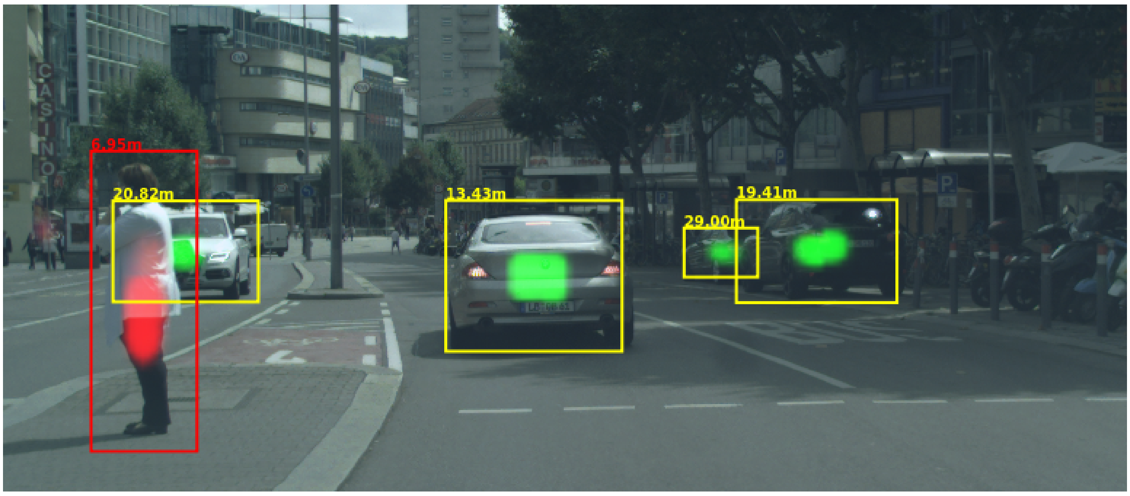
CHALMERS
UNIVERSITY OF TECHNOLOGY



Vehicle Detection and Road Scene Segmentation using Deep Learning

Master's thesis in Complex Adaptive Systems

AMRIT KRISHNAN, JONATHAN LARSSON

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

MASTER'S THESIS EX029/2016

Vehicle Detection and Road Scene Segmentation using Deep Learning

AMRIT KRISHNAN, JONATHAN LARSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Signals and Systems
Signal processing and Biomedical engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Vehicle Detection and Road Scene Segmentation using Deep Learning
AMRIT KRISHNAN, JONATHAN LARSSON

Supervisors: Samuel Scheidegger, Autoliv
Lennart Svensson, Department of Signals and Systems
Examiner: Lennart Svensson, Department of Signals and Systems

Master's Thesis EX029/2016
Department of Signals and Systems
Signal processing and Biomedical engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A traffic scene with cars, pedestrian and cyclist detections using a deep learning approach.

Typeset in LaTeX
Printed by [Name of printing company]
Gothenburg, Sweden 2016

Vehicle Detection and Road Scene Segmentation using Deep Learning

AMRIT KRISHNAN, JONATHAN LARSSON
Department of Signals and Systems
Chalmers University of Technology

Abstract

There has been significant progress in applying deep learning techniques to computer vision problems for perception scenarios, specifically for autonomous driving. This thesis explores many of these techniques and presents a detailed analysis of how deep learning tools can be used to perform computer vision for self driving vehicles. It is an attempt to replicate concurrent research work, while discussing the advantages and disadvantages of different approaches. The thesis also combines different methods and components, and presents a custom vehicle detection approach. The approach is based on generating a spatial grid of classifications, and then regressing bounding-boxes for pixels with a high object confidence score. The custom detection approach was tested on the KITTI object detection benchmark, and was able to successfully detect objects of varying scale, lighting conditions and orientation. Additionally, an analysis of semantic segmentation techniques that use deep learning is presented, and a few hand-picked approaches from literature are evaluated and compared. One of the approaches is replicated and tested on the Cityscapes benchmark for pixel level semantic segmentation. Detailed discussion and insights into future work are presented, which could be interesting for both academia and industry, especially in the area of deep learning and autonomous systems.

Keywords: Deep learning, object detection, classification, semantic segmentation

Acknowledgements

We would like to thank our industrial supervisor, Samuel Scheidegger, for his support, guidance and technical help throughout the thesis work. We also thank our examiner, Lennart Svensson, for his invaluable guidance and input during our work. We would like to also thank participants in the research meetings for Autonomous Driving, Lars Hammarstrand, Erik Stenborg, Karl Granström, Fredrik Kahl, Anders Karlsson, Luca Caltagirone and Henrik Wallenius for the interesting discussions. Furthermore, we would like to acknowledge the useful input received from our opponents Jessy Nassif and Saudin Botonjic. Finally, we would like to thank Autoliv AB for the opportunity to perform this work.

Amrit Krishnan, Jonathan Larsson, Gothenburg, June 2016

Contents

List of Figures

List of Tables

1
Introduction

Research and development in the area of autonomous vehicles has been fast-paced in the recent years and is currently approaching a tipping point, and active safety is one of the leading modes of research within the area of autonomous driving. On a different side of the spectrum, research in computer vision has also been transformational in the recent past, where deep learning has become a “de facto” tool to achieve many tasks. In this thesis, deep learning methods in computer vision are explored and applied to one specific task that concerns the development behind autonomous vehicles. The thesis aims to understand deep learning methods, and attempts to create a knowledge transition process where contemporary research work in computer vision and learning representations are connected with a real-life application.

1.1 Background

In recent years, there has been tremendous progress in the area of deep learning for computer vision. GPU computing has enabled the training of very advanced models or “deep neural networks”, in relatively short time scales. The automotive industry realises the importance of using deep learning approaches for object localisation and detection, since it can play a vital role in the development of autonomous driving capabilities, specifically related to active safety. Currently, deep learning algorithms require a lot of computational power, and using a Graphical Processing Unit (GPU) on-board a vehicle is not ideal due to the sheer power consumed and heat released by the processors. This limitation is expected to be overcome in the future, as models become more lightweight, and processors for deep learning are specifically designed to consume less power. For example, Field Programmable Gate Arrays (FPGAs) are being considered as a practical computational alternative to support deep learning models. The research work and interest in the field of deep learning is increasing rapidly, as newer algorithms and ideas re-define the state-of-the art within short time intervals, indicating a great potential and need to understand the underlying concepts.
Although the research behind self-driving vehicles is quite extensive, there is room for a lot of improvement before they are integrated into mainstream commercial use. Currently, one of the short-comings of research-grade autonomous vehicles is the lack of a robust learning mechanism that is generalisable over different conditions. For humans, it is very easy to distinguish between cars, pedestrians and other objects on the road irrespective of lighting conditions, surroundings and other variations.

To achieve such a high degree of insensitivity to variations, a powerful and invariant learning mechanism is required. Deep learning offers the potential to construct such a learning algorithm, removing a great deal of hand-engineering, while still retaining a large modelling capacity.

1.2 Purpose

An in-depth study of the use of deep learning tools for the specific tasks of vehicle detection and road scene segmentation would be valuable, enabling a stronger connection of contemporary research work to the needs of the industry. Furthermore, the pace of research and development in the field of deep learning is quite exhilarating and overwhelming, and a thesis that brings together different methods, techniques and architectures can help frame a broader perspective of deep learning for practical applications. Hence, the main purpose of this thesis was to explore one aspect of constructing an autonomous vehicular system, which primarily involves perception of information from a vehicle's surrounding, and the ability to detect other vehicles and interesting objects on the road.

1.3 Delimitations

The scope of this thesis was to understand deep learning methods specifically for vehicle detection using images from a camera. Models were to be trained in a supervised manner, implicating that a considerable amount of annotated data, providing ground truth information was necessary. This limited the thesis work to only consider scenarios for which large data sets were available. Furthermore, the type of data used limited the scope of the applicability of the model. For instance, if a model was trained using images from a highway driving environment, it might not have performed as well on urban driving scenarios. This limited the potential of creating a more generalisable model which is more invariant to scenarios and local environmental changes.

Decidedly, only public data sets were used, namely the KITTI data set [8] and the Cityscapes data set [4]. From both of these data sets, only mono-camera images were considered. Also, no inter-frame information was used, i.e. images were considered to be independent of each other.

Ideally, a deep network would have to be trained from scratch. However, the project was limited by time and computing resources available, and thus transfer learning was applied by using pre-trained networks. This drastically reduces the training time of models, since they only had to be fine-tuned to fit the purpose of the thesis.

1.4 Methodology

Prior to the start of the thesis, different research papers connected to deep learning were discussed in a study group, which gave a broad perspective of the field of deep learning. The thesis work began with literature study, covering various aspects of deep learning, including approaches to detection and semantic segmentation. In the earlier weeks, a small-scale detection implementation was attempted using the MNIST data set [16]. During this phase, familiarity with Torch7, a deep learning framework, combined with LuaJIT was achieved, allowing continued work using the framework throughout the thesis. The main task of the thesis, namely object detection using the KITTI data set, was attempted next. This task consumed most of the time and effort in the thesis. Finally, approaches to semantic segmentation were studied in the later weeks and one of the approaches from literature was replicated to gain a proper understanding of image segmentation using deep learning.

1.5 Related Work

Computer vision for active safety in vehicles has been a topic of research and development for over a decade. Before the advent of deep learning methods, deformable parts based model [7] was the most advanced and reliable technique for vehicle detection. However, due to the low precision of the traditional approaches in computer vision, other sensors such as LiDAR and radar were fused with a camera to get vehicle and other road object detections. Furthermore, road models and motion models of the vehicle had to be used to eliminate false positives. A major drawback of traditional computer vision approaches was that the features for constructing object classifiers and detectors were hand-designed.
Deep learning on the other hand, using Convolutional Neural Networks (CNNs), has been quite successful in classification tasks. CNNs have also been used quite effectively for localisation and detection tasks and progress in this direction has been immensely promising in the last year.
Recent work using CNNs has been shown to be very effective for highway driving scenarios [12]. The research work led by Ng, A.Y., used a modified version of *Overfeat* [23], a CNN architecture for feature extraction, and built on top of it, a method to achieve more dense predictions. The *Overfeat* architecture introduced a significant improvement to making CNNs efficient, which reduced the time of inference on large images immensely. The approach in [12] increased the resolution of predictions, while simultaneously reducing the number of predicted bounding boxes, compared to the original approach to detection using the *Overfeat* architecture.
Additionally, extensive work has gone into improving the configurations and training schemes whilst using CNNs. An approach using an improved architecture and hard negative mining of data is described in [11]. The implementation incorporated a more careful training scheme, seemingly suitable whilst training on smaller data sets.
The task of object localisation and detection has been largely addressed by using CNNs and formulating the detection problem as that of classification of a smaller region; thus obtaining a reduced spatial map of classifications. Extending this concept, classifying every pixel in an image is a more fascinating task and the result is a spatial semantic map, that attempts to give rich semantic information about the entire scene contained in the image.
Significant work on semantic segmentation using CNNs has been done in the recent past. One of the pioneering deep learning based semantic segmentation ap-

proaches [17] used the fully-convolutional network. Using this architecture, information extracted from different layers of the standard CNN was combined, and a novel segmentation architecture was then used to perform upsampling and achieve dense pixel level predictions. Another approach, which was more end-to-end, described in [18], introduced the idea of using a dedicated "shape-reconstruction" network to perform the upsampling and give pixel level predictions. A more recent approach [33], used a different type of convolution operation to aggregate multi-scale context information and achieve dense prediction, without loss of resolution. Besides these approaches, there have been other techniques using deep learning for segmentation, which offer potential for future work.

2

Theory

This chapter introduces deep learning in general and covers the theoretical concepts, terminologies and components that are relevant for the thesis. Many of the components were used to construct the model architectures for the methods in the thesis, while some of the components were experimented with to gain further understanding, and the learning from such experimenting is covered in the discussions.

2.1 Artificial Neural Networks

In biology, a neural network is a set of connected, interacting nerve cells. The network transmits electrical signals via axon terminals from the synapses (output) of some neurons to the dendrites (input) of other neurons. Each neuron has an activation threshold, for which the sum of all input signals must surpass in order for the neuron to fire and transmit the signal further through the network. Biological neural networks are highly complex; the most extreme example being the human brain consisting of roughly 10^{10} neurons, each connected to 10^4 other neurons on average [1].
As suggested by the name, *artificial* neural networks (ANN) are simulated networks, inspired by biological neural networks. The connection strength between neurons is modeled with a numeric weight and the activation threshold is encoded by an internal numeric bias. In addition, each neuron is usually associated with a non-linear activation function. To make the network a computational unit, a set of neurons are decided to be input neurons and some to be output neurons, thus using the network as a non-linear function. An artificial neural network can have any number of input and output neurons, making it a highly diverse operational structure which could be used in numerous ways for many different tasks.

2.2 Deep Feedforward Networks

Deep feedforward networks represent a set of modelling tools that have proven to be very successful in pattern recognition, classification, regression and other tasks generic to machine learning [13]. A main property of a feedforward network is that information flows through the network along a single direction, and hence the network does not contain any feedback or self-connections. The neurons are often arranged in layers so that the network has a dedicated input layer, output layer and potentially some hidden layers, providing a systematic method of calculating the

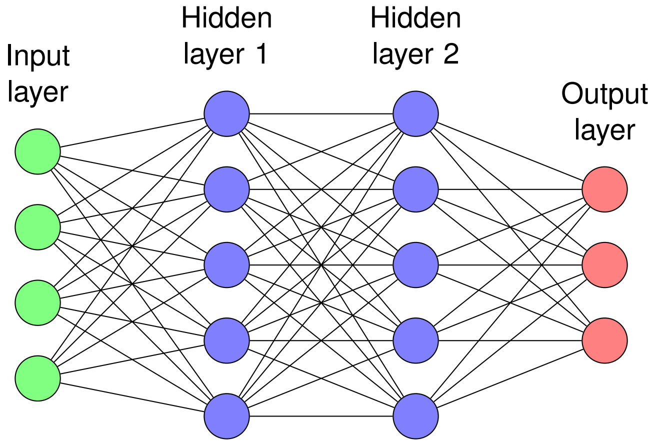


Figure 2.1: Simple example of a feed forward neural network.

activations of the output layer, given the input and the weights and biases of all intermediate layers.
The network presented in figure 2.1 is an example of a feedforward neural network with *fully connected layers*. Mathematically, the process of feeding information through a network with fully connected layers is represented by a series of matrix-vector multiplications and element-wise activation functions. Denote $\mathbf{x}$ to be the input arranged in an N dimensional vector, $\mathbf{W}_1$ to be the matrix of weights $w_{i,j}$ connecting input neuron j with hidden neuron i, $\mathbf{b}_1$ to be a vector with the biases of the neurons in the first hidden layer and σ_1 to be an activation function. Then, the output vector $\mathbf{a}_1$ from the first hidden layer is

$$\mathbf{a}_1 = \sigma_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1), \tag{2.1}$$

which will act as the input to the second hidden layer of the network. Analogously, the output from layer n is

$$\mathbf{a}_n = \sigma_n(\mathbf{W}_n\mathbf{a}_{n-1} + \mathbf{b}_n). \tag{2.2}$$

A *deep* feedforward neural network is simply a feedforward network with many layers. As the information flows through the layers, it forms a hierarchical representation of the input features. Hence, a deep feedforward neural network is an artificial neural network with a high modelling capacity that allows for mapping greater non-linearity.

2.2.1 Activation Functions

The activation functions are essential components of ANNs, used to perform non-linear mappings of the input data and are typically applied element-wise to all neurons in a hidden layer. This section describes a few commonly used activation functions and their properties. The activation functions commonly used in the intermediate layers of ANNs are presented in figure 2.2

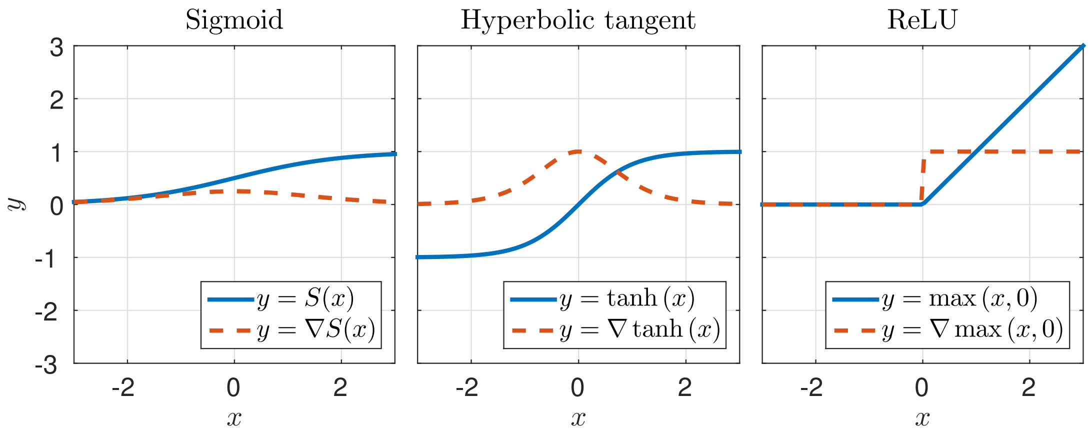


Figure 2.2: The sigmoid, hyperbolic tangent and ReLU activation functions and their derivatives.

Sigmoid

The sigmoid activation function is defined as

$$S(x) = \frac{1}{1 + e^{-x}}. \tag{2.3}$$

It takes values from 0 to 1 and is linear close to the origin. It has a "squashing" property such that large positive or negative input values result in values close to 1 or 0 respectively.

Hyperbolic Tangent

The hyperbolic tangent,

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{2.4}$$

is also commonly used as activation function in ANNs. It has very similar properties to the sigmoid function except $\tanh(x)$ takes values from -1 to 1.

Rectified Linear Unit

The Rectified Linear Unit (ReLU) is an activation function commonly used with deep neural networks, and is simply given by

$$f(x) = \max(x, 0). \tag{2.5}$$

Compared to the commonly used sigmoid activation function, the ReLU has the advantage of not saturating for large inputs. While sigmoidal functions take values in the range $(0, 1)$, ReLUs take values in $[0, \infty)$ making them less prone to be either "on" or "off".

Softmax

The softmax function is a normalising function commonly used as an activation function for the last layer of a neural network. Given an input vector $\mathbf{x}$ with values x_i, the corresponding output element y_i is calculated

$$y_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}. \tag{2.6}$$

By construction, the elements of the output vector sums to 1. This property is useful when the output of the neural network should encode probabolities for different cases, such as in classification tasks.

2.2.2 Backpropagation

To make the output of a neural network a useful mapping of its input, a common strategy is to train its parameters using supervised learning. This requires having predetermined desired outputs for given input data, which can be compared to the actual output of the ANN. The desired output $\mathbf{y}$ along with the actual output $\hat{\mathbf{y}}$ is passed to a differentiable *cost function* C, which is minimised by adjusting the parameters (weights and biases) of the network.
Let Θ be the set of all parameters in the network; then the objective when training in a supervised manner is to minimise

$$\min_{\Theta} \frac{1}{N} \sum_{i=1}^{N} C(\mathbf{y}_i, \hat{\mathbf{y}}_i | \Theta), \tag{2.7}$$

where $\hat{\mathbf{y}}_1 \dots \hat{\mathbf{y}}_N$ and $\mathbf{y}_1 \dots \mathbf{y}_N$ are the network outputs and labels corresponding to input *training data* $\mathbf{x}_1 \dots \mathbf{x}_N$. The procedure of passing data through the network, calculating the cost and adjusting the parameters continues until the network has reached an acceptable accuracy when evaluated on the *validation data set* which is separated from the training data set.
For a feedforward neural network, the method used for adjusting the parameters is referred to as *backpropagation*. With gradient descent, backpropagation propagates the gradients of the cost function with respect to the parameters back through the network using the chain rule [9].

Stochastic Gradient Descent

While only one sample of input data, desired output and the actual output is required to calculate the gradients for all parameters of the network, it is common practice to include several samples and take an average of the obtained gradients. This batch of samples is comonly referred to as a *mini-batch*. Training networks using mini-batches is beneficial in several ways. Firstly, an average of gradients is a better approximation to the gradients of the minimisation problem (2.7), which includes gradients over the entire training set. Secondly, computing the gradients of several inputs at the same time is typically more efficient when using the high level of parallelism available in modern computers and GPUs.

Stochastic Gradient Descent (SGD) is the process of randomly selecting samples from the training set, computing the gradients and then updating the parameters as

$$\Theta \leftarrow \Theta - \frac{\alpha}{m} \sum_{i=1}^{m} \frac{\partial C(\mathbf{y}_i, \hat{\mathbf{y}}_i | \Theta)}{\partial \Theta}, \tag{2.8}$$

where α is the *learning rate* and m is the mini-batch size.
Typically, the amount of training done is measured in *epochs*, defined as the number of times all training samples have been used to update the network parameters. If the number of available training samples is N, one epoch is completed after using N/m mini-batches for training. If the input data is subject to some augmentation before being passed to the network, the number of training samples actually used might be intractable and thus the measure of one epoch cannot be defined as in the original sense. In such cases, the amount of training done is measured directly by the number of mini-batches used.

Vanishing Gradient problem

Training deep neural networks with backpropagation means propagating the gradients through many layers. Using the chain rule, the gradient with respect to any given layer is multiplied by the gradients of the activation functions for each layer already passed. Since the gradients of the activation functions are typically restricted in the range $(-1, 1)$, the resulting gradient is a product of numbers with magnitude less than one, and thus tends to zero. Effectively, layers close to the input layer of the network learn slower compared to layers closer to the output of the network, resulting in an inefficient and slower learning process. This problem is referred to as the *vanishing gradient problem* [13]. One technique that has significantly helped in tackling the vanishing gradient problem is the usage of ReLUs in deep neural networks.

2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are feedforward neural networks with a layout and architecture specifically designed to handle data arranged in a spatial grid (tensors), such as 2D or 3D images. The inspiration of the architecture comes from the mechanism of biological visual perception [1]. The networks, like any other ANN, are composed of neurons with learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with an activation function. The architecture is typically composed of several layers, which gives them the characterisation of being “deep” and thus research work on CNNs fall under the domain of deep learning. Essentially, the network computes a mapping function that relates image pixels to a final desired output.
In a general CNN, the input is assumed to be an RGB image, i.e. consisting of three channels, corresponding to the red, green and blue color intensity values. Consecutive layers of the CNN may consist of even more channels referred to as *feature maps*. The number of feature maps typically increase through the layers of a CNN, while the spatial dimension of them decreases until reaching the desired

output size. The over-all idea behind this structure is that the representation of the input image is gradually increased in abstraction as it progresses through the layers. Later layers contain more information about the “what” and “how” of objects and things in an image, and less of “where”. Similar to the definition of a feature map, a *feature vector* at a specific layer of a CNN is defined to be the elements across all feature maps at a given spatial location. An example of the structure of a CNN is presented in figure 2.3

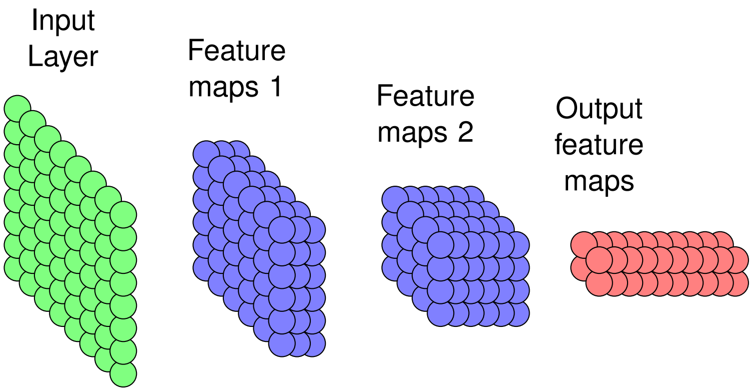


Figure 2.3: Illustration of how neurons are structured in a CNN. This simple example shows a 8×8 single channel (grayscale) image input and the arrangement of intermediate feature maps of the network. Connections between neurons have been removed for clarity.

2.3.1 Components of CNNs

Besides the fully connected layer presented in section 2.2, there are other types of layers and connections that can be used to construct deep convolutional networks. Typically, the purposes behind some of these components are to reduce the dimensions of intermediate layers, reshaping spatial dimensions, simulating fully connected layers and more. Following are some sections describing important components in the CNN framework used in this thesis.

Convolution

The discrete 2D convolution operation, illustrated in figure 2.4, is defined by a convolution kernel $\mathbf{k}$ of size $k \times k$. Given an $N \times M$ input image (tensor) $\mathbf{X}$, the convolution kernel is run along all the pixels in the image, multiplying the surrounding pixel values with the kernel and adding them. The resulting output is a $(N - k + 1) \times (M - k - 1)$ image $\mathbf{Y}$, where the output at pixel location i, j is calculated

$$Y_{i,j} = \sum_{i'=1}^{k} \sum_{j'=1}^{k} k_{i',j'} X_{i-(k+1)/2+i',j-(k+1)/2+j'}. \tag{2.9}$$

The operator is denoted using $*$ and thus

$$\mathbf{Y} = \mathbf{k} * \mathbf{X}. \tag{2.10}$$

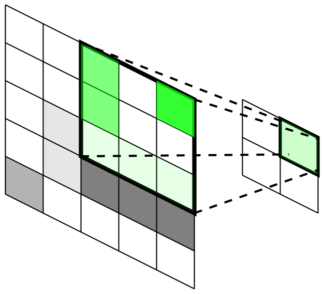


Figure 2.4: Illustration of the convolution operation using a 3×3 kernel, where multiple input activations are associated to a single output activation

The convolution operation between two layers in a CNN is defined by a kernel $\mathbf{k}_{ij}$ for each pair of feature maps in the two layers. Similar to the fully connected layer, the result from all convolutions related to feature map i in the later layer is summed and a bias b_i is distributed and added across the entire feature map. Let $\mathbf{X}_j$, be a feature map in a layer with $N \geq j$ feature maps, then the feature map $\mathbf{Y}_i$ in the next layer is calculated

$$\mathbf{Y}_i = \sigma_i \left(\sum_{j=1}^{N} \mathbf{k}_{ij} * \mathbf{X}_j + b_i \mathbf{1} \right), \tag{2.11}$$

where $\mathbf{1}$ is an all-ones matrix and σ_i is an element-wise activation function. The elements of all kernels $\mathbf{k}_{ij}$ and the biases b_i are learnable parameters and are updated through the process of training the network.

As mentioned, the convolution operation has the effect of reducing the dimensions of the feature maps. However, there are some methods which can be used to avoid or modify this effect. A common strategy to preserve the spatial dimensions is to apply zero-padding, i.e. covering the feature maps before the convolution operation with a border of $(k-1)/2$ zeros, which counters the spatial reduction exactly. On the other hand, to reduce the spatial dimensions of intermediate layer activations further, a *stride* can be associated with the convolution. The stride is applied by sliding the convolution kernel by s number of steps between each "multiply and sum" operation and the result is that the feature maps are reduced in size by a factor s. Most commonly, the convolutions are applied without stride, i.e. $s = 1$.

An important concept introduced by applying a convolution layer to a CNN is the *receptive field*. The receptive field is a measure of how much information from the input image is available to the feature vectors of a specific layer in a CNN. If the first component of a CNN is a convolutional layer with kernel width k_1, the receptive field of the first layer is $k_1 \times k_1$ – i.e. each element in the second layer was provided information from the $k_1 \times k_1$ nearest pixels in the input image. Following this pattern, a series of n convolution layers with kernel sizes $k_1, \ldots, k_n$ results in a receptive field of

$$\left(1 + \sum_{i=1}^{n}(k_i - 1)\right) \times \left(1 + \sum_{i=1}^{n}(k_i - 1)\right). \tag{2.12}$$

Up-convolution or backwards convolution

An operation analogous to the convolution operation, but reversed, referred to as up-convolution can be used for upsampling. It is achieved by convolution operation reversed, illustrated in figure 2.5. Hence, if upsampling by a factor f is desired, it can be formulated as a convolution with a fractional input stride of $\frac{1}{f}$. Such a layer can associate a single input activation to multiple output activations. This “up-convolution” layer has learnable filter parameters that could correspond to bases for reconstructing shapes of an object [18]. Hence, an end-to-end learning mechanism can be constructed by repeated downsampling and then upsampling to achieve dense predictions, and this technique has been successfully used for dense pixel-level predictions [17].

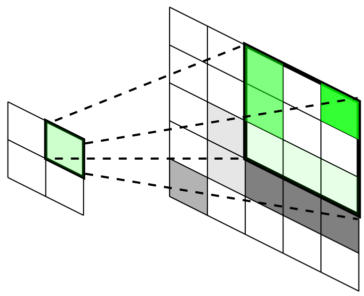


Figure 2.5: Illustration of the upconvolution operation, where a single input activation is associated to multiple output activations

Pooling

Pooling layers are non-learnable layers used to reduce the spatial dimensions of the feature maps as they pass through the network. Similar to the convolution layer, they are associated with some kernel of size $k \times k$ and a stride s. There are two commonly used types of pooling layers; the max-pooling layer and the average-pooling layer. The max-pooling layer, illustrated in figure 2.6, performs a $\max()$ operation with the elements of the feature map at each position of the kernel, thus discarding the information of the non-max neurons.

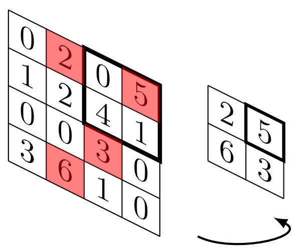


Figure 2.6: Illustration of the max-pooling operation, where the maximum of activations in a window of input activations is recorded and the output activation is simply the maximum value recorded.

The average-pooling layer performs an average at each position of the kernel, i.e. a normal convolution with the kernel values all set to $1/k^2$. Typically, the stride of the pooling layers is set to $s = k$, thus achieving a dimensional reduction of a factor s without using any zero-padding.

Apart from reducing the spatial dimensions of the feature maps, pooling layers also provide an efficient way of increasing the receptive field in a CNN. A pooling layer with stride s have the effect of increasing the receptive field of a factor s. This effect can also we achieved by including stride in a convolutional layer.

Unpooling

A reverse operation to pooling, "unpooling" aims to recover the original size of activations that were lost due to a pooling operation. A version of unpooling specifically for reversing the max-pooling operation, records the maximum activations selected during the pooling operation and uses them to recreate the original activations by placing the recorded values in their original positions, as illustrated in figure 2.7. This strategy is useful to recover the structure of objects and have a parameter-free upsampling. [18].

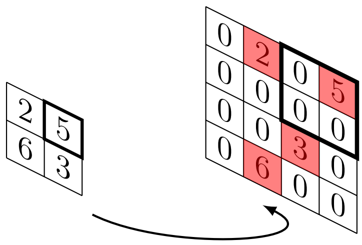


Figure 2.7: Illustration of the max-unpooling operation, where the recorded maximum activations from an associated max-pooling layer are used to recover the maximum input activations. The maximum value is recovered and the rest of the values within a window are set to zero.

1×1 Convolutions

Technically, 1×1 convolutional kernels are no different from any $k \times k$ kernel in the way they are applied. However, there is a conceptual difference between them in that $k \times k$ convolutions are usually thought of as edge/feature detectors while 1×1 kernels can only combine activations of each feature vector. The interpretation of such an operation is that it is simulating the effect of a fully connected layer, applied to each feature vector [12]. Since the convolution operation is not dependant on the spatial size of its input, transforming fully connected layers to 1×1 convolutions is a useful way of generalizing the network for different input sizes.

Batch Normalisation

Training deep neural networks can be quite tricky in practice due to the fact that the distribution of each intermediate layer's inputs changes during training. Such a change is caused by the changes in the parameters of the previous layers. This problem is referred to as *internal covariate shift*. When the input distribution changes, the activations tend to move into the saturated regimes, and this effect is amplified as the network depth increases, thus also causing the vanishing gradient problem. To tackle this, a normalisation technique in [14] was introduced, where normalisation of the inputs to the activation layers are done over the mini-batch. The batch normalising transform algorithm is briefly presented and is explained in detail in [14]. Consider a mini-batch of n inputs in a mini-batch $\mathcal{B} = \{x_1, x_2..., x_n\}$. Let the normalised values be $\{\hat{x_1}, \hat{x_2}..., \hat{x_n}\}$ and the resulting linear transformation of the batch normalisation can be represented by $\{y_1, y_2..., y_n\}$. Then, the batch normalising transform given by

$$\mathbf{BN}_{\gamma,\beta} : \{x_1, x_2..., x_n\} \rightarrow \{y_1, y_2..., y_n\}, \tag{2.13}$$

where γ and β are learnable parameters that correspond to the scaling and shifting in the transformation. Hence the mini-batch mean and variance which are given by

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{n}\sum_{i=1}^{n} x_i \quad \sigma^2_{\mathcal{B}} \leftarrow \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_{\mathcal{B}})^2, \tag{2.14}$$

are used to achieve the normalisation. Thus, the normalisation looks like

$$\hat{x_i} \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \tag{2.15}$$

and the resulting transformation can be given by

$$y_i \leftarrow \gamma\hat{x_i} + \beta. \tag{2.16}$$

Inception module

The inception module introduced in [27] consists of an architectural improvement that aims to increase the depth of the network, while remaining efficient. The idea is to have convolutional building blocks, and build up a deep architecture by carefully studying the correlation statistics of the previous layer. A typical inception block or module could consist of:

- A layer of 1×1 convolutions which are meant to cover the entire spatial extent of the output from the previous layer.
- A layer of 3×3 convolutions which are meant to cluster small groups of units with high spatial correlation
- A layer of 5×5 convolutions which are meant to cluster bigger groups of units with high spatial correlation

In the earlier stage of the network, the feature maps tend to have varying degrees of correlation and hence tend to have all three types of convolution layers in the subsequent inception module. However, as the depth of the network increases, the

1×1 and 3×3 convolution layers make better "design choices". An advantage of this architecture is that the module could have dimensionality reduction by introducing 1×1 convolution layers before the more expensive 3×3 or 5×5, where the feature map dimension could be significantly reduced for the larger spatial convolution layers. This avoids the computational bottleneck while using inception modules. Figure 2.8 shows the inception module, which is a building block in various inception architectures [26, 27].

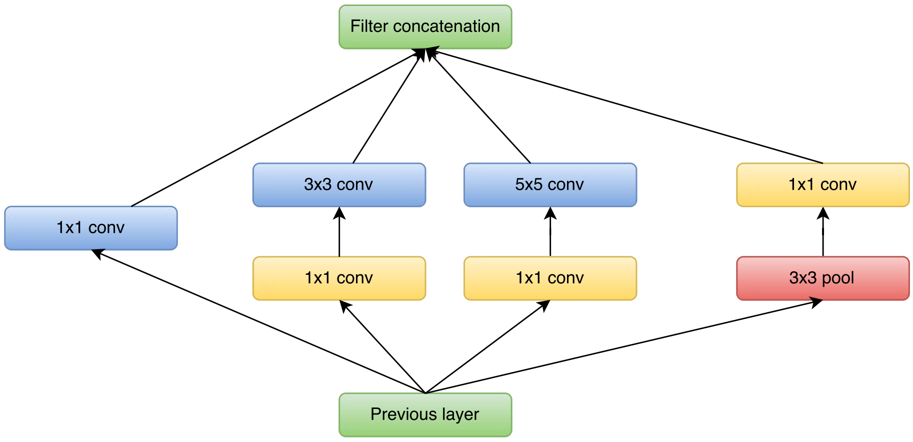


Figure 2.8: Illustration of the inception module.

Residual learning

The success of deep convolution networks encouraged researchers to try and construct even deeper architectures. However, with depth, the learning tended to saturate, and this was attributed to the vanishing gradient problem. Deep residual learning, introduced in [10] presents an architecture which aims to solve the degradation problem observed in very deep architectures. The idea is to introduce skip connections between multiple convolutional layers such that the few stacked layers in-between learn a residual mapping instead of the original one. More concretely, if a few stacked layers were to fit a desired mapping say $\mathcal{H}(\mathbf{x})$, through the modified architecture the stacked layers would now try to fit another mapping $\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x}$. Thus, the original mapping is re-formulated into $\mathcal{F}(\mathbf{x}) + \mathbf{x}$. The intuition behind the architecture is that it is easier to optimise a residual mapping than to optimise the original, unreferenced mapping. Figure 2.9 shows the basic building block in a residual learning architecture. This building block of residual layers with skip connections allows for the construction of very deep architectures, where the building block essentially is stacked on top of each other.

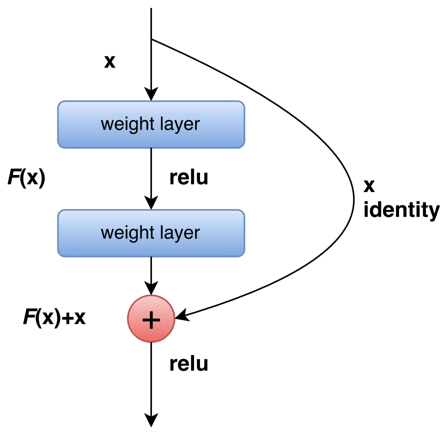


Figure 2.9: The building block in a residual learning scheme with skip connections.

Tiling

Tiling is a non-trainable form of upsampling based on combining and reshaping feature maps. The operation is associated with a single parameter t with the restriction that t^2 must be a factor of the number of feature maps in the layer input. Given m feature maps of size $N \times M$, the reshaping is applied to all feature vectors, rearranging them into m/t^2 matrices of size $t \times t$. The matrices from all feature vectors are then concatenated, forming m/t^2 feature maps of size $tN \times tM$.
There is a particularly useful interpretation of the tiling operation when it is applied directly after a 1×1 convolution layer. Since the weights of the convolution kernels are not shared across feature maps, the result can be viewed as applying t^2 independent fully connected layers to each spatial location and arranging the output in a $t \times t$ grid [12].

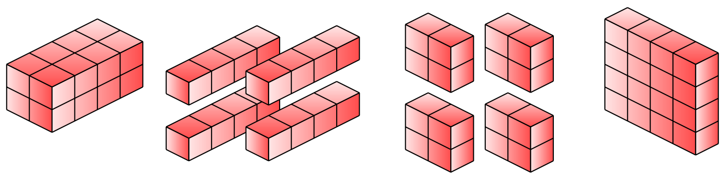


Figure 2.10: Illustration of the tiling operation, where a tensor that extends in the feature dimension is reshaped such that it tiles across in the spatial dimension. The example shows a tiling operation of size 2 applied to 4 feature maps of size 2×2.

Dilated Convolutions

Dilated convolutions [33] are modified convolutions which allow the increase of a network's receptive field size, while retaining the spatial size of the intermediate feature maps. The general convolution operator $*$ can be extended to define dilated convolution filters, where expressions for the general discrete convolution and dilated convolution can be defined as follows. Let $\mathcal{F} : \mathcal{Z}^2 \to \mathcal{R}$ be a discrete function and $\Omega_r = [-r, r]^2 \cap \mathcal{Z}^2$. Also, let a discrete filter k be defined as $k : \Omega_r \to \mathcal{R}$ whose size is $(2r+1)^2$. Using this definition, the discrete convolution operator $*$ and the more generalised dilated convolution operator $*_l$ can be given by:

$$(F * k)(\mathbf{p}) = \sum_{s+t=p} F(s)k(t) \tag{2.17}$$

$$(F *_l k)(\mathbf{p}) = \sum_{s+lt=p} F(s)k(t), \tag{2.18}$$

where $*_l$ is referred to as the l-dilated convolution operation. When $l = 1$, the expression becomes the standard 1-dilated discrete convolution operation.
The dilation filters support exponential increase in receptive field as depicted in figure 2.11 and can be denoted by

$$F_{i+1} = F_i *_{2^i} k_i. \tag{2.19}$$

This trick is most useful for scene segmentation, where multi-scale contextual information needs to aggregated without throwing away spatial information. As depicted in figure 2.11, the dilated convolutions are a convenient workaround to the idea of downsampling feature maps through successive pooling or strides in convolution layers.

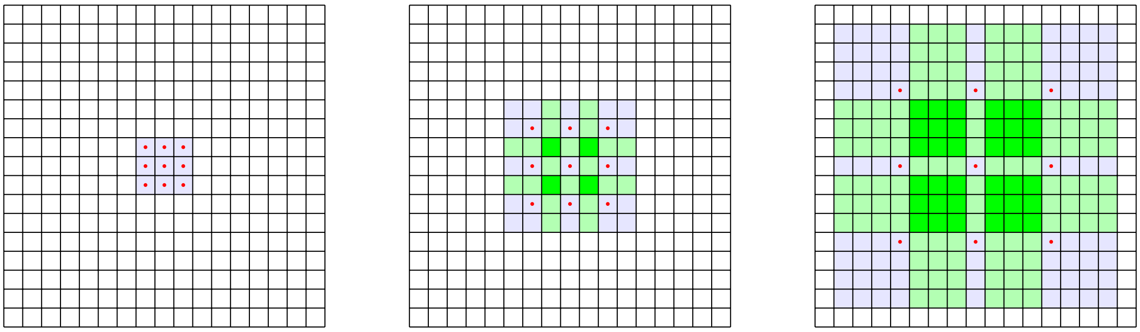


Figure 2.11: Illustration of a 3×3 kernel, systematically increased in dilation size from left to right. The first filter has 1-dilated convolutions with 3×3 receptive field, the second with 2-dilated convolutions with each element then having a receptive field of 7×7, the third filter with 4-dilated convolutions and an effective receptive field of 15×15. The green regions show overlapping regions of the elements, while the blue region shows the receptive field. The architecture supports exponential growth of receptive field.

2.4 Classification, Detection and Localisation

The ImageNet challenges [6, 22] popularised deep learning methods for object classification, localisation, detection and segmentation. The challenge consists of multiple tasks where the classification task is to assign images one of 1000 distinct classes, provided a training set with over 5 million annotated images. The approach and architecture using CNNs described in [15] achieved state of the art results for classification in 2012, bringing about a transformational change in computer vision methods.

The other tasks involve detection and localisation, which require to give information about the location of the classified object in the form of bounding box coordinates in the image pixel grid. Hence, CNNs were originally shown to be promising for classification tasks. However, soon after, they were modified to perform localisation and detection. In this section, we present briefly the corpus of theory that explains some concepts to perform localisation and detection using CNNs. These concepts are the essence of the thesis objective, and will provide a base for the approach used throughout the thesis.

2.4.1 Detection Through Classification

As a stepping stone between classification and localisation, many approaches rely on classifying spatial regions of the image independently, providing a "detection through classification" approach [12,23,28]. The detector typically assigns the spatial regions as either "Object" or "Background", reducing the amount of regions passed on for classification [20]. However, if the computational complexity of the classification is not of concern, classifier and detector can be combined to give a higher level of classification of the spatial regions directly [23].
With a CNN, a detection through classification approach can be achieved by setting the number of feature maps in the final layer equal to the number of classes to be distinguished. Each feature map can then be associated with a specific class. By applying a softmax function as activation function, the resulting feature vectors are corresponding to the probability distribution of the classes. The predictions of such a CNN would all have a corresponding spatial location in the image and a receptive field centered at the same location. As an effect of the stride of the layers in a CNN, the receptive field of the predictions are separated by some pixels. This defines the effective stride of the network which is obtained by multiplying the strides of the networks layers.
Typically, the later layers of the network have a series of 1×1 convolutions, so that independent predictions are made, given each feature vector in the layer before the 1×1 convolution layers [23, 28]. The sequence of earlier layers in a CNN is referred to as a *feature extractor* and the later layers as a *classifier*. Viewing the CNN pipeline as two separate networks is especially useful in the context of *transfer learning*. Transfer learning is a process of pre-training a network architecture on large and general data sets such as the ImageNet data set, and then replacing the final layers of the network to be able to train on another (typically smaller) data set. The idea behind such a process is that a pre-trained feature extractor is able to extract feature vectors with high conceptual information of the input image [32]. The technique regularises the network and is useful when the data set of interest is small.

2.4.2 Bounding Boxes

Even though the detection approach described in the previous section provides an indication of where the object is located in the image, it does not give information about the size of the object. A common strategy to give more detailed information about the object geometry is to use bounding box regression [12, 20, 23, 28]. The idea is to attach a smaller network to the output of the feature extractor which is run in parallel with the classifier as depicted in figure 2.12

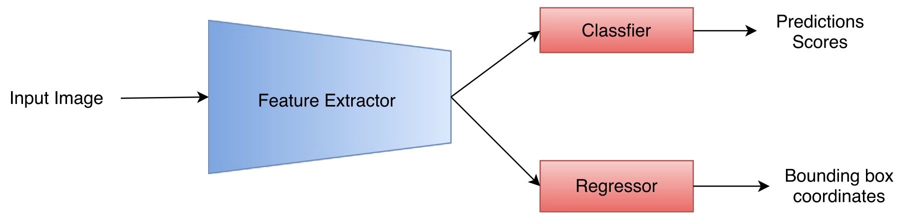


Figure 2.12: Illustration of the network architecture for perfoming detection through classification

The structure of the bounding box regressor is similar to the classifier. However, since a bounding box requires a pair of coordinates in relation to the image pixels, the number of output feature maps is set to 4. Thus, each element in the output feature vectors can be associated with one of the bounding box coordinates.
Combining the output from the bounding box regressor and the classifier gives information about both class and location of the objects in the image. Since there is a one-to-one correspondence of the spatial predictions made by the classifier, and the output coordinates from the regressor, the output of regions classified as "No class" can be neglected. The result is a set of spatial detections of objects, each associated with a bounding box. However, if the object is relatively large, the detector might classify neighbouring regions with the same class and thus yielding multiple bounding boxes corresponding to the same object.

3

Methods

In this chapter, the specific details of implementations and the approaches adopted will be explained. All the code for the implementations was written in LuaJIT, using the deep learning package Torch7 [3] and is available at `https://github.com/amrit110/masterThesis`. Torch7 was chosen primarily because of the highly active development community and support, which is especially useful, considering the pace at which the field of deep learning is evolving at the moment.

3.1 MNIST Digit Detection

As a proof of concept to the classification for detection approach, a small-scale network was implemented for the task of classifying and locating hand written digits in grayscale images. The approach used for this task was chosen such that it would resemble the scheme used for the main objective of detecting vehicles.

3.1.1 MNIST data set

The MNIST (Mixed National Institute of Standards and Technology) data set is a collection of handwritten digits commonly used to train and test various image processing models. It is one of the most commonly used data sets in machine learning research and is ideal to test proof of concepts. Models can be applied quite easily on the MNIST data set, and the hypotheses quickly tested [16]. There have been numerous scientific papers on attempts to achieve the best classification on the data set, and the best results have been achieved using CNNs.
The data set consists of 60,000 training images and 10,000 testing images. The digits have been size-normalised and centered in a fixed-size image, and the inputs are available as (28×28) grayscale images, encoded as integers between 0 and 255. Figure 3.1 shows examples of the images of digits taken from the data set.

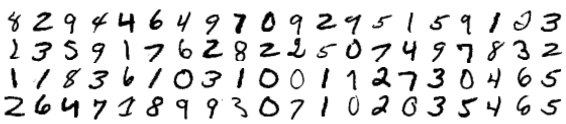


Figure 3.1: Examples of the digits from the MNIST data set.

3.1.2 Model architecture

For the digit detection task, the CNN model chosen was minimalistic in comparison to state-of-the art implementations used for vehicle detection. The intention was to have a classifier that produced highly accurate predictions and a bounding box regressor that was able to achieve detection. The model would also allow for understanding the construction of a basic CNN architecture where some of the different components presented in chapter 2 could be tried. The choices of filter sizes in the convolution layers were not optimal and was made arbitrarily to achieve the desired downsampling of the input. The network structure had a receptive field of 26×26 pixels and an effective stride of 4. This way, the receptive field of the classification network was larger than the size of the digit and the output had a resolution of one prediction per each 4×4 pixel region. Figure 3.2 depicts the architecture that was used. The number of feature maps for each convolution layer was increased by factor of two after each pooling operation. The reasoning behind this was that the features in later layers are richer and semantically more informative, and more feature maps are required to have distinguished representations. Additionally, the network was implemented with and without batch normalisation applied after each convolution layer and the performance was compared.

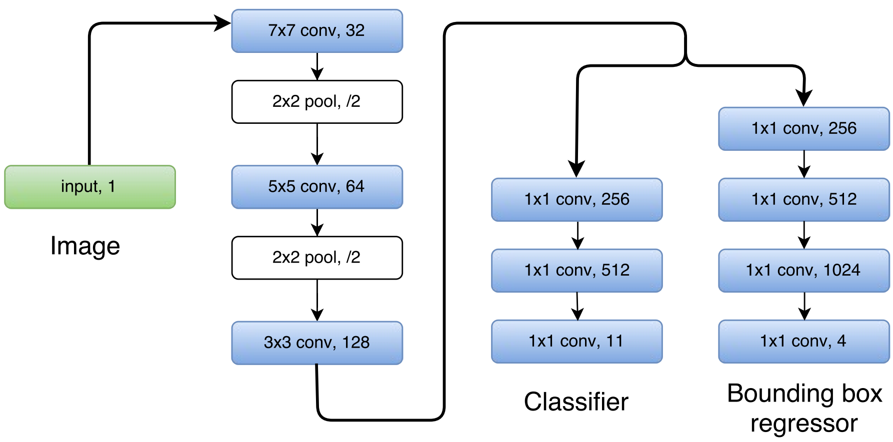


Figure 3.2: Model architecture used for MNIST digit detection implementation. In addition to the convolution and pooling layers, ReLUs are applied after each convolution layer.

3.1.3 Implementation details

To use the MNIST data set in a classification for detection approach, synthetic data sets for training, validation and testing were created by randomly scattering some of the original MNIST images onto a larger image. 2-5 MNIST images were randomly selected, downsampled to a size of 16×16 and then randomly placed on a blank, single-channel (grayscale) 72×72 image. To avoid overlap, the digits were placed in sequence, rejecting proposed locations if they were too close to the digits already placed. Additionally, a mask of Gaussian noise with zero mean and standard deviation 5 was added to each image, after which they were normalized to

zero mean and unit variance using statistics from the training set. Labels for the synthetic data set were created by storing the class and bounding box coordinates to each digit placed in the image.
Training, validation and test data sets were created using this technique, consisting of 9000, 1000 and 1000 images respectively. An example of the used images is presented in figure 3.3.

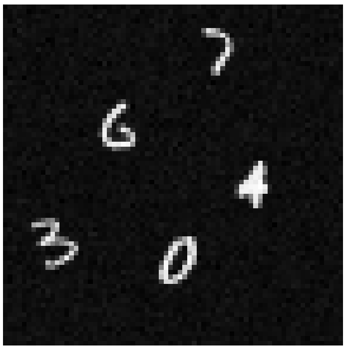


Figure 3.3: A random sample from the synthetic data set used for the MNIST digit detection task.

3.1.3.1 Training

The approach used for training the network was to use patches of the synthetic data set and classify them individually. Since feeding a 26×26 image to the network resulted in a single output prediction, patches of 26×26 were sampled from the synthetic training set. The patches were created by considering each digit in the image as a centralised object in the patch and then adding a random offset to the patch location ranging from -5 to $+5$ pixels. Additionally, patches labeled as "Background" was extracted by random sampling from the image, rejecting proposed patches with more than 20% overlap with any digit bounding box. For each image in the synthetic data set, 5 patches were extracted, resulting in a 3:7 ratio between the "Background" and "Object" classes across the training, validation and test data set with 45000, 5000 and 5000 patches respectively.
The network was trained using a mini-batch size of $m = 50$ and learning rate $\alpha = 0.1$ for a total of 12 epochs. The classifier and bounding box regressor were trained simultaneously with stochastic gradient descent, using two different cost functions. The cost function used for the classifier was the cross-entropy cost function,

$$C_c(\mathbf{p}, \hat{\mathbf{p}}) = \sum_{i=1}^{11} p_i \log \hat{p}_i, \tag{3.1}$$

where $\hat{\mathbf{p}}$ is the classifier output and $\mathbf{p}$ the corresponding label. For the bounding box regressor, a "smooth L1" cost function was used given by

$$C_b(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{4} \sum_{i=1}^{4} \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2 & \text{if } |y_i - \hat{y}_i| < 1 \\ |y_i - \hat{y}_i| - \frac{1}{2} & \text{otherwise ,} \end{cases} \tag{3.2}$$

where $\hat{\mathbf{y}}$ is the regressor output and $\mathbf{y}$ the corresponding label given in pixel coordinates. To even out the resulting gradients from these two functions, they were weighted 10:1 in the combined cost function used for training

$$C(\mathbf{p}, \mathbf{y}, \hat{\mathbf{p}}, \hat{\mathbf{y}}) = C_c(\mathbf{p}, \hat{\mathbf{p}}) + \frac{1}{10} C_b(\mathbf{y}, \hat{\mathbf{y}}). \tag{3.3}$$

3.1.3.2 Inference

Feeding the 72×72 sized images through the network resulted in a spatial grid of predictions. Each of these predictions was associated with a receptive field of the input image. However, since there were no receptive fields extending beyond the image edge, none of the output predictions could have a corresponding receptive field which was centralised at the edges of the image. To solve this issue, the input to the network was zero padded with 11 pixels on the left, right, lower and upper edge. The result was that the edge predictions now had a receptive field centralised to the 4×4 region aligned with the edge of the image, thus providing a 18×18 grid of predictions covering the *entire* image.
For inference, the output of the bounding box regressor also needed adjustments. As mentioned above, each prediction and corresponding bounding box was associated with a receptive field in the image. Thus, the pixel coordinates produced by the regressor referred to the top left corner of the associated receptive field. To account for this, all output coordinates from the regressor were transformed to the reference frame of the input image top left corner by adding the coordinates to their respective receptive field.
Finally, boxes with less than 0.95 probability of belonging to any of the digit classes were suppressed, resulting in a final set of output predictions, where each bounding box was associated with a digit.

3.2 Object Detection

The main task of this thesis was to be able to detect cars, pedestrians and cyclists, in a mono-camera image captured from a camera, mounted on the front of a vehicle capturing the traffic scene ahead. The detection was to be achieved by predicting a bounding box around the target object and additionally, also estimating the range to the object. The following sections explain the data set, model architecture and implementation details concerning the detection task.

3.2.1 KITTI data set

The KITTI object detection benchmark consists of 7481 training images and 7518 test images. The total number of objects in the training set is approximately 52,000, of which cars account for over 50 %. The images were obtained from sequences captured at different locations and different traffic conditions. A key difficulty inherent to the data set is that there are numerous vehicles, which are distant relative to the ego-vehicle, and thus quite small in the image (roughly 40 pixels in height

and width). Furthermore, the urban environment contains numerous parked vehicles along the side of the road, and a challenge is to distinguish them and achieve instance-detection. According to the KITTI vision benchmark suite [8], the images were acquired at a resolution of 1392×512 pixels. However, the images were then rectified and corrected for distortions and ambiguous regions, and the images available in the training and test set are respectively smaller.
The images in the training set are annotated with bounding boxes for cars, pedestrians and cyclists. The bounding boxes are expressed as pixel coordinates with the upper left corner of the image as reference. Since an exclusive validation set is not available for the object detection benchmark, 500 images from the training set were set aside and used as a validation set to monitor training efficiency, and decide when to stop training.

3.2.2 Model architecture

The custom architecture consisted of a version of the residual network "ResNet" [10], pre-trained on the ImageNet classification challenge. Residual networks can be constructed to be deep with many stacked convolution layers, and different pre-trained versions are available, consisting 18, 34, 50, 101, 152 and 200 layers. The architecture used for detection task consisted of the 34-layer ResNet, where the final layers were removed to form a feature extractor. The output of the feature extractor was fed to the "heads", which constitute of;

- A classifier with softmax activation function. The classifier outputs classification scores for "Car", "Pedestrian", "Cyclist" and "Background" for each output pixel.
- A bounding box regressor without an activation function, outputting 4 real numbers encoding the upper-left and lower-right pixel coordinates of a bounding box.
- A range regressor without an activation function. The output of this head was set to predict the distance to any classified object, expressed in meters. Since the main objective was to predict classified bounding boxes, this head was trained separately after the complete training process of the classifier and bounding-box regressor. Hence, the rest of the network except for the range regressor head was frozen and not trained. The range regressor was added out of curiosity, to see weather the features extracted could be used to regress for other information about objects.

The output of the respective "heads" was upsampled by a factor of 8 using the tiling operation. This was achieved by extending the feature map of the heads by a factor of 64 and then applying tiling to form an 8×8 spatial grid for each input pixel to the tiling layer. The architecture is illustrated end-to-end in figure 3.4.

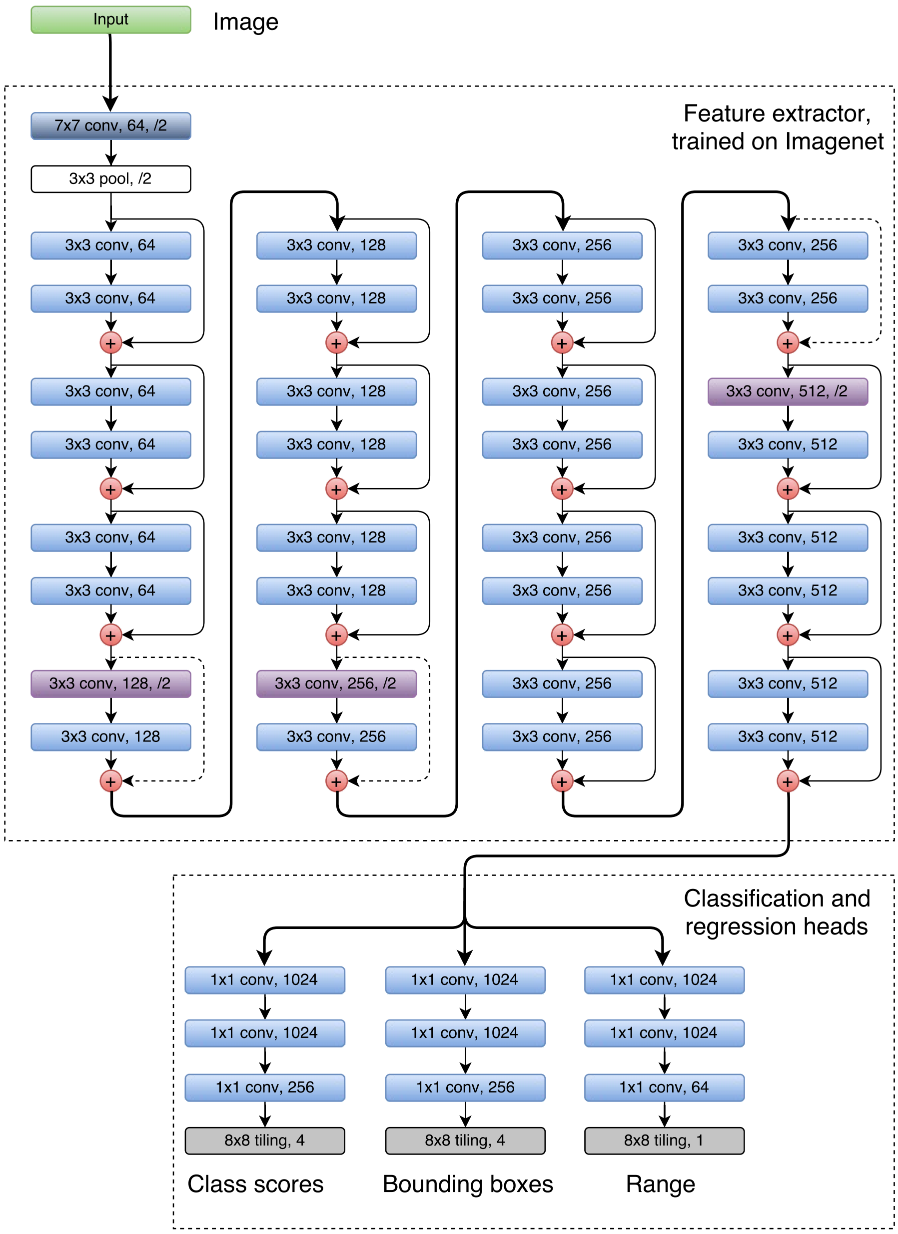


Figure 3.4: Model architecture for the KITTI object detection implementation. In addition to the convolution and pooling layers, batch normalisation and ReLUs are applied after each convolution layer. The dashed lines represent skip connections using 1×1 convolutions with stride 2 used to achieve appropriate downsampling.

3.2.3 Implementation details

The motivation for the approach to detection originated from the implementations in [12] and [11]. Since the feature extractor down-sampled the image by a factor of 32, an upsampling operation through tiling was introduced such that the effective stride of the end to end classifier is 4. This means that the classifier was capable of producing predictions of resolution 4×4 on the input image.
The pre-trained ResNet model was trained on the ImageNet data set. Hence, the images to be used for training or testing from the KITTI data set, had to be normalised

using the same normalisation parameters of the pre-trained network. For ResNet-34 [10], the mean to be subtracted from the input RGB channels is $\{0.485, 0.456, 0.406\}$ and the standard deviation to be adjusted for $\{0.229, 0.224, 0.225\}$.

3.2.3.1 Training

The approach to training consisted of constructing masks corresponding to the resolution of dense predictions that the network was designed to make, and train in an end-to-end manner. Crops of size 224×224 were extracted from the large KITTI images and masks of size 56×56 were created corresponding to effective stride of the network. The masks for training were constructed by considering the object bounding box labels, and shrinking them to 20% of their original height and width. Some examples of the crops and corresponding masks are presented in figure 3.5.

Figure 3.5: Examples of the crops and the training masks used for the object detection task. The masks show the ”Grey-zone” class (marked in red) on the boundary of the object mask and background. Far away objects tend to have the ”Grey-zone” class associated to them since they are meant to be ignored while training.

When there was an object occluded by another object, then the mask label was constructed to contain the object class of the object closer to the ego vehicle. The crops were constructed such that 70% of them belonged to centred objects and 30% were random patches extracted from the image, which could in turn contain objects or be pure background. The class of the object-centered crops were drawn such that the ratios of “Car”, “Pedestrian” and “Cyclist” were roughly equal. Furthermore, the object-centred crops were created to have slight offsets from the centre of the crop by adding or subtracting a random number of pixels drawn uniformly in the range -20 to 20 pixels, to the coordinates of the crop to be extracted. This is a data augmentation technique adopted to improve the robustness of the network [11]. The KITTI data provides the occlusion status of objects, and the object-centred crops were always ensured to be non-occluded. Another aspect of the KITTI data is that it contains a “Grey-zone” class which consists of far-away objects, and such regions in the test set are ignored while evaluating. Hence, by assigning those regions in the training set with a “Grey-zone” class, hard negative training on those ambiguous regions was avoided.
The technique of using the “Grey-zone” class on the boundaries of objects was also adopted. The mask corresponding to an object had a 1 pixel wide grey-zone surrounding it, separating the object mask area from the background. The intuition behind adopting this strategy was that vehicles which are very close to each other could be distinguished better.

Labels for the bounding box regressor were constructed to have a similar structure as the mask used for classification. Figure 3.6 illustrates how the original coordinates of the bounding box were encoded as relative coordinates from the pixel location. Pixels outside of the mask region were encoded as $x'_1 = y'_1 = x'_2 = y'_2 = 0$ and ignored during training by suppressing the output of the bounding box regressor at these locations.

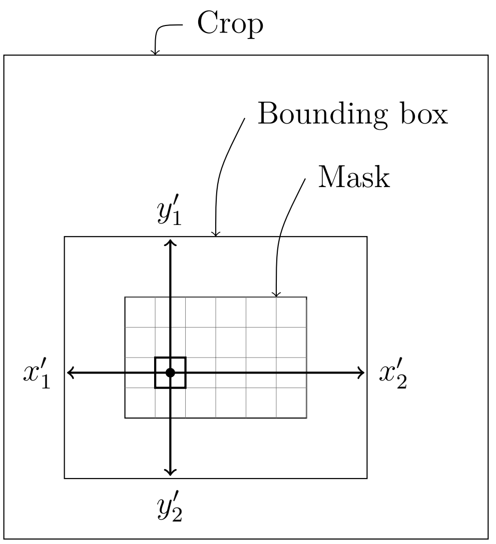


Figure 3.6: Illustration of how bounding box coordinates are encoded in the bouding box masks.

L2 cost functions were used for both classifier and bounding box regressor, given by

$$\begin{aligned} C_c(\mathbf{p}, \hat{\mathbf{p}}) &= \frac{1}{4}\sum_{i=1}^{4}(p_i - \hat{p}_i)^2 \\ C_b(\mathbf{y}, \hat{\mathbf{y}}) &= \frac{1}{4}\sum_{i=1}^{4}(y_i - \hat{y}_i)^2 , \end{aligned} \tag{3.4}$$

where $\hat{\mathbf{p}}$ and $\hat{\mathbf{y}}$ are outputs from classifier and regressor corresponding to labels $\mathbf{p}$ and $\mathbf{y}$. Since the output of the classifier ranges from 0 to 1, and the regressor output could vary between roughly 5 to 200 pixels, the cost functions were weighted respectively to form a combined cost function, given as

$$C(\mathbf{p}^{(1:N_p)}, \mathbf{y}^{(1:N_p)}, \hat{\mathbf{p}}^{(1:N_p)}, \hat{\mathbf{y}}^{(1:N_p)}) = \sum_{k=1}^{N_p}\left(w_c C_c(\mathbf{p}^{(k)}, \hat{\mathbf{p}}^{(k)}) + \frac{w_b}{N_p} C_b(\mathbf{y}^{(k)}, \hat{\mathbf{y}}^{(k)})\right), \tag{3.5}$$

where $w_c = 0.001$ and $w_c = 0.01$ are constants and $N_p = 56 \cdot 56$ is the number of output pixels.
The mini-batch size was set to $m = 10$. Transfer learning was applied by first allowing the gradients to propagate only through the classifier and regressor "heads". A learning rate of $\alpha = 0.01$ was used during this phase of training. After 6×10^4 mini-batches, gradients were allowed to propagate through the entire network, and

after a total of 10×10^5 mini-batches, the learning rate was reduced to $\alpha = 0.001$. The network was then trained additionally for 4×10^4 mini-batches.
The range regressor was trained after the training of the classifier and bounding box regressor. Much like the mask of coordinates used as label for the bounding box regressor, the range regressor was trained with a mask encoded with the distance to objects in meters. Also, pixels outside the mask region were encoded 0 and ignored during training. Similarly, an L2 cost function was used, given by

$$C_r(r^{(1:N_p)}, \hat{r}^{(1:N_p)}) = \frac{1}{N_{est}} \sum_{k=1}^{N_p} \left(r^{(k)} - \hat{r}^{(k)} \right)^2, \tag{3.6}$$

where N_{est} is the number of range estimates to be predicted. Thus, the cost function is a direct measure of the average squared deviance from the true distances.
The range regressor was trained with a mini-batch size of $m = 10$ and learning rate $\alpha = 0.01$ using 2×10^4 mini-batches, allowing the gradients to only propagate through the regression head to leave the state of classifier and bounding box regressor unaffected.

3.2.3.2 Inference

Figure 3.7 presents a flowchart describing the process of inference. The output from the network is a classification "heat map", and for each pixel of the heat map, the network also predicted bounding box coordinates. Since there is a one-to-one correspondence of the class predictions and bounding boxes, the classification scores from the heat map was used to suppress bounding boxes below a threshold of 80% probability of belonging to any of the object classes. By setting an appropriate threshold, more confident predictions are retained and the number of predictions to consider becomes less in number.

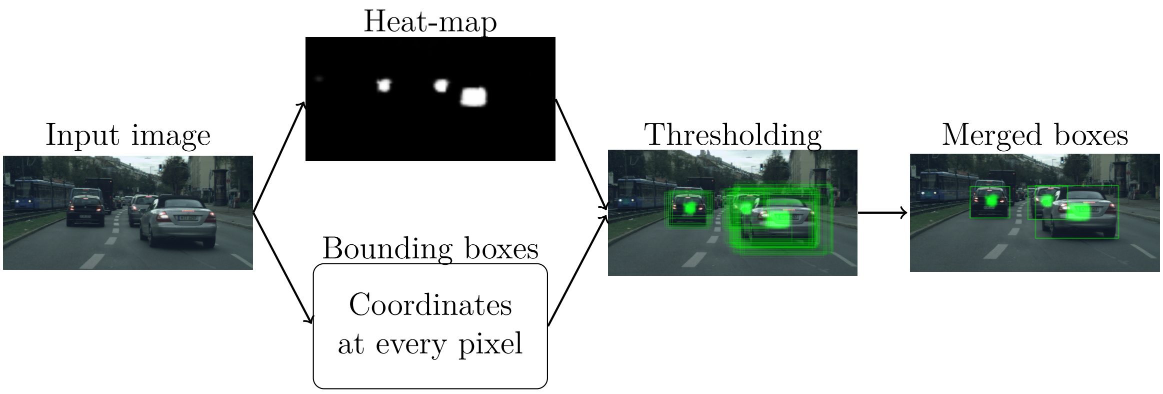


Figure 3.7: Flowchart of the inference process to create bounding boxes with corresponding class.

While thresholding reduces the amount of bounding boxes for a given test image, there are still several predicted bounding boxes per object. To get a single bounding box per object, clustering techniques are required. To this purpose, two different approaches were tried; *non-max suppression* and OpenCVs `groupRectangles` [2].

- The approach using non-max suppression orders all the bounding boxes for a given class by their classification score. Starting from the box with the highest score, bounding boxes are suppressed if they have an intersection-over-union with any another bounding box above 0.3. Technically, the final boxes originates from isolated output pixels and thus, the algorithm does not merge any bounding boxes but suppresses the majority of them.
- The second approach was using OpenCVs `groupRectangles` [2]. This is a cascade classifier which merges bounding boxes based on their overlap, controlled by a tunable parameter $\epsilon \in [0, \infty)$. After some manual inspection of results given different choices of ϵ, this parameter was set to $\epsilon = 0.25$. In addition, `groupRectangles` suppresses isolated boxes to remove potential false-positives.

The output from the range regressor was included by using output from the same pixel locations as the bounding boxes used when merging. The final output is a set of bounding boxes with classification scores and range estimates.

3.3 Semantic Segmentation

Semantic Segmentation is a pixel-level classification task, that aims to give rich semantic information about the scene captured by the camera. In the detection approach presented in the previous section, an object is detected by a classifier and we are mostly interested in a few object classes such as "Car", "Pedestrian" and "Cyclist". A bounding box regressor is used to actually detect the objects of interest in the image. In contrast, semantic segmentation provides for a scene, image segmentation into many different classes. Hence, the background is no longer just considered as "Background", but divided into more informative classes such as "Road", "Sidewalk", "Nature", "Building", etc.
This section presents a case study describing three approaches from literature which consist of different CNN architectures and implementation methods used for semantic segmentation. Furthermore, one of the three case studies is replicated and trained on the Cityscapes data set to achieve segmentation of images captured in urban environment.

3.3.1 Cityscapes data set

The Cityscapes data set is a recently released large-scale data set with images captured by a camera mounted on the front of a car. It contains a mixed set of stereo video sequences recorded in street scenes from 50 different cities. The images are annotated pixel-wise so that each pixel is labeled as one of the classes presented in table 3.1.
The data set contains 5000 finely annotated images (presented in figure 3.8) and about 20000 coarsely annotated images. The coarse annotations were generated by polygonal labelling and thus do not clearly distinguish between objects and instances in a scene, while the fine annotations were fine-tuned to pixel-level. The images were acquired and provided at a resolution of 2048×1024 pixels. The finely annotated

data is divided into training, validation and testing sets, consisting of 2975, 500 and 1525 images.

Category	**Classes**
Flat	Road, Sidewalk, Parking†, Rail track†
Human	Person, Rider
Vehicle	Car, Truck, Bus, On rails, Motorcycle, Bicycle, Caravan†, Trailer†
Construction	Building, Wall, Fence, Guard rail†, Bridge†, Tunnel†
Object	Pole, Pole group†, Traffic sign, Traffic light
Nature	Vegetation, Terrain
Sky	Sky
Void	Ground†, Dynamic†, Static†

Table 3.1: The 30 classes included in the Cityscapes data set, divided into 8 categories. Classes marked ”†” are not included in the Cityscapes benchmark for eveluation and treated as ”Void”/”Grey-zone”.

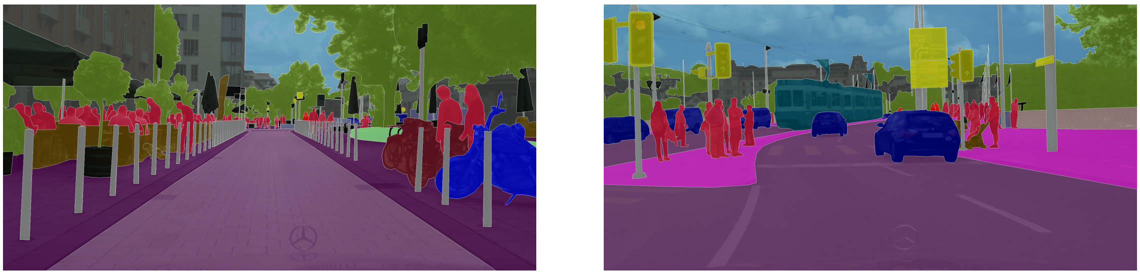


Figure 3.8: Examples of finely annotated images from the Cityscapes data set.

3.3.2 Case Studies

Three approaches to semantic segmentation were studied from previous work in literature. The approaches discuss different deep learning architectures which are presented as case studies.

- **Case Study A:** A “deconvolution” network based approach [18] that performs upsampling to reconstruct shapes of objects. The approach advocates a shape extractor that learns to perform dense predictions by reconstructing objects from a feature vector through repeated up-convolutions and max-unpooling operations. The architecture consists of convolutional and max-pooling layers extracted from the pre-trained VGG 16-layer model [24], followed by a mirrored version of the downsampling network, which uses the feature vectors as inputs and reconstructs the shapes of objects to perform dense predictions.
- **Case Study B:** An approach that uses Fully Convolutional Networks (FCNs) [17] to achieve dense predictions, by upsampling using a deconvolution layer. The deconvolution layer upsamples stride 32 predictions to the size of the original input in a single step. Different versions that combine predictions from the final layer and other strides are compared. FCN-8 combines stride

8 predictions and stride 32 predictions by upsampling the stride 8 predictions with a factor 4 and summing with the stride 32 predictions. Similarly FCN-16 combines stride 16 predictions and the stride 32 predictions by upsampling the stride 8 predictions with a factor 8 and summing with the stride 32 predictions.

- **Case Study C:** An approach that uses dilated convolutions to systematically aggregate multi-scale contextual information without losing resolution [33]. Introducing dilated convolutions to a CNN removes the need for pooling layers and strided convolutions to increase the receptive field of the network. By doing so, feature maps are never downsampled and the dimensions of the output is the same as the input which is mostly desirable for the task of semantic segmentation. This approach is also interesting given that networks using strided convolutions can be converted to dilated networks by removing the stride and increasing the dilations of subsequent convolution kernels accordingly [30].

Following the case studies, it was decided to replicate the deconvolution network approach. The prescribed architecture was modified from [18], by removing the last max-pooling and convolutional layers. The modified architecture is depicted in figure 3.9. The intention behind this is to compare the performance to the original architecture. Additionally, the performance of the replicated approach was compared to the other approaches, using the benchmark set by the authors of the approaches on the Cityscapes data set.

3.3.3 Implementation details

The architecture prescribed in [18] uses the 16 layer VGG network, pre-trained on the ImageNet. Hence, the Cityscapes training images were pre-processed using the normalisation parameters used when pre-training the VGG network. The network takes as input BGR images, where the mean subtracted from the BGR channels is $\{123.68, 128.779, 103.939\}$. Full-image training was employed, and to increase robustness and augment data, random horizontal flips were used on the input images with a 50% probability. Due to memory constraints of the GPU, training and inference on the original resolution of 2048×1024 was not possible. To circumvent this, the images were downsampled by a by a factor of 2. Further, the number of output classes was set to the 19 classes used by the Cityscapes benchmark, and one additional class as “Void” or “Grey-zone”.

The cost function used during training was the cross-entropy cost function

$$C(\mathbf{p}^{(1:N_p)}, \hat{\mathbf{p}}^{(1:N_p)}) = \frac{1}{N_p} \sum_{k=1}^{N_p} \sum_{i=1}^{20} c_i p_i^{(k)} \log \hat{p}_i^{(k)}, \tag{3.7}$$

using a set of weights $c_{1:20}$ associated with the different classes. The weights are roughly inverse proportional to the occurrence of the particular class and were introduced to the cost function to adjust for the uneven distribution of classes. The weights are summarised in table 3.2 and were roughly estimated using statistics from 100 images in the training data set. The unnormalized weights $\hat{c}_i$ were normalised in the implementation using a constant C so that $\frac{1}{C} \sum_i \hat{c}_i = \sum_i c_i = 1$.

The mini-batch size was set to $m = 2$ due to the limited memory on the GPU. The deconvolution part of the network was trained alone using 2×10^4 mini-batches with

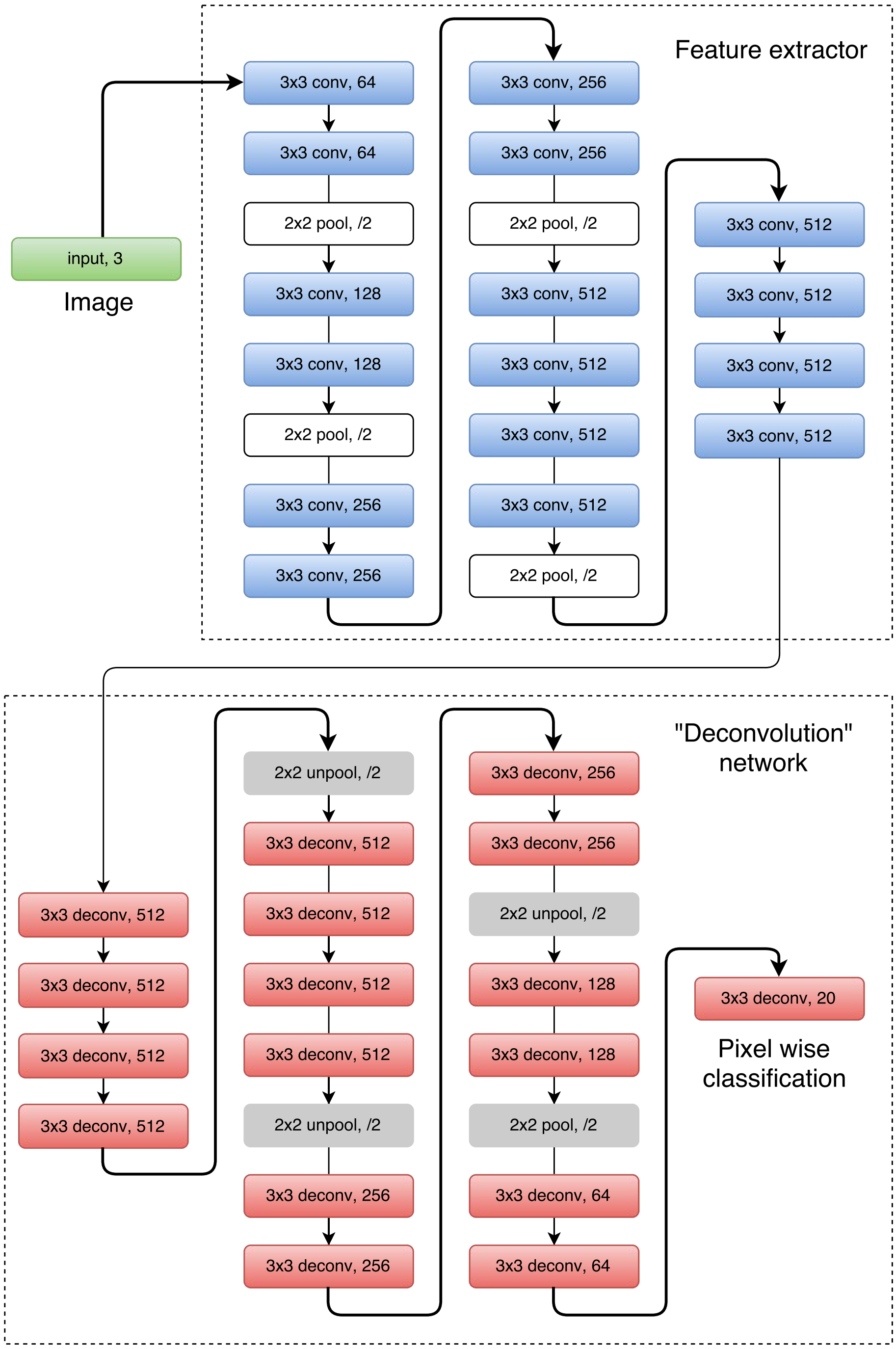


Figure 3.9: Network architecture used for the segmentation task, consisting of a pre-trained feature extractor and a shape-extractor network that upsamples feature vectors and reconstructs shapes of objects. Between all convolution and deconvolution layers are ReLUs and batch normalization layers.

learning rate $\alpha = 0.01$. After this phase, the network was trained using 2×10^4 mini-batches additionally, were the gradients were allowed to propagate through the entire network and the learning rate was lowered to $\alpha = 0.001$.
For comparison, the original architecture of the deconvolution approach [18] was implemented and trained in the same fashion.

Class	**Weight** $\hat{c}_i$	**Class**	**Weight** $\hat{c}_i$
Bicycle	108	Sidewalk	6
Building	2	Sky	9
Bus	480	Terrain	36
Car	6	Traffic light	185
Fence	47	Traffic sign	70
Motorcycle	371	Train	128
Person	37	Truck	212
Pole	31	Vegetation	2
Rider	300	Void/Grey-zone	0
Road	1	Wall	55

Table 3.2: Unnormalised weights $\hat{c}_i$ for balancing the labels of the Cityscapes dataset. Note that the class ”Void”/”Grey-zone” has a weight of 0 and thus ignored in training.

4
Results

This chapter presents the results from the different implementations. Graphs showing the cost during training and validation are presented for the MNIST digit detection and the KITTI object detection tasks. Qualitative image results and results from benchmarking are also presented for the KITTI object detection task and the Cityscapes semantic segmentation task.

4.1 MNIST digit detection

This section presents results of training the network used for MNIST digit detection, which was evaluated on a test set of 5000 patches. To present qualitative results and justify the detection through classification aproach, the network was also applied to a larger image with multiple digits.

4.1.1 Classification

The process of training the network for classification is presented in figure 4.1. The figure shows the error percentage of the network predictions from the training data set and the validation data set, with and without batch normalisation. Minimum error on the validation set occurs after 10 and 12 epochs respectively, at which time the state of the network is saved. Evaluating the network on the test set results in 3.10% and 3.22% prediction error with and without batch normalisation respectively. While the networks give similar results after training, the descent of the validation set prediction error is faster and more smooth when using batch normalisation.

Figure 4.2 shows the classifier output when feeding a 72×72 sized image with scattered MNIST digits using the network trained with batch normalisation. Note that the network gives strong confidence for the absence of digits, properly assigning blank regions of the input image as “Background”. While the distinction of the individual digit classes are correct for most locations, the network struggles to make a clear prediction close to the edges between digit and background. Results from the network trained without batch normalisation shows very similar characteristics.

4.1.2 Detection

The cost of the bounding box regressor evaluated on the training set and validation set respectively is presented in figure 4.3. The figure shows a similar descent as the

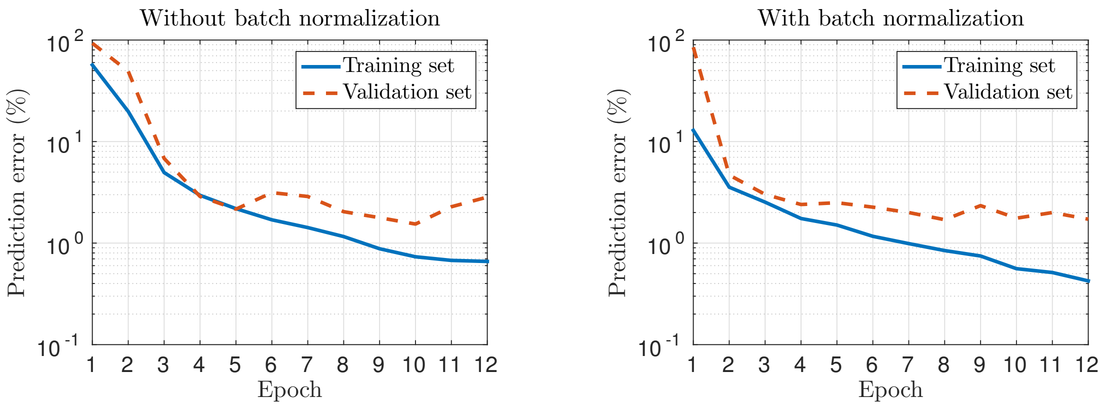


Figure 4.1: Prediction error plotted against epoch number when training with and without batch normalisation.

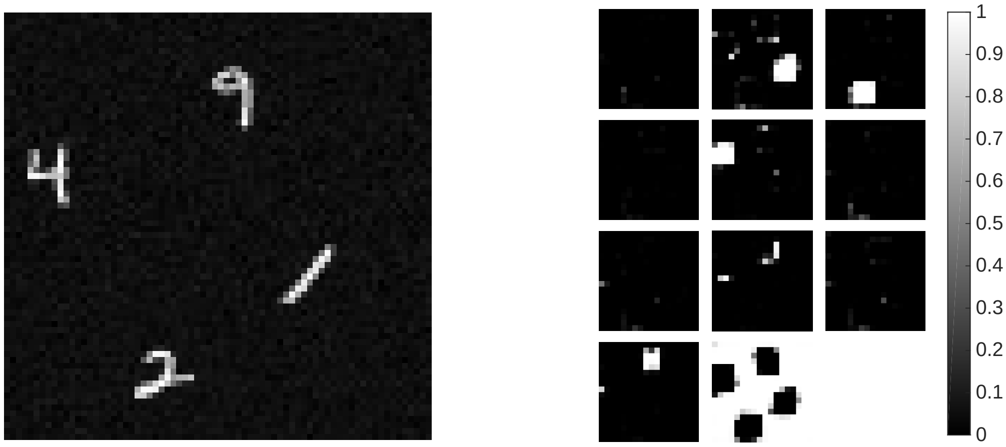


Figure 4.2: 72×72 sized image with 4 MNIST digits scattered at random locations (left) and resulting prediction heat maps from the network (right). The heat maps are sorted from left to right, top to bottom corresponding to the classes $0, ..., 9$ and "Background".

classification error (figure 4.1), where the validation set cost reduces faster when using batch normalisation.

Figure 4.4 presents the combined result of classifier and regressor when feeding a 72x72 sized image. The plotted bounding boxes correspond to locations where the classifier predicts more than 0.95 confidence of a digit being present. Note that no clustering technique has been implemented and thus there are a lot of bounding boxes corresponding to the same digit. However, the boxes are separated in distinct

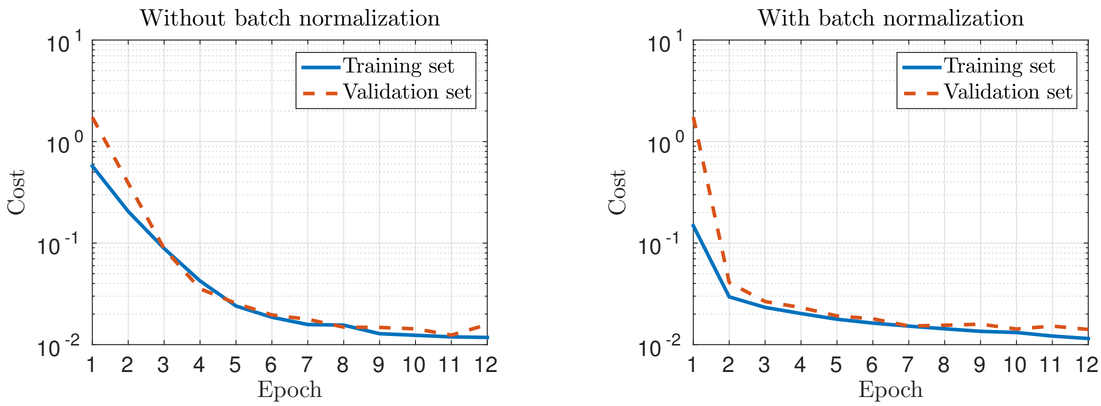


Figure 4.3: Cost of bounding box regressor plotted against epoch number when training with and without batch normalisation.

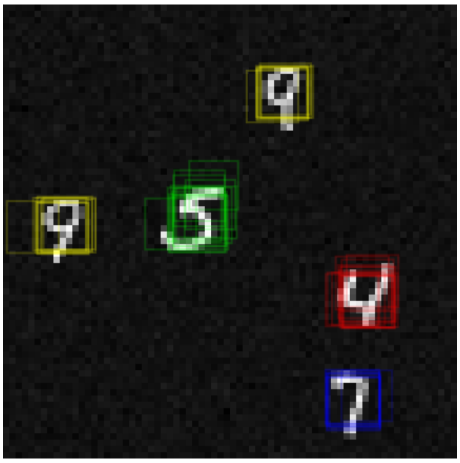


Figure 4.4: Bounding boxes corresponding to confidence above 0.95 for classes "4" (red), "5" (green), "7" (blue) and "9" (yellow).

groups around the digits which is promising for a potential clustering algorithm.

4.2 KITTI object detection

This section presents the results from the object detection implementation on the KITTI data set. The implementation was tested on the KITTI object detection test set both for qualitative measures and using the KITTI object detection benchmark. To explore the the capabilities of the method even further, the network was also applied to some images from the Cityscapes data set.

4.2.1 Training

Figure 4.5 shows the training process of the network for classification and bounding box regression. The average cost over 500 mini-batches from the training set and validation set respectively is plotted against the number of mini-batches used for training. The figure shows how the cost of the validation set starts saturating after roughly 5×10^4 mini-batches and then decreases rapidly after 6×10^4 mini-batches once gradients are allowed to propagate through the entire network. The trend of decreasing cost of the validation set stops after roughly 11×10^4 mini-batches, while the cost of the training set decreases further.

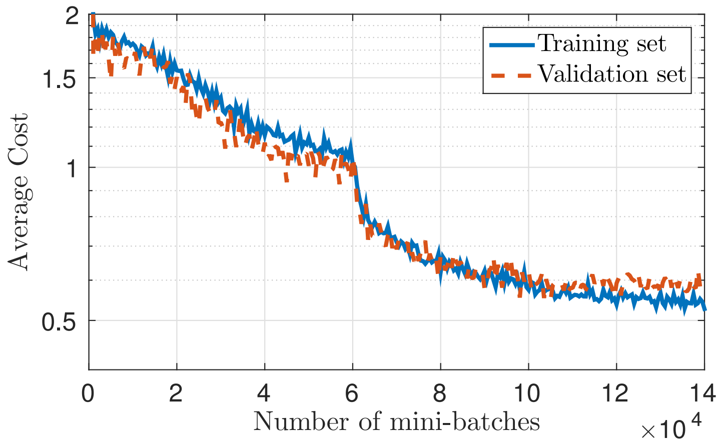


Figure 4.5: Average cost from classifier and bounding box regressor using 500 mini-batches plotted against the number of mini-batches used for training.

The process of training the range regressor is presented in figure 4.6. The range regressor "head" is trained alone without propagating gradients through the feature extractor, and thus the average cost does not show the same characteristics as the cost from classifier and bounding box regressor. The cost of the validation set saturates after approximately 1.5×10^4 mini-batches, reaching an average squared distance error of roughly 10 m^2.

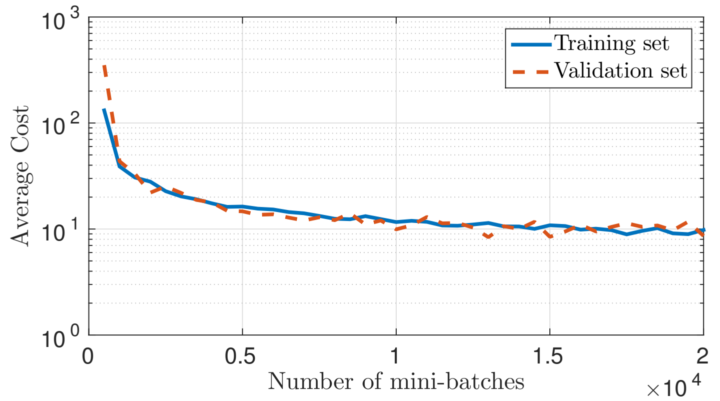


Figure 4.6: Average cost from range regressor using 500 mini-batches plotted against the number of mini-batches used for training.

4.2.2 Qualitative results

Figure 4.7 presents a sample of images from the test set with the classification score of the "Car" class projected on top and the resulting bounding boxes after suppression and merging using non-maximum suppression. The figure shows the network is successfully able to detect objects of varying scale, lighting conditions and orientation. The network also manages to detect partially occluded objects, resulting in overlapping bounding boxes.

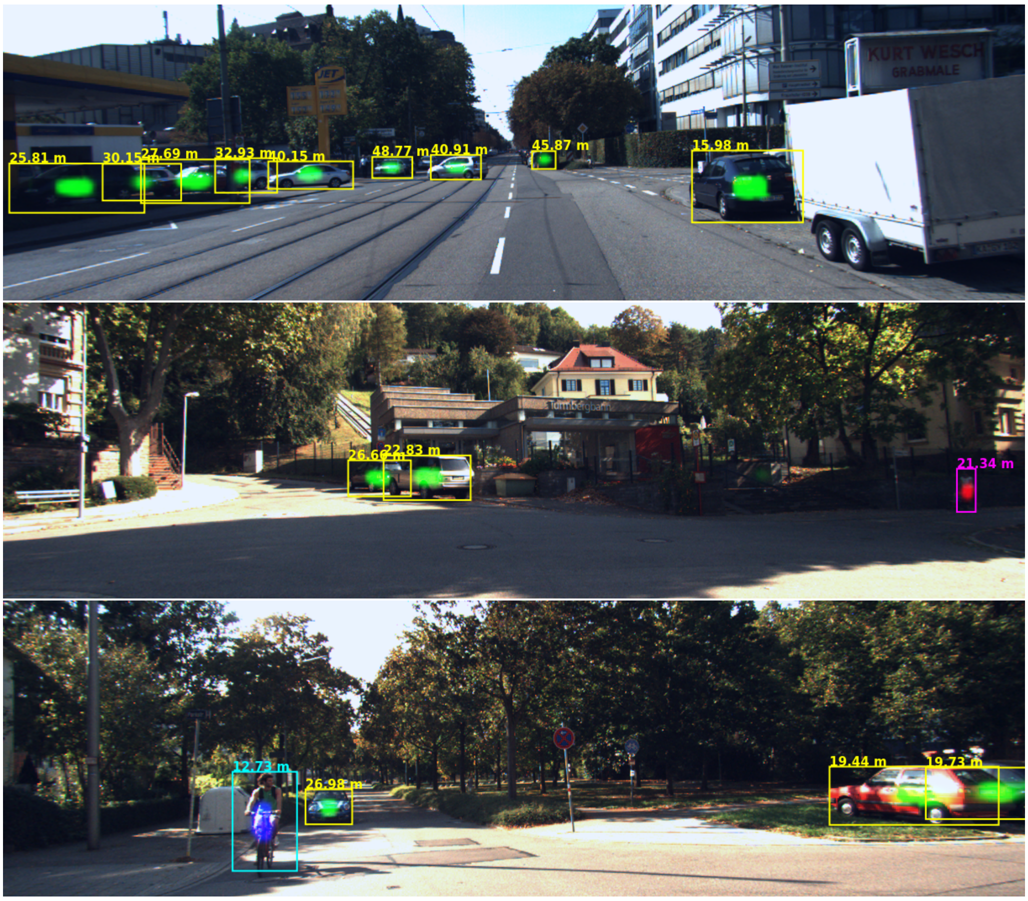


Figure 4.7: Qualitative results of the detection network for the different classes from the KITTI test set. The markers in the middle of objects are the classifier predictions scaled up to match the resolution of the image. The bounding boxes also show the range predictions to the respective objects.

Figure 4.8 shows the network successfully detecting cars, pedestrians and cyclists on a sample of images from the Cityscapes data set. Furthermore, the range to the objects is also predicted, as highlighted in a more closer inspection of one of the test images in figure 4.9. The range predictions appear qualitatively promising, both for test images from the KITTI data set as well from the Cityscapes data set.
Figure 4.10 present qualitative results when using the two considered merging approaches. Results show `groupRectangles` and non-maximum suppression produce similar results.

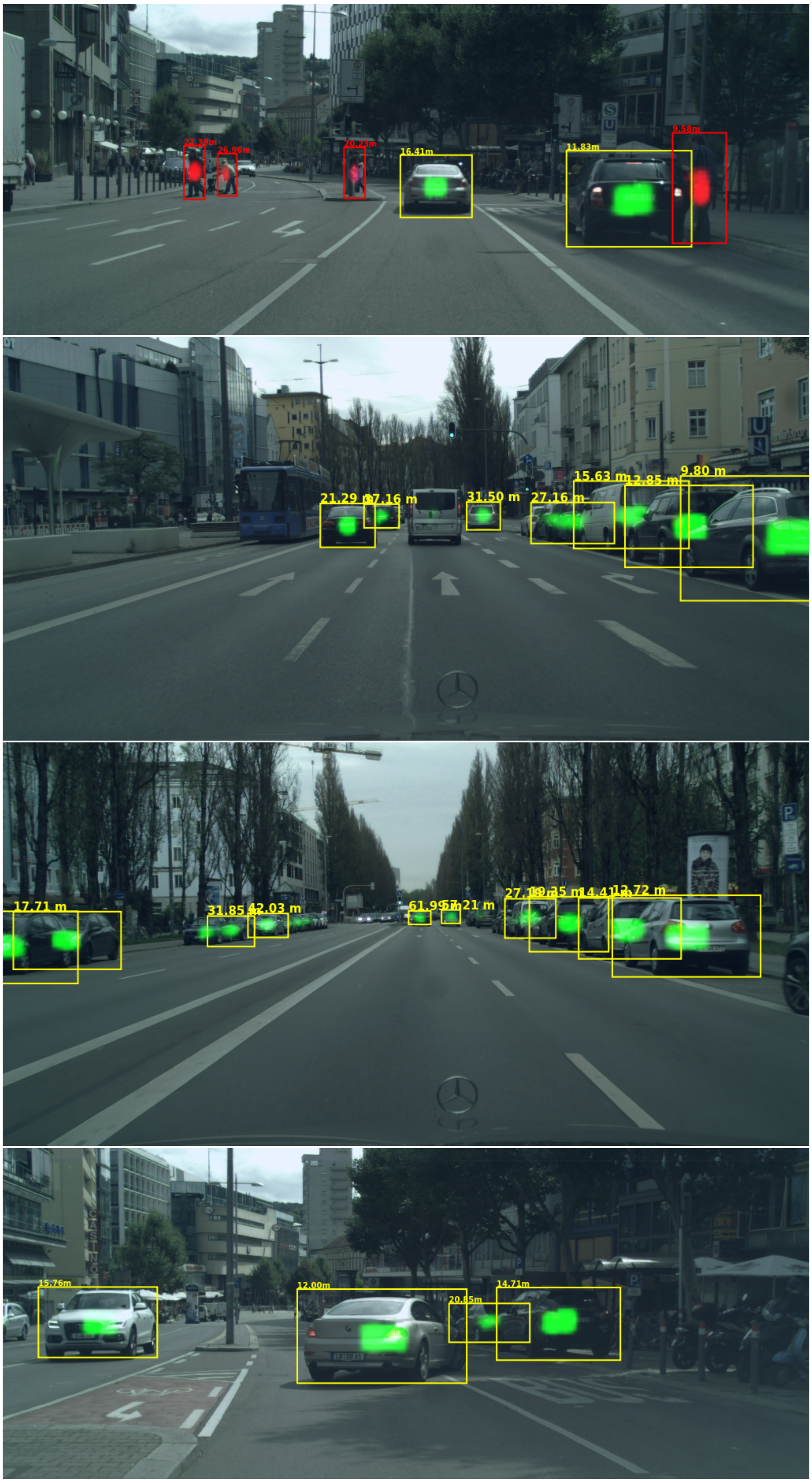


Figure 4.8: Qualitative results of the detection network for "Car", "Pedestrian" and "Cyclist" classes evaluated on test images from the Cityscapes data set. The markers in the middle of objects are the classifier predictions scaled up to match the resolution of the image.

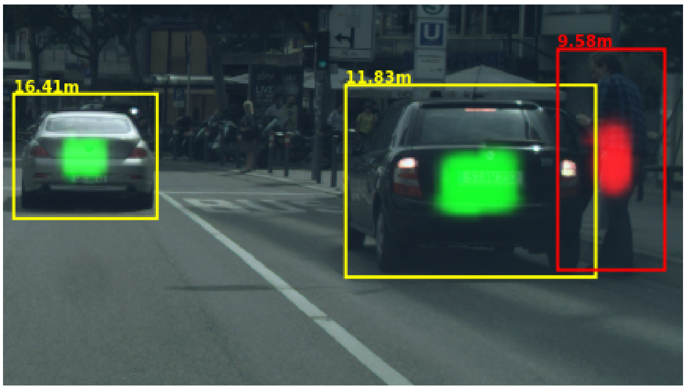


Figure 4.9: Closer inspection of the range predictions for a test image from the Cityscapes data set.

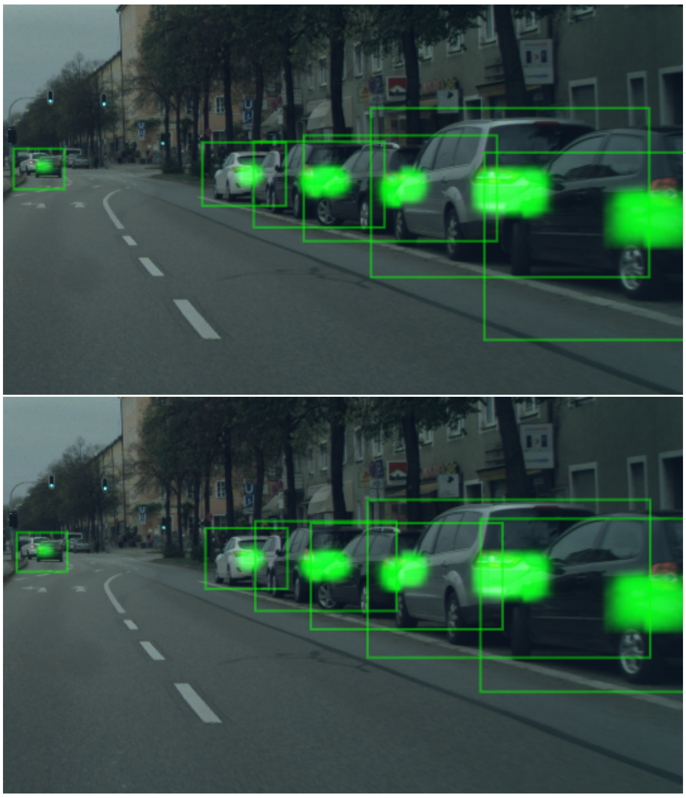


Figure 4.10: Example of non-maximum suppression (top) and merging using `groupRectangles` (bottom). The two methods of merging bounding boxes yield similar results, where the merging using `groupRectangles` is marginally better.

4.2.3 Benchmarking

To present a quantitative result on the performance and compare the approach with other benchmarks, the network was evaluated on the KITTI object detection benchmark. The benchmark evaluates the network predictions on 7520 annotated test images, using the PASCAL criterion [5]. According to the criterion, the intersection-over-union of predicted bounding boxes and ground truth labels has to be larger than 70% for an object to be detected correctly. In addition, the benchmark has three difficulties; “Easy”, “Moderate” and “Hard” with different specifications for objects to be included in the evaluation, summarised in table 4.1.

	Easy	**Moderate**	**Hard**
Min. bounding box height	40 pixels	25 pixels	25 pixels
Max. occlusion level	Fully visible	Partly occluded	Difficult to see
Max. truncation	15%	30%	50%

Table 4.1: Specifications for objects to be included in the three difficulties in the KITTI object detection benchmark.

The result of the benchmark is the precision $P = \frac{T_p}{T_p+F_p}$ and recall $R = \frac{T_p}{T_p+F_n}$, where T_p is the number of true-positive, F_p the the number of false-positive and F_n is the number of false-negative. With the bounding box class scores, these metrics are evaluated using samples above a given score, resulting in a precision-recall curves presented in figure 4.11. As seen in the figure, an increased difficulty results in lower recall, i.e. less objects being detected, while the precision remains similar. This suggests higher difficulty results in fewer objects being detected amongst the objects fulfilling the difficulty specifications.

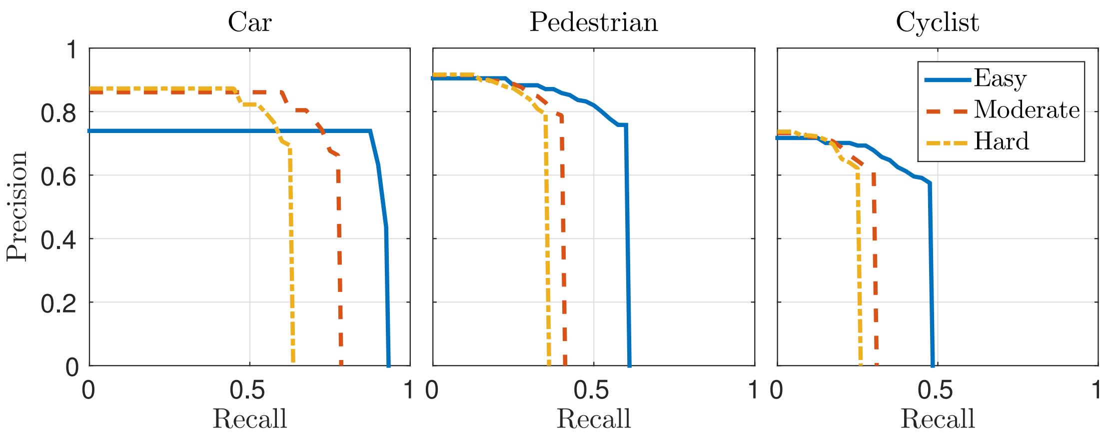


Figure 4.11: Precision-recall curves for the ”Car”, ”Pedestrian” and ”Cyclist” classes on the KITTI object detection benchmark given three different difficulties.

The average precision on the KITTI object detection benchmark is presented in table 4.2. Note that the overall detection accuracy for cars is much higher compared to pedestrians and cyclists.

Benchmark	**Easy**	**Moderate**	**Hard**
Car	66.24 %	61.83 %	53.57 %
Pedestrian	54.85 %	39.74 %	32.38 %
Cyclist	31.14 %	25.00 %	19.21 %

Table 4.2: Average precisions on the KITTI object detection benchmark given three different difficulties.

4.3 Cityscapes Semantic Segmentation

This section presents the results of evaluating the example case study model on the Cityscapes benchmark. The results present the qualitative performance on the test images and quantitative performance assessment based on the metrics from the benchmark.

4.3.1 Qualitative results

Figure 4.12 presents a set of images from the Cityscapes test set with predicted pixel-level classifications by the network used for semantic segmentation. Pixel classes has been associated as the highest class score from the softmax output and encoded as a specific color for each class. The resulting segmentation correctly distinguish between road and sidewalk, segments cars and pedestrians from the background surroundings, and is able to detect traffic signs and traffic lights. The network struggles to make clear and stable classifications of objects which are small and far away from the camera.

4.3.2 Benchmarking

To quantitatively evaluate the performance of the model, evalution on the Cityscapes pixel-level benchmark was done. According to the benchmark [4], the Jaccard index or PASCAL VOC intersection-over-union (IoU) metric is used to assess the performance of the model. The IoU is defined as

$$IoU = \frac{T_p}{T_p + F_p + F_n}.$$

Furthermore, the benchmark also evaluates the semantic labelling using an instance-level intersection-over-union metric

$$iIoU = \frac{iT_p}{iT_p + F_p + iF_n},$$

where iT_p and iF_n are computed by weighting the contribution of each pixel by the ratio of the class average instance size to the size of the respective ground truth instance.

The evaluation requires pixel labels at the original resolution. Hence, for evaluation, nearest-neighbour upsampling was used to increase the scale of the predictions by a factor of 2. Clearly, this is disadvantageous, and a more rigorous approach would have been to evaluate the network on parts of the image, and stitch them to create the final predictions at the original resolution. Since the aim was to evaluate the model to get a rough idea of the performance, the nearest-neighbour upsampling approach was deemed satisfactory. It also allowed for more discussion on the model performance when trained on lower resolutions.

Table 4.3 presents the benchmark results of the replicated more lightweight deconvolution network approach (A) and the original deconvolution architecture (A*)

Figure 4.12: Qualitative results of the segmentation network on test images from the Cityscapes data set. The different colors which are superimposed on the original images represent the different semantic labels extracted by the network. The class–color correspondence is assumed to be self explanatory.

compared with the other two case studies (B & C). The table shows the intersection-over-union for each of the semantic classes, and also the instance intersection-over-union for classes where instances are more relevant. Additionally, table 4.4 presents the results of the case study models on different semantic categories. It is to be noted that the results for the case studies B & C were directly obtained from the benchmarks set by the authors of the approaches on the Cityscapes data set.

	(A)		**(A*)**		**(B)**		**(C)**	
Class	IoU	$iIoU$	IoU	$iIoU$	IoU	$iIoU$	IoU	$iIoU$
Bicycle	46.3	37.2	46.7	30.8	66.8	49.6	66.0	49.1
Sidewalk	64.7	-	64.2	-	78.4	-	79.2	-
Building	80.2	-	79.9	-	89.2	-	89.9	-
Sky	91.5	-	89.9	-	93.9	-	93.7	-
Bus	36.6	29.9	51.2	24.7	48.6	30.8	53.4	32.7
Terrain	59.19	-	55.3	-	69.3	-	69.4	-
Car	87.8	68.9	86.5	70.9	92.6	83.9	93.3	85.8
Traffic light	26.8	-	26.7	-	60.1	-	58.6	-
Fence	29.5	-	29.6	-	44.2	-	47.6	-
Traffic sign	41.5	-	36.6	-	65.0	-	65.2	-
Motorcycle	34.9	23.2	36.9	17.0	51.6	31.1	52.2	28.0
Train	24.7	13.4	42.8	21.3	46.5	26.7	47.7	27.6
Person	61.6	44.0	61.1	43.3	77.1	55.9	78.9	56.3
Truck	20.5	16.1	39.6	15.8	35.3	22.2	45.5	21.8
Pole	33.5	-	27.5	-	47.4	-	53.2	-
Vegetation	85.0	-	82.6	-	91.4	-	91.8	-
Rider	42.9	29.4	38.8	22.6	51.4	33.4	55.0	34.5
Wall	31.3	-	29.1	-	34.9	-	37.3	-
Road	93.89	-	94.0	-	97.4	-	97.6	-
Average	52.3	32.8	53.6	30.8	65.3	41.7	67.1	42.0

Table 4.3: Intersection-over-union (IoU) and instance intersection-over-union ($iIoU$) from the Cityscapes benchmark for different classes.

The results indicate that the deconvolution network approach shows promise compared to the other approaches. For all three approaches, it can be seen that there are easier classes and harder ones to segment. For example, “Road” and “Sky” are easier classes while “Truck” and “Train” were clearly harder to segment. Comparing results from the lightweight architecture with the original one (A*) also show how over-all, making the deconvolutional network slightly more lightweight has little effect on performance. Most notably, the performance accuracy of classes with large objects such as “Bus”, “Train” and “Truck” is better when using the original architecture.

	(A)		**(A*)**		**(B)**		**(C)**	
Categories	IoU	$iIoU$	IoU	$iIoU$	IoU	$iIoU$	IoU	$iIoU$
Flat	96.9	-	96.3	-	98.2	-	98.3	-
Human	62.8	48.4	60.6	45.4	78.6	58.0	79.8	58.3
Vehicle	85.3	76.0	84.0	71.6	91.3	82.3	91.8	83.9
Construction	81.5	-	80.1	-	89.6	-	90.2	-
Object	38.5	-	34.0	-	57.0	-	60.5	-
Nature	84.9	-	82.0	-	91.1	-	91.4	-
Sky	91.5	-	89.9	-	93.9	-	93.7	-
Average	77.3	62.2	75.3	58.5	85.7	70.1	86.5	71.1

Table 4.4: Intersection-over-union (IoU) and instance intersection-over-union ($iIoU$) from the Cityscapes benchmark for different categories.

5

Discussion

This chapter presents a detailed discussion of the different implementations carried out in the thesis. The following sections analyse the different approaches and its respective deep learning components. The discussions are formulated to question design choices and methodologies, and aims to provide the reader with a deeper understanding of the vast amount of research work pertaining to the area. The discussion also goes on to analyse the results obtained, finally leading up to present the scope for future work.

5.1 Detection Implementation

Object detection achieved through predicting bounding boxes gives information about the location of an object of interest in the image. It also gives information about the scale of the object, which in turn is determined by the object's size and its distance from the ego vehicle. Using bounding boxes for detection is a common approach to detection, and is an effective one since it also separates the instances of objects in the image.

5.1.1 Design Analysis

The choice of the model architecture for the detection task was inspired from previous work [11, 12]. An important consideration while designing a CNN architecture is the performance while inference. It is clear that the feature extractor is the more bulky part of the detection network pipeline. The feature extractor consists of pre-trained layers which were originally part of a classification network, trained on the ImageNet data set. Hence, an investigation into an appropriate choice for the feature extractor was made by comparing different pre-trained architectures in terms of their inference time and accuracy on the ImageNet validation set. Table 5.1 shows this comparison.
A noteworthy observation is that the Inception-v3 [29] architecture achieves the highest accuracy to time for inference ratio. The choice of using the ResNet-34 as a feature extractor in the model architecture was made, since it offers a good trade-off between time for inference and accuracy, and has a receptive field of 209×209 which agrees well with the objects to be detected in the KITTI data set.
A natural question is "How well does transfer-learning work?". To answer this, one can consider the amount of data that was used to train the pre-trained classification networks. The networks were trained on the ImageNet data set, which consists of

Model	Top-1 err (%)	Time (ms)
VGG	26.8	9
ResNet-18	30.4	17
ResNet-34	26.7	28
ResNet-101	22.4	71
Inception-v3	21.2	76

Table 5.1: The top-1 single-crop validation error on the Imagenet-2012 data set for the models and the time taken to pass an image of size 640×480 through their respective feature extractor layers.

approximately 14 million images. Compared to this, the amount of training data available from the KITTI data set is quite less. Hence, it makes a lot of sense to use layers from a pre-trained network that is good at extracting rich hierarchical information. If training data of such scale was available for the classes of interest such as cars, pedestrians and cyclists, then a network could be trained from scratch on the more relevant data. Furthermore, such a network could potentially be designed to be lightweight compared to the pre-trained feature extractors that we used in the thesis. This argument is supported by the work done in [12], where the network was trained on $17,000$ frames and $140,000$ objects. However, the frames from the data set were highly correlated, since most of the data was captured on a highway driving environment and thus a more generalisable model requires the data to contain a variety of scenes and objects.

5.1.2 Implementation Analysis

Detection through classification is an approach commonly observed in literature [11,12], however other approaches to detection also offer competitive results [19,31]. A main issue with the detection through classification approach is that multi-scale objects are not handled optimally. Typical examples are cars that are far away and are thus very small, and cars that are near and occupy a large region in the image. Since the network has a fixed receptive field, the information captured while trying to detect small-scale objects is very little compared to all the available information in the receptive field, while the information captured while trying to detect large-scale objects is only that of part of the object. Typical classification CNN pipelines achieve downsampling through pooling layers or convolutional strides. Downsampling is important since it reduces the computational overhead, whilst also incorporating increased invariance to small translations of objects in the image. However, it throws away lots of spatial information, which in turn can hurt the detection performance for multi-scale objects. To handle this, [31] developed scale-dependent pooling, which effectively looks at the scale of different regions of interest, and uses feature maps from the appropriate scale of the downsampling network. Intuitively, this makes a lot of sense and the exploitation of information from multiple-layers is a key aspect that makes this approach more powerful.
Another strategy is to use dilated convolution kernels in the convolution layers, thus avoid throwing away spatial information, although this approach needs to be

investigated further.
The implementation details affect the training process of the model and hence its overall performance. Until recently, deep convolutional networks relied on large amounts of data, which in turn called for data augmentation, followed by regularisation such as dropout [25]. With the introduction of batch-normalisation, the need for regularisation has been reduced, since according to [14], the normalising layer inherently has regularisation properties over a mini-batch. This has further reduced the need to augment data through techniques such as random flipping, color jitters and adding random noise. The implementation presented in this thesis does not use any data augmentation, and present a baseline performance evaluation using only the training images from the KITTI data set. Hence, a simple extension to improve the results, would be to augment data, especially for the “Pedestrian” and “Cyclist” classes which had fewer instances compared to “Car”.
It was also observed that the setting up of training data plays a crucial role in determining the performance of the model. For example, patch-wise training and full-image training are two methods of training observed in literature [17]. While the idea of full-image training seems less cumbersome and neater, patch-wise training has its advantages for a detection implementation. The most obvious advantage is efficiency, since object centred crops allow for more training examples of object classes and better control of the ratio of positive to negative examples whilst training.

5.1.3 Detection Accuracy and Range Estimation

The evaluation on the KITTI object detection benchmark showed that the Average Precision (AP) of the detection approach is competitive, compared to the state-of-the-art benchmarks [8]. It is noteworthy that the high precision is similar to state-of-the-art deep learning approaches, while the lower recall suggests scope for improvement. An immediate observation was the reduced performance of the network on “Pedestrian” and “Cyclist” classes compared to the “Car” class. This could be attributed to the fact that there were fewer training examples of pedestrians and cyclists compared to cars, and it was tricky to weight the classes equally while training. Furthermore, the approach used a single network to learn the three different classes and simultanesouly perform detection on them. In some instances, especially for objects far away, the distinguishment between pedestrians and cyclists was poor. This further suggests the need for more training examples to improve the classification.
The setting of hyper-parameters affected the average precision quite drastically. For instance, a lower global threshold to suppress predictions resulted in a lower precision, but increased recall, since there were more false positives but increased detections (true positives) as well. The non-maximum suppression threshold and parameters used for `groupRectangles` also had a similar effect, where a higher threshold resulted in more detections, especially of vehicles that were close together, since it allowed more boxes that were previously suppressed because of their overlap with other boxes. However, it reduced precision, since additional instances of boxes that were of the same object were counted as false positives.
It was also observed that the average precision was not drastically affected by the

difficulty rating of the objects, and increased false positives were observed only for the "Easy" objects. This is because they were closer and hence, there was a tendency to have multiple detections for the same object due to non-optimal suppression of the boxes. Therefore, results can potentially be improved with more careful tuning of hyper-parameters.
The range prediction reduces the ambiguity in determining the scale of objects, by directly estimating the distance in meters to the objects. Furthermore, since mono-camera images were used, the range prediction was an exercise to see whether the features extracted by a CNN can actually be used to estimate distances to objects directly. Although distances to objects can be easily obtained by computing disparity maps from stereo images, the results from the range prediction show that other aspects about objects such as the orientation and size could be reliably estimated, using regression networks that use the features from a CNN.

5.2 Semantic Segmentation

Semantic segmentation offers detailed information of traffic scenes which can be useful for mapping and localisation tasks. Classifying pixels in an image into very fine classes can offer the potential to gather points from the scene which are static so that they can be used as landmarks for localisation. Furthermore, classes such as "Road", "Side-walk" and "Parking" give information that can be directly used for path planning. The deep learning approaches presented as case studies are promising and achieve competitive results on the PASCAL VOC segmentation task [5], as mentioned in the respective literature. The approaches also achieve competitive results on the Cityscapes evaluation benchmark, although the dilation kernel based approach clearly generates better results compared to the others.

5.2.1 Design Analysis

The deconvolution network based approach was replicated using the architecture suggested in [18], with small changes. The approach suggested a downsampling network with stride 32, followed by the mirrored upsampling network. The last max-pooling operation of the downsampling network is followed by a 7×7 convolution layer and 1×1 convolutions with feature vectors of size 4096 and are computationally heavy. While replicating the approach, the last max-pooling layer and all the following layers were removed, thus having only a stride of 16 in the downsampling network. The upsampling network was constructed with upconvolution layers corresponding to the remaining layers of the downsampling network. The idea was to compare the performance of the modified architecture with the original, and discuss the effects of an increased stride. As observed in table 4.3 and 4.4, the effect of a lower stride does not hurt the performance to a large extent. Intuitively, the original architecture with a larger stride condenses more information from a larger receptive field, and the feature vectors describing the receptive field also have more elements (4096) compared to the lightweight network (512). Hence, one can expect better segmentation on larger objects when the receptive field is greater, but then again loss of more spatial information also hurts performance on smaller objects.

This issue of multi-scale context information is better handled using dilated kernels, and hence the better performance of the dilation network can be attributed to the exponential increase in receptive field without having to throw away spatial information. The approaches in the different case studies offer the potential for in-depth analysis of deep learning techniques for segmentation. The deconvolution network approach builds upon the fully-convolutional network approach. The idea of sequentially downsampling feature maps and then reconstructing shapes of objects from highly representative vectors that give class information is very interesting. At the same time, a question that arises is "Is the downsampling truly necessary?".

5.2.2 Implementation Analysis

The training of the deconvolution network model relied on transfer learning, where the downsampling network was a pre-trained network. Since the upsampling network has the same number of parameters as the downsampling network, a natural procedure of training would have needed similar quantities of data as that of the ImageNet, on which the pre-trained network was trained. To tackle this issue, the original approach in [18] suggests pre-training the model on images from PASCAL-VOC segmentation data set, and then fine-tuning on a relatively smaller data set such as the Cityscapes data set. This was not attempted and is left for future work, as more data tends to improve model performance.

5.2.3 Segmentation Accuracy

The accuracy given by the intersection-over-union metric shows that the approach performs better on many static classes, and thus the approach can be reliably used to extract key points for mapping and localisation. Furthermore, the clear boundaries between the static classes allow for creating reliable maps with the driveable surface marked out, and can be used to generate mesh grids or occupancy grids. On the other hand, the dynamic objects are harder to segment and offer scope for improvement. The overall performance on segmenting vehicles is quite promising and thus can be seen as a natural extension to the detection task. However, instance segmentation would need to be achieved to be able to truly benefit from segmenting vehicles on the road.

5.3 Future Work

The applications of deep CNNs for general image-based classification, detection and localisation tasks are vast, and so is also the case in the context of road scenes. With enough labeled data, which can be represented as a spatial grid, training a network in supervised manner is a possible approach. However, there are also some obvious techniques and methods to extend the framework presented in this thesis.

5.3.1 Detection using bounding boxes

The detection approach presented in this thesis is trained to output two-dimensional bounding boxes. For complete understanding of position and geometry of surrounding objects, three-dimensional bounding boxes would be an ideal extension to this approach. While this extension was discussed during the thesis project, it was not considered due to the limited amount of time. One possible approach to three-dimensional bounding boxes is to use regression for the angle of which the objects are oriented in the road surface plane, and expand the two-dimensional bounding box coordinates to specify coordinates of the object edges.
Another natural extension is to introduce some recurrent components in the CNN, which are able to preserve and make use of temporal information through many frames. This has potential of resulting in a more stable tracking approach and can also increase performance of classification and localisation since decisions made by the network are based on multiple frames in sequence. However, this approach requires a data set of labeled image sequences.

5.3.2 Image Segmentation

The approaches for semantic segmentation presented in this thesis are all essentially pixel-level classification, distinguishing pixels belonging to different pre-defined classes. An extension to the task of pixel-level classification is instance segmentation. This task involves associating pixels of the image to belong to individual objects. A difficulty with this task is that the image might contain any number of objects, and thus having a pre-defined number of object instances is not sufficient. Recent work combining CNNs and recurrent neural networks [21], segmenting one object at a time, shows promising results and could be useful for future work in relation to the content of this thesis.

6

Conclusion

The results and discussions in this thesis shows that deep learning methods can be effectively applied to achieve object detection and semantic segmentation. The thesis work covered the practical application of different deep learning components, specifically related to the use of image data for perception in an autonomous vehicle. The results from the implementations reveal that deep learning methods hold high promise for the future and also much scope for improvement. The approaches offer a lot of advantages, while the biggest challenge to practical applicability is the high demand for computational resources. Rapid progress in research and development of both hardware and algorithms is expected, to ease the computational requirements in the future. Hence, the detailed analysis of deep learning methods presented in the thesis offer the potential for more understanding and continuing research to improve current methods.

Bibliography

[1] Dave Anderson and George McNeill. Artificial neural networks technology. *Kaman Sciences Corporation*, 258(6), 1992.

[2] Gary Bradski. The opencv library. *Dr. Dobb's Journal of Software Tools*, 2000.

[3] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

[4] 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.

[5] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.

[6] Li Fei-Fei. Imagenet: crowdsourcing, benchmarking & other cool things. In *CMU VASC Seminar*, 2010.

[7] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(9):1627–1645, 2010.

[8] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[9] Simon Haykin and Neural Network. A comprehensive foundation. *Neural Networks*, 2(2004), 2004.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[11] Lichao Huang, Yi Yang, Yafeng Deng, and Yinan Yu. Densebox: Unifying landmark localization with end to end object detection. *CoRR*, abs/1509.04874, 2015.

[12] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Pranav Rajpurkar, Toki Migimatsu, Royce Cheng-Yue, Fernando Mujica, Adam Coates, and Andrew Y. Ng. An empirical evaluation of deep learning on highway driving. *CoRR*, abs/1504.01716, 2015.

[13] Yoshua Bengio Ian Goodfellow and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.

[14] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[16] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[17] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014.

[18] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. *CoRR*, abs/1505.04366, 2015.

[19] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, pages 91–99, 2015.

[20] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[21] Bernardino Romera-Paredes and Philip H. S. Torr. Recurrent instance segmentation. *CoRR*, abs/1511.08250, 2015.

[22] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[23] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[24] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[25] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[26] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*, 2016.

[27] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[28] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep neural networks for object detection. In C.j.c. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2553–2561. 2013.

[29] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *arXiv preprint arXiv:1512.00567*, 2015.

[30] Zifeng Wu, Chunhua Shen, and Anton van den Hengel. High-performance semantic segmentation using very deep fully convolutional networks. *CoRR*, abs/1604.04339, 2016.

[31] Fan Yang, Wongun Choi, and Yuanqing Lin. Exploit all the layers: Fast and accurate cnn object detector with scale dependent pooling and cascaded rejection classifiers. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, 2016.

[32] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *CoRR*, abs/1411.1792, 2014.

[33] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.