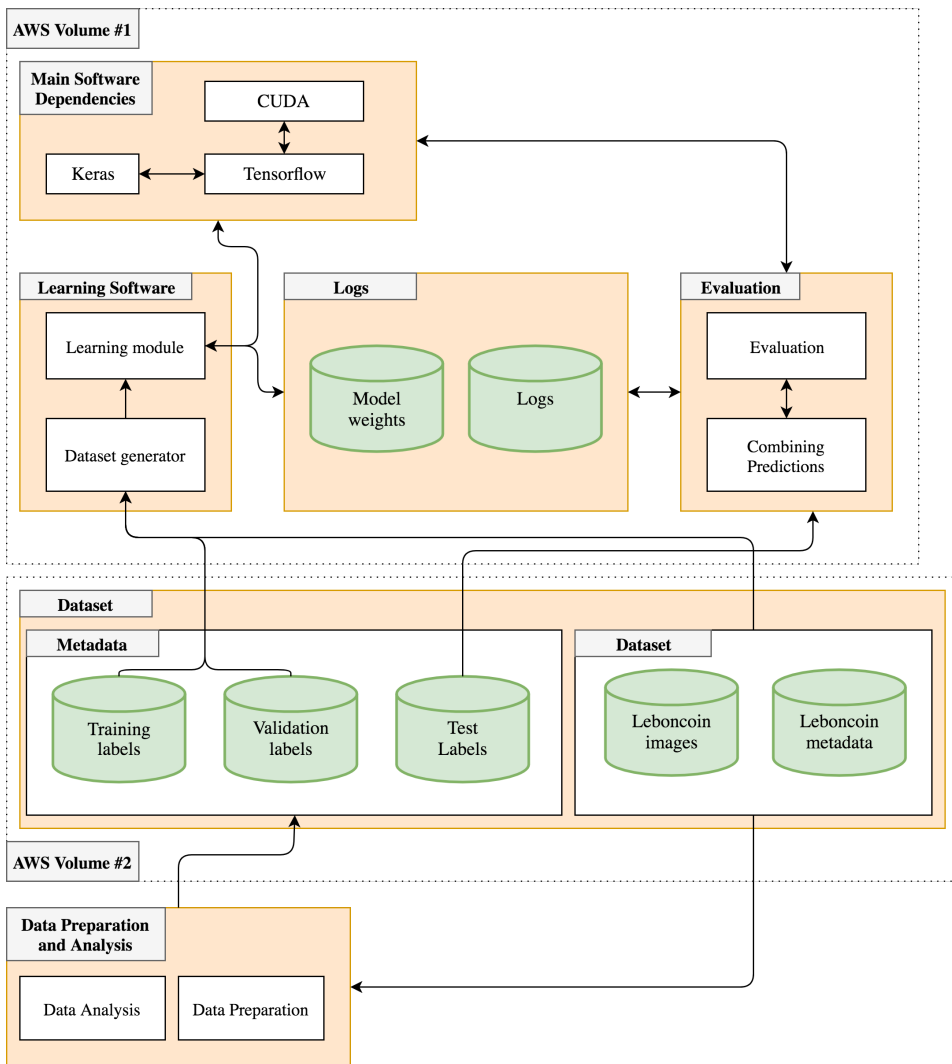
Figure 4.1 displays the components of the software used in the research. This section will break down the architecture and explain each individual component. The pipeline is sequential where each component is dependent on the components preceding it in the pipeline. Each component is presented in the order they appear in the pipeline.

There are multiple reasons behind the sequential pipeline. The first reason is limited system memory. Only one instance of the model can be loaded at the same time without exceeding max allowed memory usage. This limitation disallows running different components in parallel.

The other reason to decouple the components is to enable evaluation for all iterations and runs of the model as well as compare evaluation between two different runs. This approach enforces the storage of intermediate data.

### 4.1.1 Software Dependencies

There are many software dependencies for this research project. This section presents the most important ones. The program is written in Python, using the high-level machine learning API Keras [5] with a Tensorflow backend [1]. All of the needed packages can be installed as Python packages. I chose to use Python virtual environments to easily confine and manage the dependencies, as well as run the same program on multiple platforms. I chose Anaconda [8] as my virtual environment manager due to familiarity and Pip as package manager due to availability of all the necessary packages.



**Figure 4.1:** An overview of the main architecture.

## Keras

The Keras API includes two different backends, Theano [32] and Tensorflow. The choice of Tensorflow was arbitrary, although it was influenced by the popularity in media and by other researchers. Keras serves as a high-level API on top of Tensorflow. A neural network model can be built by stacking layer of neurons on top of each other. The layers in Keras corresponds to layers in Tensorflow.

Keras already implements multiple neural network models, including our choice of Inception-v3. Inception-v3 is designed to classify images in the ImageNet dataset [7].

There are two main versions of the network. One includes the top layer while the other does not. The reason to include a model without the classification layer is so other researchers easily can adapt the model to their own learning problem. For instance does the default model classify images into 1000 classes from ImageNet. Custom datasets might yield a different number of target classes.

Keras also includes pre-trained versions of the models. As we have seen in section 3.3, transfer learning can speed up the training of the network. The Inception-v3 model is pre-trained on the ImageNet dataset, achieving a 21.2% top-1 and 5.6% error on the ILSVRC 2012 classification challenge [31].

### Tensorflow

Tensorflow is a software library for numerical computation. It implements the concept of data flow graphs, where nodes are mathematical functions and edges are flow of multi-dimensional data arrays. It scales well between platforms, and can be run on everything from mobile devices to clusters of high-power GPU's. The Tensorflow architecture makes it suited for neural network models [1].

### CUDA and Tensorflow-GPU

In addition to the standard dependencies for Tensorflow and Keras, we also used CUDA graphic computing library [23] as well as *tensorflow-gpu* for the GPU computing. CUDA is a parallel computing platform created by Nvidia. It enables general-purpose computing on the GPU and can speed up the training substantially.

### iPython Notebook

We would like an interactive platform which provides direct access to the dataset as well as system and graphing libraries. iPython notebook provides such a solution. The whole project is placed on a server without a graphic user interface. However, we can forward the iPython Notebook via ssh to our own computer to cope with the lack of GUI. The following approach will give and graphical interface for the iPython Notebook on your own localhost. The necessary commands are in listing 4.1 and 4.2. The command in listing 4.2 should be run in the projects root folder on the server, while command in listing 4.1 is run on your local computer.

---

```
1 user@local_host$ ssh -N -f -L localhost:8888:localhost:8889 \  
2   -i .aws/martinhallen.pem username@serveraddress
```

---

**Listing 4.1:** To be run on personal computer

---

```
1 user@remote_host ipython notebook --no-browser --port=8889
```

---

**Listing 4.2:** To be run on remote host

## 4.1.2 Data Preparation and Analysis

This section will include the metadata and image processing which were used in the project. All data preparation and analysis was implemented in iPython Notebooks, except for image augmentation which are a part of the Keras library.

### Data Analysis

The main focus of the analysis is to identify the distributions of the dataset. This is important to make choices about which subset to use for training and testing, and to explain possible challenges or problems that appear. There are multiple interesting distributions, including the different viewpoint of the images as well as the number of images in each class and sales ad.

The metadata of the dataset includes all the relevant information to correctly parse and structure the data. Table 4.1 and 4.2 shows the structure of the metadata files.

ad_id	brand	model	year	ad_name
1082999147	Hyundai	Matrix	2003	Hyundai matrix 1.5crdi
1082999117	Audi	A5	2007	AUDI A5 2.7 v6
1022131942	Nissan	Patrol	2002	Nissan patrol gr 3.0
1082999117	Peugeot	Partner	2008	Partner Tepee II

**Table 4.1:** car\_ads.csv

ad_id	image_id
1082999147	45b2691b3a3dab97603a22540d830c04e7b94de2
1082999147	efaa632b361c85e9c1a75ab0dc6a5e90fd70072f
1082999117	36e28208151421c1dcc58b1ce2ac6db6b8e26279
1082999117	a5c704de4521264ce66603ec9b318eb0750ff4a1

**Table 4.2:** car\_images.csv

A full outer join of these tables will provide us with all the metadata-information we need. However, we need to add the path to the stored image to analysis the view-points. Each files is stored by the convention `/path_to_dataset/38/01/e5/380...8a9c.jpg`. Note that the folder structure is hierarchical. This is done simply for convenience. When browsing the dataset, we avoids big folders. This is identical to using hash mapping, except that the id of the image works as the unique hash.

### Data Preparation

We want our learning module as easily configurable as possible. One of these configurations is the selection of the dataset which is used for training. Our approach is to define a mapping from the images to the label of the images. This approach makes it possible to change the mapping at any time. These mappings are stored in a file which is fed to the data-generator explained in section 4.1.3. An example of one such file

is shown in listing 4.3. Notice that each file path includes the id of the picture, i.e. 3801e5476460ce35d41-070bdc40ac38e55fc8a9c. There are three different such files, one each for the *training*, *validation* and *test* set.

---

```
1 /path_to_dataset/38/01/e5/3801e5476460ce35d41...070.jpg 12
2 /path_to_dataset/cd/fe/e2/cdfce201a6eaa7fd6cd...2a6.jpg 1
3 /path_to_dataset/59/8e/fa/598efa3486eac8307c3...747.jpg 0
4 /path_to_dataset/8e/d5/4c/8ed54c2dd096b73b113...567.jpg 1
5 /path_to_dataset/86/11/ee/8611ee3d51ce5f0881b...4f3.jpg 10
6 /path_to_dataset/99/0f/e1/990fe197cab6f421f07...b7b.jpg 15
7 /path_to_dataset/da/48/dd/da48dd54b2bf808e460...9f3.jpg 16
8 /path_to_dataset/da/01/72/da0172b1b943b0b53e5...b33.jpg 3
9 /path_to_dataset/14/ea/44/14ea4464a2269f74bfb...ea8.jpg 10
```

---

**Listing 4.3:** Example of labels file. The image-hashes are shortened for presentation purposes.

The images in the dataset are of a variety of different sizes, including high resolution pictures. A resized copy of the dataset was therefore created, each image with a resolution of 400x400 pixels. There are two main reasons to do resizing of the images.

- Most machine learning libraries requires a fixed input size. This is because of the fixed definition of the layer sizes in the networks [15].
- Smaller images require fewer pixels to be processed during training, and hence faster training. A large input floods the network with irrelevant information [2].

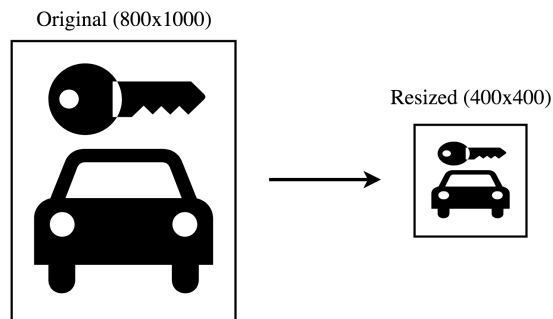
The original image is fitted inside the a bounding box of 400x400 pixels, without preserving the aspect ratio. An illustration can be seen in figure 4.2. Ideally, we should find a square bounding box of the car in an image. The we could crop and rescale to keep the aspect ratio. However, the bounding box information is not available. Warping the images are a common approach to produce fixed sized input, and does not affect classification accuracy significantly [9, 11].

### 4.1.3 Learning Software

This section provide an explanation of the architecture responsible for training the model. The input to the learning software is the mapping between image-paths and class labels. The learning software saves the model weights and log files when the training finishes.

#### Dataset generator

The dataset-generator is responsible of feeding the dataset to Keras. The images are read and augmented by the CPU in a generator function. This makes it possible for the CPU and GPU to work in parallel. The prepared images are placed in a queue, available for the Tensorflow library to process. The training and validation images are simultaneously fed by two different generators to two different queues. Managing the queue size can be



**Figure 4.2:** Example of resizing image. Note that the whole original image is fit in the resized image, not preserving aspect ratio.

crucial. The size of the two queues combined must both fit in main memory. The data generator is also responsible for the following tasks:

1. Normalize each image to have unit length and be centered at zero. However, centering does not involve looking at the pixels in the image. We assume that the full value range is used (0-255) for all colors. This assure that all images are normalized the same way.
2. Augment the images while preserving resolution. The following operations are performed:
  - Random zoom
  - Random shear
  - Random width shift
  - Random height shift
  - Random rotation

### Learning module

The learning module consists of two parts. One is responsible of building the neural net model, while the other is responsible of training it. The training program takes data generator and a model as input, in addition to the parameters of the training. The available parameters of the model in our implementation are:

- **Objective function:** The objective function is responsible of calculating the loss of the model. The loss is a metric to describe how well the model fit the current mini-batch.
- **Optimizer:** The optimizer is responsible of minimizing the objective function using back-propagation to change the model weights. The optimizer takes multiple parameters, depending on the specific optimizer



- **Callback functions:** The callback-functions are called between each batch of the training. These functions are responsible for logging and/or changing the parameters of the model during training, such as a learning rate schedule.

#### 4.1.4 Evaluation

The evaluation module is responsible for testing and analyzing the trained model. It takes the test data and the trained model as input. The evaluations are done in iPython notebook by the same reasoning as the dataset analysis.

The testing is done by predicting each test sample with the trained model. These predictions are stores in a table for analysis. Note that we save not only the top predictions, but also the predicted value for all the labels. The table has size  $n * m$ , where  $n$  is the number of samples and  $m$  is the number of classes.

##### Combining images to classify ad

As we have seen in section 3.4, we can enhance our model by combining the predictions within each subset of the data. The following functions are implemented to combine the predictions and make the final ad prediction.

- **Averaging:** The first method is to average the predictions within one bag of instances. Given the set  $X_i = \{x_1, x_2, \dots, x_n\}$  of images in ad  $i$ , where  $P(Y_j|X_i)$  is the prediction of class label  $Y_j$  on set  $X_i$ , and  $p(y_j|x_i)$  is the prediction of class label  $y_j$  on instance  $x_i$  given by our single instance classifier. We calculate the prediction as in equation 4.1.

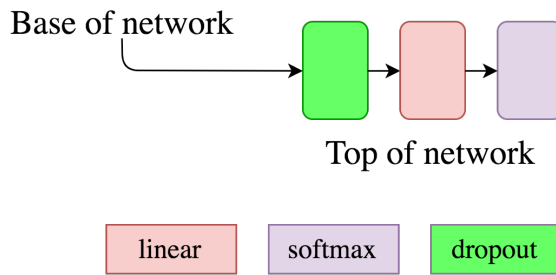
$$P(Y_j|X_i) = \frac{1}{n} \sum_{i=1}^n (p(y_j|x_i)) \quad (4.1)$$

- **Weighted averaging:** This method is similar to averaging the predictions. The set is now ordered to define the sequence  $X_i = [x_1, x_2, \dots, x_n]$  together with the weights  $W_i = [w_1, w_2, \dots, w_n]$  of same length. The prediction is then defined by equation 4.2.

$$P(Y_j|X_i) = \frac{1}{n} \sum_{i=1}^n (p(y_j|x_i) \cdot w_i) \quad (4.2)$$

- **Noisy-or:** The noisy-or model is explained in section 3.4 and defined in equation 3.11.
- **Majority voting:** Each of the individual predictions have  $n$  votes for its top  $n$  predictions. The final prediction is defined in 4.3, where  $U(x_i, y_j)$  is 1 if  $y_j$  is one of the top  $n$  predictions on image  $x_i$  and 0 otherwise.

$$P(Y_j|X_i) = \max_j \sum_i U(x_i, y_j) \quad (4.3)$$



**Figure 4.3:** The three top layers of the model. The "base of network" refers to the base of the Inception-v3 model described in section 3.2.6.

## 4.2 Neural Network Model Structure

The model for this project is a deep convolutional neural network (CNN), based on the Inception-v3 architecture described in section 3.2.6. Table 3.1 and figure 3.6 displays the original architecture of Inception-v3.

There were two main reasons for selecting the Inception-v3 architecture.

1. Inception-v3 has proven to be a state-of-the-art classifier for image classification [31].
2. Open source implementations of the Inception-v3 architecture is available in libraries such as Tensorflow and Keras. Additionally, pre-trained versions of the network makes it possible to make use of transfer learning.

The Inception-v3 architecture is designed for the ILSVRC2012 challenge on the ImageNet dataset. Each image would be classified with one out of 1000 classes. Because of this, the output layer has a total of 1000 nodes. We therefore have to modify it to fit our own classes. In addition to the difference in number of target classes, the class labels are different as well. The very last layers of the Inception-v3 model are trained to see the differences between animals and vehicles, but not the nuances that differ two car brands. Therefore, we have to substitute the top layers of the model.

We transfer the base of the Inception-v3 model, as defined in 3.6. In addition, we add a dropout layer, a fully connected layer and a final softmax layer with the same amount of outputs as we have number of classes.

# Experiments and Results

This section will describe the experiment process and contain all the necessary steps to reproduce the results. It can be regarded as a step to step guide to reproduce the results.

## 5.1 Experimental Plan

We deduced an experimental plan based on the background theory as well as the dataset analysis.

In the preface of the project, we did not yet have access to the Leboncoin dataset. Krause et. al. [19] provides a dataset of cars containing 16185 images of 196 classes of cars. The first tests of the model were therefor done on this dataset. The results are omitted from the report, mainly due the fact that this part of the research was about making everything run correctly, and not get good results.

The experimental plan is divided into three parts, the single instance classifier, multi instance classifier and an improved single instance classifier. We present the background reason for each experiment as well as the expectations of the experiment.

### 5.1.1 Single Instance Classifier

The single instance classifier refers to the neural network architecture described in section 4.2. The main experiments are listed below:

- **Experiment 1: Train the classifier on the newly added layers of the model.**

This experiment freezes the weights for all the original layers of the inception model. The experiment is originated in the theory of transfer learning (section 3.3). The idea is that the original neural network has learned the low level features in the images. Those features are therefore transferable to the new learning task. We are interested in knowing if the shallow top network is enough to perform at a high accuracy.

The expectations for this experiment is that the model will be able to converge and classify images to a certain extent. However, there are only two layers that are being

trained. We can therefore not expect the model to perform as well as the state of the art models.

- **Experiment 2: Train the last 5 inception blocks of the model.** <sup>1</sup>

The results from Experiment 1 were promising, but did not achieve the accuracy we want. The hypothesis is that only training the top layers does not improve the model a lot, as the trained network is too shallow. Deeper training should help the model to adapt the new dataset better, as mid-level features might be specific to our dataset.

The original intent in experiment 2 and 7 was to train the last 2 inception blocks of the model. However, an error in the Keras documentation led to more layers being trained. The error was reported, confirmed and fixed during the project. <sup>2</sup>.

### 5.1.2 Multiple Instance Classifier

The second part of the experiment is to combine the predictions on images originating from the same ad. The experiments are based on the background theory presented in section 3.4. However, since the images in each sales ad has an order, an additional experiment was deducted to account for this information.

The experiments in this section is done on the best single instance classification model found in section 5.1.1. There are four different experiments done to combine predictions. The experiments are listed below:

- **Experiment 3: Average the predictions of images originating from the same sales ad.**

This experiment uses the concept presented by Xu & Frank [35]. We assume that the individual image predictions are independent and contribute the same amount to the answer.

The expectation is that averaging the predictions will enhance the results, as we can see the erroneous predictions as noise in the model. Averaging predictions from multiple samples will also average the noise. If the noise is random, then the expected value of the average will converge to uniform as the instance count increases.

- **Experiment 4: Use the noisy-or model to combine the predictions of the images originating from the same ad**

The idea of this experiment is similar to the one in Experiment 3. However, as we have seen in section 3.4, the noisy-or model will increase the importance of confident predictions. This can also be seen as decreasing the weight of less confident predictions. The expectations is that the less confident pictures are the detail-oriented images, such as engine or backseat of the vehicle. These images can be hard to distinguish between classes.

---

<sup>1</sup>More precise, the first 172 layers were frozen, and the rest trained. The trained layers correspond to the last 4.5 inception blocks in addition to the newly added layers. See discussion later in this section.

<sup>2</sup>Github issue: <https://github.com/fchollet/keras/issues/6910> Github pull request: <https://github.com/fchollet/keras/pull/6918>

- **Experiment 5: Use weighted average to combine the predictions of the images originating from the same ad**

As we have seen in section 2.1, the images within one sales ad are ordered. We hypothesize that images earlier in the sequence are easier to classify than the last images. This is due to the fact that sales ads often show pictures from the outside of the car in the beginning of an ad, while the detail-oriented images appear last. In other words, the difficulty of the classification is not uniformly distributed within a class.

- **Experiment 6: Use voting to combine the predictions of the images originating from the same ad**

This experiment is an attempt to solve some of the problems that appeared because of the overconfident single instance classifier. We deduce a simple experiment, where each image contributes with votes towards a final prediction. The final prediction is simply a majority vote between the images. This experiment was inspired for the multi-attribute image classification problem of Sing and Garzon [27], discussed in section 3.4.

### 5.1.3 An Improved Model for Classification

The analysis of the results from Experiments 1-6 showed some possible shortcomings, as discussed in section 5.3.1 and 5.3.2. The single instance classifier was very confident of its predictions, which led to problems when the predictions were combined. Also, the single instance classifier showed a big gap between training accuracy and validation/testing accuracy. The problems are likely to originate in a slightly overfit model.

- **Experiment 7: Train the last 5 inception blocks of the model on augmented images.**

This experiment tries to solve two problems. First, we would not like the model to overfit. In addition, the gap between training accuracy and validation accuracy proposes that the model is unable to generalize well.

The proposed solution was to augment the images to produce a more versatile training set. At the same time, we hypothesize that the training accuracy will increase slower, as the model can not learn exact inputs. The expectations of this experiment is therefor increased accuracy on both the single instance classifier and the multi instance classifier.

## 5.2 Experimental Setup

This section will break down the setup of the experiments. First, it will introduce the common setup of all the experiments. Unless stated otherwise, the setup is the same for all the experiments. Further, the section is divided into the single instance classification task, the multi-instance classification task and the improved model. These two sections include all experiments from section 5.1, including all the parameters. The results of the individual experiments will be presented and discussed in section 5.3.

### 5.2.1 Hardware

The demands of processing power in machine learning can be high, especially as the datasets grow. The complexity of the model also affects the demands. However, hardware is expensive. Selecting the correct hardware is therefore a trade off between computing power and cost. Managing this balance can be difficult. We were provided with the opportunity to use Amazon Web Services (AWS) for computation and storage.

Training of deep neural networks has proven to be far more efficient on GPU's compared to conventional CPU architectures. The presence of GPU was therefore a major advantage in this project.

Storage space is also a factor that contributes to selecting the correct platform. The dataset in this project is close to 1TB. In addition, scaling and augmenting the dataset was a possibility. We therefore need to have the possibility to scale the storage space after demand.

The program runs on AWS as a P2 Accelerated Computing Instance, specifically the p2.xlarge model. It has one NVIDIA K80 GPU with 12GiB of GPU memory. The choice of computing service and model was made based on price and privacy options. The datasets was on mounted volumes in computing centers in Ireland. AWS makes it easy to attach, duplicate and rescale volumes if needed. We used in total 3 attached volumes. One for the original dataset, one for the scaled dataset and one for all the program code and log files.

### 5.2.2 Dataset

We chose to select a subset of the images because of the skewed distribution of ads per car brand, as represented in figure 2.3. The dataset was filtered by number of images per brand. Only brands with more than 10000 images were selected. Furthermore, 1500 sales ads per brand were selected. This was to make a balanced dataset.

The dataset was then divided into training, validation and test set. 60% went to the training set and 20% in both the validation and test set. All images within the same sales ad were placed in the same subset.

In total this yielded 19 brands with a total of  $19\text{brands} \cdot 1500\text{ads}/\text{brand} = 28500\text{ads}$ . The sizes of the training, validation and test set were then 17100, 5700 and 5700 respectively. The sets varied slightly in size regarding number of images due to the variable count of images per sales ad.

All images were prepared according to the description in section 4.1.2.

### 5.2.3 Single Instance Classifier

The Inception-v3 model was modified as described in section 4.2. The dropout layer in the model has a probability of retaining a unit of 0.5, as proposed in [29] for the last layers of the model. The fully connected layer has 1024 units, and uses ReLu as activate function. The final softmax layer has 19 target classes to fit our dataset.

All the experiments used RMSProp as optimizer with momentum  $\gamma = 0.9$  as suggested by the inventors of the function. The loss was calculated as categorical cross-entropy. The learning rate were set to the specific experiment. Some trail and error led to the learning

rates, however, no precise grid-search was done due to the time it takes to train a model and the time frame of the project. Unless stated otherwise, the batch-size was set to 100 and the validation size to 200<sup>3</sup>.

- **Experiment 1: Train the classifier on the newly added layers of the model**

The learning rate of all the layers in the base of the Inception-v3 model was set to 0. This is the same as freezing the layers, as the weights will not change during training. The learning rate for the top 3 layers were set to  $2e-4$ . The network was trained for 50 epochs, without an early stopping scheme or learning rate schedule.

- **Experiment 2: Train the last 5 inception blocks of the model.**

The first 172 layers<sup>4</sup> of the model were frozen and the rest trained. The learning rate was set to  $2e-5$  and number of epochs to 50. However, early stopping was configured to stop the training if the validation loss did not improve for 10 epochs.

## 5.2.4 Multiple Instance Classifier

All the experiments were set up as describe in section 4.1.4. The evaluations were done on the complete test set and tested on the model from Experiment 2.

Two of the experiments needs definitions for the hyper-parameters:

- **Experiment 5: Use weighted average to combine the predictions of the images originating from the same ad**

The weights used for the weighted average in Experiment 4 and 9 was set linearly in the range  $[1, 0.5]$ . For instance, if the sales ad had 6 images,  $W_i = [1, 0.9, 0.8, 0.7, 0.6, 0.5]$ .

- **Experiment 6: Use voting to combine the predictions of the images originating from the same ad**

Each of the images contribute one vote towards the final prediction. If multiple classes has the same number of votes, then a random class is selected out of them

## 5.2.5 An Improved Model for Classification

The setup for this experiment is very similar to Experiment 2. However, there are some minor differences when it comes to implementation, as we have to pay extra attention to how the CPU and GPU work parallel. The batch size is therefore slightly smaller.

- **Experiment 7: Train the last 5 inception blocks of the model on augmented images.**

The learning rate was set to the same as in Experiment 2. The number of epochs were set to 20 and no early stopping was configured. The batch size were set to 60.

The images were augmented with the following scheme:

---

<sup>3</sup>The whole validation set was not used due to a bug in the software. This will affect early stopping if used. However, the test results are performed on the complete test set.

<sup>4</sup>This number is implementation specific. It refers to the model found in Tensorflow 1.1.0.

- Random zoom: In the range  $[0.8, 1]$
- Random shear: Up to a factor of 0.1
- Random width shift: Up to 5%
- Random height shift: Up to 5%
- Random rotation: Up to 90 degrees in each direction
- Random horizontal flip: 50% of the images were flipped horizontally

We decided not to do color shifting, even though it is often done when augmenting images for machine learning. The reasoning behind this decision was that some cars have unique nuances of color. However, there are possible issues with the model learning a color as a brand, as this might not be a good generalization.



## 5.3 Experimental Results

Task	Top-1 accuracy	Top-5 accuracy
<b>Model from Experiment 2</b>		
Single image classification	0.757	0.915
Ad classification (average)	0.901	0.966
Ad classification (noisy-or)	0.898	0.966
Ad classification (weighted average)	0.890	-
Ad classification (voting)	0.868	-
<b>Model from Experiment 7</b>		
Single image classification	<b>0.823</b>	<b>0.947</b>
Ad classification (average)	<b>0.945</b>	<b>0.985</b>
Ad classification (noisy-or)	0.944	0.985
Ad classification (weighted average)	0.940	-
Ad classification (voting)	0.865	-

**Table 5.1:** Accuracy on the different learning tasks.

This section will provide the results of the individual experiments. First, we present the results of the single image classification task. Then we will show the result for combining predictions on the different classifiers. We have decided to test the top-1 as well as top-5 classification accuracy, as this is widely done in other classification tasks.

The results in this section is presented in the same order as the experimental plan and setup. An overview of the results are given in table 5.1.

### 5.3.1 Single Instance Classifier

The single instance classifier refers to the neural network model described in section 4.2.

#### Experiment 1: Train the classifier on the newly added layers of the model

The first experiment of the research was to train the newly added layers. Figure 5.1 shows the progression through 50 epochs of training. As we can see, the model converges to about 0.35 validation accuracy. Due to the low accuracy, we have omitted further analysis of the results.

The low accuracy can be explained by the shallowness of the trained model. The last two layers can not correctly distinguish the different brands.

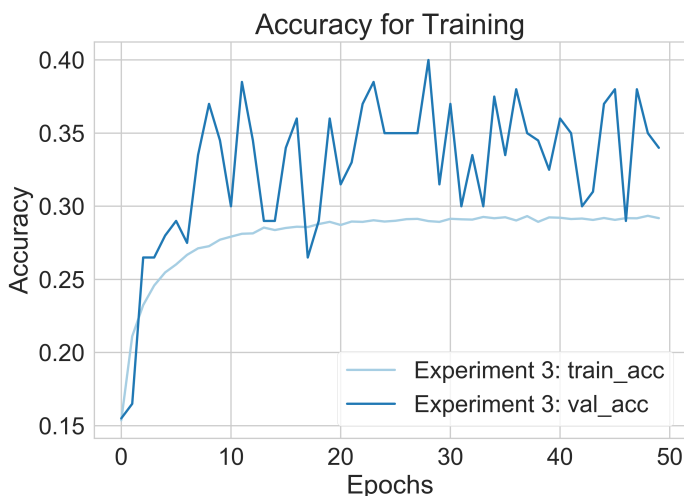


Figure 5.1: Experiment 1 training accuracy.

#### Experiment 2: Train the last 5 inception blocks of the model.

The model trained in Experiment 2 yielded an top-1 classification accuracy of 0.757 and top-5 classification accuracy of 0.915 on the test set. We can see the training and validation accuracy in figure 5.2. As we can see, there is a gap between the training and validation accuracy. The gap tells us that the model learns the input more than generalizing on the correct concepts.

Furthermore, we can look at figure 5.3. Each subplot shows the predictions for all images within an ad. As we can see the model seems to be overconfident about its predictions. The model is confident even when it is wrong. This might be a symptom of overfitting.

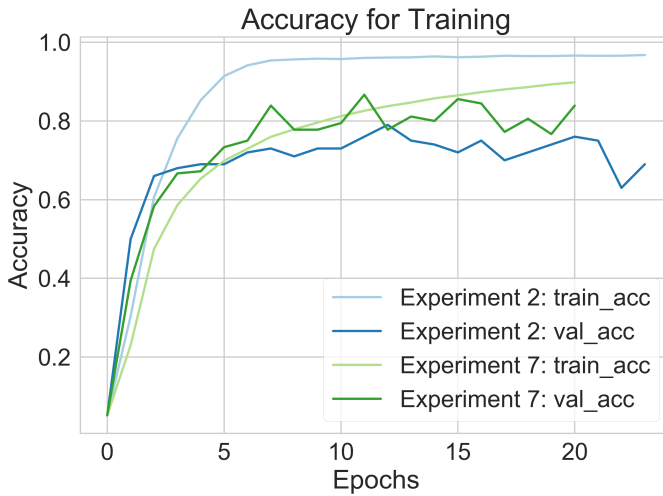


Figure 5.2: Experiment 2 and 7 training accuracy.

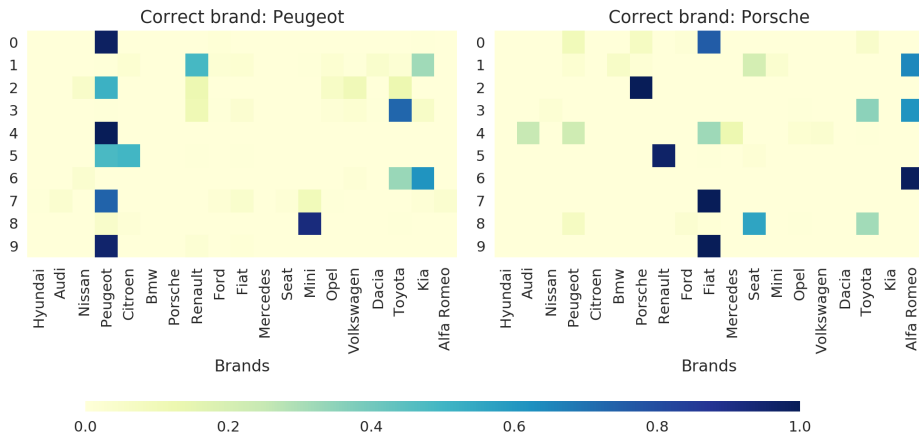
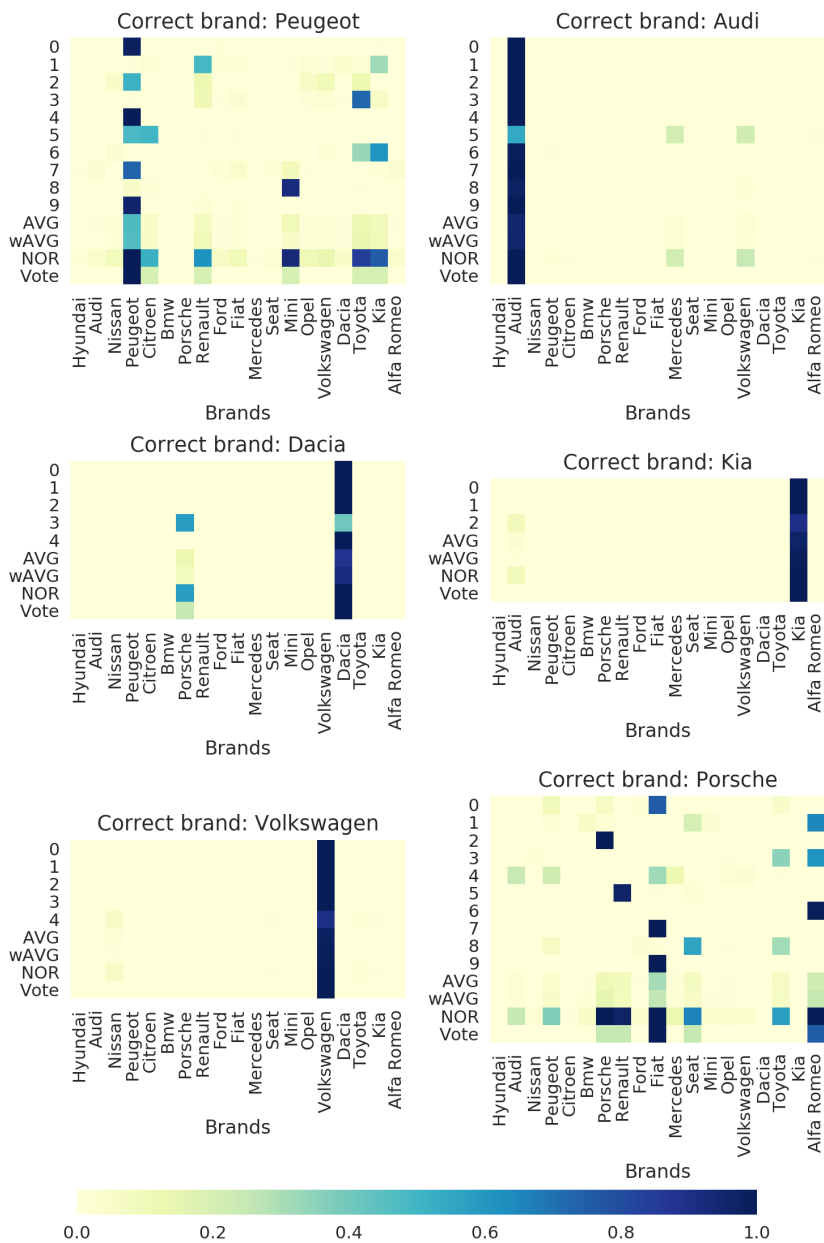


Figure 5.3: Visualization of the predictions of images in two different sales ads. The color represent the output of the model



**Figure 5.4:** Combining predictions on 6 randomly selected sales ads. Each row correspond to an image from the sales ad, except the last 4 rows which shows the combined predictions. The color represent the output from the model. The correct class label is shown in the title of each subplot. The abbreviations stands for average (AVG), weighted average (wAVG), noisy-or (NOR) and voting (Vote).

### 5.3.2 Multiple Instance Classifier

This section will show the results for the four conducted experiments regarding the combination of predictions. An overview of the results can be seen in table 5.1. The results will be discussed in view of the expectations of the experimental plan in section 5.1. An visual representation of the experiments in this section can be seen in figure 5.4.

#### **Experiment 3: Average the predictions of images originating from the same sales ad**

Averaging the predictions for all the images within the same ad gave a ad-wise top-1 accuracy of 0.901 and top-5 accuracy of 0.966. As expected, this accuracy is higher than the single instance classifier. As we can see in figure 5.4, the average function does a good job of catching the essence of the individual predictions.

#### **Experiment 4: Use the noisy-or model to combine the predictions of the images originating from the same ad**

The expectations for the noisy-or function were high. However, it yielded slightly worse results than the average function, an ad-wise top-1 accuracy of 0.898 and top-5 accuracy of 0.966. However, the difference is not significant.

Figure 5.4 reveals some possible problems. The bottom right subplot shows the predictions for a Porsche. As we can see, there are four classes that show a high confidence. The exact values of the prediction is shown in table 5.2. We have made the high values bold to clarify the problem.

The problem is a combination of the way the noisy-or model work and the overconfident single instance classifier. If only one of the images are wrongly classified with a high confidence, the combined prediction will be confident of this class. This means that a single image can mess up the prediction for the whole ad.

There is no way to tell a wrong confident classification for a correct confident classification with the current model. However, the problem is a result of the overconfident single instance classifier.

#### **Experiment 5: Use weighted average to combine the predictions of the images originating from the same ad**

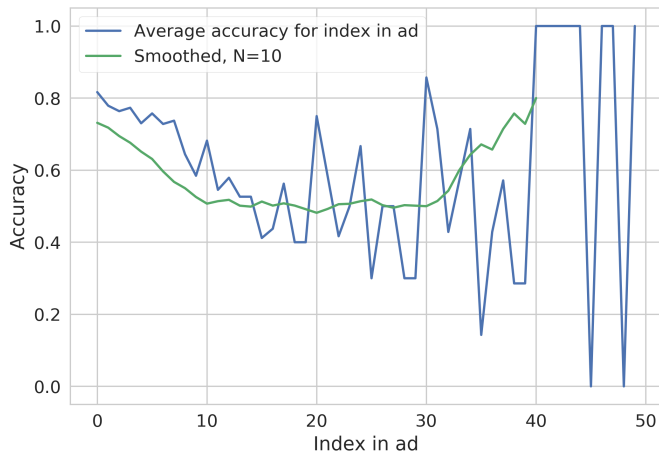
This experiment is based on the hypothesis that images earlier in the ad are more important than the last images. However, the results did not improve compared to the uniformly weighted average in Experiment 3. Experiment 5 gave an ad-wise top-1 accuracy of 0.890. Different weights were tested, but the more they differed from uniform weights, the worse the results got.

We can visualize the average accuracy of images at a certain index in the ads. Figure 5.5 shows us that images appearing early in an ad has a higher accuracy than later images. This confirms our hypothesis, however, it does not explain the low accuracy of the weighted average.

We can also see that the accuracy increases for high indexes. However, this might be a result of few data points, and should not contribute to much to the overall accuracy.

Prediction	Class
0.002	Hyundai
0.250	Audi
0.025	Nissan
0.369	Peugeot
0.008	Citroen
0.058	Bmw
<b>0.998</b>	Porsche
<b>0.969</b>	Renault
0.043	Ford
<b>1.000</b>	Fiat
0.131	Mercedes
0.658	Seat
0.035	Mini
0.042	Opel
0.035	Volkswagen
0.003	Dacia
0.581	Toyota
0.001	Kia
<b>0.999</b>	Alfa Romeo

**Table 5.2:** Noisy-or predictions on a sales ad.



**Figure 5.5:** Average accuracy versus index in ad.

Number of images	$P(X > n/2)$	$P(X \geq n/2)$
2	0.573	0.941
4	0.750	0.953
6	0.841	0.965
8	0.895	0.975
10	0.930	0.983

**Table 5.3:** Probability of getting 50% of the votes

**Experiment 6: Use voting to combine the predictions of the images originating from the same ad**

Voting gave an ad-wise top-1 accuracy of 0.868. The improvement of classification accuracy from the single instance classifier was smaller than expected.

We know that each image has a probability of 0.757 for being correctly classified from the single instance classifier. An ad-wise prediction of a label needs maximum 50% of the votes to be selected. If we consider the binary classification problem of predicting the correct label versus the wrong, we can model this as binomial distribution. We are interested in the probability that more than 50% of the ads are correctly classified. We can calculate this as in equation 5.1.  $X$  is the number of correctly classified images and  $n$  is the total number of images.

$$\begin{aligned}
 Pr(X > n/2) &= 1 - Pr(X \leq \lfloor n/2 \rfloor) \\
 &= 1 - \sum_{i=0}^{\lfloor n/2 \rfloor} \binom{n}{i} p^i (1-p)^{n-i}
 \end{aligned} \tag{5.1}$$

Table 5.3 shows the probabilities of getting 50% and more than 50% of the votes. Note that getting more than 50% of the votes guarantees that the label is selected. This is a strict calculation, as an ad with 10 images and 4 correctly labeled images still can get the correct majority vote if the remaining 6 votes are relatively uniformly distributed on the other labels. We see from the table that voting should yield a higher accuracy than we expected, considering the distribution in chapter 3. However, there is an assumption in the calculations that does not hold. The calculations assume that the images are equally difficult to classify independent on which ad they appear in. Section 5.3.3 addresses this assumption.

### 5.3.3 An Improved Model for Classification

This section will describe the results on the single instance classifier as well as updated results of the different experiments for the multi instance classifier.

#### **Experiment 7: Train the last 5 inception blocks of the model on augmented images.**

As expected, the model trained slower in this experiment than Experiment 2, as we can see in figure 5.2. However, the validation accuracy stays closer to the training accuracy, and also surpasses the experiment without augmented images. This behavior shows us that the model is not overfitting as much as before.

On the single instance classifier, the test accuracy was 0.823 and 0.947 on top-1-accuracy and top-5-accuracy respectively. This is a significant improvement, and an increase of 6.6 percent point. There were 5700 ads in total, which means that 5388 ads were correctly classified.

The predicted labels from the test set were plotted in a confusion matrix. The value of each cell visualize the fraction of images within the true class which were classifier into the predicted class. We can see the confusion matrix in figure 5.6a.

As we can see, the diagonal is prominent. This means that most of the images were correctly classified. However, the nuances of the false positives disappear because of the strong diagonal. Figure 5.6b shows the same confusion matrix without the diagonal. As we can see, the false positives are relatively uniformly distributed throughout the confusion matrix. However, some information can be induced. For instance, the model finds it difficult to distinguish between Kia and Hyundai.

We ran all the multiple instance classifier experiments on the new improved model. All the results are listed in table 5.1. The results improved for all multi instance experiments with the new single instance model. As we can see, uniform averaging still gives the best results, with an ad-wise classification accuracy of 0.945 and 0.985 for top-1 and top-5 respectively. However, noisy-or is only slightly worse.

A manual look at correctly and wrongly images gave an impression that the accuracy were biased regarding viewpoint. To verify this impression, we manually labeled 400 images (50/50 correct/wrong) by viewpoint from the test set. For each image, we wrote down if the image were correctly classified or not. The results are shown in table 5.4. Note that the dataset does not contain viewpoint metadata, so the a priori viewpoint distribution is unknown. However, we can estimate this distribution from the manually labeled images.

We should be careful to draw any hard conclusions by this analysis, as the sample is small. Also, stating that images of the front has  $12/16 = 3/4$  accuracy is wrong, since images with this viewpoint has different probability of appearing in the two categories (correct/wrong).

We define  $B$  as the viewpoint. We can calculate  $P(B)$  as  $P(B) = P(B|\text{Correct}) * P(\text{Correct}) + P(B|\text{Wrong}) * P(\text{Wrong})$ . We know  $P(\text{Correct})$  and  $P(\text{Wrong})$  from the classification accuracy of the model,  $P(\text{Correct}) = 0.823$ . We infer  $P(B|\text{Correct})$  from the values in the table. We can then use Bayes Theorem to calculate the probability that an image from a viewpoint is correctly classified,  $P(\text{Correct}|B)$ .

Pictures of trunks and engines are mostly classified wrong. This is also true, not surprisingly, for images that does not contain a car. The accuracy is significantly biased in



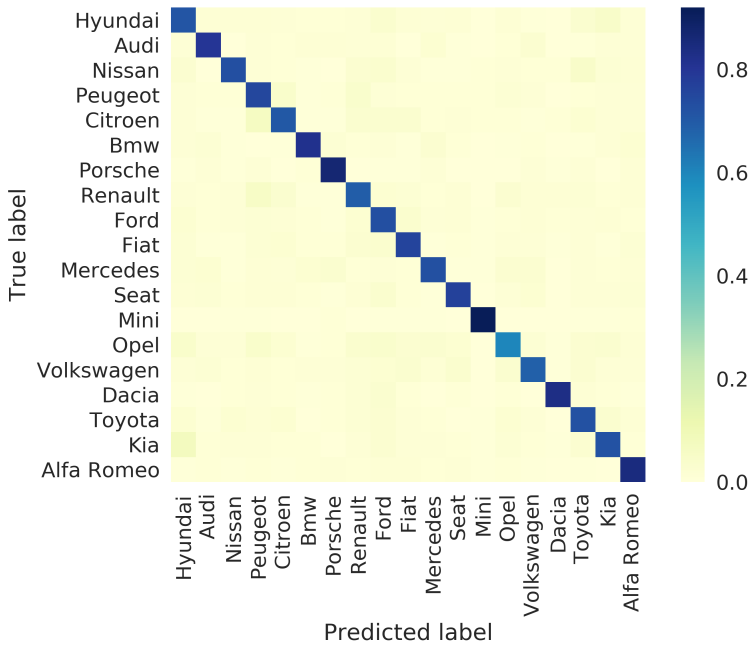
Viewpoint ( $B$ )	Correct	Wrong	$P(B)$	$P(B \text{Correct})$	$P(\text{Correct} B)$
Front	12	4	0.05	0.06	0.93
Side	13	16	0.07	0.07	0.79
Back	12	2	0.05	0.06	0.97
Front/Side	55	33	0.26	0.28	0.89
Back/Side	32	18	0.15	0.16	0.89
Trunk	1	15	0.02	0.01	<b>0.24</b>
Engine	0	10	0.01	0.0	<b>0.0</b>
Back seat	4	24	0.03	0.02	0.44
Front seat	21	30	0.11	0.11	0.76
Dashboard	30	6	0.13	0.15	0.96
Detail (inside)	16	20	0.08	0.08	0.79
Detail (outside)	4	10	0.03	0.02	0.65
No car in image	0	12	0.01	0.0	<b>0.0</b>
<b>Total</b>	<b>200</b>	<b>200</b>	<b>1</b>	<b>1</b>	-

**Table 5.4:** The viewpoint’s bias on accuracy. 200 correctly and 200 wrongly classified images were randomly selected from the test set. ”Front”, ”Side” and ”Back” refers to images taken directly from the direction. If the viewpoint is slightly different, the image falls into the category ”Front/Side” or ”Back/Side”.

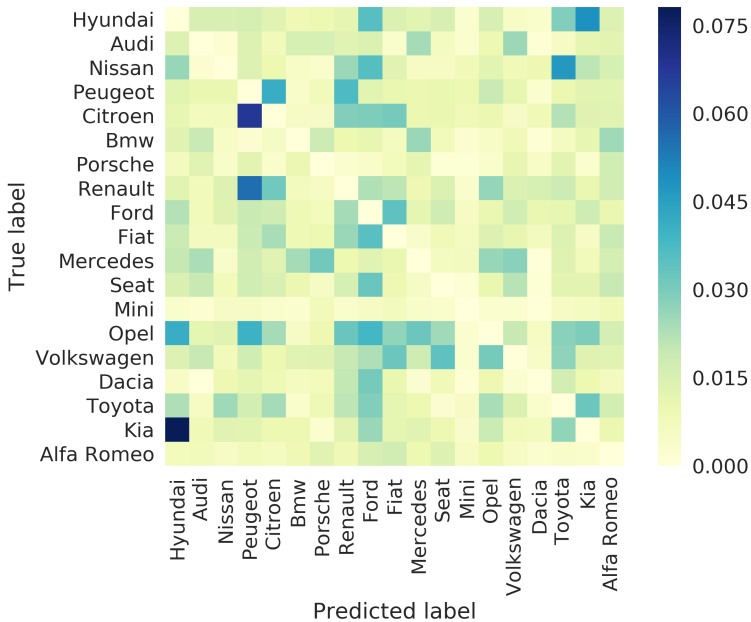
regards of viewpoint.

Another perspective to the results is to see how many correctly classifier images there are in an ad. Since the ads have different number of images, we plot the fraction of correctly classifier images in figure 5.7. If the ads were independent and identically in regards of the difficulties of classifying an image, then we would expect a distribution centered around the single classifier accuracy of 0.832. However, as we can see in figure 5.7a, most of the ads have images which were all correctly classified, while other had almost none. Figure 5.7b is the same histogram, but focused on the ads that had less than 50% correctly classified images. Almost 200 ads had less than 10% correctly. Not surprisingly, all of them were wrongly classified.

In addition, we expect the year model of the vehicle to affect the accuracy of the classification. Figure 5.8 shows the percentage of correctly classified ads in relation to the year model. As we can see, older cars has a less accuracy than newer cars. This is probably because there are less data for early years.

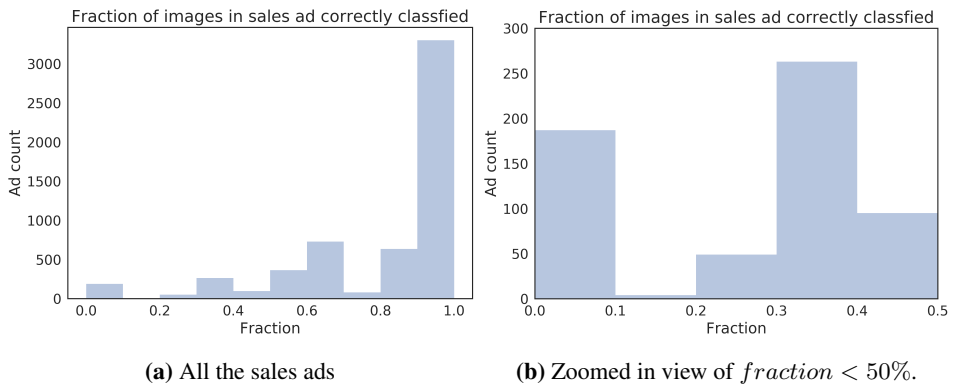


(a) Confusion matrix

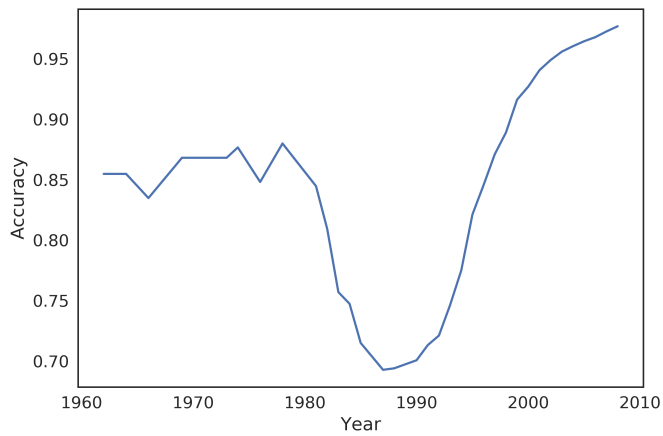


(b) Confusion matrix where the diagonal is removed to better see nuances.

**Figure 5.6:** Confusion matrices for the single instance classification task.



**Figure 5.7:** Histograms of the fraction of images in an ad that were correctly classified.



**Figure 5.8:** Accuracy of sales ads correctly classified in regards of model year. A running mean of 10 years is used, since there are few ads with a specific year in years before 1990.



# Discussion

This section will provide a discussion about the experimental results of the project. It will discuss possible shortcomings, problems with the classifier and the research process in total.

## 6.1 Performance

This section will evaluate the results across all the different experiments and investigate possible explanations for the results. The section is divided into evaluation of the single instance classifier and the multi instance classifier. However, the latter refers to the former section to provide reliable explanations.

### 6.1.1 Single Instance Learning

It is important to validate the performance of the classifier. It is however difficult to quantify how good the solution is without a good benchmark test. One possible benchmark is the ILSVR 2012 classification challenge. The inception-v3 model achieved a classification error of 21.2%, top-1 and 5.6% top-5 error for single crop image [31].

The benchmark results in [31] are comparable to our results of 17.3% and 5.3% classification error for the single image classification task. However, the results presented for the Inception-v3 were on the ImageNet ILSVRC 2012 classification task containing a total of 1000 classes. Thus we would expect or model to perform at least as well as the benchmark test.

The comparison between Experiment 1 and 2 shows that we need to train more layers than the newly added when we perform transfer learning. This is not surprising. Ideally, we should fine-tune all the layers after we have trained the network. However, training a deep network like the Inception-v3 is demanding in regards of computational time.

Another important factor to discuss is the quality of the dataset. Some of the pictures do not contain a car at all. Table 5.4 is based on a small sample, but around 1% of the

images are of this type. We cannot expect to correctly classify images without a car. Similarly, the images of the trunk and the engine of the car are classified with a low accuracy. These images are difficult for humans to classify as well. However, the model is good at picking up small details that humans may overlook.

### 6.1.2 Multiple Instance Learning

Combining the predictions on the different images improved the results substantially. There were no significant difference in averaging the predictions and using the noisy-or method. However, a single instance classification accuracy of 0.823 makes an expectation that we should be able to classify ads with a higher accuracy than we do.

Although the accuracy for ads were higher than for individual images, it did not increase very much. The explanation for this is probably that some ads are more difficult to classify than others. In other words, some ads contain difficult images. Combining them is not going to help. Figure 5.4 visualizes this. 2 of the 6 ads have a much more widespread predictions than the other 4 ads.

Consider the scenario with a test set of 10 sales ads, each with 10 images. The single instance classification model has an accuracy of 0.7. If all the wrongly classified images are placed in 3 of the ads, then the ad-wise classification will also have an accuracy of 0.7. This might explain why combining the images into a multi-instance model did not improve the result more than it did.

If we look at figure 5.3, we can see that the single instance classifier is very confident of its predictions. This is true even if the prediction is wrong. This seems to be a case of overfitting, but the overfitting does not affect the classification accuracy. Usually a certain amount of overfitting can be tolerated, or even necessary for the model to fully converge. The problem arises when we try to combine the predictions from different images. A smoother distribution of the predictions may improve the combined classification. In other words, the model would be under-fit. The correct class would appear when we average the different predictions. We hypothesize that it would be preferable to intentionally under-fit the single instance classifier to produce a better multi-instance classifier.

# Conclusion and Future Work

This chapter will summarize the research project and conclude on the results and process. It will highlight the contributions of the project and present some of the aspects that should be investigated further.

## 7.1 Conclusion

In this thesis we show that we can classify sequences of images using a single instance classifier and average the results. Even though the dataset is created by users and contains noisy images, the classification accuracy is high. Averaging the predictions of individual images increases the accuracy substantially. We conclude that we can achieve a high accuracy on sales ad classification, only based on the images.

The dataset contains noise in the sense of unusual images which are hard to classify. The main challenge is that the noise is not evenly distributed between the ads. This affects the result when we combine the images.

Averaging the predictions proves to be the most accurate combiner of predictions, as questioned in Research Question 2. The main challenge when it comes to combining the images is that some ads contains a statistically higher fraction of difficult images. This may be because of a vehicle in bad condition, an old vehicle or that the images are not available.

## 7.2 Contributions

This theses shows that multiple instance classification can be used with a conventional single instance classifier and still achieve high accuracy. This method can be applied on marketplace websites, on medical imagery and video without increasing the complexity of training the model. If a trained model is already available, the predicted output can be combined without modifying or retraining the original model.

## 7.3 Future Work

There are multiple aspects of the projects that were not investigated due to a limited time frame for the project. This section will highlight the most prominent possible improvements.

The experiments in this process was simplified by excluding large parts of the dataset. The subset of the dataset that were selected was done in order to balance the dataset and make the possible problems fewer. However, more data means that the model can generalize better if the model is designed correctly.

The bias of the viewpoint of the image on accuracy shows us that this information is valuable. One method to utilize this information is to train an additional model which identify the viewpoint. This information can be used to weight the average when we combine the predictions. A model that identify the viewpoint should also identify if there isn't a car in the image. We could either train this new model on a different dataset that already includes viewpoint information, such as [36], or we could manually label our training set. This could be done using services such as Mechanical Turk from Amazon to quickly label thousands of images.

Although this project combines bags of instances to an ad-wise prediction, it does not utilize the information about the groups during training. An interesting addition would be to extend the model to calculate loss across bags of instances. We hypothesize that this would increase the ad-wise prediction accuracy.

Consider the following example. Each ad has 10 images. 5 of these images are of the actual object we try to classify, while the 5 other images are there to simply distract the model. A custom loss function could potentially understand the underlying structure of the dataset better, and ignore the random images that are only there to distract. A custom loss function could calculate bag-wise loss as the average loss for the top 50% of the images. This would in effect ignore difficult images and make the model only predict the easiest images.



# Bibliography

- [1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [2] S. Abrahams, D. Hafner, E. Erwitte, and A. Scarpinelli. *TensorFlow for Machine Intelligence*. Bleeding Edge Press, 2016.
- [3] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. Visual tracking with online multiple instance learning. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 983–990. IEEE, 2009.
- [4] Y-Lan Boureau, Francis Bach, Yann LeCun, and Jean Ponce. Learning mid-level features for recognition. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2559–2566. IEEE, 2010.
- [5] François Chollet et al. Keras. <https://github.com/fchollet/keras>, 2015.
- [6] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.
- [7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

- 
- [8] Anaconda Software Distribution. Anaconda: Computer software. vers. 2-2.4.0. <https://continuum.io>, 2016.
- [9] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *Icml*, volume 32, pages 647–655, 2014.
- [10] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, Berlin, 2001.
- [11] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.
- [12] Schibsted Media Group. Annual report 2016. <http://hugin.info/131/R/2096898/793519.pdf>, 2015. Accessed: 2017-06-13.
- [13] Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12(10):993–1001, 1990.
- [14] Simon S Haykin, Simon S Haykin, Simon S Haykin, and Simon S Haykin. *Neural networks and learning machines*, volume 3. Pearson Upper Saddle River, NJ, USA:, 2009.
- [15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *European Conference on Computer Vision*, pages 346–361. Springer, 2014.
- [16] Geoffrey Hinton, NiRsh Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent.
- [17] Svein-Erik Holesvein. Norges 25 største nettsted. *Teknisk Ukeblad*, 2012. [Online; accessed 9-May-2017].
- [18] Andrej Karpathy. Stanford University CS231n: Convolutional Neural Networks for Visual Recognition. 2015.
- [19] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 554–561, 2013.
- [20] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.
- [21] Yann LeCun, LD Jackel, Léon Bottou, Corinna Cortes, John S Denker, Harris Drucker, Isabelle Guyon, UA Muller, E Sackinger, Patrice Simard, et al. Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural networks: the statistical mechanics perspective*, 261:276, 1995.

- 
- [22] Andrew Ng. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.
- [23] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda. *Queue*, 6(2):40–53, March 2008.
- [24] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1717–1724, 2014.
- [25] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.
- [26] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.
- [27] Diveesh Singh and Pedro Garzon. Using convolutional neural networks and transfer learning to perform yelp restaurant photo classification.
- [28] A Smola and Vishwanathan S.V.N. *Introduction to Machine Learning*. Cambridge University Press, 2008.
- [29] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
- [30] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.
- [31] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.
- [32] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.
- [33] Paul Viola, John C Platt, Cha Zhang, et al. Multiple instance boosting for object detection. In *NIPS*, volume 2, page 5, 2005.
- [34] Sun-Chong Wang. *Artificial Neural Network*, pages 81–100. Springer US, Boston, MA, 2003.
- [35] Xin Xu and Eibe Frank. Logistic regression and boosting for labeled bags of instances. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 272–281. Springer, 2004.

- 
- [36] Linjie Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. A large-scale car dataset for fine-grained categorization and verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3973–3981, 2015.
- [37] B Yegnanarayana. *Artificial neural networks*. PHI Learning Pvt. Ltd., 2009.
- [38] Manel Zoghlami, Sabeur Aridhi, Haitham Sghaier, Mondher Maddouri, and Engelbert Mephu Nguifo. A multiple instance learning approach for sequence data with across bag dependencies. *arXiv preprint arXiv:1602.00163*, 2016.

Appendix **A**

## Example of Leboncoin car ad


---

---

Leboncoin ACCUEIL DÉPOSER UNE ANNONCE OFFRES DEMANDES MES ANNONCES BOUTIQUES Se connecter

Accueil > Marne > Voitures > Mégane

### Mégane



leboncoin

3 photos disponibles

Mise en ligne le 16 juin

KELLY

Prix	3 800 €
Ville	
Marque	Renault
Modèle	Megane
Année-modèle	2008
Kilométrage	
Carburant	
Boîte de vitesse	

Description :

[Sauvegarder l'annonce](#)
[Signaler l'annonce](#)
[Partager l'annonce](#)

[Voir le numéro](#)
[Envoyer un email](#)

Gérer votre annonce

[Mettre en avant](#)
[Remonter en tête de liste](#)

[Modifier l'annonce](#)
[Supprimer l'annonce](#)

**Figure A.1:** An example of how car ads are displayed on the Leboncoin website.