NTNU
Norwegian University of
Science and Technology

Combining Image and Depth Data for Efficient Semantic Segmentation

Lars Erik Junge

Master of Science in Cybernetics and Robotics
Submission date: June 2018
Supervisor: Kristin Ytterstad Pettersen, ITK
Co-supervisor: Idar Dyrdal, FFI

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Problem formulation

Thesis description:

Scene understanding is necessary for unmanned ground vehicles (UGVs) to operate autonomously. By equipping an UGV with sensors, the vehicle can obtain an overview of its surroundings. This information can be used to build up a model of the scene, describing where it is feasible for the UGV to drive and where there are obstacles to be avoided. The focus of this project is on how cameras can be used to obtain scene understanding.

Deep convolutional neural networks (CNNs) have been successfully applied to several tasks related to scene understanding. Semantic segmentation is one such task, where an image is segmented into relevant classes. While great performance has been achieved by working only with RGB data, the quality of the segmentation may potentially be increased by including information captured from multiple sensor modalities. This project investigates how depth information can be fused with color information to perform semantic segmentation.

The following tasks should be considered:

1. Perform a literature review of convolutional neural networks applied to semantic segmentation, and how these networks can be extended to incorporate depth information.
2. Evaluate relevant network architectures and approaches, and propose a suitable solution for UGV scene understanding.
3. Implement a convolutional neural network, and perform training on a relevant dataset.
4. Evaluate the performance, present the results, and discuss limitations and challenges.

Start date: 08.01.2018
Submission deadline: 04.06.2018
Thesis performed at: Department of Engineering Cybernetics, NTNU
Supervisor: Professor Kristin Ytterstad Pettersen, NTNU
Co-supervisor: Idar Dyrdal, FFI

Abstract

Unmanned ground vehicles (UGVs) and other autonomous systems rely on sensors to understand their environments. These systems are often equipped with cameras which capture detailed descriptions of surrounding scenes. Computer vision systems which are concerned with extracting meaningful information from digital images are frequently used for scene understanding purposes for such systems. Modern deep learning techniques, and in particular convolutional neural networks (CNNs), have been successfully applied to analyzing and understanding images. CNN models are state-of-the-art for solving computer vision tasks such as image classification and semantic segmentation. Semantic segmentation involves classifying each pixel in an image as belonging to one of a set of classes. Modern CNN models have achieved impressive results on popular benchmarks for semantic segmentation when working with RGB image inputs. Incorporating data from other sensor modalities has the potential to improve perception capability further.

This thesis studies how ENet, a CNN model for real-time semantic segmentation from RGB inputs, can be extended to incorporate depth information. The network is modified by adding a feature extraction branch, which learns features from depth images. The depth features are fused into the feature maps from the RGB feature extraction branch at several points. The fusion is implemented as element-wise summation layers, placed throughout the encoder part of the network. Two new variants of the architecture are proposed which fuse features one and three times. The performance of both models are compared with the baseline ENet model which operates on only RGB inputs. We use two datasets to assess the performance of the different models. First we benchmark on the popular Cityscapes dataset to evaluate performance in urban scenes. A smaller dataset from forested scenes, the Freiburg forest dataset, is used to assess potential in more challenging off-road environments. The models are evaluated in terms of segmentation quality, using common metrics, as well as in terms of efficiency.

Fusing depth and RGB features at one location gave poorer segmentation quality than the baseline model which operates only on RGB inputs. Performance was improved by fusing depth and RGB features at several locations. The improvement is most noticeable for segmentation of spatially small classes, which the baseline model struggles with. The most conclusive results stem from the experiments performed on the Cityscapes dataset. The extended ENet model with improved results has a significantly slower inference speed, however the model remains small in size compared to other models.

Sammendrag

Ubemmanede bakkefartøy (unmanned ground vehicles - UGVs) og andre autonome systemer er avhengig av sensorer for å forstå omgivelsene sine. Slike systemer er ofte utstyrt med kameraer som kan gi detaljerte beskrivelser av omgivelsene. Datasynsystemer, som har som oppgave å hente ut meningsfull informasjon fra bilder, brukes ofte for sceneforståelsesformål. Moderne teknikker som bruker dyp læring, og spesielt konvolusjonelle nevrale nettverk (convolutional neural networks - CNNs) har vist imponerende resultater når det kommer til å analysere og forstå bilder. CNN-modeller er de beste tilnærmingene for å løse datasynproblemer som bildeklassifikasjon og semantisk segmentering. Semantisk segmentering innebærer å klassifisere alle piksler i et bilde som å høre til én av et sett med klasser. Moderne CNN-modeller som bruker RGB bilder har oppnådd imponerende resultater på populære benchmarks for semantisk segmentering. Ved å inkludere data fra andre sensortyper kan man potensielt forbedre ytelsen ytterligere.

Denne oppgaven studerer hvordan ENet, en CNN-modell for semantisk segmentering i sanntid av RGB-bilder, kan utvides til å inkludere dybdeinformasjon. Nettverket modifiseres ved å legge til en ekstra gren for uthenting av dybdeegenskaper (features) fra dybdebilder. Dybdeegenskapene slås sammen med RGB-egenskapene flere steder. Sammenslåingen implementeres som elementvise summasjonslag, som plasseres flere steder i enkoder-delen av nettverket. To nye varianter av nettverket presenteres, som slår sammen egenskaper henholdsvis én og tre ganger. Ytelsen til begge modellene sammelignes med grunnmodellen som bare bruker RGB-bilder. Vi bruker to datasett for å evaluere ytelsen til modellene. Først bruker vi det populære datasettet Cityscapes for å vurdere ytelsen i byscener. Et mindre datasett fra skogsscener, Freiburg forest, brukes for å vurdere ytelsen i mer utfordrende terreng. Modellene evalueres både basert på segmenteringskvalitet ved å bruke vanlige mål, samt basert på effektivitet.

Å kombinere dybde- og bildeegenskaper i ett punkt ga dårligere resultater enn grunnmodellen som bare bruker RGB bilder. Ytelsen ble forbedret ved å kombinere dybde- og bildeegenskaper i flere punkter. Forbedringen er mest synlig for segmenteringen av små klasser, som grunnmodellen sliter med. De fleste resultatene stammer fra eksperimenter på Cityscapes-datasettet. Den utvidede ENet-modellen som ga bedre resultater har en betydelig tregere inferenstid, men modellen forblir liten og effektiv sammenlignet med andre modeller.

Preface

This master's thesis is submitted as part of the requirements for the engineering cybernetics M.Sc. degree. It has been carried out at the Department of Engineering Cybernetics, at the Norwegian University of Science and Technology during the spring of 2018. The contributions of this thesis are within the area of practical applications of semantic segmentation from multimodal data. The background and contributions of the project are described in section 1.3 Background and contributions.

I would like to thank my supervisor Professor Kristin Ytterstad Pettersen and my co-supervisor from FFI, Idar Dyrdal for helpful advice and guidance throughout this project. I would also like to thank my family for all their support and encouragement throughout my years as a student in Trondheim.

Trondheim, June 2018

Lars Erik Junge

Table of Contents

Problem formulation i

Abstract iii

Sammendrag iv

Preface v

Table of Contents vii

List of Tables xi

List of Figures xiv

Abbreviations xv

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Problem description and goals . . . 2
- 1.3 Background and contributions . . . 3
- 1.4 Outline . . . 3

2 Theoretical background 5
- 2.1 Computer vision and image processing . . . 5
  - 2.1.1 Basic image processing - convolution . . . 6
  - 2.1.2 Computer vision tasks . . . 7
- 2.2 Deep learning and machine learning . . . 10
- 2.3 Artificial neural networks . . . 11
  - 2.3.1 Neurons and layers . . . 12
  - 2.3.2 Activation functions . . . 14
  - 2.3.3 Learning . . . 15
- 2.4 Convolutional neural networks . . . 19
  - 2.4.1 A brief history of CNNs . . . 19
  - 2.4.2 Distinguishing features . . . 20
  - 2.4.3 The layers of a CNN . . . 21

2.4.4 Training . . . 24
2.4.5 Architectural approaches . . . 24

3 Related work **27**
3.1 Deep learning applied to semantic segmentation . . . 27
3.1.1 Fully Convolutional Networks . . . 28
3.1.2 Encoder-decoder architectures . . . 29
3.1.3 Dilated convolutions . . . 32
3.2 Multimodal deep learning . . . 33
3.2.1 Multimodal semantic segmentation using RGB-D data . . . 33

4 Method and implementation **37**
4.1 Overall approach . . . 37
4.1.1 Choice of base architecture . . . 37
4.2 The ENet architecture . . . 39
4.2.1 Multimodal extension . . . 40
4.2.2 Fusion strategy . . . 42
4.3 Learning description . . . 44
4.4 Implementation . . . 45

5 Experiments **47**
5.1 Datasets . . . 47
5.1.1 Cityscapes dataset . . . 48
5.1.2 Freiburg forest dataset . . . 49
5.1.3 Dataset comparison . . . 50
5.2 Experimental design . . . 51
5.2.1 Training configuration . . . 51
5.3 Evaluation metrics . . . 53
5.3.1 Quantitative segmentation metrics . . . 53
5.3.2 Qualitative evaluation . . . 54
5.3.3 Performance metrics . . . 55

6 Results and discussion **57**
6.1 Cityscapes dataset . . . 57
6.1.1 Training results . . . 57
6.1.2 Quantitative results . . . 59
6.1.3 Qualitative results . . . 61
6.2 Freiburg forest dataset . . . 63
6.2.1 Training results . . . 63
6.2.2 Quantitative results . . . 63
6.2.3 Qualitative results . . . 65
6.3 Feature fusion . . . 67
6.4 Performance results . . . 71
6.5 Discussion . . . 71

7 Conclusion and further work **73**
7.1 Overview 73
7.2 Conclusion 74
7.3 Future work 74

Bibliography **77**

List of Tables

4.1 The ENet architecture 39
4.2 System specifications 46

5.1 Description of the network models 51
5.2 Dataset partitioning and image resolution. 51
5.3 Hyperparameters for training 52

6.1 Aggregated test results for the Cityscapes dataset. 59
6.2 Class-wise IoU results for the Cityscapes dataset 60
6.3 Aggregated test results for the Freiburg forest dataset 63
6.4 Class-wise IoU results on the Freiburg forest dataset 65
6.5 Performance results for the networks 71

List of Figures

2.1 Representation of a digital image. 6
2.2 Using convolution for edge detection. 7
2.3 Traditional and deep learning approach to image classification. 8
2.4 Moving from classification to semantic segmentation 9
2.5 Neurons and layers in ANNs 12
2.6 Popular activation functions 14
2.7 The LeNet-5 architecture. 20
2.8 Arrangement of neurons in a CNN. 21
2.9 Multiple neurons connected to the same region. 22
2.10 Example of max pooling. 23
2.11 Generic CNN architecture for classification. 25
2.12 The Inception module from the GoogLeNet architecture. 26
2.13 The residual block from the ResNet architecture. 26

3.1 Convolutionalizing classification networks. 28
3.2 A convolution and its associated transposed convolution. 29
3.3 The DeconvNet architecture. 30
3.4 Illustration of max pooling and unpooling. 30
3.5 The U-Net architecture 31
3.6 A 3×3 filter with various dilation rates. 32
3.7 Environmental challenges 34
3.8 Early and late fusion 35
3.9 The FuseNet architecture. 36

4.1 Accuracy per parameter for various networks. 38
4.2 ENet bottleneck modules 40
4.3 The proposed architecture and intermediate volume sizes. 41
4.4 The effect of the proposed fusion strategy 44
4.5 A Caffe model and the visualization as a computation graph. 45

5.1 Example data from the Cityscapes dataset 48
5.2 Cityscapes class distribution 49
5.3 Example data from the Freiburg forest dataset. 50

6.1 Training curves for the networks trained on the Cityscapes dataset. 58
6.2 Result from the Cityscapes dataset 61
6.3 Result from the Cityscapes dataset 62
6.4 Result from the Cityscapes dataset 62
6.5 Training curves for the networks trained on the Freiburg forest dataset. . . 64
6.6 Result from the Freiburg forest dataset 66
6.7 Result from the Freiburg forest dataset 67
6.8 The depth image for the example in figure 6.7 67
6.9 Input image for the feature fusion analysis. 68
6.10 Activations at the first fusion point 69
6.11 ENet-RGB decoder output. 70
6.12 ENet-MF decoder output. 70

Abbreviations

ANN	Artificial neural network
BN	Batch normalization
CNN	Convolutional neural network
DCNN	Deep convolutional neural network
GPU	Graphics processing unit
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IoU	Intersection over union
MIoU	Mean intersection over union
ReLU	Rectified linear unit
SGD	Stochastic gradient descent
SGM	Semi-global matching
UGV	Unmanned ground vehicle

Chapter 1

Introduction

This chapter gives an introduction to the thesis. The first section presents the motivation for the work presented in this thesis. This is followed by a description of the problem to be solved, and the goals of the thesis. Next the background for the work and the main contributions of are presented, and finally an outline of the report is given.

1.1 Motivation

Autonomous vehicles have received much attention and generated a great deal of discussion in recent years. Big companies like Google and Tesla have spent a considerable amount of resources developing relevant technology, and their self-driving cars have already taken to the roads. Advocates of autonomous vehicles promise reduced mobility costs, increased safety and improved mobility. The field is actively being researched and the technology is improving rapidly. Most research has gone into driving in structured environments such as urban scenes, partly due to commercial interest. While driving off-road is generally considered more challenging, performance of autonomous vehicles operating in this domain is also improving steadily. Autonomous vehicles operating in the terrain have great potential, and may aid in tasks such as rescue and search missions or military assignments.

Any unmanned ground vehicle (UGV) that operates autonomously, either on road or in the terrain, needs to be equipped with sensors that enable it to sense its environment. The scene understanding system of the vehicle is tasked with processing information from the sensors and building a model of the surroundings. This model can in turn be used to carry out tasks such as path planning in a safe and efficient manner. Various types of sensors are relevant for scene understanding systems, and the use of cameras has received much attention. Cameras are relatively low-cost, yet they can capture detailed information about the scenes. Image data from cameras mounted on the vehicle can be used to classify different parts of the surroundings. Stereo camera systems can also capture dense depth maps which provides 3D information about the environment.

Gaining high-level understanding from digital images falls under the field of computer vision. In recent years, modern deep learning techniques have been successfully applied to several visual scene understanding tasks. These approaches are considered to be state of the art in most computer vision tasks, and generally the go-to method. Deep learning rose to prominence when deep convolutional neural networks (CNNs) showed their strength at image classification. The most successful modern networks can take an input image and classify the content at the same level as humans, and sometimes even better. After the initial success of classification CNNs, these networks have been successfully adapted to other relevant perception tasks such as semantic segmentation. Instead of assigning a single label such as "car" to an image, a label is assigned to every pixel. This is a very powerful tool for scene understanding systems for autonomous vehicles. The images from a camera mounted on a car can be segmented into classes such as "pedestrian", "road" and "car", which promotes safe and efficient operation.

One of the main challenges for scene understanding systems using passive optical sensors, is the changing environmental conditions both throughout the day, and across seasons. From the viewpoint of a camera, a scene may look completely different in the summer when compared to the winter. The lack of light may also pose problems when the system is operating during the night. Another challenge arises from the fact that CNNs are often trained on datasets which do not capture such changes. This means that good results on offline benchmarks, do not necessarily transfer to good performance in the real-world. A novel solution is to create a larger training set which includes different types of environmental conditions. However, this approach is both time-consuming and inefficient, especially for data for semantic segmentation. Deep learning approaches need large datasets with densely annotated data which takes a long time to generate.

A more efficient approach to improve the robustness of the scene understanding, is to include one or more additional sensor types. Different sensor modalities capture different types of information, which can complement each other. Deep learning techniques applied to semantic segmentation have typically focused on only working with RGB images. One extension is to include depth information, which may be captured either with dedicated depth sensors or stereo cameras. The depth data can be used as additional input to scene understanding systems to improve performance. Recent related work has shown promising results, yet the optimal technique for combining different types of data remains an open question.

1.2 Problem description and goals

The overall goal of this project is to develop a system for performing semantic segmentation, based on image and depth information. The system should be based on deep learning techniques, which currently are the best-performing methods for this task. Convolutional neural networks in particular are considered to be state of the art when the input is only RGB images. The work aims to extend one such CNN architecture to incorporate depth information. Experiments should be performed to reveal how well the system performs compared to approaches based on only RGB inputs.

The developed system should be suitable for practical scene understanding applications. One of the highest priorities for such applications is real-time execution time. When developing a system for scene understanding, and in particular semantic segmentation, factors such as accuracy, speed and memory requirements need to be balanced. State-of-the-art methods achieve impressive results in terms of accuracy, often at the expenses of efficiency. The main focus of this thesis will be on implementing a system which is efficient in terms of speed and memory requirements. Thus the system will be evaluated both in terms of segmentation accuracy, but also processing speed and memory requirements.

1.3 Background and contributions

The background for the work in this thesis is the ENet model introduced by Paszke et al. [1]. It is a CNN model designed for real-time semantic segmentation which operates only on RGB inputs. ENet was designed with efficiency in mind, and has a small number of model parameters, while also having very fast inference speeds.

This thesis makes contributions to the area of practical applications of semantic segmentation from multimodal data, by proposing an extended ENet model which operates on RGB and depth data concurrently. Techniques for extending the architecture, while maintaining efficient performance are identified, incorporated and evaluated. The thesis provides insight into which approach for combining RGB and depth data is best suited for practical applications.

1.4 Outline

This thesis is organized in seven chapters. A short description of the contents in each chapter is given below:

Chapter 1 presents the motivation for the thesis, as well as the problem description and goals for the work. It also describes the background and contributions.
Chapter 2 presents the background theory for this project. The fields of computer vision and image processing are introduced, and deep learning methods for solving the computer vision task of image classification are presented.
Chapter 3 presents related work on how semantic segmentation can be performed using CNNs. It also covers work on multimodal deep learning, and in particular how depth information can be incorporated in CNN architectures for semantic segmentation.
Chapter 4 presents the methods used in the work, and in particular the proposed architecture. The chapter then describes how the network learns semantic segmentation, and also covers the implementation of the system.
Chapter 5 presents the datasets which are used for training and testing the network, and an overview of the experiments. The evaluation metrics are also presented here.
Chapter 6 presents the results of the experiments, together with a discussion of the results.
Chapter 7 concludes the thesis, and presents suggestions for further work.

Chapter 2

Theoretical background

This chapter presents the theoretical background for the work in this thesis. We begin the chapter by briefly covering the fields of computer vision and image processing. This is followed by a section on machine learning and deep learning, which serves as a foundation for the sections to follow. We then present artificial neural networks, before covering convolutional neural networks.

2.1 Computer vision and image processing

The visual system enables humans to process visual detail and understand the world around us. It detects and interprets visible light, to construct a representation of the surrounding environment [2]. Computer vision is concerned with how computer systems can achieve visual understanding similar to the human visual system [3]. The field was born in the 1970s in the early days of modern artificial intelligence as the visual component of intelligent robotic systems. The computer vision field was motivated by the desire to recover the three-dimensional structure of the world from images. The idea was that this information could be used to achieve full scene understanding [4].

Computer vision is closely related to the field of digital image processing, which already was an established field of research when computer vision was conceived. There is no general agreement as to where image processing stops and other fields such as computer vision and image analysis, start. However a common distinction is to define image processing as the discipline where both the input and output is an image. Computer vision systems on the other hand, may produce both an output image as well as qualitative and/or quantitative information such as size, intensity, object shape or object classification. For both computer vision and digital image processing tasks we consider the images to simply be matrices. Each matrix element is referred to as a pixel, and the value of the pixel is referred to as the intensity value of the pixel. A grayscale image is nothing more than a two-dimensional matrix, while an RGB image is represented by a three-dimensional matrix. This is illustrated in figure 2.1.

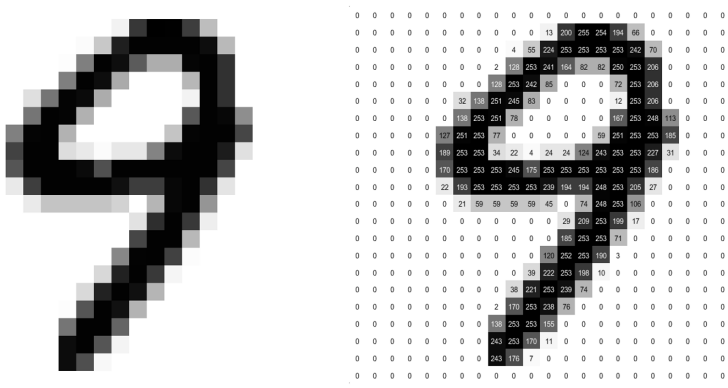


Figure 2.1: Representation of a digital image as a matrix of intensity values.

2.1.1 Basic image processing - convolution

Convolution is a fundamental image processing operation which operates directly on the pixel intensity values of an image. A filter, also referred to as a kernel, is applied to every pixel of an input image. By sliding the filter over all the pixels of the input image, and at each pixel compute a sum of all the local neighbours weighted by the filter, an output image is produced. For an input image f of size $M \times N$, and a filter w of size $m \times n$, the operation for a single pixel $[x, y]$ is given by the expression:

$$g(x,y) = w(x,y) * f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s, y-t) \tag{2.1}$$

where $a = (m-1)/2$ and $b = (n-1)/2$. The minus signs on the right side has the effect of flipping the input image, i.e. rotating it by $180°$. The result is the same if we flip the filter instead of the image. Applying this to the entire image, requires computing equation 2.1 for $x = 0, 1, 2, ..., M-1$ and $y = 0, 1, 2, , ..., N-1$, where the pixel origin is located in the upper left corner. We assume that the image f has been padded appropriately, e.g. with zeros, and that m and n are odd integers.

Convolution filtering has many uses, and different filters are used for different applications. Convolution can for example be used to sharpen an image or detect edges. Some examples of filters are shown in equation 2.2. The left filter can be used for sharpening, while the right filter is used for applying a blur to an image.

$$w = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \qquad w = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.2}$$

Equation 2.3 illustrates one step of the convolution operation. Here we compute the value of the output center pixel, at location $[1, 1]$. By considering the image and filter as matrices, the process involves multiplying corresponding elements, and summing the products, after having rotated either the image f or the filter w. This can be written as:

$$g(1,1) = w(1,1) * f(1,1) = \left(\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \right) [1,1] = \tag{2.3}$$
$$(i \cdot 1) + (h \cdot 2) + (g \cdot 3) + (f \cdot 4) + (e \cdot 5) + (d \cdot 6) + (c \cdot 7) + (b \cdot 8) + (a \cdot 9)$$

Repeating this procedure for every single pixel in the image, would produce a complete output image. Figure 2.2 illustrates an example where an edge detection filter has been applied to the input image. This example shows how the convolution operation can be used to extract features, in this case edges, from an image. Extracting features from images is a fundamental step to solving many computer vision tasks. Convolution is also an essential part in convolutional neural networks, which are covered at the end of this chapter.

(a) Original image **(b)** Edge map

Figure 2.2: Using convolution for edge detection. Image courtesy of [5]

2.1.2 Computer vision tasks

Computer vision has a wide range of applications, and is already well integrated into modern society. Medical computer vision is one of the most highlighted applications. Through extracting information from image data, the diagnosis of patients can be improved. Another prominent application is in navigation systems for autonomous vehicles or mobile robots. Computer vision systems are present both in many self-driving cars on the roads [6], as well as on vehicles involved in space exploration, such as NASA's Mars Exploration Rover [7].

The applications of computer vision employ a range of computer vision tasks, which can be viewed as common and well-defined problems. These tasks all share a goal of producing numerical information which represents a visual understanding of images. They are based on methods for processing and analyzing digital images, to transform them into suitable descriptions of the world. Two computer vision tasks relevant for scene understanding are presented below.

Image classification

Image classification, also called object recognition, is one of the core computer vision problems. It is concerned with making a prediction for the whole input image, classifying the object in the image, or producing a list of all the recognized objects. The task of classifying the content of an image is a trivial task for humans, and it takes virtually no effort to recognize for example the presence of car in an image. While the field of image classification has been studied extensively for many years, it is still considered a difficult task for computer vision systems.

There are numerous challenges which makes image classification difficult for computer algorithms. A classifier deployed in the real world is sure to encounter variations in factors such as viewpoint, scale, occlusion and intra-class variance. For the example where the output prediction should be the label "car", there is a huge amount of inputs which should produce the same result. The digital representation of a car will vary greatly depending on whether it is viewed from the front or the back, whether it is far or close, if it is fully visible, and depending on the type of car. An image classification system needs to be invariant to these variations, even when they appear combined.

(a) Traditional approach to image classification.

(b) Deep learning approach to image classification.

Figure 2.3: Traditional and deep learning approach to image classification.

Traditional approaches to solving the classification task often used feature-based methods. These methods extract human-engineered features, such as edges, colors and corners, from images to produce feature descriptors. Such features were deemed relevant in solving computer vision tasks, as they have shown to capture useful information. These features can be be gathered from a set of training images, providing a feature description of that specific class. Learning algorithms can then use the training images to learn relevant classes. This is illustrated in figure 2.3a.

The most successful modern approaches are based on deep learning, and are called convolutional neural networks (CNNs). These networks also make use of features extracted from training images, however they are no longer human-engineered features such as edges and corners. The general idea is that such networks learn more and more abstract representations of the training images, as it proceeds throughout the network architecture. This is illustrated in figure 2.3b. It turns out that such abstract feature representations work much better than traditional feature descriptors, and CNNs are currently state of the art in image classification.

Semantic segmentation

While solving the task of classification provides information about the "what" of an image, it does not tell us "where" the detected object is. In the context of autonomous driving, the "where" part, i.e. localization, is a crucial component for providing full scene understanding. When we include location of the objects, we move to the domain of object detection. Object detection algorithms will both assign a class label, as well as provide localization information about the detected objects, for example by using bounding boxes.

Motivated by the goal of full scene understanding, it is possible to move even beyond object detection. Instead of producing bounding boxes for the detected objects, a more powerful description is acquired by assigning a class label to every single pixel in an image. This is known as semantic segmentation, and is illustrated in figure 2.4, together with the progression from image classification. The bottom right image illustrates instance-level segmentation, where each pixel is assigned both a class label, as well as an instance label.

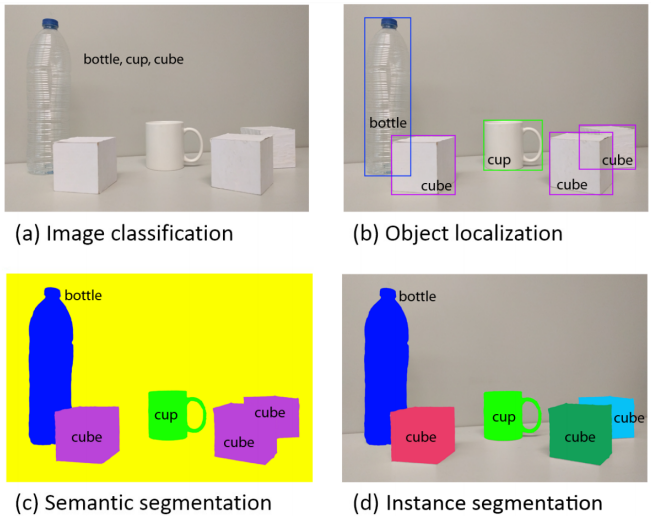


Figure 2.4: Moving from classification to semantic segmentation. Image extracted from [8].

Several approaches exists for solving the task of semantic segmentation using traditional computer vision methods, summarized in [9] and [10]. Any segmentation algorithm should aim to divide an image into "meaningful" parts, according to some measure. The different parts may be generated based on properties such as intensity level, texture or gradient value, which is shared among the grouped parts. One approach is based on classic bottom-up methods. By using some local homogeneity measure, such as color or texture, nearby pixels are grouped together. This can create regions which are similar according to this measure, through merging pixels in an iterative fashion. These methods are simple and straight-forward to implement, but suffer due to the fact that they do not explicitly take into account the information about the different objects. This means that without using any prior assumptions, these methods struggle to produce object regions.

Another approach is similar to the traditional feature-based methods for image classification. This involves using a classifier which is trained on, and operates on, fixed size images. The features used for training the classifier are often the same as for image classification. A sliding-window technique is used to extract small parts, "windows", of an image, which are classified individually. The window-wise classification may then classify only the center pixel, or a subset of the window, as belonging to one of the semantic classes. Such patch-wise classification needs to be applied many times for even small windows, and while it may be sped up using strides combined with interpolation, it is a computationally expensive approach.

The modern approaches which have achieved the best results for semantic segmentation are based on deep learning and CNNs, like in the image classification case. In fact the networks which were first designed for image classification, were later adapted to perform semantic segmentation, and significantly outperformed the traditional methods covered above.

2.2 Deep learning and machine learning

The most successful modern approaches to solving different computer vision tasks are based on deep learning, and in particular CNNs. Before introducing deep learning in further detail, it is advantageous to briefly cover the machine learning field, which deep learning is a sub-class off. The aim of this section is to provide an overview of the fields of machine learning and deep learning, and how they relate to each other. This will make it clearer how deep learning can be used as a tool to solve the computer vision tasks.

Machine learning uses statistical techniques to give computers the ability to improve their performance on specific tasks without being explicitly programmed, but rather through experience. Machine learning algorithms parse large amounts of data to learn from it during a learning phase, which can be used to adjust the algorithm and improve performance. A model is built for the specific task at hand, which can be applied to new data.

Machine learning tasks are typically said to belong to one of two broad categories, determined by the way learning is performed:

- **Supervised learning:**
  The system is provided with a labeled dataset for training, which consists of N training samples. This takes the form of $\{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$, where x_i is the feature vector of the i-th example, and y_i is its corresponding ground truth label. The goal is to learn a function which maps the inputs to the desired outputs, $f : X \mapsto Y$, where X and Y respectively is the input and output space. This function can in turn be used for the mapping of new unseen inputs. A requirement for the algorithm to correctly map new inputs to the desired outputs, is its ability to generalize from the training data to new data in a sensible fashion.

- **Unsupervised learning:**
  The system is provided with an unlabelled dataset for training, i.e. the system only gets a set of inputs, but not desired outputs. Based on these inputs, the system needs to learn the structure in the data, and find a function which maps the inputs to the desired outputs.

Deep learning is a subset of machine learning, which is based on learning representations of data instead of task-specific algorithms. This allows deep learning algorithms to solve problems from end to end. The learning can be performed in both a supervised and an unsupervised manner, where the representations are learned from labelled and unlabelled data respectively. Common to these methods is that they use models which have multiple stages, or levels, for extracting features. Each stage performs some transformation to the features, and pass the result to the next stage. This allows the methods to learn several levels of feature representations, which corresponds to increasing levels of abstraction, as we move deeper through the model [11].

Many modern deep learning models are based on artificial neural networks (ANNs) which are considered in detail in the next section. When such networks are exposed to vast amounts of data, they can efficiently learn hierarchies of features. One specific class of ANNs, convolutional neural networks (CNNs) are especially good in the context of computer vision. These networks are considered at the end of the chapter.

2.3 Artificial neural networks

Artificial neural networks (ANNs) are computing systems which are loosely inspired by how the brain works. These networks learn how to perform a given task by considering training examples specific to the task. Without being provided with a priori knowledge about the task, they are still able to develop a set of relevant feature representations from the training data. One of the strengths of ANNs is their representational power. These networks can be considered as a set of functions, which are parameterized by the parameters of the network. Each such function defines a mapping between the inputs and outputs. Under some mild assumptions on the network structure, the Universal approximation theorem states that such neural networks can approximate any continuous function [12]. As such, the networks have a wide range of possible applications, and have been successfully

applied to fields such as computer vision, speech recognition and automatic text generation.

In this section we cover the most important aspects for all neural networks, the neurons and layers, as well as the activation functions associated with the neurons. Following this, we consider how such networks can be trained, before we cover the specifics of CNNs in the next section. The theory of those networks is based on the general theory for ANNs.

2.3.1 Neurons and layers

The early research into artificial neural networks was originally inspired by the goal of developing models of biological neural systems such as the human brain. The fundamental computational unit of the human brain is the neuron. Each neuron receives one or more input signals from its dendrites via synapses. The synaptic strength controls the strength of the signal, and thus the influence of one neuron on another. Each neuron can produce a single output signal along its axon, which in turn connects via synapses to the dendrites of other neurons. Each neuron considers the sum of its input signals, and if the sum is larger than a certain threshold, the neuron "fires", sending an output signal along its axon.

A mathematical model of these biological neurons, is the basis for artificial neural networks. The model is illustrated in the left part of figure 2.5. Each such neuron is a computational unit, and the neurons may be connected to each other to through its inputs and outputs, thus forming a network. In the network, the neurons are typically organized in layers, as shown in the right part of figure 2.5. These neural networks consists of a single input layer, a single output layer, and potentially one or more hidden layers connecting the input and output layers. The example architecture has an input layer with three inputs, two outputs in the output layer, and two hidden layers, each with four neurons.

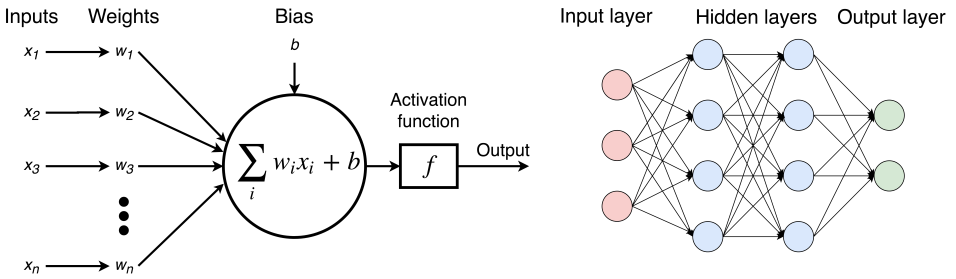


Figure 2.5: Neurons and layers in ANNs

The arrangement of the neurons forms a directed weighted graph. When this graph is acyclic, as in figure 2.5, the network is commonly referred to as a feed-forward neural network. The information in this type of network always moves in one direction. The flow is from the input layer, through the hidden layers (if there are any) and to the output nodes. An artificial neural network with multiple hidden layers is referred to as a deep

neural network (DNN) [13]. This property enables the network to model more complex relationships in the data, as opposed to their shallow counterpart-networks with at most one hidden layer.

Each neuron in the network can be expressed as a mathematical function. This function takes one or more input values from neurons in the previous layer, and produces a single output value, which is passed to neurons in the next layer. The function can be abstracted into two computational stages. First a weighted sum is computed, before a non-linearity is applied, using an activation function.

Associated with each neuron h_j is a weight vector $\mathbf{w}$, which controls the influence of the inputs. The input vector to the neuron is denoted $\mathbf{x}$. The linear output of the neuron is computed as a weighted sum

$$o_j = \sum_{i=1}^{n} w_i x_i + b_j = \mathbf{w}^T \mathbf{x} + b_j \tag{2.4}$$

where w_i is the "strength" of the connection between the neuron and x_i, the i^{th} output from the previous layer. A bias term b_j is added, which allows us to shift the sum, which can be helpful during the learning phase. A non-linearity is then applied to this sum by using a nonlinear activation function f. The following expression for h_j gives the final output of the neuron, a real-valued number which can be passed to neurons in the next layer:

$$h_j = f(o_j) = f\left(\sum_{i=1}^{n} w_i x_i + b_j = \mathbf{w}^T \mathbf{x} + b_j\right) \tag{2.5}$$

The output from a single neuron depends on the weights and bias for that neuron, and the output from a single layer depends on the weights and biases of all neurons in that layer. For convenience, all parameters for all neurons in layer m can be gathered in a weight matrix $\mathbf{w}^{(m)}$. This allows us to express all the network parameters as $\mathbf{W} = [\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, ..., \mathbf{w}^{(L)}]$, where L is the number of layers. This means that we can view the network as a function $f_{\mathbf{W}}$, with parameters $\mathbf{W}$, which transforms an input x to an output y,

$$y = f_{\mathbf{W}}(x) \tag{2.6}$$

The key insight of this model, which makes artificial neural network so strong, is the idea that the parameters of the network, the weights and biases, are learnable. By feeding a network training examples, an output signal is produced, which can be compared to the desired output. By continuously adjusting the parameters, based on how good or bad the produced output is, we move towards optimal weights for solving the task.

2.3.2 Activation functions

The activation function associated with a neuron defines the output signal. There are several desirable properties of an activation function that leads to better performance. The most important property is that it is non-linear, which allows for the network to be a universal function approximator [12]. Another important property is that the activation function should be continuously differentiable, which enables gradient-based optimization algorithms. There are several choices for the activation function, and the most popular choices are summarized below, and illustrated in figure 2.6.

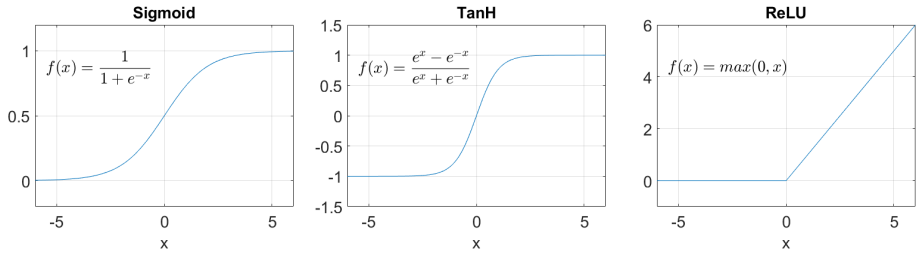


Figure 2.6: Popular activation functions

Sigmoid:

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.7}$$

The sigmoid function squashes the input values the range $(0, 1)$. This property makes it convenient to use when the output is interpreted as a probability. This activation function has one major drawback, which is the fact that it suffers from vanishing gradients. When the activation is near either of the tails, close to 0 or 1, the gradient is almost zero. When training a network, the gradients of the activations are used extensively. This means that the sigmoid function can slow down training significantly, and for this reason, the function has fallen out of use.

TanH - Hyperbolic tangent:

$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.8}$$

The TanH function is quite similar to the sigmoid function, as it squashes the inputs to the range $(-1, 1)$. Another property it shares with the sigmoid function, is that it suffers from the problem of vanishing gradients. However, one edge it has, is that the output is centered around zero. For this reason, the TanH function is in practice preferred over the sigmoid function.

Rectified Linear Unit - ReLU:

$$f(x) = max(0, x) = \begin{cases} 0 & \text{for } x < 0, \\ x & \text{for } x \geq 0 \end{cases} \tag{2.9}$$

The ReLU function thresholds the activations at zero, giving outputs in the range $[0, \infty)$. It has become a very popular activation function, due to some of the advantages it has over the sigmoid and TanH functions. The most important edge is that it does not suffer from vanishing gradients, which allows for greatly accelerated training. It is also less computationally expensive to compute, as we can avoid the exponential function.

Leaky ReLU is a modified variant of the ReLU function, which has proven to improve performance. It provides a small, positive gradient when the input is non-positive, by introducing a small leakage coefficient, here set to 0.01.

$$f(x) = \begin{cases} 0.01x & \text{for } x < 0, \\ x & \text{for } x \geq 0 \end{cases} \tag{2.10}$$

Parametric ReLU - PReLU lets the leakage coefficient be learnable, similar to the other parameters of the network.

$$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0, \\ x & \text{for } x \geq 0 \end{cases} \tag{2.11}$$

2.3.3 Learning

The ability to learn is a key feature of artificial neural networks. For a specific task and a set of training data for that task, this involves using the training data to find a function represented by the network $f^* \in F$, where F is a class of functions, such that f^* solves the task in an optimal manner. The traditional approach for doing this is to use mathematical optimization. We can define a loss function, also called a cost function, $C : F \mapsto \mathbb{R}$, such that for f^*, $C(f^*) \leq C(f) \forall f \in F$. No functions have a smaller loss (cost) than the optimal solution. While the terms loss function and cost function are sometimes used interchangeably in the literature, we use the term loss from here on. A small loss, equivalently a small cost, implies a good fit.

The learning may be performed in both a supervised or unsupervised fashion. We will focus on the supervised approach to learning, as this is the most successful method for solving the computer vision tasks. This means that when searching for the optimal function f^*, we look for a mapping implied by the data itself. When defining a loss function C, it should be related to how well the output of the network matches the ground-truth label for the training samples.

Loss function

When defining the loss function, it is convenient to limit the possible functions by making two assumptions about the properties of the function. These assumptions allows the use of the powerful backpropagation method during the training process [14]. This is an efficient method for computing gradients of an ANN, and is covered in the next section. The first assumption is that the loss function can be expressed as an average over the loss for each of the n individual training examples:

$$C = \frac{1}{n} \sum_{i=1}^{n} C_i \tag{2.12}$$

The backpropagation algorithm needs the loss function in this form as it computes the gradient of the loss function individually for each training example. The form in equation 2.12 generalizes this to the total loss function. The second assumption is that the loss function can be expressed as a function of the output of the network. This output depends on the parameters of the network, and therefore the second assumptions allows us to minimize the loss function with regard to the network parameters. We use the notation $C(\mathbf{W})$ for the loss function, given the current weights $\mathbf{W}$.

Depending on the task to solve, certain loss functions may be more suitable than others. For regression problems, a common approach is to measure the error between the output and the true label, then compute the L2 squared norm. For a single example, the loss function takes the form:

$$C_i = \|\hat{y_i} - y_i\|_2^2 \tag{2.13}$$

For classification tasks, a common choice of loss function is the cross-entropy loss:

$$C_i = -[y_i \log(\hat{y_i}) + (1 - y_i) \log(1 - \hat{y_i})] \tag{2.14}$$

For both the equations, we have denoted the output of the network for training example i as $\hat{y_i}$, and the true label as y_i.

Gradient descent and backpropagation

Having defined a loss function which indirectly depends on the weights of the network, we can look for the optimal weights. The goal of the training process is to determine the weights which gives the smallest loss. This optimization problem can be solved using different algorithms, and one common choice for finding the weights, is the gradient descent algorithm. It is an iterative procedure for finding the minimum of a function, by taking steps in the parameter space proportional to the negative of the gradient of the function. In each iteration, the weights of the network are modified by adding a small change $\Delta\mathbf{W}^t$,

$$\mathbf{W}^{t+1} = \mathbf{W}^t + \Delta\mathbf{W}^t \tag{2.15}$$

where t is the iteration step. This updating scheme is continued until some stopping criteria is met. In order to apply this method, we need to compute gradients of the network.

Backpropagation is an efficient method for computing these gradients. The learning algorithm takes the form of two phases, propagation and weight update. A training sample is propagated through the network, to produce an output. The "quality" of this output can be found by comparing it with the desired output, using the loss function. This produces a total error, and we are interested in finding the contribution to this error for each of the neurons in the network. By computing the error value for each of the neurons in the output layer, we can propagate these values back throughout the network to find the contribution

of each neuron. These values allows us to compute the gradient of the loss function. The gradient descent method can then be used to perform the weight updates during the second phase. It uses the gradient of the loss function to update the weights, such that the loss is minimized,

$$\Delta \mathbf{W}^t = -\gamma \nabla C(\mathbf{W}^t) \tag{2.16}$$

In this equation, γ is the step size. The step size, also called the learning rate, is an important parameter which can heavily influence how fast the training of the network converges. A small learning rate may lead to slow training, as many iterations are needed to find the minimum of the function. A larger learning rate may speed up training, at the cost of potentially "overshooting", and missing the minimum. Therefore care must be taken when choosing this value.

Extensions to backpropagation

Using gradient descent together with backpropagation in its standard form is fairly common, however several extensions exists and are often used. Some extensions aim to speed up the process, while others aim at making the learning process more reliable. Two important such extensions is to introduce a momentum term, and to use weight decay.

By introducing a momentum term α, the weight adjustment can be modified to also depend on the adjustment performed in the previous iteration. The expression for the weight change can be expressed as

$$\Delta \mathbf{W}^t = (1-\alpha)\gamma \nabla C(\mathbf{W}^t) + \alpha \Delta \mathbf{W}^{t-1} \tag{2.17}$$

where the momentum term α is in the range $[0, 1]$. This technique alleviates the problems that arise due to challenging regions in the weight space, where there are flat plateaus and steep "valleys". In such regions the gradient of the loss function may be very small or very large. The momentum term helps the gradient descent algorithm to not get stuck in these regions.

Weight decay is a regularization technique which helps prevent the network from overfitting during training. It involves adding a regularization term to the loss function, such that the new form becomes

$$\tilde{C}(\mathbf{W}) = C(\mathbf{W}) + \frac{\lambda}{2}\mathbf{W}^2 \tag{2.18}$$

where λ is the regularization parameter, and $C(\mathbf{W})$ is the normal loss function without weight decay. The expression for the weight change becomes

$$\Delta \mathbf{W}^t = -\gamma \nabla C(\mathbf{W}^t) - \gamma \lambda \mathbf{W}^t \tag{2.19}$$

The effect of the new term is that large weights are penalized, and the network is forced to use parameters of small magnitude. Effectively, this limits the freedom of the model, and in effect its ability to overfit.

Stochastic and batch learning

When using the backpropagation for training a network, a choice between two modes of learning needs to be done. The two approaches are called stochastic and batch learning, and they differ in how the training inputs affect the weight update. Stochastic learning leads to a weight adjustment for each input, and is the approach described above. As stochastic learning computes local gradients from single inputs, it naturally introduces "noise" into the optimization process. This noise makes the network less likely to get stuck in local minima. The alternative approach, batch learning, uses batches of inputs for computing the weight adjustment. This involves averaging the error over the batch, which gives a faster and more stable weight update process. Both approaches have their pros and cons, and a common compromise is to use "mini-batches". This is batch learning, where the batches are restricted to being small, and the samples for each batch are stochastically selected from the entire training set.

Batch normalization

Batch normalization [15] is a technique introduced in 2015, which made deep ANNs significantly easier to train. One of the issues which complicates training is the phenomenon known as internal covariate shift. The inputs to each layer is affected by the parameters of all the layers prior to it, and as parameters are adjusted during training, the distribution of activations change. This means that a small change in network parameters can be amplified significantly in the deep parts of networks. Internal covariate shift can be dealt with by using low learning rates, and initializing the weights with care, however this slows down training considerably.

Batch normalization addresses this problem by forcing the activations of all layers to take on a unit Gaussian distribution. This normalization procedure is performed for each training mini-batch. The activation output from one layer is the input to the next layer, and the normalization step fixes the means and variances of all layer inputs. Each of the activations is made to have a mean of 0 and a variance of 1.

The main advantage of using batch normalization is that it can drastically speed up the training phase. Using high learning rates may in many cases increase the scale of the network parameters. This can amplify the gradient during backpropagation, thus destabilizing the learning process. When the batch normalization is applied, backpropagation through a layer is not affected by large parameters in the layer. This means that higher learning rates can be used.

Dropout

Dropout [16] is a regularization technique which reduces overfitting. At each training stage, neurons in the network are either kept active with probability p or "dropped out" with a probability $1 - p$. The result is a reduced network which is trained on the data in that training stage. Before proceeding to the next training stage, the removed neurons are

reinserted with their original weights.

The dropout technique can be viewed as a form of ensemble learning, where a number of reduced networks are trained separately. During testing, the predictions of all these reduced networks are combined using an averaging method. While all of these reduced networks are weaker than a network without dropout, combining them results in an overall stronger network. By not training all the neurons on all the training data, the network is less likely to overfit to the data. The method has also shown to greatly reduce training time, even for deep networks.

2.4 Convolutional neural networks

Convolutional neural networks (CNNs) is a class of deep feed-forward artificial neural networks, which have a wide range of applications. These networks have shown very good performance at classical computer vision tasks such as image classification and detection. In fact, these networks outperforms traditional methods in terms of accuracy, performance and sometimes even efficiency. In this section we focus on networks designed for image classification. We discuss how they differ from the regular ANNs and the most important layer types, before giving examples of typical network architectures.

2.4.1 A brief history of CNNs

The early development of CNNs was inspired by the work of neurophysiologists Hubel and Wiesel in the 1950s and 1960s [17, 18]. They showed that neurons in the visual cortexes of cats and monkeys respond individually, and differently, to small sub-regions of the visual field, called receptive fields. Neurons located close to each other have overlapping receptive fields, which are similar in structure. The idea that certain parts of the visual system seemed to have certain tasks was further expanded on in their 1968 paper [18]. They found that the some neurons in the brain responded when exposed to straight edges with particular orientations at certain locations. Other neurons responded to oriented edges regardless of their exact locations, thus having a degree of spatial invariance. These discoveries would later have a large impact on the development of CNN architectures.

One of the first successful convolutional neural networks was LeNet-5, introduced in 1998 by Lecun et al. [19]. It was designed for document recognition, classifying hand-written numbers on checks. The architecture is shown in figure 2.7. Using convolutional layers to extract features from digitized images, fully-connected layers to classify them, and the backpropagation algorithm to train the network, other state-of-the-art methods at the time were outperformed. The work also revealed how the network architecture was heavily constrained by the lack of availability of computer power.

In 2012, the AlexNet architecture was presented by Krizhevsky et al. [20]. It was similar to LeNet-5, but was deeper and much wider. Thanks to advances in GPU technology, the network could be trained using GPUs which led to a much shorter training time than earlier

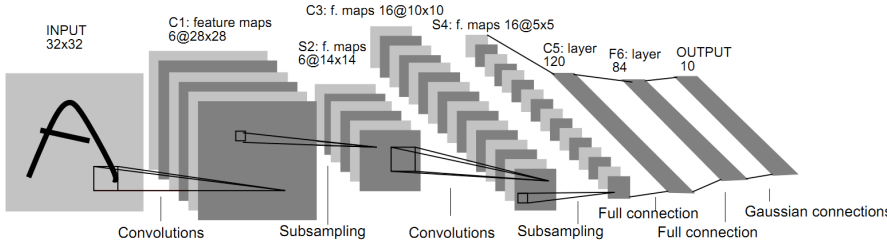


Figure 2.7: The LeNet-5 architecture. Figure extracted from [19].

approaches. The same year, AlexNet competed in the annual ImageNet Large Scale Visual Recognition Competition (ILSVRC), a prestigious computer vision competition. AlexNet won the 2012 competition with a large margin, which sparked a small revolution. The work had shown the power of CNNs in the context of computer vision, and since 2012, deep CNNs have dominated the annual ILSVRC. Some of the most successful networks for image classification are the VGG-16 architecture [21], ResNet [22] and GoogLeNet [23].

2.4.2 Distinguishing features

Just like a regular artificial neural network, a CNN is a collection of neurons organized in layers. However, we now make the explicit assumption that the inputs are digital images to be classified. The network now represents a score function, which transforms raw image pixels to class scores at the output. Certain design choices regarding the network architecture allows us to exploit fundamental properties of the images. As opposed to regular ANNs, CNNs have the following distinguishing features: spatial arrangement of neurons, local connectivity and shared weights. These features are briefly summarized below, and elaborated on later in the context of the different layers.

Spatial arrangement of neurons

Unlike regular ANNs, the layers of a CNN have neurons arranged in 3 dimensions: width, height and depth. This is illustrated in figure 2.8. Just like before, layers are stacked to form a network architecture. The neurons can have varying levels of connections to its previous layer, depending on the layer type. For some layers, every neuron is connected to every neuron in the previous layer. For other layers, the neurons are only connected to a small region in the previous layer.

Local connectivity

The neurons in regular ANNs are typically connected to all neurons in the previous layer. It is the weights associated with these connections which make up most of the parameters

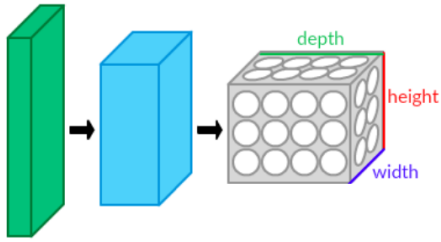


Figure 2.8: Arrangement of neurons in a CNN. Image courtesy of [24]

of the network. Fully-connected layers lead to the most parameters, and this approach scales poorly when working with high dimensional inputs such as images. For a relatively small image of size 200x200x3 pixels, a single fully-connected neuron in the first hidden layer would have $200 \times 200 \times 3 = 120000$ weights, without counting the biases. The number of parameters increases drastically as more neurons and layers are introduced. A large number of parameters leads to increased memory requirements and more computations, while also making the network more susceptible to overfitting. Another downside with fully-connected neurons is that they don't exploit the spatial structure of the data. A fully-connected approach treats input pixels which are far apart in the same way as pixels that are located close to each other. Regions in images are spatially correlated, and to exploit this, a more fitting approach is desirable.

Instead of fully-connected neurons, a local connectivity pattern is used, loosely inspired by the concept of receptive fields discussed earlier. Each neuron has a small receptive field, and is only connected to a small region of the previous layer. The receptive field is spatially small along the width and height dimension, but extends through the entire depth dimension of the previous layer. The receptive field of a neuron is an important parameter in convolutional layers.

Shared weights

Most of the learnable parameters in a CNN are held by neurons in the convolutional layers. The parameters of these layers, i.e. the weights and biases, can be defined as small filters. By sliding these filters across the input volume, the filters can learn to respond to certain features, and different layers will respond to different features. These filters are shared by neurons in each convolutional layer, and the same filter is used by each neuron with its own receptive field. All the neurons in a layer will therefore respond to the same features. As features thus can be detected across the entire visual field, the shared weights makes the feature detection invariant to translation .

2.4.3 The layers of a CNN

A CNN is a sequence of different layers, where each layer accepts an input, performs some sort of transformation, and produces an output. As we are now working with three-

dimensional data, we refer to this as "volumes" of data. Each layer transforms an input 3D volume to an output 3D volume. There are many different types of layers, however a few are considered essential layer types, and are included in most CNN architectures. Different layer types have different properties, based on their purpose. Some layers simply implement a fixed function, without having any parameters, while other layers have many parameters which are subject to optimization during training. In the following sections we describe the most important layers of a CNN architecture.

Convolutional layer: The core operation in the CNN is the convolution, as introduced in section 2.1.1. This operation involves sliding filters across input volumes to produce output volumes. One of the filters presented previously was designed to extract one specific feature, namely edges. The values of the filter coefficients specified what feature to look for. When performing convolution in the context of CNNs, the filters no longer have static values. Instead they are regarded as parameters associated with the neurons in the convolutional layers. The network learns optimal weights for these neurons, which in turn means that the filter values are learned. The filters can learn to respond to abstract features, which results in accurate output values from the network.

The learnable filters are defined by the receptive fields of the neurons. Each neuron has a small receptive view, which extends through the full depth of the input volume. Different neurons which are arranged along the depth-dimension of the layer connect to the same region of the input. This is illustrated in figure 2.9, where 5 neurons in a "depth column" are connected to the same region. Each of these neurons learn to activate for different features, producing different activation maps.

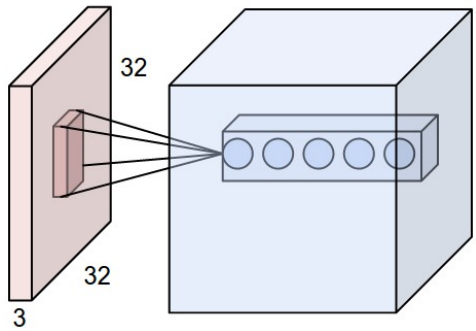


Figure 2.9: Multiple neurons connected to the same region. Image courtesy of [25]

All neurons which are organized in the same depth column are constrained to use the same weights and biases. The reasoning behind this is that if a filter activates at some spatial location, it should also be able to activate at other positions. In this way, the parameter sharing contributes to making the CNN translation invariant. As the filter is convolved across the width and height of the input volume, it produces a 2-dimensional activation map. Each filter produces an activation map, and by stacking these along the depth dimension, a three-dimensional output volume is formed.

Multiple convolutional layers are typically used in a CNN, and their characteristics are often different, depending on where in the network the layer is placed. We can fully specify each layer using 4 hyperparameters:

1. **K** - The number of filters. This is the number of neurons in the layer which has a receptive field of the same region of the input volume.
2. **F** - The filter size. This is equivalent to the size of the receptive fields.
3. **S** - The stride. This controls the distance we slide the filters at a time.
4. **P** - The zero-padding. This is the amount of zero-padding used on the borders of the input volume.

Given an input volume of size $[\mathbf{W_1} \times \mathbf{H_1} \times \mathbf{D_1}]$, the size of the output volume is $[\mathbf{W_2} \times \mathbf{H_2} \times \mathbf{D_2}]$, where

$$\mathbf{W_2} = \frac{(\mathbf{W_1} - \mathbf{F} + 2\mathbf{P})}{\mathbf{S}} + 1 \tag{2.20}$$

$$\mathbf{H_2} = \frac{(\mathbf{H_1} - \mathbf{F} + 2\mathbf{P})}{\mathbf{S}} + 1 \tag{2.21}$$

$$\mathbf{D_2} = \mathbf{K} \tag{2.22}$$

Pooling layer: Pooling layers are downsampling layers used to reduce the spatial size of an activation volume. This leads to a reduced number of parameters in the network, as well as fewer necessary computations. As a large number of activations are discarded, the use of pooling layers helps prevent the network from overfitting to the training data. The most common type of pooling employed is max pooling. This operation works by returning the maximum value of each sub-region of the input activation volume. This value is used in the output activation volume. The max pooling operation requires that we specify two parameters: the filter size $\mathbf{F}$, and the stride $\mathbf{S}$. An example of max-pooling, using 2x2-filters and stride 2 on a 4x4 input volume is shown in figure 2.10.

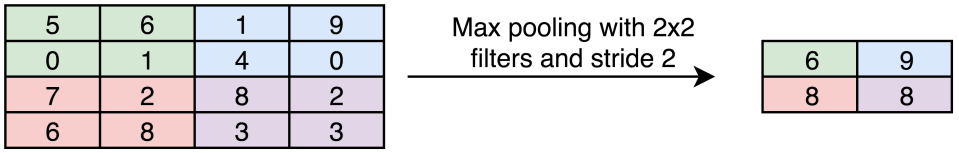


Figure 2.10: Example of max pooling.

ReLU layer: This layer applies the ReLU activation function introduced earlier, element-wise to the activation map. This operation does not change the size of the input volume, but only thresholds the activations at zero. It is the preferred activation function for use in CNNs, for the same reasons as discussed earlier.

Fully-connected layer: The fully-connected layers of a convolutional neural network are the same as those found in traditional ANNs. Every neuron in such a layer is connected to every neuron in the previous layer. All these connections make fully-connected layers extremely computationally expensive when used in CNN-architectures with a large number of neurons. For this reason, the fully-connected layers are typically only used as final layers in classification-CNNs, where they transform the final activations into class scores.

The final layers of a CNN are often accompanied by the softmax function, which is defined as

$$\begin{aligned} \sigma : \mathbb{R}^K &\mapsto \{\sigma \in \mathbb{R}^K \mid \sigma_i > 0, \sum_{i=1}^{K} \sigma_i = 1\} \\ \sigma(\mathbf{z})_j &= \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{for } j = 1, ..., K \end{aligned} \tag{2.23}$$

This function takes a K-dimensional vector $\mathbf{z}$ with elements of arbitrary real values, and transforms it to a new K-dimensional vector $\sigma(\mathbf{z})$. The elements of this new vector all lie in the range $(0, 1)$, and add up to 1.

This function is convenient to use as the activation output of the final fully-connected layers can take on arbitrary real values. We want to transform these into final classification scores, as a measure of relative activation strength of each class compared to the other classes. The output of the softmax funcation can be used to represent a probability distribution over the K possible classes.

2.4.4 Training

Training a CNN for classification is similar to training regular ANNs. A loss function is used to measure how well the prediction fits the ground truth. The cross-entropy loss is commonly used to measure the error at softmax layer. The output vector from this layer holds the predicted class probabilities for each of the possible classes. This is compared to the ground truth vector, where the class probability is 1 for the true class, and 0 for the other classes.

2.4.5 Architectural approaches

The convolutional, max-pooling, ReLU and fully-connected layers are an essential part of most successful CNNs for classification. These can be put together to form an architecture in numerous ways. There is no "optimal" architecture for solving the classification problem, and certain architectures may work well with some data, but struggle with different data. However, there is a general approach to the ordering of the different layers that have proven to result in good performance.

In practice, ReLU layers almost always follow convolutional layers. Most common approaches stack a few convolutional-ReLU pairs followed by a max pooling layer. This

pattern is repeated a couple of times, depending on the size of the images, and how much downsampling is needed. After this, the network transitions to a few fully-connected layers, each followed by a ReLU layer. The final fully-connected layer can be followed by a softmax layer to produce classification scores. Figure 2.11 shows an example of a typical classification architecture.

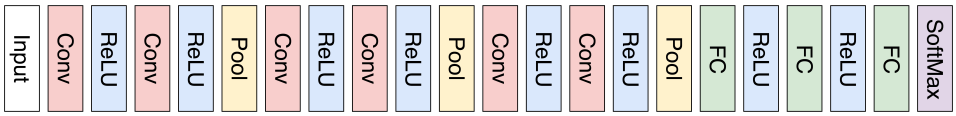


Figure 2.11: Generic CNN architecture for classification.

Many popular architectures follow this pattern closely. One example is the successful VGG-16 architecture, which is quite similar to the network in figure 2.11, but slightly deeper. Different architectures vary in factors like filter sizes, stride, the number of convolutional layers to stack before pooling and the amount of downsampling. Beyond this pattern, certain architectures have made significant contributions to the field by introducing new techniques and approaches. We will briefly cover two such architectures, which have become widely known standards.

GoogLeNet

GoogLeNet [23] won the ILSVRC in 2014, beating VGG-16. This was one of the first architectures that really strayed away from the general pattern of stacking the layers in a sequential order. Compared to previous approaches, the network structure was much more complex, and the creative structuring led to better results and efficient computations.

The main contribution was the development of a building block called the Inception module, which is shown in figure 2.12. The success of this module showed that the CNN layers didn't need to be stacked sequentially to achieve good performance. Each module performs a pooling operation, a large and a small convolution, all performed in parallel. 1x1 convolutional layers are used before the larger convolutions to reduce to dimensionality of the inputs. The outputs of all these operations are concatenated at the end. These modules dramatically reduced the number of parameters in the network. They also replaced the fully-connected layers with average pooling, which saves a huge number of parameters.

ResNet

ResNet [22] won the 2015 ILSVRC after achieving 3.57% error on the ImageNet test set, which exceeds typical human-level performance on the dataset. The architecture is well known for its depth, having a total of 152 layers, and for introducing the residual block. ResNets are among the best performing architectures for many computer vision tasks.

ResNet introduced an innovative new module called the residual block, shown in figure 2.13. These blocks include identity skip-connections, which allows layers to copy their

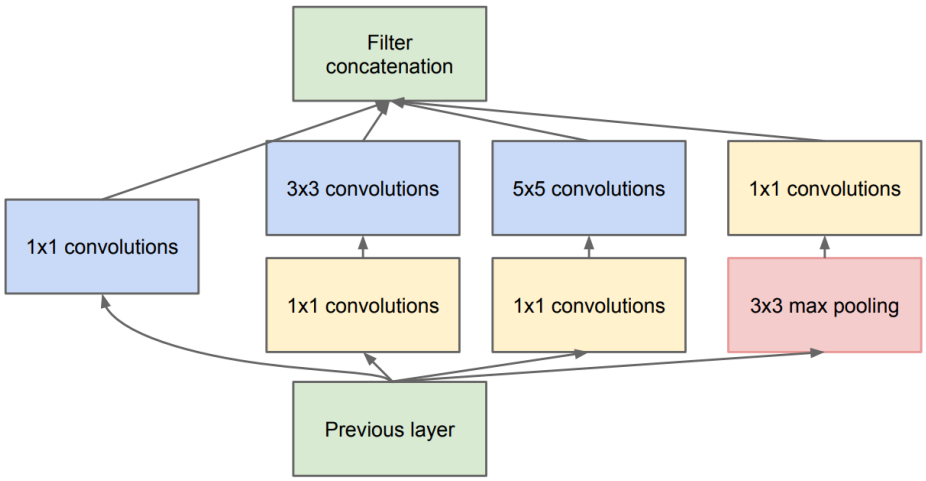


Figure 2.12: The Inception module from the GoogLeNet architecture. Figure extracted from [23].

inputs to the subsequent layer. This makes it possible to effectively train really deep networks. As a layer is fed both the processed output of the previous layer, as well as the unchanged input to the previous layer, the residual block ensures that the layer learns something new. The identity skip-connection also deals with the issue of vanishing gradients during backpropagation. This is due to the addition-operation which distributes the gradient as it flows backwards through the network.

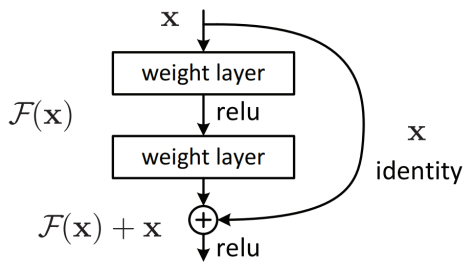


Figure 2.13: The residual block from the ResNet architecture. Figure extracted from [22].

Chapter 3

Related work

This chapter presents related work for this thesis. We begin this chapter by describing how CNN models designed for image classification can be modified to perform semantic segmentation. We cover the Fully Convolutional Network, encoder-decoder architectures, and dilated convolutions, which are all important approaches. Following this, we present multimodal deep learning, with focus on how depth information can be incorporated into CNNs for semantic segmentation.

3.1 Deep learning applied to semantic segmentation

Since the AlexNet architecture won the 2012 ILSVRC, deep learning has been the go-to approach for high-level computer vision tasks such as image classification and object detection. The impressive performance of the CNNs motivated research into how capable they were of solving other computer vision tasks, such as semantic segmentation. Moving from image classification to semantic segmentation is a natural step from coarse to fine predictions. The successful CNNs for classification were a natural basis for the work on applying deep learning to semantic segmentation.

The main challenges with using traditional CNN architectures for semantic segmentation, is that they discard spatial information along the way. Downsampling by pooling layers discard spatial information as they increase the receptive fields. The fully-connected layers at the end completely discards the last remaining spatial information, to produce one output vector with classification scores for the entire image. However, in the context of semantic segmentation this spatial information is important. For the classification task where we are only interested in the "what" of the image, now we also need to know the "where". Several techniques to deal with this have been introduced over the years. In this section we cover different approaches for using CNNs to perform semantic segmentation.

3.1.1 Fully Convolutional Networks

The Fully Convolutional Network (FCN) proposed in 2014 by Long et al. [26] was one of the first successful approaches to using CNN architectures for semantic pixel-wise labelling. They used network architectures previously used for classification, such as VGG-16 and GoogLeNet, and modified them to produce dense pixel-wise predictions. This work is considered a milestone as it showed that CNNs could be trained end-to-end to perform semantic segmentation from input images of arbitrary size.

The fully-connected layers at the end of typical classification network have fixed dimensions, and discard spatial information. However, they can also be viewed as convolutional layers, where the receptive field covers the entire input region. By removing the fully-connected layers, and replacing them with convolutional layers, the network can make predictions on arbitrary inputs. This conversion also means that instead of outputting class scores, we get spatial output maps, or heat maps. This technique to "convolutionalize" a classification network is illustrated in figure 3.1.

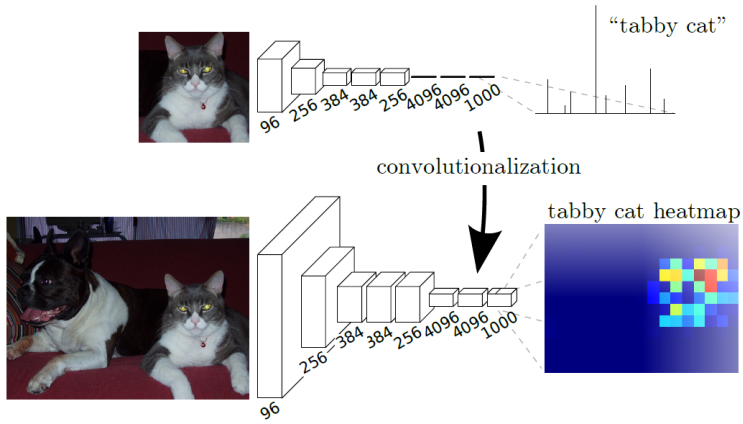
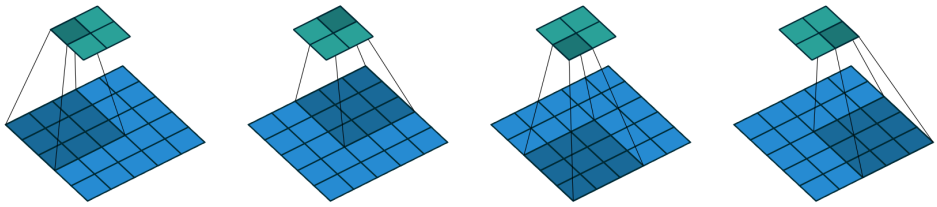
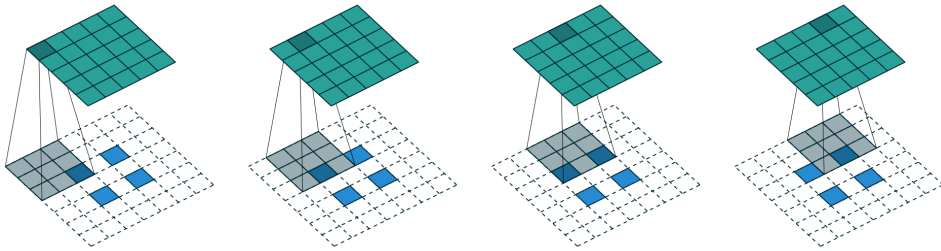


Figure 3.1: Convolutionalizing classification networks to produce heat maps instead of classification scores. Figure extracted from [26].

Due to downsampling in the network, the output dimensions are accordingly reduced compared to the input. To produce dense outputs of the same size as the input, the coarse heat maps need to be upsampled. To accomplish this, the FCN uses transposed convolutions. The terms deconvolutions and fractionally strided convolutions are sometimes used in the literature, however in this thesis we will refer to these as transposed convolutions. This operation can be viewed as a reverse of a convolution, and is illustrated in figure 3.2. When the filters are fixed, the operation can simply be viewed as bilinear interpolation. An alternative approach is to let the filters be learnable, just as the filters in the regular convolutional layers. This means that the network learns both to downsample and upsample, which often gives better results than fixed upsamling.



(a) Convolving a 3×3 filter over a 5×5 input using 2×2 strides, with no zero padding. Using equations 2.20 and 2.21 with $\mathbf{W_1} = \mathbf{H_1} = 5$, $\mathbf{F} = 3$, $\mathbf{P} = 0$, $\mathbf{S} = 2$, gives an output volume of size $[\mathbf{W_2} \times \mathbf{H_2}] = [2 \times 2]$.



(b) The transpose of the convolution in figure (a). It is equivalent to padding the 2×2 input with a 2×2 border of zeros and inserting 1 zero between the inputs, then convolving a 3×3 filter over the input using unit strides. With $\mathbf{W_1} = \mathbf{H_1} = 3$, $\mathbf{F} = 3$, $\mathbf{P} = 2$, $\mathbf{S} = 1$, equations 2.20 and 2.21 give an output volume of size $[\mathbf{W_2} \times \mathbf{H_2}] = [5 \times 5]$.

Figure 3.2: A convolution and its associated transposed convolution. The convolution reduces the dimensions of the input, and the transposed convolution recovers it. Figures extracted from [27].

While using transposed convolutions for upsampling is able to produce an output with the same resolution as the input, it requires refinement. This upsampling produces coarse segmentation maps as a result of the loss of spatial information from max pooling layers. One effect of this is that small objects are not considered at all. In order to refine the upsampling, the FCN makes use of skip-connections to recover some of the spatial information from the earlier layers. Features from the earlier layers are combined with deeper layers. By combining coarse, high level information with fine, low level information, the resulting segmentation maps are improved.

3.1.2 Encoder-decoder architectures

The FCN architecture can be viewed as having two main parts. The first part is a traditional classification architecture which produces low-resolution image representations by encoding them as feature maps. The second part maps the feature maps to pixel-wise predictions, by learning to decode them. This type of architecture is commonly referred to as an encoder-decoder architecture. The success of the FCN inspired many similar encoder-decoder architectures, and is widely considered to be a common forerunner for

such networks. While different architectures often have very similar encoders, such as VGG-16, it is the decoder which sets them apart.

DeconvNet

DeconvNet [28] is a network which builds on the same principles as the FCN. The architecture is relatively symmetrical, as opposed to FCN which has a large encoder and a small decoder. The encoder uses the VGG-16 architecture with the last classification layer removed. DeconvNet uses a decoder that is substantially larger than the FCN-decoder, and also uses unpooling together with the transposed convolutions. The architecture is illustrated in figure 3.3

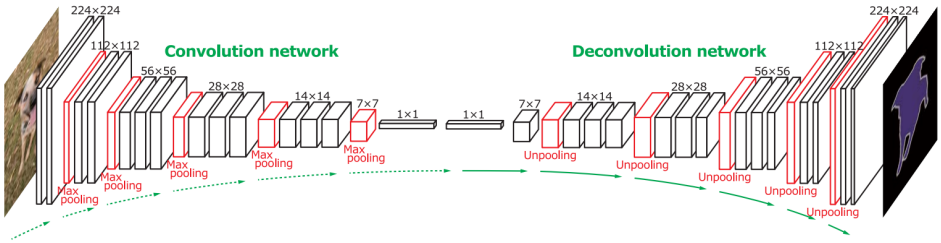


Figure 3.3: The DeconvNet architecture. Figure extracted from [28].

The unpooling layers works as a counterweight to the max-pooling layers, as shown in figure 3.4. The max pooling indices are saved by the encoder, and the unpooling layers re-use them to accurately upsample feature maps. As seen in the figure, the unpooling layer only produces sparse feature maps. For this reason, these layers are followed by transposed convolutional layers. These layers densify the feature maps, by using learned transposed convolutional filters. This approach gave better segmentation results compared to the FCN. However, DeconvNet had a significantly higher number of parameters, as it uses two fully-connected layers in the encoder. This led to higher memory requirements.

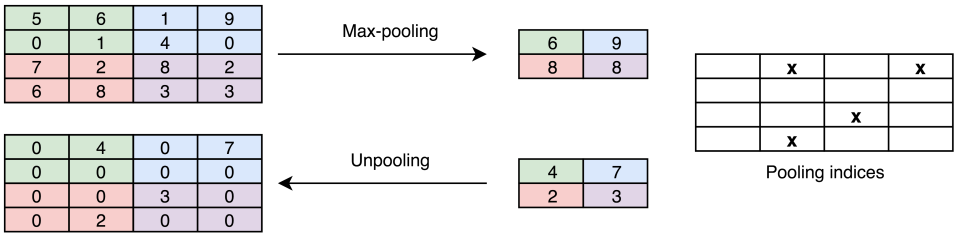


Figure 3.4: Illustration of max pooling and unpooling.

SegNet

The development of the SegNet architecture [29] was primarily motivated by road scene understanding applications. For such a system to be applicable in a real-world scenario, it should be able to perform in real-time. Thus, the architecture was designed to be efficient both in terms of memory usage and inference time. It shares many similarities with DeconvNet, with some important modifications. SegNet is fully-convolutional, and does not use any fully-connected layers in the encoder such as DeconvNet. This gives a much smaller and faster network, as those layers typically hold a huge number of parameters. SegNet also uses unpooling layers for up-sampling, but does not use transposed convolutional layers. These have been replaced by convolutional layers, and the sparse feature maps from the unpooling layers are convolved with a set of learnable filter banks. This means that the network does not learn up-sampling.

U-Net

The U-Net architecture [30], originally proposed as an approach for medical imaging, is similar to the SegNet architecture, but with some important differences. Instead of reusing pooling indices to relate corresponding parts of the encoder and decoder, they transfer entire encoder feature maps to the corresponding decoders. Here they are concatenated with the decoder feature maps, which have been upsampled using transposed convolutions. These "skip-concatenation-connections" means that features which are lost during pooling can be recovered. The architecture is illustrated in figure 3.5.

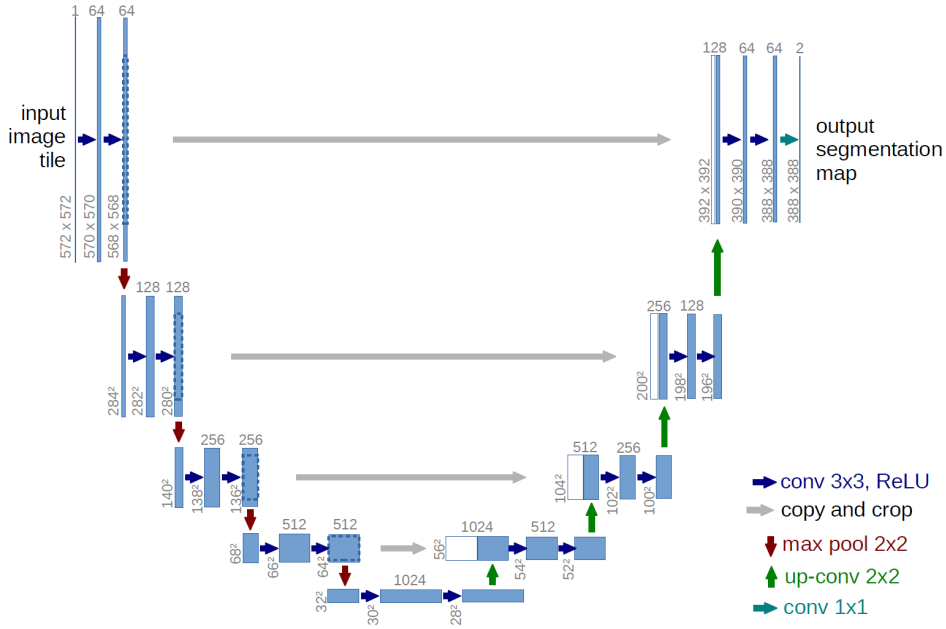


Figure 3.5: The U-Net architecture. Figure extracted from [30].

ENet

ENet [1] is an architecture inspired by SegNet, but reworked to make it more efficient in terms of both speed and memory. They achieve this by introducing "bottleneck modules" with skip connections, as inspired by ResNet. The bottleneck module is used throughout the entire encoder-decoder architecture, and different forms of convolutions are used at different locations. While demonstrating similar performance to SegNet, it is 18x faster and has 79x less parameters. ENet performs below state-of-the-art in terms of accuracy, but it is one of the most efficient networks. It has a reported inference time of 13 ms on the popular Cityscapes benchmark dataset [31]. This result was achieved on a Nvidia Titan X GPU.

3.1.3 Dilated convolutions

The FCN and most other encoder-decoder architectures are constructed by re-purposing classification networks for dense predictions. Producing dense predictions requires integrating context knowledge from different spatial scales, and balancing local and global information. Local information is important for good pixel-level accuracy, while global information can help resolve ambiguities in local regions. The pooling layers are beneficial in classification networks, because the receptive fields increases. However, they dispose of global context information, and are ill-suited for tasks which require dense predictions. A technique which is better suited for dense predictions is to use dilated convolutions.

Dilated convolutions, also known as atrous convolutions, is a special type of convolution which lets us integrate context knowledge efficiently. Instead of the normal convolution presented earlier, we make use of upsampled, or dilated, filters. A parameter called the dilation rate, defines a spacing between each element in the filter. Figure 3.6 shows examples of dilated filters.

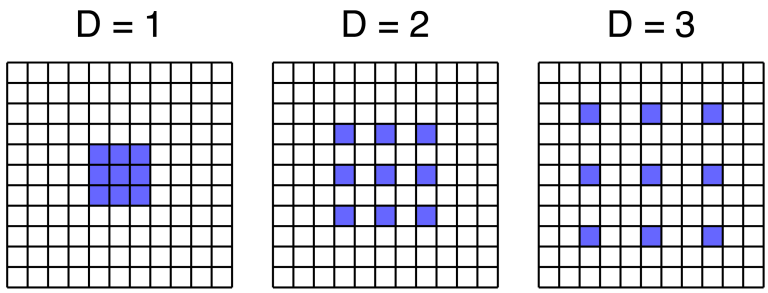


Figure 3.6: A 3×3 filter with various dilation rates.

In practice, the filter is expanded, and the empty elements are filled with zeros. This means that for a dilation rate greater than one, the filter weights are matched with non-adjacent elements. If the filter has size 3x3, and the dilation rate is 2, this corresponds to the same receptive view as a 5x5 filter, while only using 9 parameters. This means that by stacking

dilated convolutions, the receptive field grows at a faster rate than the number of parameters. This can be used to produce efficient dense feature extraction for any resolution.

DeepLab

The DeepLab architecture has been proposed in several variants [32, 33, 34], and uses dilated convolutions extensively. It is based on architectures for image classification, such as VGG-16 and ResNet-101, which are re-purposed for dense predictions, by introducing dilated, or atrous, convolutions. They also propose the Atrous Spatial Pyramid Pooling (ASPP), which encodes objects and image context at multiple spatial scales. In order to improve the localization performance of the segmentation, they combine the outputs with fully connected Conditional Random Fields. This approach refines the segmentation significantly. DeepLab is one of the best performing networks for semantic segmentation, and outperforms most other architectures on almost all datasets.

3.2 Multimodal deep learning

When humans experience the world, we say that the experience is multimodal. In this context, a modality refers to the way in which something is experienced. We see objects and hear sound using different sensory modalities such as vision and hearing. The information from the different modalities is highly correlated, and complement each other when processed. Many technological systems rely on different sensory modalities, and research problems are characterized as multimodal, when it includes multiple such modalities. Multimodal deep learning aims to build deep learning models that can process and relate information from multiple modalities.

In the context of autonomous driving, multimodal approaches can contribute to improving the robustness of scene understanding systems. A unimodal approach, such as relying only on RGB image data for perception, may perform poorly in challenging environments. Image data is not well-equipped to deal with natural factors such as shadows, sun glare and poor illumination. The changing environmental conditions throughout the days and seasons, is especially challenging for UGVs operating off-road in unstructured environments. Examples of environmental challenges that an UGV may encounter are shown in figure 3.7.

3.2.1 Multimodal semantic segmentation using RGB-D data

The deep learning approaches presented in section 3.1 are concerned with learning semantic segmentation entirely from RGB data. The models which are currently state-of-the-art for this task operate using only RGB images as inputs. Relating information from multiple sensory modalities has the potential to increase performance. Many autonomous vehicles are equipped with multiple sensors which provides more information about the environment. Depth information is of special interest, as many systems are equipped with either

Figure 3.7: Examples of challenges in various environments. The first image illustrates a scene with poor illumination and very prominent sun glare. The second and third images illustrates significant appearance variations across seasons for forested scenes.

stereo cameras or dedicated depth sensors such as Lidars. Depth data encodes structural information about the scene, and provides information which can compliment the RGB data. Depth and RGB data can also be seen as correlated, e.g. when depth discontinuities manifest themselves as strong edges in images. Information from these two modalities combined provides a richer description of a scene. In this section we cover related work for combining RGB and depth data in CNNs for semantic segmentation.

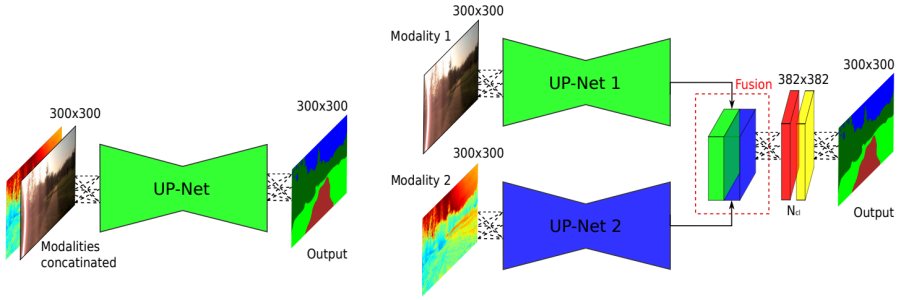
Depth encoding

The networks considered until now are designed to take three-channel RGB images as inputs. Depth data can be encoded in various manners to make it applicable in such networks. The simplest approach is to encode it as a single-channel image, where the intensity of each pixel represents the corresponding depth. The network can be modified to accept this input, or the data can be replicated over three channels to mimic an RGB image.

Gupta et al. presented an alternative technique known as Horizontal Height Angle (HHA) [35]. HHA encodes the depth into three channels, where each pixel encodes the horizontal disparity, the height above ground, and the angle with respect to gravity of the surface. Encoding depth into three channels, similar to an RGB image, makes it possible to input the depth images into CNN models such as those seen before. The model can then learn features from the depth data, just as it would for RGB data. This technique for representing depth has shown good results, however it involves a high computational cost.

Fusing modalities

CNNs learn feature representations from the data it is presented with, and when working with multiple modalities, features from each modality need to be combined or fused. Different approaches to multimodal deep learning from RGB-D data differ in several ways, and one of the most distinct ways is where the fusion takes place. The approaches are generally classified as early or late fusion, illustrated in figure 3.8.



(a) Channel Stacking (b) Late-fused Convolution

Figure 3.8: Early fusion, using channel stacking, is shown on the left, and late fusion is shown on the right. Figure extracted from [36].

Early fusion approaches expose the network to multiple modalities already from the input or from early layers. A trivial way to include depth information is to simply stack it as a separate color channel to the RGB data, as shown in figure 3.8. The network can then be trained on this data, assuming the input has four instead of three channels. This approach lets the network learn combined features from end to end. Extending an existing architecture to accept one additional input channel is a small task, and this approach is the simplest extension. The increase in computational cost is also small compared to processing only RGB data.

When Long et al. presented the FCN architecture, they experimented with channel stacking as an early-fusion approach. Depth data in the form of a single-channel image was stacked as a separate color channel in the corresponding RGB image, to produce a 4-channel input. However this approach was not very successful, due to the difficulty of propagating meaningful gradients through the entire network [26].

Late fusion approaches typically make use of network architectures with several data streams, one for each modality. The data streams are processed individually up to a point, then followed by a late fusion network, which finally produces a semantic segmentation. Each branch extract features from separate modalities, and may be symmetrical, or designed specifically for each modality. Furthermore, the branches may be trained separately, before the late fusion network is trained to produce a combined prediction.

Eitel et al. [37] proposed a late-fusion approach for object detection. They use a network with two data-streams, one for RGB data and one for depth data. Each of the streams use a typical classification CNN architecture to extract features. The two streams converge in a fully-connected layer and a softmax classifier.

The FuseNet architecture [38] is a multimodal architecture inspired by popular encoder-decoder approaches. It uses two encoder branches which extract features from the two modalities. The depth features are fused into the RGB features at several points, as we

progress deeper through the network. The fusion layers are implemented as element-wise sums. Two different approaches were investigated, dense and sparse fusion, depending on how often depth data is fused into the RGB data. Both fusion approaches led to similar results, which were competitive results with state-of-the-art approaches on the SUN RGB-D benchmark. The architecture is illustrated in figure 3.9.

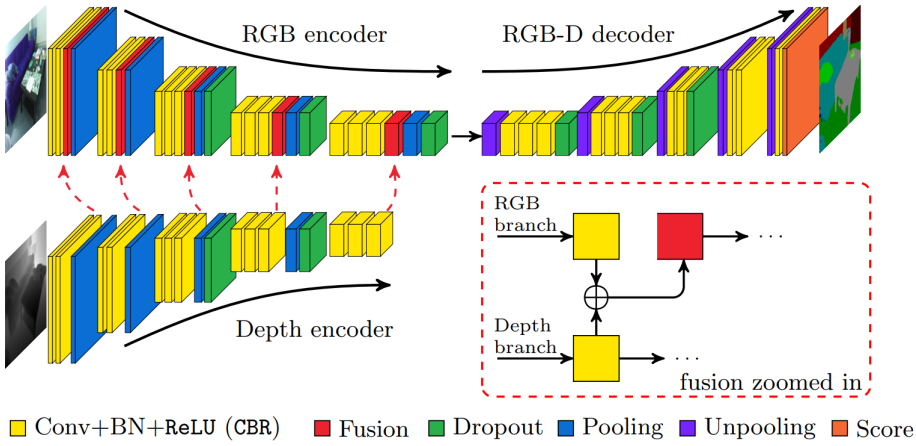


Figure 3.9: The FuseNet architecture. Figure extracted from [38].

Chapter 4

Method and implementation

This chapter presents the methods used in the work presented in this thesis. First an overview of the work is presented and related to the overall goals of the project. We then present a base architecture, before covering how it can be extended to operate on multimodal data. In the last sections we describe how the network is trained and the implementation.

4.1 Overall approach

This thesis investigates how to fuse depth information with RGB data using convolutional neural networks for semantic segmentation. This involves proposing an architecture suitable for practical applications, such as scene understanding systems for autonomous vehicles. Based on the context of the project and a review of the related work presented in chapter 3, a relevant architecture is chosen as a basis for the work. The base architecture which is designed for RGB inputs will be extended to incorporate depth information. Furthermore, an approach for combining RGB and depth information needs to be chosen. This involves both the location of fusion, as well as how to implement the fusion.

4.1.1 Choice of base architecture

Numerous CNN architectures have been proposed for semantic segmentation, and while they are all based on the same building blocks, each architecture has its distinguishing features. Different architectures have different strengths, and choosing an architecture often involves a trade-off between accuracy and efficiency. In terms of these factors, two architectures stand out. The DeepLab architecture outperforms most other architectures in terms of accuracy on almost every RGB dataset by a significant margin. In terms of speed and efficiency, ENet provides some of the fastest inference times on many popular datasets, while also being a very small architecture. It does however have below state-of-the-art accuracy compared to methods such as DeepLab. Based on these insights, ENet and DeepLab are the most relevant architectures to consider for multimodal extension.

Figure 4.1 shows the accuracy per parameter for various CNNs. Most of these are classification networks, except for ENet. The DeepLab model builds on the ResNet-101 model to re-purpose it for semantic segmentation. The figure can also be interpreted as information density, which is an efficiency metric that illustrates how well an architecture takes advantage of its potential learning ability. Deep networks are known to be inefficient in exploiting their full learning power, and the figure illustrates that the ENet model is the best model for forcing all neurons to learn a given task. The ResNet-101 model performs worse in terms of this metrics, presumably due to being a very deep architecture. Based on these results we may hypothesize that the ENet architecture should be able to learn effectively also when additional modalities are introduced.

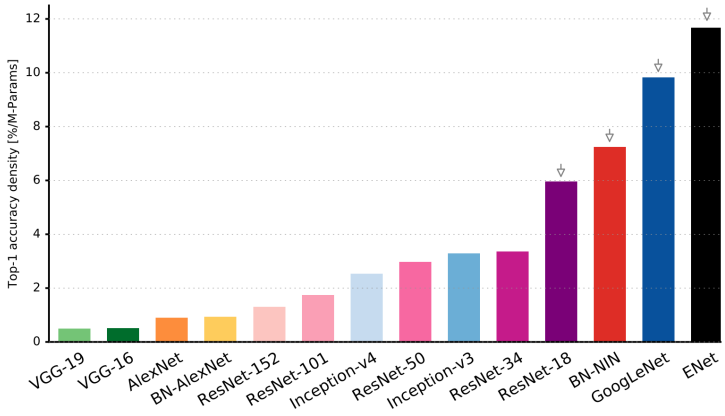


Figure 4.1: Accuracy per parameter for various networks for the ImageNet challenge. Figure extracted from [39].

In the context of autonomous driving, efficient run-time performance is an important factor. As new modalities are incorporated into the system, more data needs to be processed to produce predictions. This may lead to significantly increased computational costs and reduced efficiency. Furthermore, the size of the architecture grows, which leads to higher memory requirements. Memory is often considered to be the largest bottleneck when constructing architectures, and large networks may be too complex to fit on relevant GPUs. As the ENet architecture is the most efficient architecture, it is also the most attractive to investigate for multimodal extension. Related works such as [40] and [41] have shown promising results with extensions of the ENet model, which further motivates its use in this thesis. Other architectures such as DeepLab have the potential to achieve better results, but may also grow too large to be suitable for practical applications. We consider the efficiency factor to be more important than the accuracy, and choose the ENet architecture as a base model. The ENet architecture was developed for real-time semantic segmentation from RGB inputs, and will be extended to incorporate depth information.

4.2 The ENet architecture

The architecture of the ENet network is presented in table 4.1. The architecture has one initial stage and five main stages, or phases, as highlighted by the horizontal lines in the table and the first digit in each block name. Combined, these stages make up an encoder-decoder architecture. Stage 1, 2 and 3 make up the encoder, while stage 4 and 5 belong to the decoder. The final layer, denoted by "fullconv" is a transposed convolutional layer. C is the number of classes. The middle column describes the relevant module type and network parameter for that stage.

Name	Type	Output size
initial		$16 \times 256 \times 256$
bottleneck1.0	downsampling	$64 \times 128 \times 128$
4×bottleneck1.x		$64 \times 128 \times 128$
bottleneck2.0	downsampling	$128 \times 64 \times 64$
bottleneck2.1		$128 \times 64 \times 64$
bottleneck2.2	dilated 2	$128 \times 64 \times 64$
bottleneck2.3	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.4	dilated 4	$128 \times 64 \times 64$
bottleneck2.5		$128 \times 64 \times 64$
bottleneck2.6	dilated 8	$128 \times 64 \times 64$
bottleneck2.7	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.8	dilated 16	$128 \times 64 \times 64$
Repeat section 2, without bottleneck2.0		
bottleneck4.0	upsampling	$64 \times 128 \times 128$
bottleneck4.1		$64 \times 128 \times 128$
bottleneck4.2		$64 \times 128 \times 128$
bottleneck5.0	upsampling	$16 \times 256 \times 256$
bottleneck5.1		$16 \times 256 \times 256$
fullconv		$C \times 512 \times 512$

Table 4.1: The ENet architecture. The output sizes are given for an example input of size 512×512 pixels. Table reconstructed from [1].

The main building block of the ENet architecture is the bottleneck module which comes in three variants, and is illustrated in figure 4.2 together with the initial block. These modules bear a resemblance to the residual blocks from the successful ResNet classification architecture. Each bottleneck module has a skip-connection in parallel with various types of convolutions. A 1×1 convolution for feature reduction is followed by a main convolutional layer, which is the "conv" block. Then a 1×1 convolution for feature expansion is used, followed by a spatial dropout regularizer [42]. The two branches are merged with element-wise summation. Batch normalization and PReLU activations are used between all convolutions.

The "conv" block is either a regular, dilated or a transposed convolution. The filter size

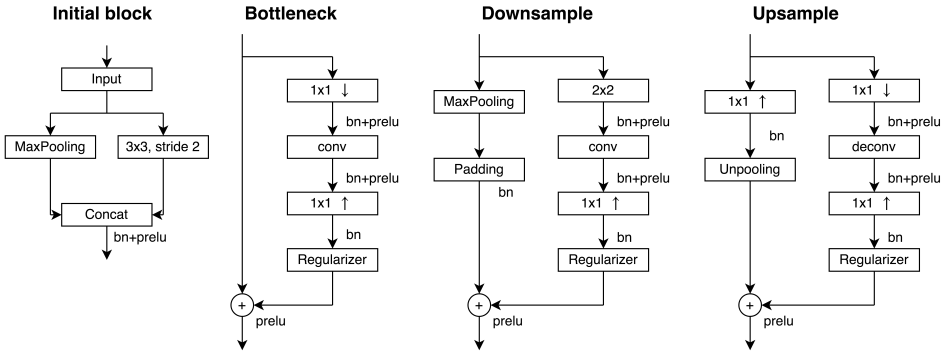


Figure 4.2: ENet bottleneck modules

is relatively small, being only 3×3 in size. Occasionally, the network uses asymmetric convolutions, which involves splitting a $n \times n$ filter into two smaller ones, one with a $n \times 1$ filter and the other with a $1 \times n$ filter. In their architecture they use $n = 5$, and the cost of those two operations is comparable to a single 3×3 convolution, while also increasing the receptive view.

A downsampling variant of the bottleneck module is used in the encoder. This has a max pooling layer in the main branch, and the 1×1 reduction is replaced by a 2×2 convolution with stride 2. A 1×1 reduction was found to discard too much information, and the increased filter size means that the full input is taken into consideration. While these layers are more expensive, they improve information flow and accuracy. Apart from the initial block, there are only two downsampling operations in the encoder. Instead of downsampling numerous times to increase the receptive fields of the filters, the network uses dilated convolutions for this purpose. This allows the filters to gather more context. The upsampling bottleneck module uses max unpooling with the pooling indices from the downsampling bottlenecks. The "conv" block is a transposed convolution. The padding is replaced with a 1×1 convolution.

4.2.1 Multimodal extension

When introducing a new modality, a natural extension is to include an additional encoder branch. The two encoder branches can extract features from RGB and depth images, using two separate inputs. To keep the model simple, we let the depth-encoder be identical to the RGB-encoder. All parameters are the same, which gives symmetric feature extraction streams. The features can then be fused into a combined representation. The decoder is left unchanged, and now operates on this combined representation of the data.

Another design choice for keeping the model efficient, is to use simple depth representations. Many approaches use HHA depth encoding which has achieved good results. However, Hazirbas et al. [38] argue that the HHA representation do not contain more in-

formation than the raw depth data itself. Furthermore, it involves a high computational cost, and is most relevant when the depth information is provided as 3D data, such as point clouds. For practical applications it may be more convenient to operate on raw depth images to avoid computationally expensive depth encoding at run-time. For this reason we design the model to operate on single-channel depth images. To make the RGB and depth features compatible, the depth images are normalized to the same range as the RGB images, [0,255]. This may also contribute to making the model easier to train.

The optimal locations to fuse features remains an open question. Different approaches have been proposed, without finding a definite best approach. It is also unsettled whether the best approach is single fusion, or multiple points of fusion. For these reasons, two variants of the proposed architecture will be considered. These differ in the locations where fusion takes place. For the ENet base architecture, we consider three locations to be potential points of interest for fusion. The first point is after phase 1, just before a downsampling operation. The first phase is relatively short compared to phase 2 and 3. However, relevant features should be extracted in both branches at this stage. Fusing before the downsampling is important, to avoid loosing important information. The second and third potential fusion points are after phase 2 and 3. These phases are relatively similar, however phase 3 does not perform downsampling at the start. Arranging the fusion points evenly throughout the encoder ensures that multiple levels of depth features are taken into consideration. Figure 4.3 illustrates the proposed architecture.

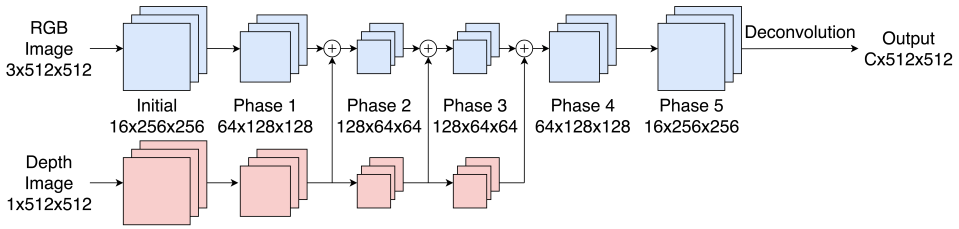


Figure 4.3: The proposed architecture and intermediate volume sizes for input size 512x512.

The first variant of the architecture fuses only at one point, namely after stage 1. This means that we only capture the low-level features extracted from the depth data, and fuse them into the RGB feature maps. The motivation for this approach is to let the depth information complement the RGB data only slightly. By fusing only this early, the network learns combined features from here and to the end. As meaningful information should be present in both modalities, the augmented RGB features may be a more suitable representation. However, it is not a given that these depth features can complement the RGB features in a positive way. If the depth data is noisy or of poor quality, fusing this early may contribute only to corrupting the RGB features. As the combined representation propagates through the network, it may result in a poorer result than what is achieved using only RGB data. In terms of efficiency, this approach does not include much additional computational cost over the baseline model. The depth-encoder becomes relatively small, as it only requires phase 1. This means that the model shouldn't grow much in size.

The second variant performs feature fusion at all the three points, an approach similar to that of [38]. The intermediate feature maps from the depth-encoder are fused into the RGB feature maps at several locations as the network goes deeper. This choice is motivated by the fact that features learned at multiple levels are valuable, as noted earlier, and in [43]. Reasoning across multiple levels of abstraction and scale has proven beneficial in many parts of computer vision. This is one of the properties which has led to the DeepLab model being the most successful architecture for semantic segmentation. Adopting a similar approach here, takes advantage of multiple levels of depth features, which may complement the RGB features. However this comes at the cost of a substantially larger encoder. For the three points described above, the depth-encoder takes the same size as the RGB-encoder, which gives larger memory requirements. The depth data needs to propagated through the entire encoder, thus increasing the computational expenses as well.

4.2.2 Fusion strategy

Two fusion strategies were considered and evaluated for the architecture, channel-wise concatenation and element-wise summation. Here we will first briefly discuss the concatenation, and why it is not used. Following this we consider the element-wise summation, and justify it as fusion strategy.

Channel-wise concatenation

The most straightforward way to combine features from the two branches is to simply concatenate them. Channel-wise concatenation effectively doubles the number of feature maps at the fusion point. The convolutional layers in the network extend through the full depth of the input volumes, and the depth of the output volume is not affected by the depth of the input. This means that the feature maps from both the branches will have the same dimensions, despite the inputs to each branch having different number of color channels. While this fusion strategy is simple, the unavoidable increase in memory requirements is a significant drawback. Another shortcoming is the fact that concatenation does not fully exploit the complementary properties of the two modalities, as features are kept separate.

Element-wise summation

Element-wise summation as a fusion strategy does not suffer from the same memory problem as concatenation. It is also straight forward to implement, as the feature maps from the two branches are of the same size. Most approaches to multimodal RGB-D deep learning use this technique. While concatenation increases the number of features to learn from, element-wise summation actually fuses the features. This technique is more attractive for modalities which contain complementing information. If both branches produce strong activations for a relevant feature, element-wise summation produces an even stronger activation. Concatenation on the other hand would represent this as two separate strong activations, not taking full advantage of the complementing information.

Special care is taken when choosing the exact point of fusion using element-wise summation. The two alternatives to consider is before or after a ReLU operation is applied to the feature maps. While the baseline ENet model makes heavy use of PReLU activations, we consider for the moment the case of ReLU activations, to justify our choice.

Consider the CNN architecture having L layers, where the ReLU activation for simplicity is not a separate layer, but merged with the associated layer. We can represent the total weights and biases of the network as $(W, b) = \{W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}, ..., W^{(L)}, b^{(L)}\}$. Here $W^{(i)}$ holds the weights associated with the connections between layer i and layer $i + 1$, and $b^{(i)}$ is the bias in layer $i + 1$. The input to layer i is $A^{(i-1)}$, and it produces the output $A^{(i)}$ in two stages. First by computing:

$$Z^{(i)} = W^{(i-1)} A^{(i-1)} + b^{(i)} \tag{4.1}$$

Applying the ReLU activation function $f(x) = max(0, x)$ gives:

$$A^{(i)} = f(Z^{(i)}) \tag{4.2}$$

As the two encoder branches are identical, we let the layer index i refer to the two corresponding layers. Each of the encoder branches will produce feature maps, $Z^{(i)}_{RGB}$ and $Z^{(i)}_{depth}$. Element-wise summation can be performed either before or after applying the ReLU activation. Fusing the the data before the ReLU operation involves computing:

$$Z^{(i)}_{fused} = Z^{(i)}_{RGB} + Z^{(i)}_{depth}, \tag{4.3}$$

$$A^{(i)}_{fused} = f(Z^{(i)}_{fused}) = f(Z^{(i)}_{RGB} + Z^{(i)}_{depth}) \tag{4.4}$$

Alternatively, we can fuse the feature maps after the ReLU operation:

$$A^{(i)}_{RGB} = f(Z^{(i)}_{RGB}), \tag{4.5}$$

$$A^{(i)}_{depth} = f(Z^{(i)}_{depth}), \tag{4.6}$$

$$A^{(i)}_{fused} = A^{(i)}_{RGB} + A^{(i)}_{depth} = f(Z^{(i)}_{RGB}) + f(Z^{(i)}_{depth}) \tag{4.7}$$

Given two real-valued feature maps, $Z^{(i)}_{RGB}$ and $Z^{(i)}_{depth}$, and the function f defined as above, the following holds:

$$f(Z^{(i)}_{RGB} + Z^{(i)}_{depth}) \leq f(Z^{(i)}_{RGB}) + f(Z^{(i)}_{depth}) \tag{4.8}$$

This inequality tells us that the activation of the fused features produces a weaker signal than the fusion of the activations. The sum can strengthen the activation map, while also preserving important features from both branches. This is illustrated in figure 4.4. Based on this insight, we adopt element-wise summation after ReLU layers, as the fusion strategy.

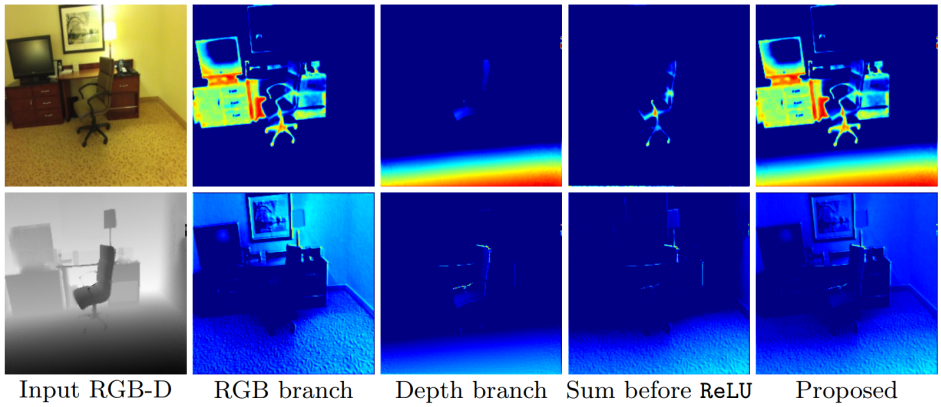


Figure 4.4: Effect of the proposed fusion strategy. The figure is extracted from the FuseNet paper [38]. The first column shows the RGB and depth inputs to the network. The second and third columns visualize 2 out of 64 feature maps produced by the same convolutional layer in each branch after batch normalization has been applied. The fourth and fifth column visually illustrates the effect of fusion before and after the ReLU layer. As seen in the last column, strong activations are preserved from both modalities.

4.3 Learning description

When training the network, a training set S of N training examples is used, which can be represented as $S = \{(\mathbf{X}_i, \mathbf{D_i}, \mathbf{Y_i}) \text{ for } i = 1, ..., N\}$. $\mathbf{X_i}$ is an RGB image, $\mathbf{X}_i \in \mathbb{R}^{H \times W \times 3}$, and $\mathbf{D_i}$ is the corresponding depth image, $\mathbf{D}_i \in \mathbb{R}^{H \times W \times 1}$, both of spatial resolution $H \times W$. $\mathbf{Y_i}$ is the corresponding ground-truth mask, $\mathbf{Y}_i \in \mathcal{L}^{H \times W}$, where the label set is defined as $\mathcal{L} = \{1, 2, ..., C\}$. Each label index corresponds to each of the C pre-defined classes.

The convolutional neural network consists of L layers, each with corresponding weights, which are all gathered in a weight matrix $\mathbf{W}$, defined as $\mathbf{W} = [\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, ..., \mathbf{w}^{(L)}]$. The final layer, produces an output of the same spatial resolution as the input, with the number of classes C, along the last dimension. Considering the input to be a four-channel image, where the first three channels are presented to the RGB encoder, and the last channel to the depth encoder, the network represents a function $f(\mathbf{X_i}, \mathbf{D_i}, \mathbf{W})$, performing the mapping

$$f : \mathbb{R}^{H \times W \times 4} \mapsto \mathbb{R}^{H \times W \times C} \tag{4.9}$$

Associated with each output pixel $\mathbf{x}$, is a vector of dimension C, holding the class scores. By using the softmax function, we can map this to a probability distribution by "squashing" the values to the range $(0, 1)$. Softmax for each pixel is computed as

$$p_c(\mathbf{x}) = \frac{e^{a_c(\mathbf{x})}}{\sum_{i=1}^{C} e^{a_i(\mathbf{x})}} \tag{4.10}$$

where $a_c(\mathbf{x})$ is the output activation in channel c at pixel x. Ideally $p_c(\mathbf{x})$ works as the approximated maximum function, where $p_c(\mathbf{x}) \approx 1$ for the c with the maximum activation $a_c(\mathbf{x})$, and $p_c(\mathbf{x}) \approx 0$ for all other c. The deviation from the true class can be computed for every pixel, using the cross-entropy loss

$$\mathcal{L}(p, y) = -[y \log(p) + (1 - y) \log(1 - p)] \tag{4.11}$$

where y is the true class, and p is the predicted probability for that class. To find the optimal weights, we need to solve

$$\mathbf{W}^* = \underset{\mathbf{W}}{argmin} = \frac{1}{N} \sum_{i=1}^{N} \sum_{\mathbf{x}} \mathcal{L}(p_{y(\mathbf{x})}(\mathbf{x}), y(\mathbf{x})) \tag{4.12}$$

where $y(\mathbf{x})$ is the true class of pixel x.

4.4 Implementation

The proposed architecture is implemented using the Caffe framework. Building an architecture in Caffe is done by specifying the various layers and the flow of data through the network, as illustrated in figure 4.5. Data flows through the network in the form of blobs, which lets us interface with the data as standard arrays. Each layer takes zero or more blobs as input, transforms the blob, and outputs zero or more blobs. This flow is visualized as flowing from bottom to top, where the input blob is the bottom blob and the output blob is the top blob. Each single layer is specified in a special file, and the top and bottom parameters define the flow of data, resulting in a fully specified architecture.

```
name: "LogReg"
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  data_param {
    source: "input_leveldb"
    batch_size: 64
  }
}
layer {
  name: "ip"
  type: "InnerProduct"
  bottom: "data"
  top: "ip"
  inner_product_param {
    num_output: 2
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip"
  bottom: "label"
  top: "loss"
}
```

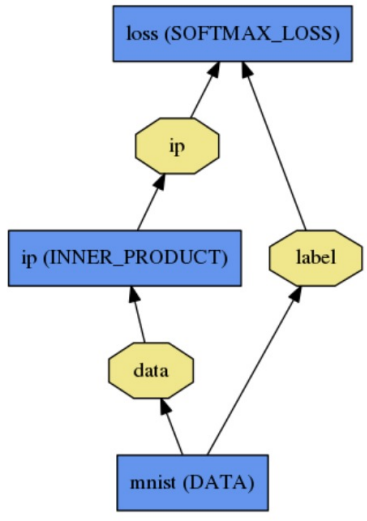


Figure 4.5: A Caffe model and the visualization as a computation graph.

To accelerate the training process, we make use of GPU-based parallel computing. The main advantage of using GPUs over traditional CPUs, is their ability to perform computing operations in parallel. While modern CPUs may typically have few, but complex and powerful processing cores, modern GPUs may have anywhere from some hundred to several thousand cores. These cores can each process simple sub-routines in parallell, which leads to a much shorter training time compared to CPU-based training.

In our implementation we use CUDA and cuDNN for paralell computing, to accelerate training. CUDA is a parallel computing platform developed by Nvidia. It enables developers to create GPU-accelerated applications, where the non-sequential portions can run on thousands of GPU cores in paralell. Nvidia has also developed a popular software library called "The NVIDIA CUDA Deep Neural Network (cuDNN) library. While CUDA has many general-purpose applications wherever data may be processed in parallell, cuDNN is developed specifically for deep learning purposes. The library provides optimized implementations of routines commonly used in CNNs, such as convolutions, pooling, normalization and activation layers. Table 4.2 describes the system which was used for the implementation.

Component	**Description**
Processor (CPU)	Intel(R) Core(TM) i7-4790k
Graphics processing unit (GPU)	Nvidia GeForce GTX 1060
GPU memory	6 GB
Memory (RAM)	16 GB
Operating system	Ubuntu 16.04
Architecture	64-bit
CUDA version	7.5
cuDNN version	5.0

Table 4.2: System specifications

Chapter 5

Experiments

This chapter presents details regarding the experiments which have been carried out. The first section covers the datasets which are used for training and testing the proposed approach. We then cover the experimental design and the training configuration. In the last section we present metrics which are used to evaluate the performance.

5.1 Datasets

Two datasets have been used to test the performance of the proposed approach. As the background context of this project is scene understanding for autonomous vehicles, datasets relevant to this task were chosen. Both of these datasets provide dense pixel-level annotations for the images. The first dataset is the Cityscapes dataset [31], which contains images from urban road scenes. This dataset is commonly used as a benchmark for semantic segmentation methods in the context of autonomous driving. Its popularity motivates its use in this thesis, as our results are easily comparable with other approaches. The second dataset is the Freiburg forest dataset, provided by Valada et al. [36], which is a less commonly used dataset for scene understanding. The Freiburg forest dataset does however provide densely annotated images from forested off-road scenes, which is an interesting domain to investigate.

Several datasets designed for scene understanding in the context of driving exist, however the subset that includes depth data is relatively small. This is another factor which motivates the use of these datasets. Apart from their relevance, the two datasets are also attractive choices as they differ in the way depth information is generated. Cityscapes provides depth maps precomputed using a stereo camera and the semi-global matching (SGM) algorithm [44]. Freiburg forest on the other hand uses depth predicted from single images, using a separately trained deep convolutional neural network. One may expect that the overall characteristics of the depth data has a strong impact on what features the network is able to learn. This motivates further investigation into how well the two different approaches can complement RGB data.

5.1.1 Cityscapes dataset

The Cityscapes dataset [31] is a large-scale dataset designed for semantic understanding of complex urban scenes. It contains a diverse collection of images from 50 different cities, captured during several months and at different daytimes. In total, 5000 images with a resolution of 2048x1024 pixels are supplied with high quality instance-level pixel-wise annotations, as well as disparity maps. Example data from the dataset is shown in figure 5.1. The dataset provides a pre-defined training/validation/test split. The data is not split randomly, but in such a way that each split contains data from different street scene scenarios. The result is 2975 training and 500 validation images, where the annotations are publicly available, and 1525 test images where the annotations are withheld for the purposes of benchmarking. An online evaluation server is used for evaluating performance on the test set.

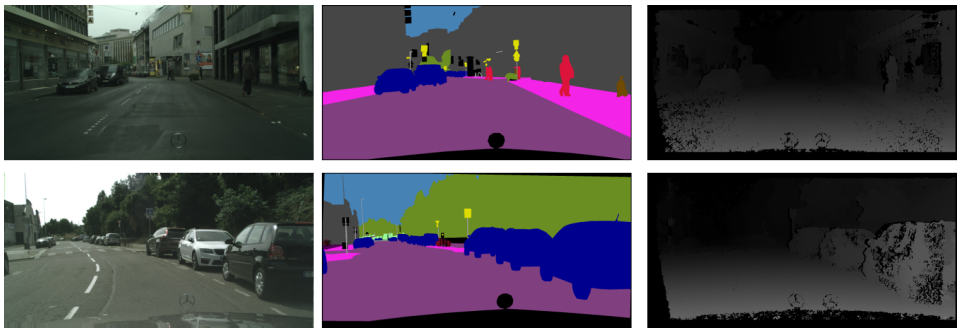


Figure 5.1: Example data from the Cityscapes dataset [31]. The first column is the input image and the second column is the corresponding colorized ground-truth image. The third column shows the disparity map, where bright pixels are close, while dark pixels are farther away.

The dataset provides dense pixel-wise annotations of 30 different classes, which are grouped into 8 categories: flat surfaces, constructions, nature, vehicles, sky, objects, humans, void. The void class indicates that the ground truth of the pixel is unclear, and such pixels should be ignored during training and evaluation. The distribution of the different classes can be seen in figure 5.2. The dataset is somewhat unbalanced, with spatially large classes such as roads, buildings and vegetation dominating. This property is also illustrated in figure 5.1. Classes that are too rare are excluded during testing, resulting in a total of 19 classes for evaluation, indicated in figure 5.2. We adopt this same approach.

The disparity images provided with each image, are computed from stereo camera systems, using the semi-global matching (SGM) algorithm [44]. Computing depth from stereo is a popular approach for systems equipped only with cameras, and the SGM algorithm is arguably one of the most widely used for computing disparity. For images of high resolution, using the SGM algorithm is typically a computationally heavy task, and it also requires a significant amount of memory. These factors pose challenges for systems with strict hardware constraints, such as embedded systems. However, fairly recent implemen-

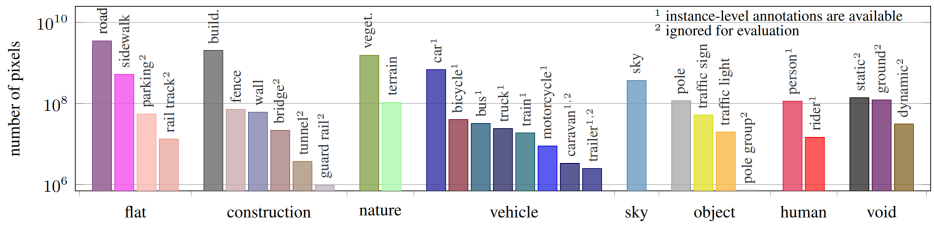


Figure 5.2: The classes present in the Cityscapes dataset, number of densely annotated pixels per class and their categories. Image extracted from [31].

tations which focuses on efficient GPU-based implementations such as [45] have achieved close to real-time performance even for images with a resolution of 640x480 pixels. This makes the depth data from the Cityscapes dataset interesting to investigate for practical applications.

5.1.2 Freiburg forest dataset

The Freiburg forest dataset from Valada et al. [36] is one of few publicly available datasets which focuses on off-road scenes. Most research on semantic outdoor scene understanding has been focused on urban scenes, and most popular large-scale segmentation datasets are gathered from such structured scenes. Off-road environments such as forested areas are generally considered more challenging due to being less structured and predictable. Not many datasets have been developed and published which focuses on off-road scenes, and those that do exist are often the work of smaller research teams. The Freiburg forest dataset was developed by the Autonomous Intelligent Systems group at the University of Freiburg.

The Freiburg forest dataset was gathered on three different days in forested areas, to include enough variability in terms of lightning conditions. 366 images are provided with pixel-level annotations of six different classes: trail, grass, vegetation, obstacle, sky and void. The images are smaller than those in the Cityscapes dataset, and typically have a resolution of around 880x480 pixels. Example data from the dataset is shown in figure 5.3. A dataset split of 230 training images and 136 testing images is provided. A key feature of this dataset is that it provides multiple different data modalities. Apart from RGB and depth data, multispectral cameras have been used to capture near-infrared (NIR) data. The NIR data provides detailed description of the presence of vegetation, and thus contributes to capture a richer representation of the environment. NIR data is not used in this thesis.

The images were captured using a stereo camera system, and disparity maps were generated using the SGM algorithm. However, due to problematic factors such as motion blur and rectification artifacts, the resulting disparity images came out noisy. Instead of using these images, Valada et al. adopted an approach from Liu et al. [46], where they use a deep convolutional neural network to estimate depth from a single image. They used a network pre-trained on the Make3D dataset [47], and found that the predicted depth maps gave better results than the noisy disparity maps.

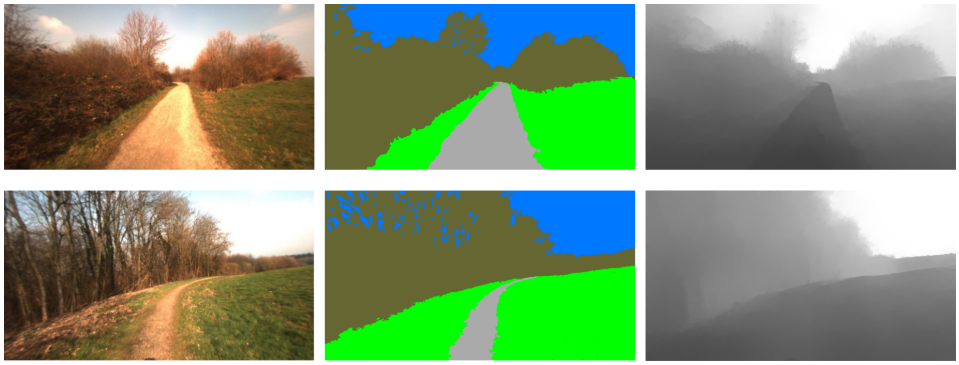


Figure 5.3: Example data from the Freiburg forest dataset [36]. The first column is the input image and the second column is the corresponding colorized ground-truth image. The third column shows the depth image, where dark pixels are close, while bright pixels are farther away.

5.1.3 Dataset comparison

In terms of annotations, the Cityscapes dataset is much more complex than the Freiburg forest dataset. The dataset contains 30 different classes, which appear in different forms. Spatially large classes such as road, building, vegetation and sky make up a great part of the annotations. Such classes often appear in the form of large homogeneous regions, as seen in figure 5.1. Many other classes are spatially much smaller, such as traffic light, pole and person. These classes often appear as separate objects in different parts of the image. The Freiburg forest dataset is characterized by a smaller number of large homogeneous regions. There is only one class which stands out, the object class, which appears relatively infrequently. Based on these differences, we expect the Cityscapes dataset to be a much more challenging dataset, and performance may be poorer compared to the Freiburg forest dataset. The Cityscapes dataset is however much larger in size, and training the network on more data may still lead to good performance.

The depth data in the two datasets is generated in different ways, and consequently the characteristics of resulting depth images differ. While analyses of depth data generation is beyond the scope of this project, some important differences should be pointed out. In boundary regions where there are depth discontinuities, the raw Cityscapes disparity maps are often inaccurate. The inaccuracies imply small regions of increased depth, which does not match the actual depth. For scenes with such challenging regions, the disparity maps consist of quite a few pixels which qualify as noise. The depth images in the Freiburg forest dataset does not contain any such regions, as depth is predicted for every pixel. Overall these depth maps are much smoother than the Cityscapes disparity maps. They do not suffer from noise in boundary regions. Such factors may have a significant impact on the features that the network learns from the data.

5.2 Experimental design

For both datasets, we train and test the baseline ENet model on RGB data, and the two variants of the proposed fusion architecture on RGB and depth data. From here on we will refer to the baseline ENet model as ENet-RGB. We refer to the fusion models as ENet-Single-Fusion and ENet-Multi-Fusion, and use the abbreviations ENet-SF and ENet-MF. The descriptions are summarized in table 5.1. By comparing the results of the different models, the goal is to investigate the effect of introducing depth. Comparing both of the fusion variants should give insight into which approach is best suited for practical applications. Finally, the results on the two datasets are compared to contrast performance in different scenarios. The models are evaluated and compared both in terms of segmentation performance and efficiency, and evaluation metrics are summarized in section 5.3.

Network model	Abbreviation	Description
ENet	ENet-RGB	The baseline ENet model which operates on RGB data
ENet-Single-Fusion	ENet-SF	The ENet fusion model which fuses RGB and depth data at a single location
ENet-Multi-Fusion	ENet-MF	The ENet fusion model which fuses RGB and depth data at multiple locations

Table 5.1: Description of the network models

5.2.1 Training configuration

Each dataset is split into training and testing partitions. For the Freiburg forest dataset, we use the provided train-test-split. For the Cityscapes dataset, the ground-truth labels of the test set are not publicly available, and the only way to evaluate a model on this data is to submit it to an online evaluation server. For this reason, we instead test our system on the provided validation set, to evaluate the performance. Furthermore, we downsample all images to reasonable size, to reduce the training time.

Dataset	Training	Validation	Testing	Image resolution
Freiburg forest	207	23	136	440x240
Cityscapes	2677	298	500	512x256

Table 5.2: Number of training, validation and testing images, as well as image resolution for the two datasets.

The training partitions for each dataset is further split into a new training set and a validation set, where 10% of the data is designated as validation data. Table 5.2 shows the partitioning of datasets, and the image resolution used for training and testing. The validation data was not available for the network to train upon. Instead it was used to evaluate the

performance at periodic intervals. The error on the validation set is used as an indication for the generalization error, and for determining when the network has started overfitting.

All network configurations are trained using the same hyperparameters, summarized in table 5.3. Tuning hyperparameters is a tedious task, as the training process is very time-consuming. For this reason the hyperparameters were chosen based on the values given in the official ENet paper, with some minor modifications. We use stochastic gradient descent (SGD) to minimize the cross-entropy loss function. We use a learning rate of 5×10^{-4} and momentum 0.9. We train for around 100 epochs, or until the training loss converges. We select the model iteration which has the highest accuracy among the evaluations on the validation set.

Parameter	**Description**	**Value**
$\mathcal{L}$	Loss function	Cross-entropy
Solver	Solver	SGD
a	Learning rate	5×10^{-4}
m	Momentum	0.9
Epochs	The number of epochs to train for	100
b	Batch size	4
γ	L2 weight decay factor	2×10^{-4}
$p^{(l)}$	Dropout rate for network stage l	0.01, 0.1, 0.1, 0.1, 0.1

Table 5.3: Hyperparameters for training

The batch sizes were chosen to be as big as possible, without exceeding the GPU memory. The different architectures require varying amount of memory. The ENet-RGB architecture requires the smallest amount of memory, as it only has one encoder branch, and each data input only consists of RGB images. The ENet-MF architecture with two full encoders and both RGB and depth inputs requires the most. Thus, this architecture determines the maximum batch size. While training possibly could be optimized by using a bigger batch size when training the smaller architectures, we avoid this. Using the same batch size may make it easier to compare and contrast the training process of the different architectures.

Several regularization techniques were used to prevent overfitting. This is especially important for the Freiburg forest dataset, which is relatively small. The network may quickly begin to overfit the data. L2 regularization was used in the form of weight decay. Spatial dropout is used in all bottleneck modules, with probability $p = 0.01$ before bottleneck2.0 and $p = 0.1$ afterwards.

Early stopping is another regularization technique used to avoid overfitting. It involves interrupting the training procedure when the network starts to overfit. If the training loss gets smaller with more iterations, but the validation loss plateaus or increases, it is a sign that the network is memorizing patterns in the samples instead of learning representations. Another indication of overfitting is the gap between training and validation accuracy. If this gap is large, or increasing from a point outwards, this indicates overfitting.

When training the network, we adopt the technique proposed in the original ENet paper. This involves a two-stage training scheme were the encoders are pre-trained before training the entire network. When pre-training, we use the same cross-entropy loss function, however the ground-truth images need to be modified. For the network configuration presented in table 4.1, we see that an input image is downsampled by a factor of eight, from 512x512 to 64x64 for the example resolution. In order to compute the loss when pre-training the encoders, the corresponding ground truth image is also downsampled by the same factor. After the initial training stage, we append the decoder, and train the network to perform upsampling and finally pixel-wise segmentation. This technique lets the network learn to extract and fuse features before learning to upsample them. For this reason, the two-stage training scheme appears promising for the multimodal extension.

Because we use batch normalization (BN) layers in our model, we need to adjust the weights before testing. As training batches propagate through the network, the BN layers normalizes the feature maps so that they have zero-mean and unit-variance. The feature maps are shifted according to the mean and variance statistics for each batch during training. After the last batch is processed during training, the statistics computed for that batch are still in use. However, these statistics may not be valid for the rest of the dataset, especially not for small batch sizes. For that reason it is necessary to use the statistics for the entire dataset before testing the network. We run batch normalization over the entire training dataset, to update the network weights before test time.

5.3 Evaluation metrics

In order to evaluate the segmentation performance of the different models, we use both quantitative and qualitative metrics. The quantitative segmentation metrics evaluate the pixel-level accuracy using different measures. The qualitative evaluation involves visual examination of the result, to uncover results which the quantitative metrics may not capture. Finally, the models are evaluated in terms of relevant performance measures.

5.3.1 Quantitative segmentation metrics

The most common evaluation metrics for semantic segmentation measures the pixel-level classification accuracy of the segmentation. All pixels in the test dataset are assigned a class label, and are compared with the ground truth labels. The main tool used for this comparison, is a pixel-level confusion matrix $\mathbf{C}$. This matrix aggregates the pixel predictions for the whole testing dataset. The element C_{ij} is the number of pixels with ground-truth label i, which were predicted as being of class j. This matrix allows us to compute relevant metrics, using the number of true positives (TP), false positives (FP), and false negatives (FN) for each class.

Accuracy:
The accuracy is the simplest and most intuitive metric, for evaluating the performance. For

each class i, the accuracy is the ratio between the number or correctly classified pixels to the total number of pixels for the class:

$$\text{Accuracy of class } i = \frac{\text{TP}_i}{\text{TP}_i + \text{FN}_i} \tag{5.1}$$

The global accuracy is the ratio of correctly classified pixels for all the K classes, to the total amount of pixels (N):

$$\text{Global accuracy} = G = \frac{1}{N}\sum_{i=1}^{K} \text{TP}_i \tag{5.2}$$

While the accuracy measurements are intuitive and may provide decent estimates of the performance, they suffer due to their simplicity. The metrics do not punish false positives, and a high class accuracy may be therefore be misleading. In the extreme case that an entire image is predicted as class j, this would give a perfect accuracy score for that class, while zero for the other classes. For datasets where the classes are relatively imbalanced, the global accuracy suffers, as it does not capture this element. Classes which appear frequently tend to dominate less frequently appearing classes. For this reason we also measure the class average accuracy to provide a better evaluation of the performance:

$$\text{Class average accuracy} = C = \frac{1}{K}\sum_{i=1}^{K} \frac{\text{TP}_i}{\text{TP}_i + \text{FN}_i} \tag{5.3}$$

Intersection over union:
The intersection over union (IoU), also known as the Jaccard Index, is the most commonly used evaluation metric. It is a more strict than all the accuracy metrics, as it penalizes false positives. For each class i, the IoU is computed by taking the intersection of the predicted pixels and the ground truth labels, and dividing by the union. This can be expressed as:

$$\text{IoU}_i = \frac{\text{TP}_i}{\text{TP}_i + \text{FP}_i + \text{FN}_i} \tag{5.4}$$

The mean intersection over union (MIoU), is found by averaging over all the K classes:

$$\text{MIoU} = \frac{1}{K}\sum_{i=1}^{K} \frac{\text{TP}_i}{\text{TP}_i + \text{FP}_i + \text{FN}_i} \tag{5.5}$$

The MIoU is the standard metric for segmentation performances, and most results are reported in terms of this metric. In our analyses we will use both the MIoU and class-specific IoU together with global and class average accuracy to report the results.

5.3.2 Qualitative evaluation

The quantitative segmentation metrics are very useful as they capture objective statistics concerning the segmentation. This allows for comparisons with other approaches which use the same datasets. However, the quantitative measures do not necessarily capture the

whole picture. The performance of the segmentation should also be evaluated through visual inspection, which may provide additional insight regarding the results.

There are certain factors which are best evaluated through visual inspection. For typical road scenes most of the pixels belong to spatially large classes such as roads and buildings, so the network should produce smooth segmentations. It should also produce clear boundaries between regions to distinguish objects, despite potentially small sizes. Visual inspection may also provide insight into how the network deals with challenging environments. The effect of different light settings and natural factors such as sun glare may be significant.

In addition to inspecting the segmentation results, it is also of interest to inspect intermediate feature maps as they pass through the network. This can be helpful for illuminating the effect of introducing depth information. By extracting feature maps at strategic points, such as before and after points of fusion, we can see both what features are detected, and how they complement each other.

5.3.3 Performance metrics

Performance metrics are useful for evaluating how applicable a system is for real-world deployment. Accurate segmentation which closely matches the ground truth is desirable, but it often comes at the cost of computational cost and memory requirements. For applications with strict real-time requirements, some accuracy in segmentation may be expendable up to a certain point, if it contributes to better performance. Depending on the context of the system, these metrics might even be of more importance than those relating to the segmentation.

While certainly useful, these metrics are more challenging to compare with different approaches. The performance of the system is highly dependent on the hardware it runs on. An approach which performs well on one system during offline testing, may be constrained by different factors in real-world applications. However, they still provide insight into how good performance is achievable. As the work in this thesis extends the ENet model, it is of special interest to investigate how the performance is affected by the enlarged architecture. Another factor which may affect the performance is the choice of deep learning framework. The various frameworks all have their pros and cons, and implementations in different frameworks may lead to different performance results. The main metrics which are considered in this thesis are inference speed and memory requirements.

Inference speed:
The inference time is the most important metric regarding execution time. This measures how fast segmentations can be produced, and consequently how many frames per seconds the system can produce. The inference speed that the system achieves depends both on the hardware of the system as well as the image resolution it operates on. This means that the results may be difficult to compare with other approaches, however the may still provide valuable insight.

Memory requirements:
Memory is often a considerable bottleneck when constructing CNN architectures. The main challenges typically arise in the offline-phase during training, which means that this is a less important factor than inference speed. However, a certain amount of GPU memory is required for the system to run on a real-world application. It is therefore of interest to investigate how much GPU memory is needed during the inference pass. The model size is also of interest, when disk space is limited. The extended architectures lead to larger models, which takes up more disk size due to having more parameters.

Chapter 6

Results and discussion

This chapter presents the results and discussion for the experiments described in chapter 5. We present the results for each dataset individually, comparing the performance of the different networks. As described in chapter 5, we refer to these as ENet-RGB, ENet-SF and ENet-MF. After this, we present an analysis of the inference process, evaluating the performance of the fusion. Next, we present the performance results, before a summarizing discussion is given.

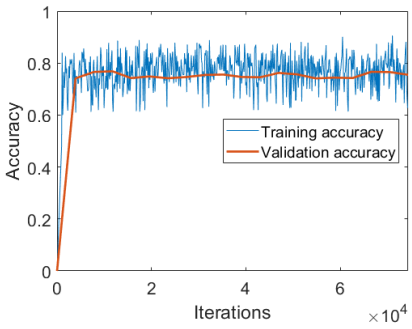
6.1 Cityscapes dataset

In this section we present the results for each network trained on the Cityscapes dataset. We first present the relevant training curves. Following this, we present the quantitative results, using the evaluation metrics described in chapter 5. The aggregated results for the entire testset are presented first, before we move on to the class-specific results. Next we present the qualitative results which are based on visual inspection of the segmentation results.

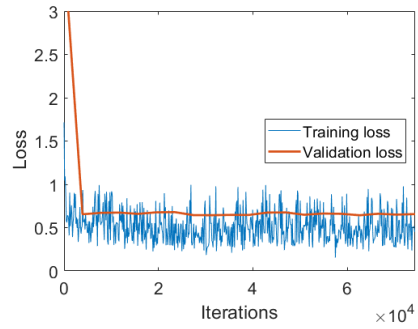
6.1.1 Training results

The training curves from training on the Cityscapes dataset are shown in figure 6.1. Each row show the plots for a single variant of the architecture. The columns compare the accuracy and loss respectively between the training and validation sets. These are used to verify that the networks has produced a valid solution which can be used for testing, and that the network did not overfit.

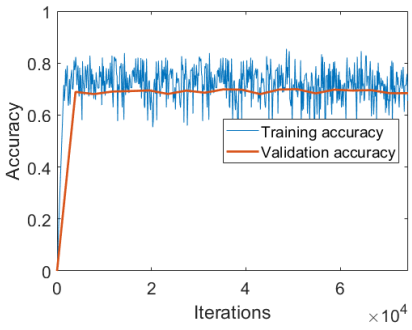
The expected form of the training accuracy curve is an initial phase where accuracy improves quickly, followed by a small but steady increase, before it eventually plateaus. In our training, the initial phase is barely visible, and an overall high accuracy is achieved after relatively few iterations for all variants. There is no distinct rise in accuracy after this, and all networks appear to converge quickly. The validation accuracy closely follows



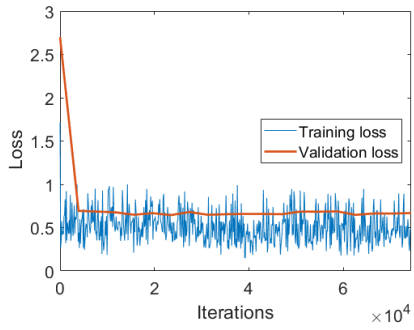
(a) ENet-RGB Training vs. validation accuracy



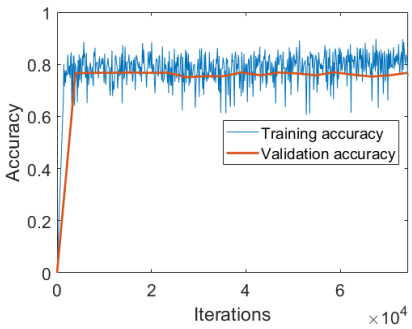
(b) ENet-RGB Training vs. validation loss



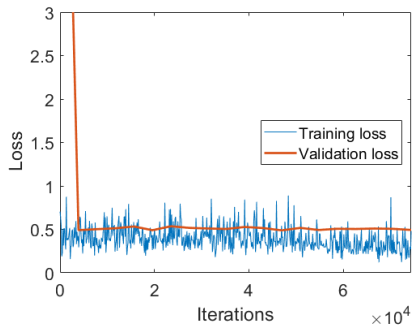
(c) ENet-SF Training vs. validation accuracy



(d) ENet-SF Training vs. validation loss



(e) ENet-MF Training vs. validation accuracy



(f) ENet-MF Training vs. validation loss

Figure 6.1: Training curves for the networks trained on the Cityscapes dataset

the training accuracy, overall being a bit lower. The small gap indicates that the network does not overfit to the data. The loss plots show a similar trend. The initial phase is barely visible, and while the initial validation loss is high, it quickly decreases. The validation loss remains overall higher than the training loss, while being relatively constant. As there is no clear divergence in the training and test loss, the network does not seem to overfit.

Overall the trend is quite similar for all architectures. The accuracy for the ENet-SF model is slightly lower than the ENet-RGB and ENet-MF models. The networks all appear quite easy to train, which may be due to the symmetric encoder-branches. The training curves appear quite noisy, both in the loss and accuracy plots. This is presumably due to the batch size being relatively small. From these plots, we can confirm that the network did not overfit, and that it has converged successfully.

6.1.2 Quantitative results

The results computed for the entire testset are presented in table 6.1. This shows that the baseline ENet-RGB model has the highest global accuracy, however it only slightly outperforms the fusion models. By comparing the global accuracies with the training accuracies in figure 6.1, we see that the test accuracies are significantly higher. This may be due to how differences in how Caffe computes accuracy during training, and the divergence was not investigated further. The ENet-MF model performs best in the more important class average accuracy and MIoU metrics. Overall, the ENet-SF model performs poorest. While the aggregated results provide some insight, they are best-considered in conjunction with the class-specific results, which will be considered next.

Model	Global accuracy	Class average accuracy	MIoU
ENet-RGB	**89.73**	50.59	44.44
ENet-SF	88.11	49.33	37.77
ENet-MF	88.62	**57.41**	**46.91**

Table 6.1: Aggregated test results for the Cityscapes dataset. For each evaluation metric, the model with the best result is reported in bold.

The class-wise IoU results are presented in table 6.2. The general trend is similar to the aggregated results, where the ENet-RGB and ENet-MF models outperform the ENet-SF model. While the ENet-MF model had the best class average accuracy and MIoU, it does not perform best on all the classes. The ENet-RGB model has the best results for five of the classes. The results also show that the ENet-SF model actually outperforms the other models in one of the classes, despite having the poorest aggregated results.

While the differences in global accuracy are too small to draw any definite conclusions from, the class average accuracy and MIoU provide better insight. These metrics show the same relative performance ordering between the models, with ENet-MF being the best,

Model	Road	Sidewalk	Building	Wall	Fence	Pole	Bus	Car	Vegetation
ENet-RGB	**92.48**	59.32	78.45	25.86	25.11	28.75	3.10	**79.32**	**82.57**
ENet-SF	88.56	48.58	70.52	17.74	15.60	23.75	5.24	73.42	75.28
ENet-MF	91.82	**62.26**	**80.06**	**28.57**	**29.48**	**35.39**	**7.98**	79.18	81.10

Terrain	Sky	Person	Rider	Traffic sign	Truck	Traffic light	Train	Motorcycle	Bicycle
47.88	**84.72**	53.31	34.70	37.61	24.54	**25.06**	4.13	15.67	41.81
41.05	79.14	37.85	24.39	25.37	19.67	19.36	**4.65**	14.65	32.86
52.49	81.48	**54.48**	**37.70**	**42.20**	**31.11**	24.79	2.49	**24.30**	**44.41**

Table 6.2: Class-wise IoU results for the Cityscapes dataset

followed by ENet-RGB and ENet-SF. The results does not undoubtedly indicate a performance boost by incorporating depth information. On the contrary, the poorer results of the ENet-SF network shows that fusing modalities may also negatively affect performance. The differences in performance for the two fusion strategies is especially interesting. The results indicate that incorporating depth information may indeed be beneficial, but the level of fusion is important.

The class-specific results gives a greater insight into the effect of depth incorporation. An important factor to consider when evaluating these results is the fact that some classes occur more frequently in the test set than others. Spatially large classes such as "road", "car", "building" and "sky" make up a large share of the pixels, while classes such as "person", "fence" and "traffic light" are generally smaller. The general trend for all networks is a high score for the large classes, and lower scores for the smaller classes.

The first point to touch upon is the spatially large classes, for which all models achieve good results. For most of these classes the difference between the best and the poorest result is quite small. The baseline ENet-RGB model has the best results on the road, car, vegetation and sky classes, while ENet-MF performs best on the building class. Such classes are characterized by large homogeneous regions, with relatively low variance in both modalities, which may be easier to learn. The road is always in the bottom half of the image, contains little texture variations and the corresponding depth region is always a smooth transition from near to far. Similarly, the sky is always a homogeneous region in the upper part, where the depth is maximum compared to other regions. The baseline model achieves good performance on these regions, and depth information may be redundant. Based on these large classes, it is not possible to judge which model is the best.

The results on the smaller classes are more interesting. Overall, the scores for many of these classes are considerably lower than for the large classes, and there is a greater amount of variation in the results. For two of these classes, "bus" and "train", all models obtain very poor scores. This is as expected, as those classes appear very infrequently in the testset. For many of the other classes, the ENet-MF model outperforms the ENet-RGB

model by a significant amount. The ENet-SF model has the poorest results overall, which is consistent with the earlier results. These results seem to indicate that segmentation of spatially small classes is where incorporation of depth information has the largest impact. While the overall poor ENet-SF results suggest that this variant of the architecture is not very useful, the ENet-MF model looks promising.

6.1.3 Qualitative results

This section presents qualitative results from the testset. Results from the networks are inspected visually and compared to the ground truth images. For illustrative purposes each class has been assigned with a color in the ground truth and predicted images. We present images which illustrate trends that are consistent across the entire testset.

The quantitative results suggests that all models may perform well for scenes dominated by spatially large classes. Some of these classes are road, sidewalk, building, vegetation, car and sky. This is confirmed by visually inspecting the segmentation results. The trend for the large classes is especially noticeable in images where there are few objects from the small classes. This is illustrated in figure 6.2, where the overall segmentation is smooth, and closely matches the ground-truth for all the models. The baseline model fails to recognize some of the smaller objects, which both fusion models detect. The ENet-MF model mislabels part of the vegetation area in the middle part of the image, which the other models more correctly label.

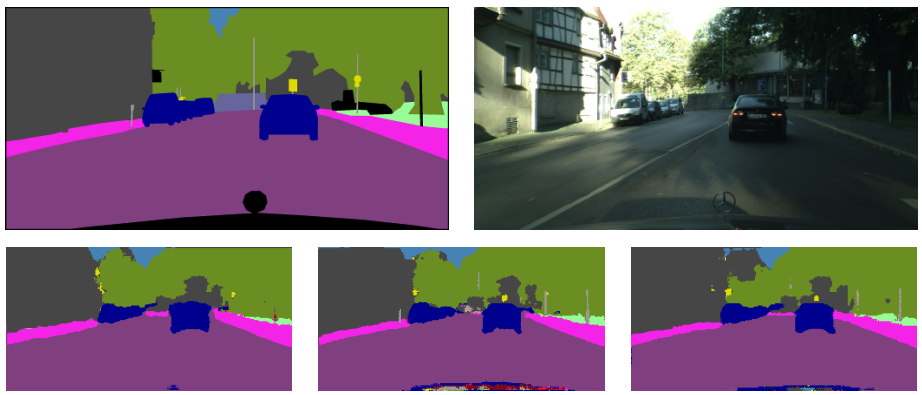


Figure 6.2: Result from the Cityscapes dataset. The upper row shows the ground truth and the input image. The bottom row shows predictions from ENet-RGB, ENet-SF and ENet-MF from left to right.

The effect of incorporating depth information becomes more apparent when inspecting more difficult and cluttered scenes. Such scenes which contains many smaller objects are illustrated in figure 6.3 and 6.4. These results confirm what the quantitative results suggested, and shows that the baseline model struggles with small classes such as poles and

traffic signs. The ENet-RGB model in many cases fails to recognize such object, and labels it similar to the background. Both fusion models show better performance, and detects most of these objects. Perhaps most impressive is the performance for the thin objects such as poles. While the quantitative results suggested that the ENet-SF model may perform poorer than the baseline model for such small classes, it actually detects them consistently across the testset. However, the overall quality of the segmentation, in terms of smoothness and clear boundaries is worse than both the ENet-RGB and ENet-MF models.

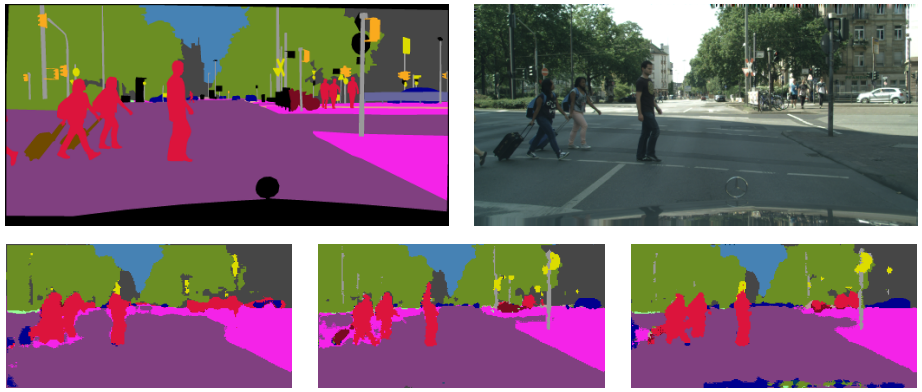


Figure 6.3: Result from the Cityscapes dataset. The upper row shows the ground truth and the input image. Bottom row shows predictions from ENet-RGB, ENet-SF and ENet-MF from left to right.

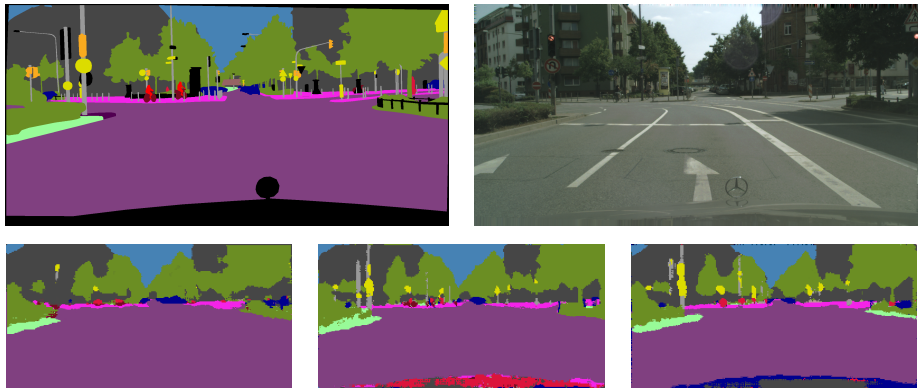


Figure 6.4: Result from the Cityscapes dataset. The upper row shows the ground truth and the input image. Bottom row shows predictions from ENet-RGB, ENet-SF and ENet-MF from left to right.

The ENet-MF model shows significantly better performance than ENet-RGB at separating small objects which appear close, such as two persons walking next to each other. While the ENet-RGB model tends to group such regions together as large blobs, the ENet-MF model does a better job at producing separate regions. ENet-MF also produces object borders which closely corresponds to the ground truth. The segmented regions of persons are

generally recognizable from their form. The same goes for classes such as cars, which are easily recognized.

Visual inspection also reveal some of the shortcomings of all models. The pixels near the bottom of each label image are labelled as void, which complicates learning these regions. This is seen in several of the images above, where these regions are consistently mislabelled. Furthermore, there are several classes which the networks struggle to distinguish between. Some of the most prevalent are cars and trucks which are obviously similar in appearance, as well as traffic lights and traffic signs. This is confirmed by inspecting the confusion matrix, which shows that a significant part of these pixels are classified incorrectly.

6.2 Freiburg forest dataset

In this section we present the results for the Freiburg forest dataset. The results will be presented in the same order as for the Cityscapes dataset. We first present the relevant training curves for each model. Following this, we present the quantitative and the qualitative results.

6.2.1 Training results

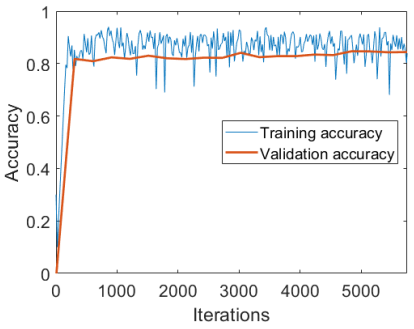
The training curves from training on the Freiburg forest dataset are shown in figure 6.5. Due to a smaller dataset, fewer iterations of training were performed to reach 100 epochs. As this dataset is considerably smaller than the Cityscapes dataset, it is more susceptible to overfitting. This makes it especially important to inspect the training curves. The plots show the same trends as for the Cityscapes dataset. The model reaches a high accuracy quickly, and converges fast. Considering the results for the validation loss and accuracy, the network does not seem to overfit the data.

6.2.2 Quantitative results

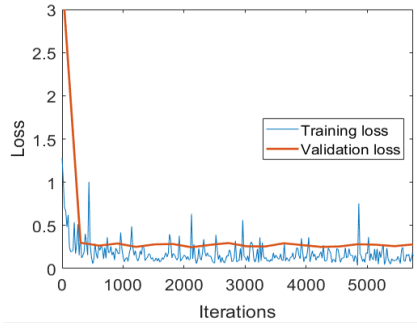
The aggregated results for the Freiburg forest testset are shown in table 6.3. The ENet-RGB model slightly outperforms both the fusion models in both global accuracy and class average accuracy, while ENet-MF has the best MIoU score. ENet-SF has the poorest results in terms of all metrics.

Model	Global accuracy	Class average accuracy	MIoU
ENet-RGB	**96.57**	**86.12**	71.98
ENet-SF	95.94	84.14	70.42
Enet-MF	96.42	84.30	**72.45**

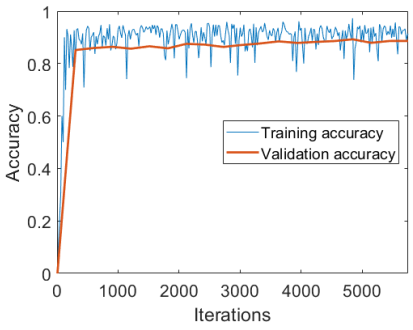
Table 6.3: Aggregated test results for the Freiburg forest dataset



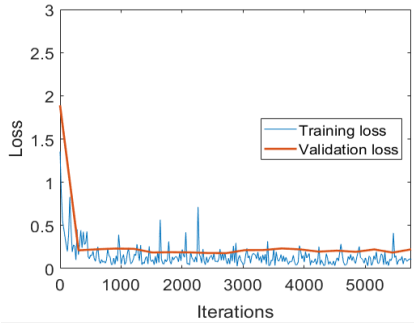
(a) ENet-RGB Training vs. validation accuracy



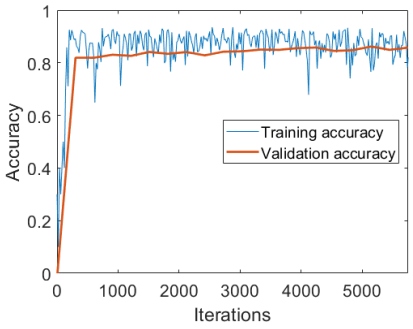
(b) ENet-RGB Training vs. validation loss



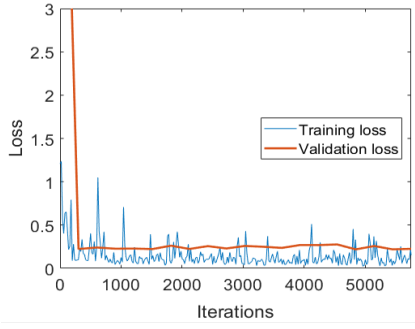
(c) ENet-SF Training vs. validation accuracy



(d) ENet-SF Training vs. validation loss



(e) ENet-MF Training vs. validation accuracy



(f) ENet-MF Training vs. validation loss

Figure 6.5: Training curves for the networks trained on the Freiburg forest dataset

The class-specific IoU results are presented in table 6.4. The ENet-RGB model has the best results for the "grass", "vegetation" and "sky" class, while ENet-MF performs best for "road" and "obstacle". ENet-SF does not have the best results for any class, however it is not the poorest in all the classes.

Model	Road	Grass	Vegetation	Sky	Obstacle
ENet-RGB	81.08	**84.14**	**85.14**	**89.05**	20.46
ENet-SF	74.41	83.79	84.69	88.57	20.66
Enet-MF	**82.08**	83.42	85.04	88.35	**23.36**

Table 6.4: Class-wise IoU results on the Freiburg forest dataset

The aggregated scores for the Freiburg forest testset are all significantly higher than for the Cityscapes dataset. The classwise IoU results are also all high, except for the obstacle class. There is little variation between the three models in all results, which is a clear difference from the Cityscapes dataset. All networks perform well, and the only result that stands out is the "obstacle" class.

Compared to the Cityscapes dataset, the Freiburg forest dataset has much fewer classes, and is characterized by large homogeneous regions. The Cityscapes results indicated for such regions even the baseline model performs well, and incorporation of depth information may be excessive. This insight helps explain the overall better results, and why no significant improvement is achieved from incorporating depth information. The obstacle class stands out from the other classes in the dataset. It appears relatively infrequently, and when it appears, it is spatially small. The results for the different models for this class are much lower than for the other classes. The ENet-MF model has the best results on this class, which is consistent with the results from the Cityscapes dataset. It appears that this model is better suited to deal with the small and more challenging classes.

Another important difference between the Freiburg forest and the Cityscapes dataset, is that Freiburg forest has much less variation. While the Cityscapes data was captured across different cities and times, the Freiburg data is relatively uniform. Comparing the training data with the test data confirms this. Having very similar data in both splits yields good results, but the model may not generalize well to new data. The small dataset size is also a limiting factor for drawing any definite conclusions from the dataset.

6.2.3 Qualitative results

Similar to the section for the Cityscapes dataset, we present a qualitative evaluation based on visual inspection of the segmentation results. It should be noted that the colors used in these images are not related to the colors for the Cityscapes results.

An example of the results from all three models is shown in figure 6.6. As the quantitative results suggested, the results for all models are overall very good. The networks produce smooth regions with clear boundaries that closely matches the ground truth images. The

most challenging regions are the boundaries between different classes such as "vegetation" and "sky", however all models produce good boundaries. There appear to be little difference in segmentation performance between the baseline model and the fusion models.

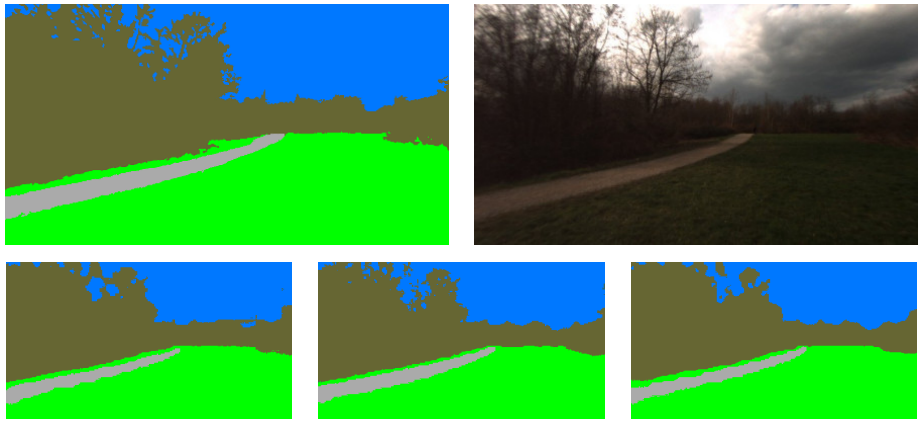


Figure 6.6: Result from the Freiburg forest dataset. The upper row shows the ground truth and the input image. Bottom row shows predictions from ENet-RGB, ENet-SF and ENet-MF from left to right.

While there is little variation in scenery across the dataset, there are certain images which stand out as more challenging than others. These are images from scenes with poor illumination or where sun glare is very prominent. Such situations are where the RGB data suffers, while the depth data should be invariant. Here we may expect the fusion models to outperform the baseline model. An example of this scenario is illustrated in figure 6.7 together with the results from the networks.

While all three models struggle with the glare, an interesting result is that the ENet-RGB model appears to be less affected by it than the fusion models. Inspecting the depth images for such images, reveal part of the reason why the fusion models do not perform as well as expected. As noted in chapter 5, the depth images are predicted from the RGB images using a separate CNN. An unfortunate consequence of this is that some of the problems of the RGB images are also present in the depth images. In particular it is interesting to note that the sun glare results in a significant depth discontinuity in the depth image, illustrated in figure 6.8. The depth modality which should be invariant to factors such as poor illumination and sun glare, is not very efficient in this dataset.

While the results are good and promising, the limited size of the dataset does not justify any definite conclusions. From these experiments it is hard to tell if depth information has any significant impact. Furthermore, the similarities between the training and test set makes it hard to tell how well the model can generalize to new data.

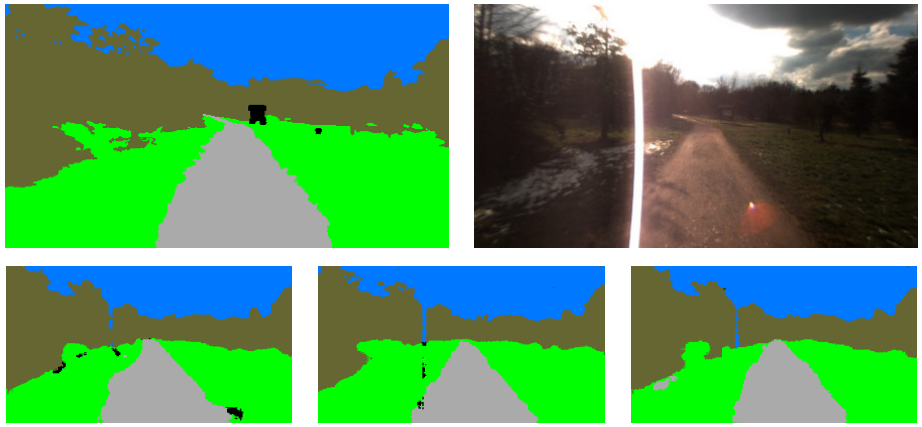


Figure 6.7: Result from the Freiburg forest dataset. The upper row shows the ground truth and the input image. Bottom row shows predictions from ENet-RGB, ENet-SF and ENet-MF from left to right.

Figure 6.8: The depth image for the example in figure 6.7, where sun glare is present.

6.3 Feature fusion

The results presented up until now have been concerned with the effect depth incorporation has on the final network outputs, i.e. the segmentation images. While analyzing the end results is useful for evaluating performance, it is also beneficial to analyze certain intermediate stages of the segmentation process. In particular we perform an in-depth analysis of a single forward pass for a relatively challenging test-sample, shown in figure 6.9. Analyzing intermediate feature maps at points of fusion may provide insight into what features the network learns from the different modalities. Such an analysis can contribute to insight regarding how the models may be improved. As the Cityscapes dataset proved to be the most useful dataset, we focus on that dataset in this section. We dedicate the majority of this section to an analysis of the best-performing fusion model ENet-MF, before comparing it with the ENet-RGB model.

Figure 6.9: Input image for the feature fusion analysis.

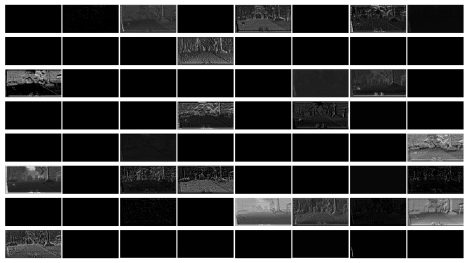
Figure 6.10 shows extracted feature maps at the first fusion point for the ENet-MF model. Feature maps are extracted from both branches, as well as after the fusion. At each of these locations the network produces 64 feature maps, each with a resolution of 32x64 pixels. Attempting to interpret the feature maps at this early stage is both impractical due to the small resolution and serves little purpose. However, comparing the fused feature maps with those produced by each branch provides some insight into how the modalities complement each other.

The RGB and depth branches learn separate features, and each branch may have a different focus for a given stage. The fused activation map shows that the network successfully combines features from both branches to produce stronger activations. While stronger activations does not necessarily lead to better accuracy, it is useful to also preserve activations at different locations. At an early stage, the network typically learns low-level features such as edges and lines. These features should be learned in both branches, and they usually complement each other. A boundary region in the RGB image between two objects, manifests itself as a depth discontinuity in the depth image. Combining the information from the two branches with the proposed fusion strategy, captures more information.

An interesting observation is that there is a considerable amount of dead filters in both branches. These are visualized as being all black, and have more or less zero activation. This can be a sign of high learning rates, but it may also just mean that relevant features are not present in this specific image. As these dead filters for the most part appear at different locations in the two branches, the effect is smaller in the fused feature map. This illustrates a surprising, but very welcome effect of this fusion strategy. The alternative fusion strategy, channelwise concatenation, would have preserved these feature maps which don't hold any information, thus wasting memory.

Figure 6.11 shows the final outputs of the decoder for ENet-RGB model, while figure 6.12 shows the outputs for ENet-MF model. Each of the 19 feature maps correspond to the probability that the network infers for the 19 classes. These results helps explain the

(a) RGB activations



(b) Depth activations

(c) Fused activations

Figure 6.10: Activations at the first fusion point. Strong activations are visualized as bright areas in the feature maps, while dark areas indicates weak activations.

different results for the two models. For large classes, such as the "road" and "building" classes, both models are able to reason well, and predict accurate regions. For the smaller and more challenging classes, the predictions from the ENet-RGB model produce weaker activations in relevant regions. This can be seen in the image for the pole class, where the network produces weak activations all over the image. The ENet-MF model on the other hand produces stronger activations only for the relevant pole regions. The ENet-MF model is also able to predict the smaller regions such as the traffic lights and traffic signs. The ENet-RGB model produces weaker activations in these regions, which explains why it performs poorer in terms of such classes.

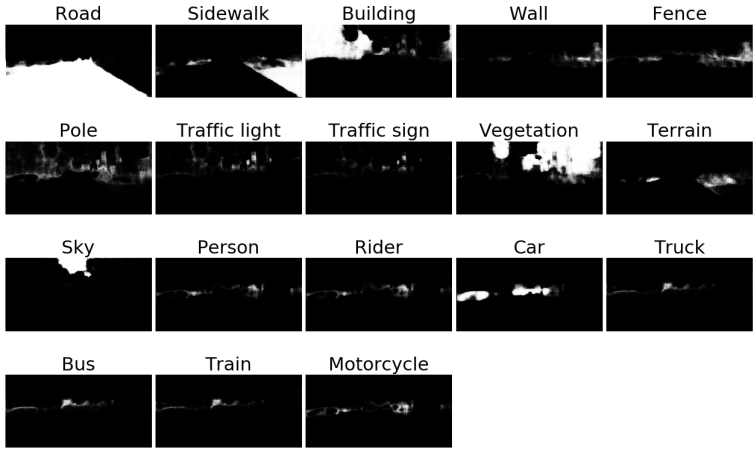


Figure 6.11: ENet-RGB decoder output.

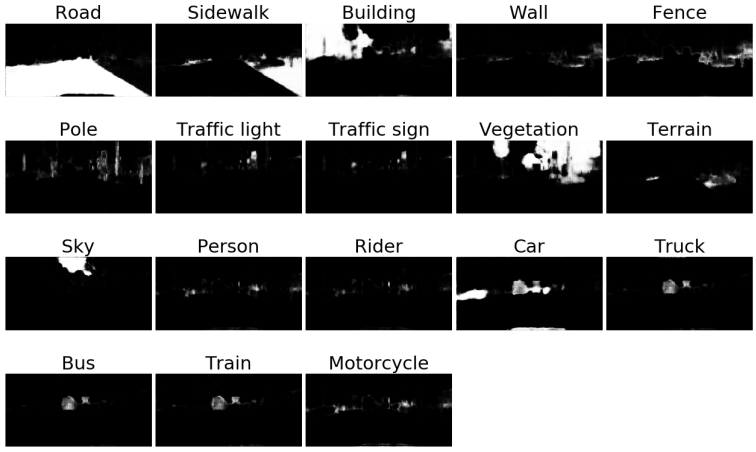


Figure 6.12: ENet-MF decoder output.

6.4 Performance results

The performance results for the three architectures are shown in table 6.5. Images from the Cityscapes dataset were used, with the same resolution as during training, 512x256 pixels. To determine the forward pass, the average time over 50 iterations with mini batch size 1 was computed, using the Caffe time command. To computer memory consumption, we used the nvidia-smi unix command, where inference memory was found using batch size 1. For the model size, we used the disk size.

Model	Forward pass (ms)	Parameters (#)	GPU inference memory (MB)	Model size (MB)
ENet-RGB	30.42	367471	728	1.5
ENet-SF	76.84	391168	953	1.6
ENet-MF	111.69	710144	1253	2.9

Table 6.5: Performance results for the networks

The relative ordering of the three networks is as expected. ENet-RGB model is the simplest model and has the fewest parameters. It has the fastest inference speed, uses least GPU memory and takes up the least amount of space on disk. ENet-SF is slightly more complex, but the increase in parameters and model size is very small. The increase in inference speed and GPU memory is more significant. The most complex model, ENet-MF, scores "worst" in all results as expected.

6.5 Discussion

The results of the various experiments all contribute to a final verdict regarding the network models. The experiments on the two datsets provided varying degrees of insight. Most of the useful conclusions can be drawn based on the experiments on the Cityscapes dataset. This is the more complex of the two datasets, which also holds the most useful depth data. The Freiburg forest dataset lacked the complexity needed to draw definite conclusions from the experiments.

The results are overall very consistent. The single-fusion model ENet-SF had the poorest results overall. For the easiest classes, the ENet-RGB and ENet-MF both performed good, while ENet-MF outperforms ENet-RGB for many of the smaller and more challenging classes. This indicates that incorporating depth may both improve and decrease performance, depending on how it is implemented. The best results were achieved when RGB and depth features were fused at multiple locations. This was initially suggested by the quantitative results, and confirmed by visually inspecting the segmentation results. The baseline ENet-RGB model achieved competitive results for several classes, and proved to be a reasonable alternative for situations where depth information is not available. For data characterized by large homogeneous regions, one may argue that the baseline model is the best choice, as depth incorporation has little impact for such data.

The improved performance on the Cityscapes dataset by incorporating depth information is motivating. Disparity images generated using the SGM algorithm are very relevant for practical application where systems are equipped with stereo camera systems. While these images were somewhat noisy, the network still seemed able to learn relevant features from them. Such images holds helpful information which is indeed useful as a complement to RGB data. While the results from the experiments on the Freiburg forest dataset are good, they did not provide much insight into the effect of incorporating depth. Even the baseline model which only operates on RGB images achieved very competitive results, and the ENet-MF fusion model only slightly outperformed it in terms of MIoU. The baseline model even had the highest IoU score for three out five classes. These results can mostly be attributed to the simplicity of the dataset. For this reason, it is difficult to say whether the proposed approach is suitable for the off-road domain. The lack of high quality dataset is a severely limiting factor, and the Freiburg forest dataset proved to be too simple.

The proposed ENet-MF model achieve better results than the baseline model, however all results are somewhat lower than the results reported on the official Cityscapes benchmark. They report an aggregate MIoU score of 58.3, while we achieved 44.44 for the ENet-RGB. The best performing fusion model achieved a MIoU score of 46.91, which is still significantly lower. The official class-specific results are overall also slightly higher than our results. While this may indicate that the training process can be optimized, this was not pursued as training is a time-consuming process. Some of the differences in results may also be attributed to different testing data. The official results are achieved on the official Cityscapes test set, while we use the validation set for testing.

Besides scoring lower than the official ENet Cityscapes scores, the results are significantly lower than the state of the art results which operate only on RGB images. The highest scoring models on the Cityscapes benchmark achieve MIoU scores of over 80, however this often comes at the expense of being slower and larger. This is as expected, and the focus has been on an efficient model rather than a very accurate model. While the fusion-models are more complex, we are still able to achieve a very small architecture. The most successful ENet-MF model is approximately twice as large as the baseline ENet-RGB model, which is still very small compared to other models.

One of the initial assumptions was that incorporating depth should help the system deal with difficult scenarios such as sun glare and poor illumination. Such images are present in the Freiburg forest dataset, but due to the nature of the depth data the results were inconclusive. The experiments did however reveal how predicting depth from a single RGB image is not a good approach when dealing with natural factors such as sun glare.

The most important shortcoming of the ENet-MF model is the significant increase in inference speed. For the hardware used in this thesis, the average forward pass for the ENet-MF model is more than 3 times that of the ENet-RGB. For more powerful hardware, the effect should be smaller. By using alternative deep learning frameworks which can further parallelize the inference process, better performance may also be achieved.

Chapter 7

Conclusion and further work

This chapter concludes the thesis. A brief overview of the work is given in the first section. Based on the results in chapter 6, a conclusion is presented in section 7.2. Finally some suggestions for further work are presented.

7.1 Overview

The goal of this project was to investigate how RGB and depth data could be combined to perform the task of semantic segmentation. The ENet architecture, originally designed for RGB inputs, was extended to incorporate depth information. The proposed architecture has two feature extraction branches, one for each modality. Feature maps from the depth branch are fused into the RGB feature maps by element-wise summation, to produce a combined data representation. The fusion scheme was designed to maintain the efficient properties of the base architecture. Two variants of the proposed architecture was considered, which fuses the depth and RGB features at either one or three locations. Both these networks were compared with the baseline model which operates only on RGB data.

Two datasets were used for the experiments, which were chosen because of their relevance to scene understanding for vehicles. These are the popular Cityscapes dataset which is concerned with urban scenes and the less commonly used Freiburg forest dataset from off-road scenes. The baseline model and the two proposed variants of the multimodal architecture were trained and tested using these datasets. For the Cityscapes dataset we used the validation partition for testing, while we used the pre-defined testing partition for the Freiburg forest dataset. Performance was evaluated using standard evaluation metrics, as well as in terms of execution time and memory footprint. An in-depth analysis of the inference pass was also performed to evaluate the fusion.

7.2 Conclusion

The results of the experiments suggest that incorporating depth information may both improve and worsen performance, depending on the implementation. The best results were achieved by the proposed ENet-MF model, while the proposed ENet-SF model performed worse than both the baseline model and ENet-MF. For spatially large classes which appear frequently in the dataset, all models perform well, and the baseline model even performs best for some classes. The small classes which appear more infrequently is where depth incorporation had the biggest impact. The ENet-MF model which operates on both RGB and depth data was able to reason about challenging small classes which the baseline model not always detected.

Element-wise summation of feature maps proved to be a successful fusion strategy, and the best results were achieved when depth and RGB data was combined at multiple levels throughout the encoder part of the network. This strategy was able to preserve important information from both modalities, and produce useful combined representations of the data. Experiments also revealed that this strategy may help deal with the problem of dead filters. Combining RGB and depth data at one single point gave overall worse results than the baseline model. The experiments were not conclusive as to why this approach performed bad, as the focus rather on investigating the good performance of the more successful approach.

Extending the ENet model to incorporate depth information leads to a significant increase in inference speed. The slower inference speed may be a limiting factor for practical applications. However, the decrease in performance may be less severe for systems equipped with more powerful hardware, in particular a more powerful GPU. The fusion models remained small in terms of parameters, and the GPU memory required during inference is little enough for practical applications equipped with modern GPUs.

7.3 Future work

As the official ENet results on the Cityscapes benchmark are somewhat better than the results achieved in this work, this motivates further experiments. More comparable results may be achieved by testing the proposed models on the official Cityscapes testing set, instead of the validation set. Furthermore, by optimizing the training procedure through fine-tuning hyperparameters and experimenting with other solvers, we may achieve more competitive results. The dead filters which were revealed near the first fusion point motivates training with lower learning rates. Approaches for initializing the weights of different parts of the network could also be investigated, and may lead to improved performance. One approach could be to pre-train the depth encoder branch exclusively on depth data, before training the entire network on RGB and depth data.

This thesis has worked with 2D representations of depth data, such as disparity maps generated from stereo camera systems. While the results indicate that depth information may

indeed improve performance, the use of stereo camera systems to generate depth data is not suitable for all practical applications. For some real-world applications, the use of dedicated depth sensors such as Lidars is more convenient. Such sensors are invariant to appearance variations caused by changes in illumination, weather and seasons, which camera systems struggle with. When depth information is provided by such sensors, the network can work with 3D depth data, for instance point clouds. This data may capture useful information which is not available in the disparity maps. A modified feature extraction branch can learn 3D features, which can then be combined with RGB features.

Further experiments are required to determine how well suited the proposed approach is for the off-road domain. The lack of densely annotated data hinder research into how applicable deep learning techniques are for the domain. Developing high quality datasets which can be used for training and testing is a time-consuming process, and most research has focused on urban scenes. However, if more relevant datasets are developed for the off-road domain, this will facilitate further research. Of special interest are multimodal datasets with data from multiple sensor modalities. One potentially relevant sensor type is multispectral cameras. These cameras can capture near-infrared (NIR) data, which can provide information about the presence of vegetation. This information may be relevant for scene understanding systems for the off-road domain. NIR data is available in the Freiburg forest dataset, however it was not used in this thesis. This data should be correlated with both RGB and depth data, and may be incorporated to further improve performance.

While this thesis has focused on extending the efficient ENet architecture, it is also of interest to investigate extensions of other architectures. The fields of computer vision and deep learning are actively being researched, and important advances are made frequently. The ENet model was presented in 2016 and while the performance still holds up in terms of speed and efficiency, some may argue that it is outdated due to newer architectures having significantly higher accuracy. Modern architectures such as the newest DeepLab networks outperform ENet considerably, even when working only with RGB data. Performance of these networks may be improved further by working with data from multiple modalities. The approach for multimodal extension proposed in this thesis may be applied to relevant encoder-decoder architectures. Other types of architectures may be extended to work on multimodal data using different techniques.

Bibliography

[1] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello. ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation. *ArXiv e-prints*, June 2016.

[2] Wikipedia contributors. Visual system — Wikipedia, the free encyclopedia, 2018. URL `https://en.wikipedia.org/wiki/Visual_system`. [Online; accessed 25-March-2018].

[3] What is computer vision? - the british machine vision association and society for pattern recognition. http://www.bmva.org/visionoverview. [Online; accessed 25-March-2018].

[4] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010. ISBN 1848829345, 9781848829343.

[5] Wikipedia contributors. Lenna — Wikipedia, the free encyclopedia, 2018. URL `https://en.wikipedia.org/wiki/Lenna`. [Online; accessed 25-April-2018].

[6] J. Janai, F. Güney, A. Behl, and A. Geiger. Computer Vision for Autonomous Vehicles: Problems, Datasets and State-of-the-Art. *ArXiv e-prints*, April 2017.

[7] Larry Matthies, Mark Maimone, Andrew Johnson, Yang Cheng, Reg Willson, Carlos Villalpando, Steve Goldberg, Andres Huertas, Andrew Stein, and Anelia Angelova. Computer vision on mars. *International Journal of Computer Vision*, 75(1):67–92, Oct 2007. ISSN 1573-1405. doi: 10.1007/s11263-007-0046-z. URL `https://doi.org/10.1007/s11263-007-0046-z`.

[8] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, and J. Garcia-Rodriguez. A Review on Deep Learning Techniques Applied to Semantic Segmentation. *ArXiv e-prints*, April 2017.

[9] M. Thoma. A Survey of Semantic Segmentation. *ArXiv e-prints*, February 2016.

[10] H. Zhu, F. Meng, J. Cai, and S. Lu. Beyond Pixels: A Comprehensive Survey from Bottom-up to Semantic Image Segmentation and Cosegmentation. *ArXiv e-prints*, February 2015.

[11] Li Deng and Dong Yu. Deep learning: Methods and applications. 7, 01 2013.

[12] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, Dec 1989. ISSN 1435-568X. doi: 10.1007/BF02551274. URL https://doi.org/10.1007/BF02551274.

[13] J. Schmidhuber. Deep Learning in Neural Networks: An Overview. *ArXiv e-prints*, April 2014.

[14] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[15] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ArXiv e-prints*, February 2015.

[16] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. 15:1929–1958, 06 2014.

[17] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154, 1962. doi: 10.1113/jphysiol.1962.sp006837. URL https://physoc.onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1962.sp006837.

[18] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*, 195 (1):215–243, 1968. doi: 10.1113/jphysiol.1968.sp008455. URL https://physoc.onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1968.sp008455.

[19] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. ISSN 0018-9219. doi: 10.1109/5.726791.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[21] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *ArXiv e-prints*, September 2014.

[22] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *ArXiv e-prints*, December 2015.

[23] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going Deeper with Convolutions. *ArXiv e-prints*, September 2014.

[24] Wikimedia Commons. Cnn layers arranged in 3-d volumes, 2015. URL https://commons.wikimedia.org/wiki/File:Conv_layers.png.

[25] Cs231n convolutional neural networks for visual recognition. https://www.http://cs231n.stanford.edu/. Accessed: 10-05-2018.

[26] J. Long, E. Shelhamer, and T. Darrell. Fully Convolutional Networks for Semantic Segmentation. *ArXiv e-prints*, November 2014.

[27] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *ArXiv e-prints*, mar 2016.

[28] H. Noh, S. Hong, and B. Han. Learning Deconvolution Network for Semantic Segmentation. *ArXiv e-prints*, May 2015.

[29] V. Badrinarayanan, A. Kendall, and R. Cipolla. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *ArXiv e-prints*, November 2015.

[30] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. *ArXiv e-prints*, May 2015.

[31] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[32] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs. *ArXiv e-prints*, December 2014.

[33] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *ArXiv e-prints*, June 2016.

[34] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam. Rethinking Atrous Convolution for Semantic Image Segmentation. *ArXiv e-prints*, June 2017.

[35] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik. Learning Rich Features from RGB-D Images for Object Detection and Segmentation. *ArXiv e-prints*, July 2014.

[36] A. Valada, G. Oliveira, T.Brox, and W. Burgard. Deep multispectral semantic scene understanding of forested environments using multimodal fusion. In *International Symposium on Experimental Robotics (ISER 2016)*, oct 2016. URL http://lmb.informatik.uni-freiburg.de/Publications/2016/OB16c.

[37] A. Eitel, J. T. Springenberg, L. Spinello, M. Riedmiller, and W. Burgard. Multimodal Deep Learning for Robust RGB-D Object Recognition. *ArXiv e-prints*, July 2015.

[38] C. Hazirbas, L. Ma, C. Domokos, and D. Cremers. Fusenet: Incorporating depth into semantic segmentation via fusion-based cnn architecture. In *Asian Conference on Computer Vision (ACCV)*, November 2016.

[39] A. Canziani, A. Paszke, and E. Culurciello. An Analysis of Deep Neural Network Models for Practical Applications. *ArXiv e-prints*, May 2016.

[40] Dong-Ki Kim, Daniel Maturana, Masashi Uenoyama, and Sebastian Scherer. Season-invariant semantic segmentation with a deep multimodal network. In Marco Hutter and Roland Siegwart, editors, *Field and Service Robotics*, pages 255–270, Cham, 2018. Springer International Publishing. ISBN 978-3-319-67361-5.

[41] Jesse Tetreault. Deep multimodal fusion networks for semantic segmentation. Master's thesis, Clemson University, 8 2017. All Theses. 2756. `https://tigerprints.clemson.edu/all_theses/2756`.

[42] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler. Efficient Object Localization Using Convolutional Networks. *ArXiv e-prints*, November 2014.

[43] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Hypercolumns for Object Segmentation and Fine-grained Localization. *ArXiv e-prints*, November 2014.

[44] H. Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):328–341, Feb 2008. ISSN 0162-8828. doi: 10.1109/TPAMI.2007.1166.

[45] Daniel Hernandez-Juarez, Alejandro Chacón, Antonio Espinosa, David Vázquez, Juan Carlos Moure, and Antonio M. López. Embedded real-time stereo estimation via semi-global matching on the GPU. In *International Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA*, pages 143–153, 2016. doi: 10.1016/j.procs.2016.05.305. URL `http://dx.doi.org/10.1016/j.procs.2016.05.305`.

[46] F. Liu, C. Shen, and G. Lin. Deep Convolutional Neural Fields for Depth Estimation from a Single Image. *ArXiv e-prints*, November 2014.

[47] A. Saxena, M. Sun, and A. Y. Ng. Make3d: Learning 3d scene structure from a single still image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5): 824–840, May 2009. ISSN 0162-8828. doi: 10.1109/TPAMI.2008.132.