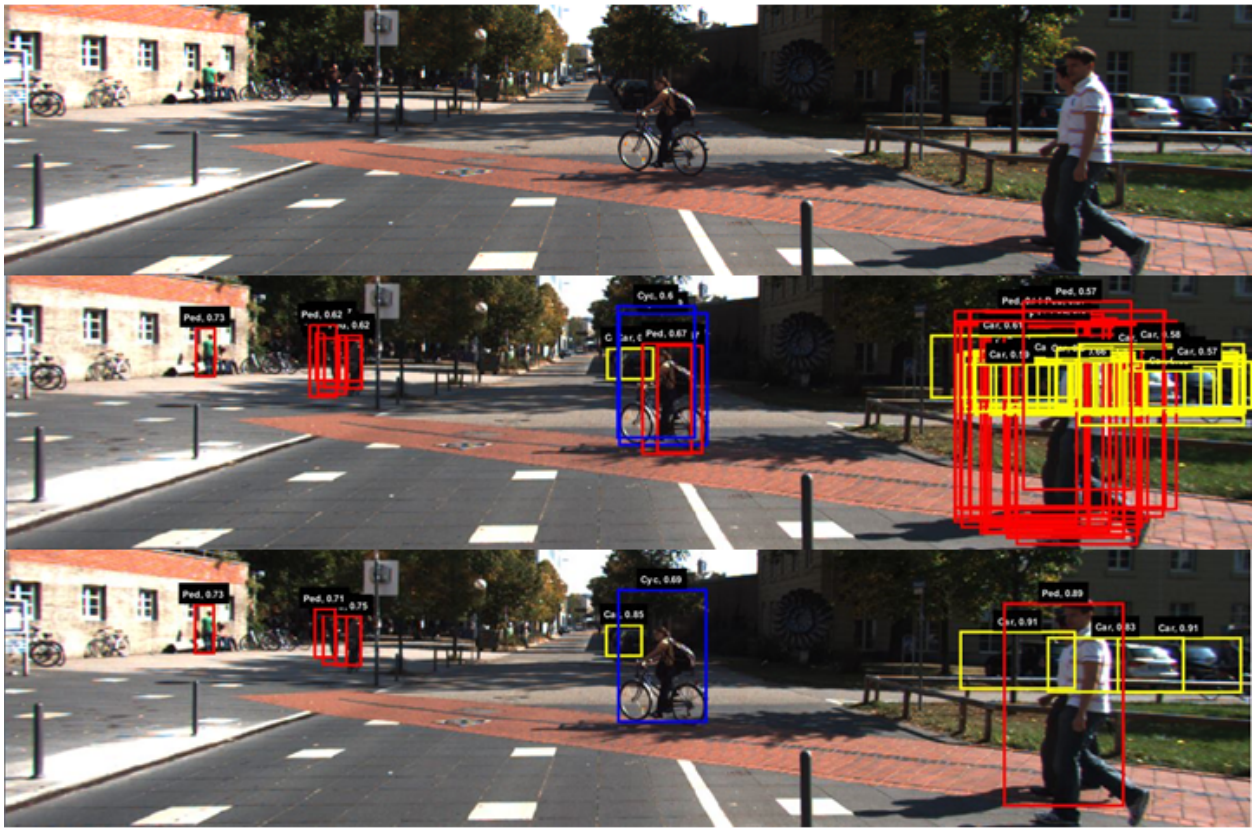


Deep Learning for Robust Road Object Detection

Using Convolutional Neural Networks to Detect Cars, Pedestrians, & Cyclists in Single Colour Images

Master's thesis in Complex Adaptive Systems

Dónal Scanlan

Master's thesis in Engineering Mathematics and Computational Science

Lucía Diego Solana

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017

Master's thesis 2017

Deep Learning for Robust Road Object Detection

DÓNAL SCANLAN LUCÍA DIEGO SOLANA

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2017

Deep Learning for Robust Road Object Detection
DÓNAL SCANLAN, LUCÍA DIEGO SOLANA

Supervisors: Nasser Mohammadiha and Ghazaleh Panahandeh, Volvo Cars Corporation
Examiner: Mats Rudemo, Mathematical Sciences

Master's Thesis 2017
Department of Mathematics Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Predictions generated by one of our networks for a KITTI test image before and after filtering.

Typeset in LaTeX
Gothenburg, Sweden 2017

Deep Learning for Robust Road Object Detection
DÓNAL SCANLAN, LUCÍA DIEGO SOLANA
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

Recent neurophysiological research has suggested that humans begin to discriminate objects in their visual field within one twentieth of a second. Our visual perception thus allows us to undertake complex, dynamic tasks, such as driving, in a seemingly effortless fashion. Emulating the efficiency with which we process our environment is an important step in the ongoing development of autonomous drive (AD) functionalities and advanced driver-assistance systems (ADAS). Important considerations in utilising convolutional neural networks (CNN) for automated object detection systems include robust performance but also high inference speed and modest memory requirements however. To this end, several light-weight CNN have emerged, designed specifically for real-time processing in embedded systems. In this study, one such model (*SqueezeDet*) is analysed in greater detail. Two modifications to the original network structure are also studied: the addition of residual connections and novel gated residual connections. Preliminary results suggest that the latter addition improves mean average precision in the *KITTI* object detection data-set by 1%, with particularly improved recall for distant objects. This is achieved with only a small inflation in the number of parameters (0.7%). On the other hand, adding standard residual connections lead to significant performance depreciation (3%). With the inclusion of a pre-training stage (classification on *ImageNet*) , the original architecture outperforms the modified network however. We suspect that saturation of the activation function in the gating mechanism during pre-training plays an important role in its under-performance following training for object detection.

Keywords: Deep Learning, Neural Networks, Machine Learning, Autonomous Driving, Convolutional Neural Networks (CNN), Object Detection, *KITTI* Database, *SqueezeDet*.

Acknowledgements

Before we begin, we would like to thank everybody that has contributed to this project. In particular, we would like to say a word of thanks to our Volvo supervisors, Ghazaleh Panahandeh and Nasser Mohammadiha, for pairing us on this project and for actively creating a fun and enriching work environment. Furthermore, to our Chalmers supervisor Mats Rudemo, we greatly appreciate his guidance on the academic aspect of our work and insight into the image analysis field.

We would also like to acknowledge everybody at Volvo Cars for daring to push the boundaries of driving technology and giving us the opportunity to contribute. In particular, we want to thank our colleagues, Christopher, Henrik, and Edvin, in the machine learning Master's group for their help and friendship. Similarly, we are grateful for all the help we received from the deep learning team at Autoliv.

Last but not least, their is no bounds to our appreciation for the support we received from our families, particularly during the completion of our Master's degree over the past two years. To our friends in Sweden and all around the world, thank you for keeping our spirits up!

Dónal Scanlan and Lucía Diego Solana, Gothenburg, June 2017

Contents

List of Figures

List of Tables

1
Introduction

1.1 Problem Description

Even with significant advances in image classification, the use of deep learning methods in locating multiple objects in a computationally efficient and robust way is still unresolved. Considerable effort has been invested as of late in the augmentation of convolutional neural network methods to propose and classify bounding boxes which encapsulate objects of interest in single images. Object detection is, however, far from a trivial task. Further to the difficulties experienced in classification; such as object occlusion, truncation, and intraclass variation, the complication of having zero or more instances of each class per frame is introduced.

Despite these issues, humans interpret visual information rapidly, subconsciously, and effectively. Recent neurophysiological research has suggested that we begin to discriminate objects in our visual field within one twentieth of a second [1]. Our visual perception thus allows us to undertake complex tasks such as driving in a seemingly effortless fashion. Considerable effort has gone into progressing deep learning as a means of emulating our ability to perceive our environment. As of late, the leaderboards of many visual perception challenges have been dominated by large convolutional neural network (CNN) models, some with upwards of 100 million parameters. Important considerations in utilising CNN for automated object detection systems include robust performance but also high inference speed and modest memory requirements however.

In the active safety department at Volvo Cars, work on sensors and systems for vehicles with AD functionalities is underway. Developing deep learning based tools which are capable of predicting the two-dimensional location of objects in an ego-vehicle's field of view can be integrated into systems for adaptive cruise control and collision avoidance.

1.2 Project Objectives

The goal of this project is to develop a system capable of detecting cars, pedestrians, and cyclist in real time without substantial memory requirements. Different deep learning-based methodologies have been proposed to achieve this, and a thorough study of them is undertaken here. This represents the first stage of this project.

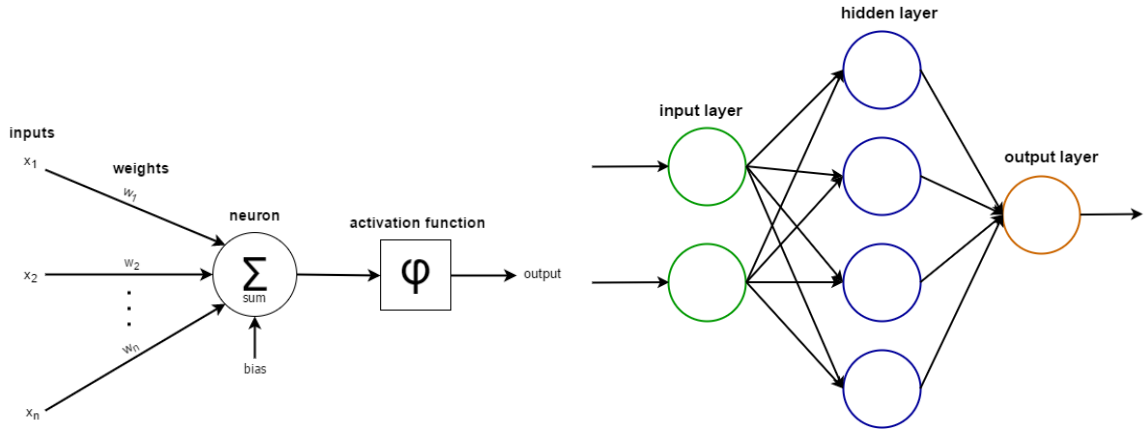


Figure 1.1: Left: an artificial neuron. Right: collection of neurons which form an artificial neural network.

Next, the model most suitable for the task is implemented and novel changes are applied with the hope of improving detection properties without significantly inflating the model size. Furthermore, different CNN training strategies are analysed. The *KITTI* object detection data-set acts as a training and testing environment for making a comparative study of the different architectures. The 2012 *ImageNet* classification challenge data-set will also be used to pre-train the network, an approach advocated by most in the CNN-based object detection field.

1.3 Machine Learning

Over the past few decades, machine learning has become more and more ubiquitous in information technology. Despite this, the branch is still in its nascence. Simply put, machine learning describes a plethora of techniques which share a data-driven approach to problem solving. In this way, algorithms *'learn'* to recognise patterns in a set of training data without being explicitly programmed to do so. Such approaches have, for the most part, replaced highly specific feature extraction programs in areas such as computer vision.

Artificial neural networks (ANN) are a class of machine learning algorithms which have shown great promise in tackling many persistent problems in computer science. Taking loose inspiration from the information processing structure of the brain, ANN consist of small inter-connected computational units called *neurons*. Neurons are generally segmented into layers: information is passed to the network through an input layer, is disseminated and transformed across multiple hidden layers, and values describing some abstract feature of the input information is given by the output layer following training. The transformations are parameterised by *weights* and *biases* which are known as trainable parameters, in the sense that they are optimised to best predict some aspect of the input data (see figure 1.1). This is done by quantifying, through a differentiable loss function, the similarity between the target labels and the outputs inferred by the ANN. Using calculus and some

optimisation strategy, the learnable parameters are modified with the aim of improving the likeness of the labels and the predictions generated by the ANN.

Convolutional neural networks (CNN) were developed to bring the robustness of ANN to image processing applications without massive inflation of the number of parameters and inference time. This is achieved through the use of the convolution operation which allows for a single set of parameters (called a *filter* in this context) to be used at multiple spatial locations of the input. Such a procedure, known as parameter sharing, fits naturally into image recognition problems when we expect local features, which may be useful in predicting more abstract properties of the data, to be present at any position in the input image [2].

1.4 Deep Learning

Deep learning can be thought of as a specific manifestation of ANN, with the discriminating factor being the number of layers used in the network. Models which have many hidden layers (more colloquially known as *deep* networks) can extract features from data at various levels of abstraction through many layers of affine transformations and non-linear functions. The success of deep neural networks (DNN) has come at the confluence of many trends: the widespread collection of various types of data, greater amounts of labelling for that data, falling hardware prices allowing for large-scale distributed computing, and more efficient network architectures just to name a few. Deep CNN have been successfully applied to many non-trivial problems; for example, aiding in the win of a computer against a world-champion at the ancient Chinese game of Go [3] and even recognising the onset of blindness through retinal images of patients with diabetes [4]. DNN have also been applied to problems which are solved intuitively by humans but confound computers. Therefore it is natural that such methods have played a major role in the development of automatic object detection systems over the past few years.

2
Background

In this section, we give an introduction to the basics of Convolutional Neural Networks (CNN). We describe the motivation for the original development of CNN and break down its structure into the key operations. Following this we go through different methodologies for applying CNN to the task of object detection. In **section 2.2**, we characterize how these networks are optimised to handle complex tasks like object detection by manipulation of the loss gradient.

2.1 Convolutional Neural Networks

While CNN have gained significant popularity in recent years, they have history which stretches back to the 1980's. These neural network models are designed specifically to efficiently process images. They do this by combining a large number mathematical operations.

2.1.1 Development

Over the last few decades, many different approaches for visual recognition have been developed. In 1980 Fukushima constructed a biologically inspired hierarchical and shift-invariant model for pattern recognition, known as a *neocognitron*. This network had the ability to recognize given patterns, including cases where there was a shift in position or a distortion in shape. The model was trained in an unsupervised manner, i.e. the target for a given training input was undefined. The network consists of an input layer followed by a hierarchy of connected layers. Neurons in these layers are either simple or complex cells, the former extracting local features of the input and the latter handling combinations of these features. An example of their inter-connection can be seen in figure 2.1, where the last layer of cells give the classification corresponding to the stimulus presented to the network [5].

In 1986, Rumelhart et al. created a supervised learning algorithm using gradient descent [6]. Four years later, LeCun et al. applied this backpropagation algorithm to a multilayer neural network for recognizing handwritten digits [7]. However, there were some outstanding performance issues: this network had poor in-built invariance with respect to translations or distortions of the inputs and therefore struggled to handle the variability in handwriting samples. Input images had to be size-normalized and centered in the input field. It became evident that in many object recognition problems that local features and the deformable organisation of

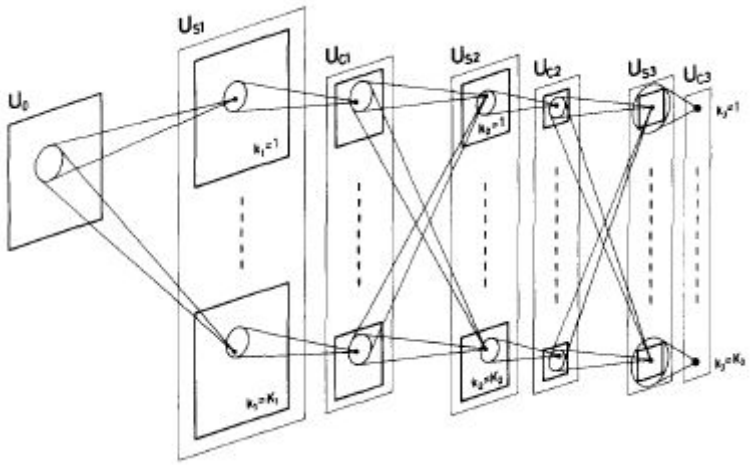


Figure 2.1: Diagram of Fukushima's *neocognitron*. Inspired by early investigations of animal visual processing, the network is composed of layers of simple cells (*'S-cells'*), which extract local features, and complex cells (*'C-cells'*), which handle the deformation and translation of these features [5].

them are of more interest, and network design should take this into account [8].

The problems described above led to the creation of a new model with stronger shift invariance which responds to hierarchies of local features. This new network was called the *convolutional neural network* (CNN) [8]. CNN's were used extensively in the 1990's, but were largely abandoned due to the lack of computing power. They were primarily replaced by support vector machines and complicated hand-crafted feature extraction approaches. Renewed hope in the perceptive abilities of CNN was awoken in 2012, when A. Krizhevsky et al. utilised a CNN to win the *ImageNet* Classification Challenge [9]. In the interim, CNN have developed significantly, and are now the default approach to solving many computer vision tasks [10].

2.1.2 Operations

A CNN is generally composed of different layers of operations, each with their own function. At test time, an image is fed to the first layer, and feature information is propagated through each layer producing an output. This is compared against a label (or ground truth) for that input. This is sometimes known as the *forward pass*.

2.1.2.1 Convolution

Convolution is the workhorse of a CNN. Assume we have an input image of width, height, and depth $W_i \times H_i \times D_i$. Convolution can be thought of as sliding a filter along the its width and height, computing dot products of the filter and the window of the input it covers at evenly-spaced locations (as seen in figure 2.2). This filter has a size of $W_h \times H_h \times D_h$, where $D_h = D_i$, and it encodes a pattern in its values (or *weights*). Filter weights are trainable parameters that are modified to extract

features of importance for the task at hand. The output of convolving one filter in multiple spatial position of the input is called a feature map and is a response to the similarity of this pattern to local features in the input. At this stage of processing, global features of the image are less of a concern. Therefore the sliding convolution, which uses the same weights across many spatial locations, is an appropriate choice in that it reduces the number of redundant, spatially-separated pixel interactions, introduces equivariant response to feature translation, and allows for variable input size.

The size of a feature map depends on the spatial dimensions of the input and filter and the stride applied to the convolution operation. Here, stride is defined as the number of pixels the filter shifts per convolution operation. To preserve the resolution, it is common to apply zero-padding to the input i.e. the width and height of the input image are framed with zeros. The size of the feature map is given by:

$$W_o = (W_i - W_h + 2P)/S + 1 \qquad H_o = (H_i - H_h + 2P)/S + 1 \tag{2.1}$$

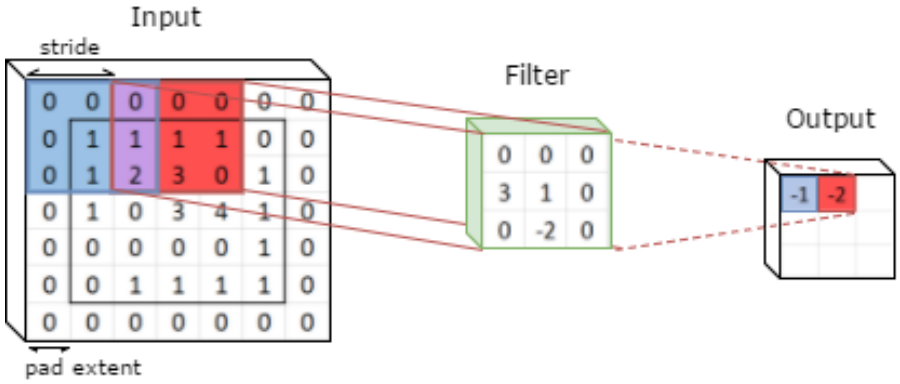


Figure 2.2: Convolutional layer of a CNN model. In this example, the image, of size $5 \times 5 \times 1$, is zero-padded and then convolved with a filter of size $3 \times 3 \times 1$. A stride of 2 was used in the convolution to produce a feature map of size $3 \times 3 \times 1$. Using k filters, our output is a volume of size $3 \times 3 \times k$.

where P is the pad extent and S is the stride. A single feature map has a depth of 1, and k filters are applied to the input per layer, giving a output feature volume of depth k. Further convolutions can be applied to this feature volume, with the output of this step describing combinations of the local features extracted in the previous layer. Thus, pixels (neurons) in an activation map of a typical CNN have progressively larger *receptive fields* with network depth (i.e. they are exposed to information from greater spatial tracts of the input image) and respond to more complex features present in the input image [11].

2.1.2.2 Activation

The activation function is used to limit the output of a neuron and to also introduce non-linearities to the linear activations generated by a convolution layer. The input

to such functions are typically the output of a convolution layer. There are several activation functions used in neural networks with some popular examples being:

1. **Sigmoid** Historically, it is the most widely-used activation function. It takes real values and transform them into the range $[0, 1]$. It is given by $\sigma(x) = \frac{1}{1+e^{-x}}$, with the output tending to 0 for large negative values and to 1 for large positive values. Nowadays, it is rarely used because it usually saturates at extreme values resulting in a vanishing gradient. Moreover it is not zero centered [12].
2. **Hyperbolic Tangent** It is calculated as $tanh(x) = \frac{2}{1+e^{-2x}} - 1$ with outputs ranging in the interval $[-1, 1]$. Like the sigmoid function, activations easily saturate, but it has the benefit of zero centered output [12].
3. **ReLU (Rectified Linear Unit)** Recommended as the default in DNN, this activation function is computed as $f(x) = max(0, x)$. Forward and backward passes can be computed far more efficiently than for the above mentioned functions. Furthermore, training algorithms have been shown to converge faster than with sigmoid and hyperbolic tangent functions as ReLU is piecewise linear. However, some neurons never become activated (i.e. only output 0) reducing the representation capacity of the network [12].

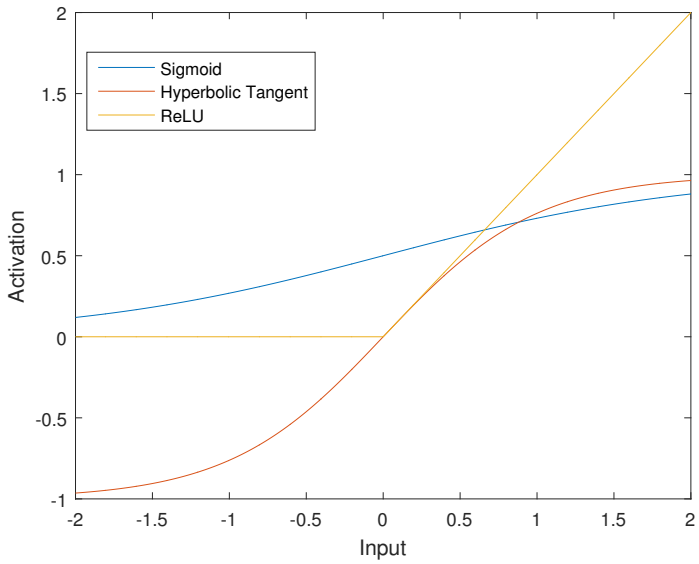


Figure 2.3: Example of activation functions commonly used in ANN.

2.1.2.3 Pooling

In CNN, it is common practice to preserve the width and height resolution during convolution. Pooling layers are responsible for reducing this resolution, making for smaller and more manageable representations by down-sampling. This operation is applied to each feature map outputted by a layer independently. The most popular pooling approach is known as *max pooling* (see figure 2.4), although variations include *average* and L_2 *norm pooling* [11].

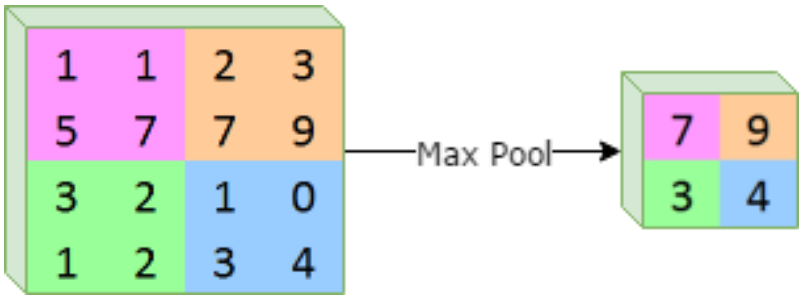


Figure 2.4: An example of max pooling with a 2×2 window and a stride of 2. At each window location, the maximum pixel value is extracted. Max pooling is typically applied to each feature map in a feature volume independently.

The results of this operation is an output volume with size $W_o \times H_o \times D_o$:

$$W_o = (W_i - W_h)/S + 1 \qquad H_o = (H_i - H_h)/S + 1 \qquad D_o = D_i \tag{2.2}$$

As such, pooling layers generate local feature-wise summary statistics. Beside compressing feature representations, another benefit of this is that it offers approximate invariance to limited feature translations; in other words, the exact spatial location of the feature with the highest activation is discarded.

2.2 Training

In training a CNN, forward passes for a number of images are computed. The outputted predictions are then compared to ground-truths to generate a loss. Using calculus, the sensitivity of this loss with respect to changes in network parameters can be found. This step is often referred to as the *backward pass*. With these gradients, the network's trainable parameters can be modified with the goal of decreasing the loss across the entire training data-set.

2.2.1 Loss function

In the context of artificial neural networks, loss functions are differentiable functions which characterise the similarity of a label with some features predicted by the network given an input. For object detection training, we typically optimise networks for a multi-task that includes classification and regression elements. In quantifying classification performance, *cross-entropy* is one of the most common loss functions used. In regression, the loss function varies from network to network, but is generally some variation of the L_1 or L_2 loss function, as can be seen in [14] and [19]. A significant issue when defining a loss function is to avoid saturating and vanishing gradients which can halt the learning process. For this reason, *log* terms appear regularly in error quantification. A detailed description of the loss function used in this study is given in **3.4.1**.

2.2.2 Backpropogation

Through backpropogation, losses calculated during training can be used to stimulate changes in network parameters, allowing the mastering of complex and varied tasks. This is done by calculating the analytic gradient of the loss function with respect to the network outputs, and using the chain rule to drive the error gradient backwards through the entire network. The logic is that parameters which have more significant effect on the final output will share more of the error gradient. A simple explanation to the back-propagation algorithm can be found in [27], where it is defined as follows. To calculate the gradient of a scalar z (e.g. the loss function) with respect to its precursor x in the graph, start by computing the gradient with respect to each ancestor of z. Then, multiply the present gradient by the Jacobian of dz. Keep multiplying by Jacobians going backwards through the network until it arrives to x. The final result is the sum of the gradients arriving from different tracks at any node reached by going backwards.

2.2.3 Updating

In optimising CNN, the loss gradient with respect to the model parameters is calculated for each sample in a subset of the training data known as a mini-batch (as described in **section 2.2.2**). Common mini-batch sizes are powers of two, with the final loss gradient given by the average over the mini-batch. This is then used to update the parameters of the network in a multitude of ways.

Intuitively, the loss gradient describes the extent and direction of change in loss experienced by increasing convolution parameters by one unit. This information can be used to alter the parameters in such a way that the network *'descends'* in the loss landscape. Gradient descent is the most common and straightforward way in which this is done. For a given filter, its weights w are updated in each iteration as follows:

$$w \leftarrow w - \eta \frac{\partial L}{\partial w} \tag{2.3}$$

Here, η is the learning rate and it is common practice to initialise it by trial and error and gradually decrease it over iterations. Despite its popularity, gradient descent is often the slowest to converge at a minimum loss setting, sometimes not even achieving convergence at all. Steep loss *'cliffs'* may encourage massive unwanted parameter changes while shallow inclines hinder speedy convergence. *Momentum* was introduced to alleviate these problems, where an exponentially decaying moving average of previous gradients is introduced:

$$v \leftarrow \alpha v - \eta \frac{\partial L}{\partial w} \qquad w \leftarrow w - v \tag{2.4}$$

Here $\alpha \in [0, 1]$ is a new hyperparameter controlling the contribution of the gradients [27].

AdaGrad is one of many updating procedures that includes an adaptive learning rate. The motivation for this is that in deep networks, the gradients for shallow layers are

typically much smaller than deeper layers, and learning rates should accommodate for this [28]. Here:

$$\Delta w = -\frac{\eta}{\sqrt{r+\delta}} \odot \frac{\partial L}{\partial w} \qquad w \leftarrow w + \Delta w \tag{2.5}$$

where $r \leftarrow r + \left(\frac{\partial L}{\partial w} \odot \frac{\partial L}{\partial w}\right)$ is a sum of the squares of past gradients. δ is a constant that avoids the division by zero. The mathematical symbol $\odot$ represents element-wise multiplication. The main problem of this method is the accumulating sum of gradients in r. Thus, the learning rate becomes iteratively smaller, decelerating the learning process. The benefit of this method over the *Momemtum* approach is a reduction in the number of hyperparameters while still retaining the desirable properties. With *RMSProp*, the updating rule looks much the same as *AdaGrad* with one key difference:

$$r \leftarrow \rho r + (1-\rho)\left(\frac{\partial L}{\partial w} \odot \frac{\partial L}{\partial w}\right) \tag{2.6}$$

r is now the weighted sum of squared gradients. ρ is a parameter that controls the decay of the moving average and it is usually set to 0.9. This configuration performs better in non-convex functions than *AdaGrad* and it was developed specifically to solve *AdaGrad's* monotonically decreasing learning rate [27].

Adaptive Moment Estimation, or *Adam*, is another adaptive learning rate optimization algorithm that can be seen as a combination of *Momentum* and *RMSProp*.

$$s \leftarrow \rho_1 s + (1-\rho_1)\frac{\partial L}{\partial w} \qquad \hat{s} \leftarrow \frac{s}{1-\rho_1^t} \tag{2.7}$$

$$r \leftarrow \rho_2 r + (1-\rho_2)\left(\frac{\partial L}{\partial w} \odot \frac{\partial L}{\partial w}\right) \qquad \hat{r} \leftarrow \frac{r}{1-\rho_2^t} \tag{2.8}$$

$$\Delta w = -\eta \frac{\hat{s}}{\delta + \sqrt{\hat{r}}} \qquad w \leftarrow w + \Delta w \tag{2.9}$$

where s and r are estimate of the first and second moments respectively. $\hat{s}$ and $\hat{r}$ are the bias correction of these estimates, as the first and second momentum are usually biased towards zero during the first steps. ρ_1 and ρ_2 are called decay rates and they are usually set to 0.9 and 0.999 respectively. Empirically, it has been shown that *Adam* is relatively robust to the choice of hyperparameters [27].

2.2.4 Regularisation

Regularisation has been defined as *"any modication we make to a learning algorithm that is intended to reduce its generalization error but not its training error"*. In other words, the goal is to encourage an algorithm to generalise well beyond the training set and, in a sense, estimate the true distribution from which the data is sampled from. Thus, we are concerned with managing the variance induced by estimating other data generating processes and the bias brought about by excluding the

true data generating process from the model respectively. One of the most prevalent forms of regularisation is the L_2 loss, more colloquially known as *weight decay*. Here, we introduce a new parameter $L_2(w) = \frac{\phi}{2}||w||^2$ to the loss function which is parameterised by the constant ϕ. The motivation behind this loss is that weight matrices with lower, more evenly distributed values are encouraged over sparse weight matrices with high values. This pushes the network to exploit all available input data regardless of whether or not it improves training error [27]. Dropout is another regularisation technique used to complement L_2 regularization. To avoid overfitting during training, neurons are *de-activated* with probability $1 - p$ i.e. their output is set to 0. This approach provides a way of simulating different architectures without requiring excessive computation or memory requirements [29]. The probability p and the constant ϕ are both hyperparameters that are fixed before starting training. p is usually set to 0.5 while *phi* is chosen by heuristics. There also exists a repertoire of data set augmentation techniques including affine transformation and colour distortions of the input which help decrease over-fitting and are discussed in **section 3.1.3**.

2.3 Object Detection

Object detection is the task of generating class labels for, and bounding-boxes which surround, one or more objects in an image. In 2015, a machine surpassed human-level performance in the *ImageNet* classification challenge for the first time [13]. In comparison to classification, object detection is a far more complex undertaking and super-human performance through deep learning remains elusive. The difficulty arises from the need to locate multiple objects, with the class and number of instances not specified *a priori*. Overcoming this in an efficient way could be an important step in the development of autonomous driving agents.

As stated, the main outputs of many CNN-based object detectors are bounding boxes coupled with confidence scores. These two concepts are important during the development of this thesis, thus, a good understanding of them is necessary. We interpret an ideal bounding box to be the minimal axis-parallel rectangle that contains all parts of an object. Depending on the visibility of the objects and the database used, the bounding boxes are treated differently. This issue will be discussed in **section 5**. Ideally, confidence scores can be used to estimate the probability that an object lies within the bounding box. These scores are normalised between 0 and 1, and are often erroneously referred to as probabilities in the literature (e.g. [14]). Furthermore, confidence in this context should not be confused with the notion of confidence in the context of statistics (e.g confidence intervals).

2.3.1 Classical Methods

Prior to the application of deep learning methods to computer vision problems, automatic object detection systems were mainly based on extracting feature descriptors such as *histograms of orientated gradients* (HOG) or *scale-invariant feature transform* (SIFT) with subsequent classification by SVM. HOG features are computed

by dividing an image into a number of overlapping blocks, computing intensity gradients for each pixel, and quantising the gradient orientation into several bins. The histograms for each block are then concatenated and fed to the classifier. In locating objects, a *'sliding window'* approach is taken i.e. features are extracted and classified from sub-images at multiple scales, aspect ratios and evenly spaced locations. While producing state-of-the-art results in the early 2000's, such methods lacked robustness with regard to occlusion and deformation, and performance did not generalise beyond a small set of classes. Deformable parts models were introduced to give flexibility to such classifiers, although the resulting accuracy and run speed were still prohibitive for many applications [15].

SIFT is a proprietary approach to object detection developed by David G. Lowe. With this method, keypoints are extracted from labelled images and stored in a database. When a new image is to be processed, a match process between the new image features and those in the database is realized using Euclidean distance. Following this, a Hough transform is performed to identify clusters from a specific object and the probability that a particular feature vector represents an object in the image is computed. The verification is performed as a least-squares solution applied to the parameters obtained from the affine transformation [16].

2.3.2 Sliding Window CNN Methods

Many of the attempts to tackle object detection using CNN going back to the early 1990's also relied heavily on sliding window processing. Such methods have been successfully applied to several domains, including face, pedestrian, and text detection. In 2013, one such network, namely *OverFeat*, won the 2013 *ImageNet* joint classification and localisation challenge (where localisation refers to predicting the location of a single object of interest) and obtained competitive results in the detection challenge. This method extracts features for each window and feeds them to a classification network and bounding-box regression network, the latter refining the window to more tightly envelop an object. The resulting class scores and regressed bounding-boxes are then aggregated using a greedy merging algorithm to produce the final class and location predictions [17].

Despite its successes, and efforts to share computations between windows, one major drawback remains; many of the windows processed are redundant or superfluous, only partially containing objects of interest and thus considerably inflating the running time. Furthermore, particularly in the case of *OverFeat*, the quality of the final output is highly reliant on the ability of an external algorithm, which cannot be optimised in the same manner as the network, to integrate the information from multiple windows.

2.3.3 Regional CNN (R-CNN)

The genesis of *R-CNN* came in 2014, when a group at UC Berkley aimed to generalise the successes achieved with CNN to the task of object detection. Their

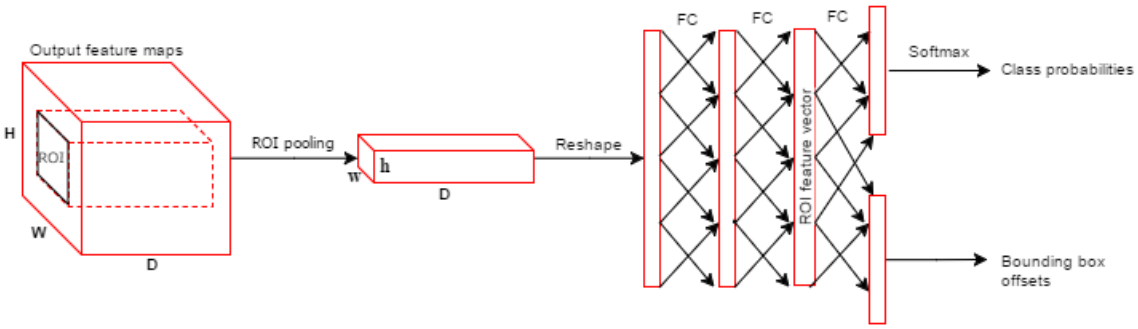


Figure 2.5: *Fast R-CNN* architecture: a ROI is projected onto the output features maps, pooled to a standard size and then passed to a fully-connected feed-forward network which generates class confidence scores as well as position and scale offsets for a single ROI.

method combines an independent region proposal algorithm and a CNN, which suggests object-containing regions and compresses them into a fixed length feature vector respectively. Each feature vector is then classified by class-specific support vector machines (SVM) and the set of classified region proposals reduced using non-maximum suppression (NMS). NMS is a greedy algorithm which sorts detections by their object confidence scores, takes the highest scoring detection and removes lower-scoring detections which have an IOU greater than some threshold, repeating the procedure until all detections have been considered. Finally, localisation is refined by linear regression of the features extracted by the CNN, leaving the expected bounding-boxes of objects in the input image. The introduction of the SVM classifier brought with it a constraint of having a fixed input size, which was surpassed by warping image regions (regardless of its size, location, or aspect) to pre-defined dimensions before processing by the CNN. Another major drawback of R-CNN was it's high computational cost; a result of passing warped sub-images individually through the CNN (with around 2,000 regions being generated at test time), the need for a separate algorithm for region proposals and refinement, and the introduction of class-specific classifiers [10].

2.3.4 Fast R-CNN

Fast R-CNN retained many of the core notions of R-CNN but introduced several refinements. One such was the *Region-of-Interest* (ROI) pooling layer. With *Fast R-CNN*, the entire image is first processed by a CNN (which we will hereafter refer to as a feature extractor in this context), producing a single set of features maps for the whole image. As with R-CNN, region proposals are generated externally. On the contrary, each spatial region is then projected onto the feature maps, and the associated volume is pooled to standard dimensions. The proceeding fully connected layers take as input these features and output softmax class confidence scores and bounding-box regression offsets (see figure 2.5) [18].

2.3.5 Faster R-CNN

Despite the streamlining achieved by *Fast R-CNN*, it still places the requirement on some external method to propose ROI's, which is a bottleneck in terms of running time. Like *Fast R-CNN*, *Faster R-CNN* exploits the feature extractor output for classification and regression, but in contrast, uses this information in generating region proposals through the novel *Region Proposal Network* (RPN). In this way, a single set of feature maps is generated from the input image, which is then passed to two sibling networks; the RPN which generates region proposals, and a *Fast R-CNN* network for classification and regression of these ROI (note that the names region proposals and ROI are used interchangeably). Thus, *Faster R-CNN* represents a consolidation of CNN object detection architecture which can be trained in a single stage end-to-end, as seen in figure 2.6.

The RPN is modelled as a fully convolutional network which operates on the feature extractor output maps. The features are first passed through an 3×3 convolution layer, followed by a 1×1 convolution layer, with the output predicting k bounding-box locations and object confidence scores for each spatial location (see figure 2.7). Here, k is a hyperparameter describing the number of *anchors*, with anchors (or *box priors*) being reference shapes of different scales and aspect ratios in the input image (see figure 2.8). Thus, $W_i \times H_i \times k$ ROI are predicted. It is important to make the distinction here that the 4 bounding-box coordinates produced by the RPN describe the region proposals with respect to deformations of the anchors. Further to the speed up in run time enjoyed by *Fast R-CNN* as a result of sharing convolutional features across region proposals, a considerable speed improvement is compounded by replacing expensive external proposal methods [14].

2.3.6 Region-based Fully Convolutional Network (R-FCN)

One unresolved issue with *Faster R-CNN* is its speed at test time, which prohibits application in many domains. To counter this, several works have driven the progression of shared convolution features between region proposals. As with *Faster R-CNN*, a *RPN* is applied to the output of the feature extractor to generate region proposals. Unlike *Faster R-CNN*, the Fast R-CNN classification sub-network is replaced by a fully convolutional network which generates class-specific and position-sensitive feature maps for the whole input image (figure 2.9). This is implemented as follows: the feature extractor output is fed to another CNN, whose last layer gives an output of depth $k^2(C+1)$, where k^2 are the number of activation maps per class and $C+1$ is the number of classes plus background. The novel ROI pooling operation (see figure 2.10) enforces strict spatial constraints on the activations. Through training, the output feature maps should have an increased sensitivity to the position of objects in the input image. The goal here is to address the dilemma of balancing a network's translation invariance for image classification versus the translation variance needed for object detection. With *R-FCN*, the final pooling layer is the only per-region operation, resulting in a significant reduction in per-region computations as well as inference and training speed [19].

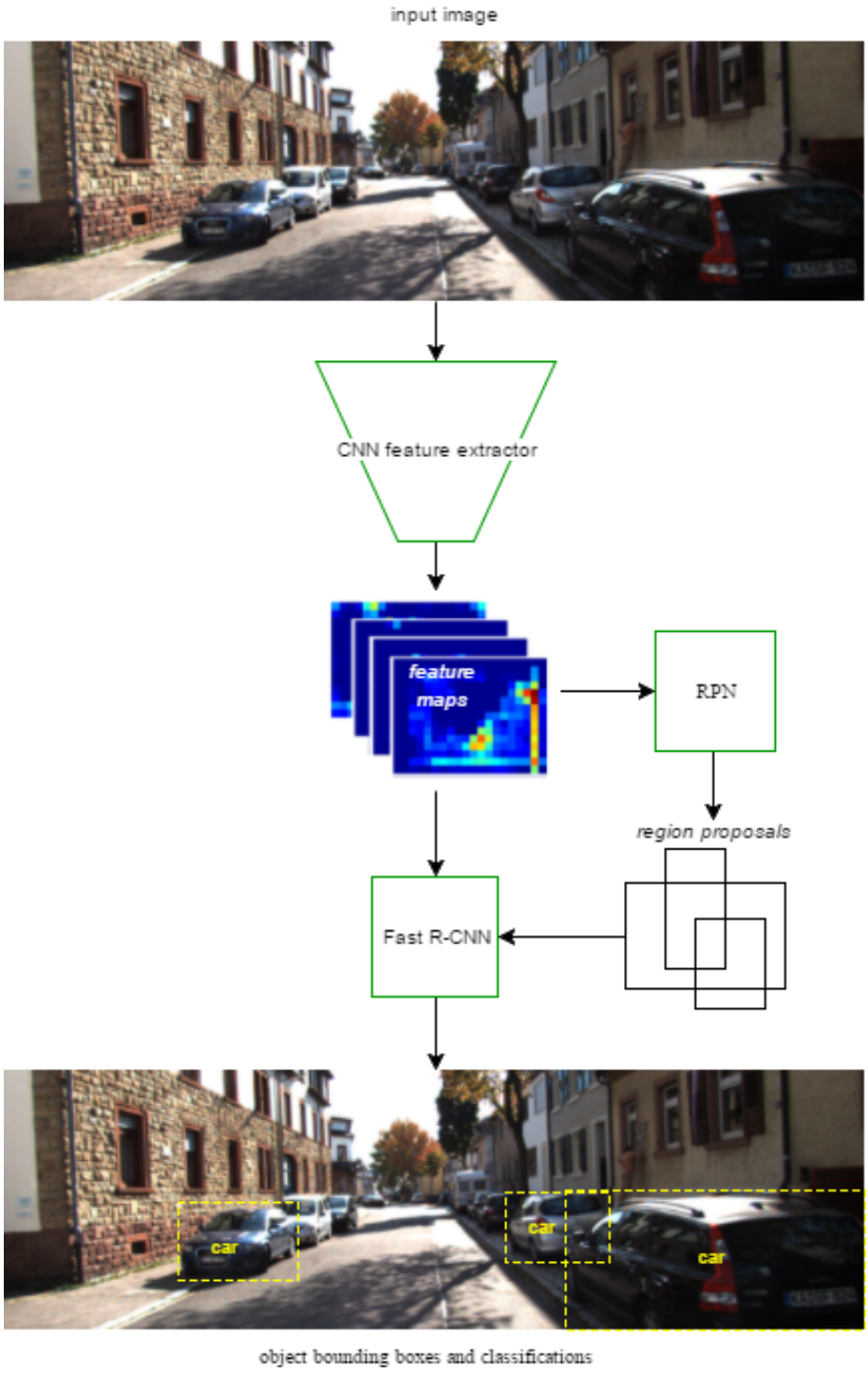


Figure 2.6: *Faster R-CNN* architecture: firstly, an input image is processed by a feature extractor. The output feature maps are then passed to an RPN to generate region proposals, which are subsequently passed to the ROI pooling layer of the *Fast R-CNN* module along with the feature maps.

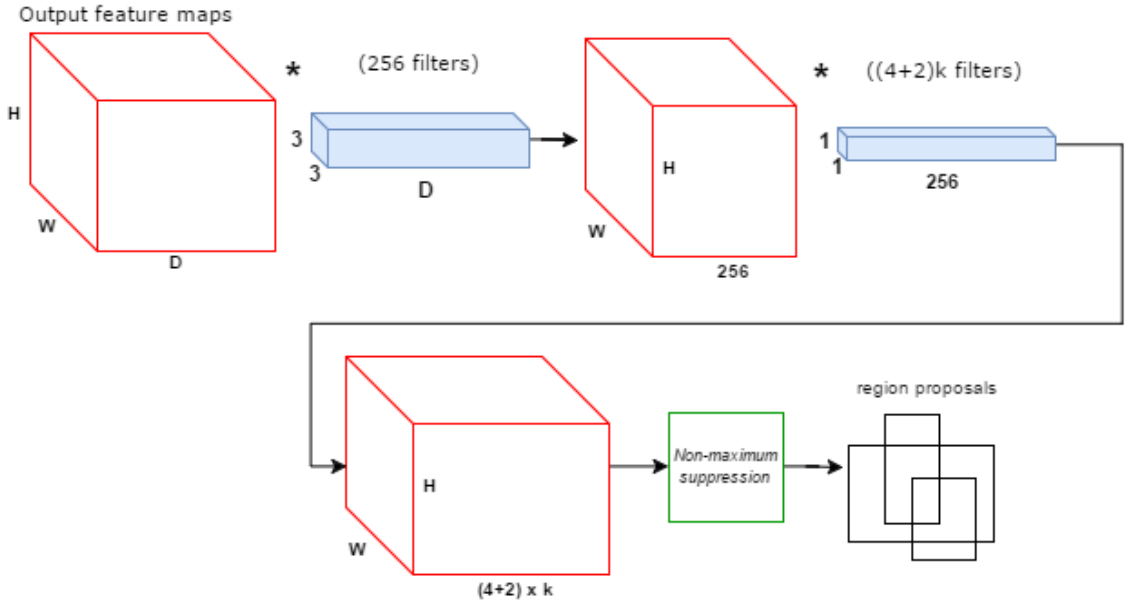


Figure 2.7: *Region Proposal Network* architecture: at each of the $H \times W$ spatial locations, class-agnostic object confidence scores and coordinate offsets are predicted for k anchors (reference shapes). The resulting set of region proposals for the whole feature volume is reduced through non-maximum suppression.

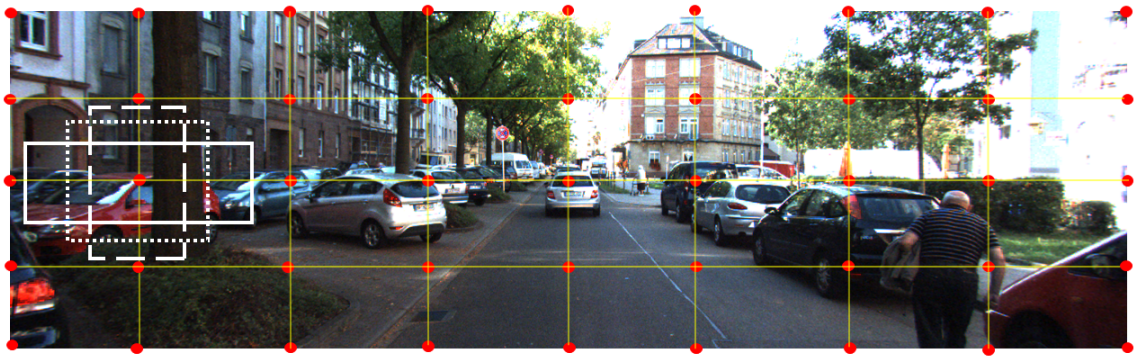


Figure 2.8: Illustrated is a sample image from the *KITTI* data set. A grid is imagined to overlay the input image, with $H \times W$ intersection points (shown in red). At each intersection point, the *RPN* predicts the translation and scaling for a set of k anchors centred at that point (three examples are shown in white), with the original anchor dimensions specified *a priori*.

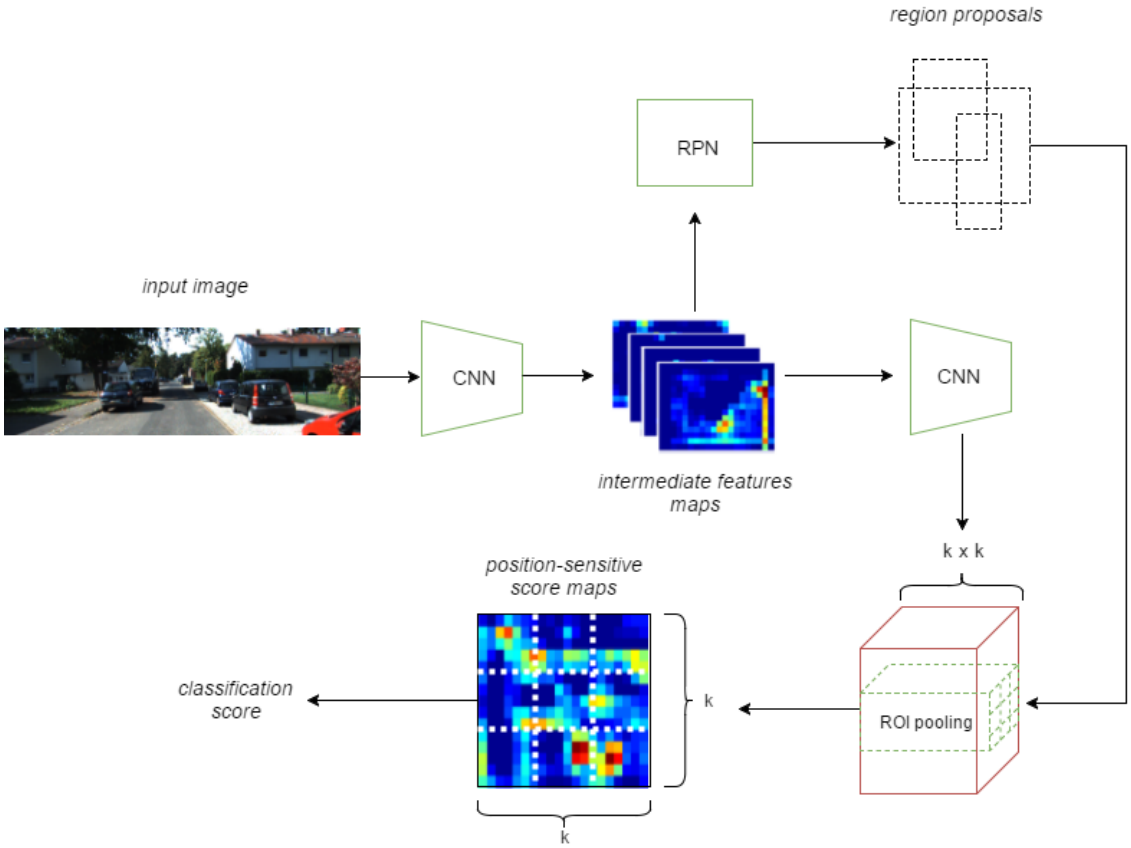


Figure 2.9: *R-FCN* architecture: the input image is processed by a feature extractor. The output feature maps are then passed to an RPN to generate region proposals. Concurrently, the feature maps are processed by a second CNN. The output feature maps of the second network have a depth of k^2 per class (including a background class). These maps are passed to a position-sensitive ROI pooling layer along with the region proposals. This pooling is illustrated in more detail in figure 2.10.

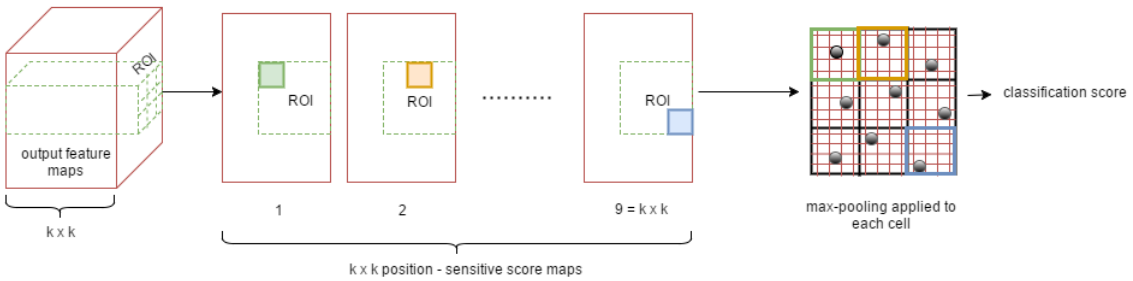


Figure 2.10: *R-FCN* pooling: the region proposals obtained in the RPN network are individually projected onto the output feature maps. The area of each ROI is divided into $k \times k$ cells, and a single cell is taken from a each feature map, i.e. the number of cells is the same as the depth of the feature maps per class. Max-pooling is applied to the set of pixels inside each cell (for illustration purposes, the black dots in the image represent the pixels with the biggest value) and then an unnormalised class confidence score is obtained by averaging the results of the pooling.

2.3.7 You Only Look Once (YOLO)

YOLO reframes object detection as a single regression problem, where there is a more direct feed-forward mapping from image pixels to object location and class predictions. The input image is imagined to be divided by a grid into S^2 cells. This image is processed by a network consisting of 24 convolutional layers followed by two fully-connected layers. The output is a volume with a width and height S, and a depth of $(C + B(4 + 1))$. Here, C is the number of classes and B is the number of bounding boxes predicted per cell of the input. The 4 and 1 relate to the offset coordinates with respect to the cell boundary and the confidence score respectively, where confidence is interpreted as the certainty that an object of interest has a centre which lies within that cell multiplied by the intersection-over-union between the ground truth and predicted bounding boxes. The need for a class-agnostic RPN has thus been bypassed. Instead, *YOLO* can be thought of as an RPN which generates class probability distributions and gives harder offset coordinates. Such methods are sometimes grouped under the name *Single Shot* detectors along with approaches like the *Single Shot MultiBox Detector* [20] and *SqueezeDet* [21] (the latter being the model analysed in this study and is discussed in greater detail in **section 3.3.1**). While exhibiting competitive inference speed, its accuracy is not acceptable for many applications [22].

2.3.8 Other Notable Works

While *Faster R-CNN* represents a step change in terms of accuracy and speed, issues still remain. A significant one is poor detection of objects which are small in the input image [23]. Two factors contribute to this under-performance. Firstly, *Faster R-CNN* only exploits features maps from the final layer of the feature extractor in generating bounding-boxes and assigning classes. However, it follows from pooling operations that feature maps deeper in the CNN have more granular spatial res-

olution. Furthermore, depending on the size of a neuron's receptive field, spatial information in the input image may be diffused to a greater extent in feature maps deeper in the network (increasing translational invariance).

Multiscale CNN [23] and *PVANet* [24] are a small selection of works which aim to address this issue. They do so by employing features maps outputted from multiple layers of the feature extractor and not just from the final layer. Other work has refined this concept further, introducing *Scale Dependent Pooling* whereby feature maps are taken from a specific layer based on the size of the region proposal. This has the benefit of reducing the amount of redundant information being processed in classifying the regions [25]. The reasoning behind such modifications is that objects which are small in the input image will have higher activations in earlier layers. Accuracy improvements, with only minor cost incursions, have also been exhibited by iteratively cycling bounding-boxes through a *Fast R-CNN* network [26].

2.4 Conclusion to CNN

To summarise, CNN is a flexible methodology which has found particular use in the task of object detection. Traditionally, results in this task were not sufficient for utilization in real time systems. However, with the development of more effective architectures and optimisation strategies, along with several other factors, the field is approaching this paradigm. CNN-based object detectors can be coarsely divided into two categories: those that propose and classify regions-of-interest (e.g. RCNN), and those that generate predictions for evenly spaced locations of the input image (e.g. *YOLO*). The latter have, in general, an advantage in terms of processing speed and will thus be explored in this thesis.

3
Data & Methods

In this section, we describe the data-sets used to train and evaluate the CNN-based object detection models. The architecture of the chosen method is explained in greater detail, along with the loss functions and the hyper-parameter values chosen to tune the network. We also characterise two novel modifications made to the network, giving motivation for these changes.

3.1 Data

Important consideration in choosing a training and testing data-set include size, but also to what degree they matches the application's target environment. Here, we are concerned with detecting road objects in standard driving environments. For this reason, the *KITTI* object detection data-set was favoured over those available from *MS-COCO* and *PASCAL*. Classification data-sets are often used as a pre-training stage to prime networks for object detection training. Here, we consider the *ImageNet* 2012 challenge data-set for this purpose.

3.1.1 *KITTI* Vision Benchmark Suite

The *KITTI* database is a project of the Karlsruhe Institute of Technology and Toyota Technological Institute of Chicago. It was created with the purpose of reducing the bias between application performance tested *in silico* versus the real world, and for providing real-world benchmarks with novel difficulties to the computer vision community. To achieve this, a standard station wagon was equipped with two high-resolution color and grayscale video cameras and driven around the city of Karlsruhe, Germany. While there is not much variation in lighting and weather conditions, the data-set contains images depicting an array of driving environments [30].

KITTI consists of eight different sets of data, each having its own specific purpose. For example, the stereo data set contains images taken from two cameras with a small separation between them, which can be used to evaluate stereo matching methods. The odometry data collection functions as a testing and training environment for determining the position and orientation of the ego-vehicle by analyzing different sequences of images. Other groups of data are designed for tracking, semantic segmentation or optimal flow purposes. Following calibration of the sensors and data-collection expeditions, the raw images are rectified. Then, distinct approaches are applied to annotate data depending on its type. In the case of the object de-

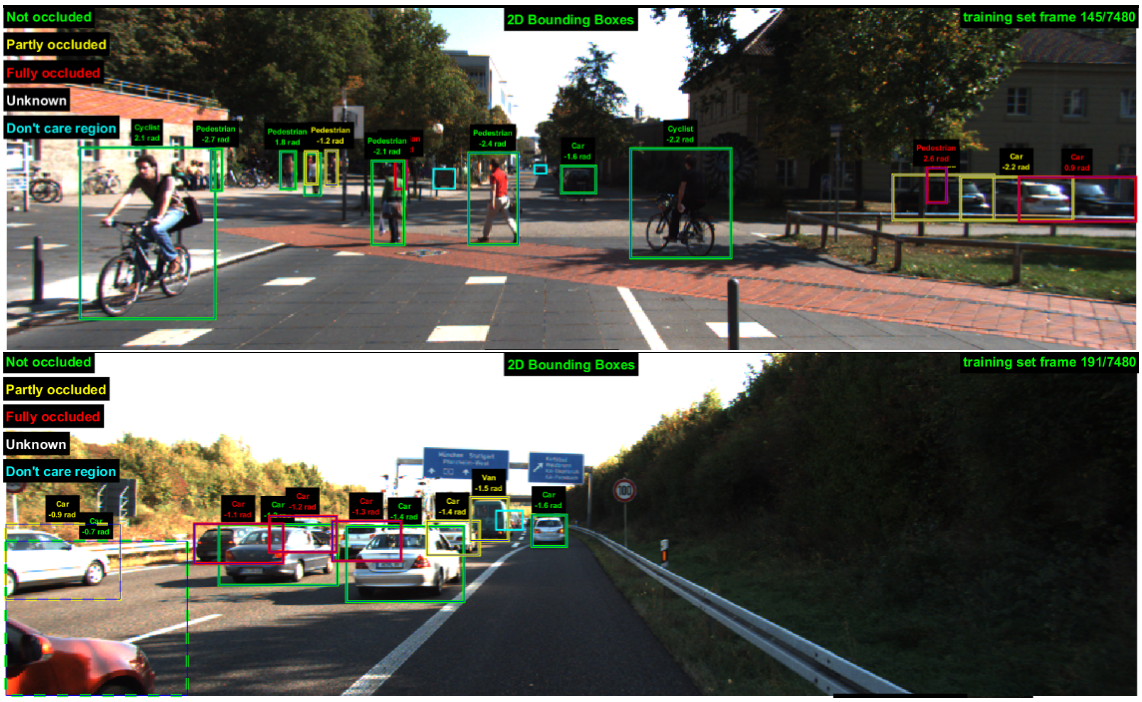


Figure 3.1: Sample images from *KITTI* object detection database with ground truth bounding box annotation and class labels.

tection images, each object was tracked manually, and the hired annotators marked the bounding boxes as visible, semi occluded, fully occluded or truncated [30].

In this thesis, the object detection database will be used. It consists of 7481 training images and 7518 test images in color *png* format of approximate size 375×1242 pixels. Training images are annotated with bounding boxes which describe the positions of 8 categories of objects (car, van, truck, pedestrian, person sitting, cyclist, tram, and miscellaneous) plus *'Don't care'* regions. The latter label refers to regions in which detections are not counted in evaluation metrics e.g. image regions containing distant objects. *KITTI* also provides an evaluation platform that calculates the precision and recall on the test data for 3 categories: namely cars, pedestrians, and cyclist. Evaluation metrics include recall, precision and mean average precision (mAP). Recall is defined as the ratio between the number of correctly assigned detections and all ground truths, whereas precision is the ratio of the number of correctly assigned detection and all detections. In object detection it is common for a CNN to output confidences on its predictions. Precision and recall can thus be evaluated at different confidence thresholds generating a *precision-recall curve* for each object class. Integrating this curve gives the average precision (AP), with numerical integration often used. The mAP is computed by averaging the AP over the set of classes and different difficulty categories: easy, moderate and hard.

In evaluating performance, detections are assigned to ground truths if they share a class label and have an intersection-over-union larger than 70% for cars and 50% for cyclists and pedestrians. A detected object with a height smaller than 25 pixels is

not considered in evaluation, and detection of vans is not identified as a false positive for cars. Each image contains ground-truths for at most 15 cars and 30 pedestrians. Performance can be evaluated at three different difficulty levels, assessed based on the minimum object size, maximum occlusion level (the extent to which an object is blocked from view), and maximum truncation level (the extent to which an object is out-of-frame) [30]. Samples images from the training set can be seen in figure 3.1. Test result submission involves sending a zip file containing 7518 text files corresponding to each image in the test set to the evaluation server. Inside the text file, the category of the object (i.e. car, pedestrian and cyclist), the bounding boxes coordinates and the confidence score must be written, alongside other values for truncation or occlusion, which are not necessary in this thesis and are set to a default value specified in the *KITTI* development kit.

3.1.2 *ImageNet*

ImageNet is a database which consists of over 14 million URL's to third party images, each with manually annotated class labels. These labels are organised according to the *WordNet* hierarchy and describe *synonym sets* or *synsets*. Each label is typically composed of several equivalent words or phrases describing a tangible concept. The tree structure of the data set allows grouping of images at various levels of granularity, with *children* or hyponym sets having more precise descriptors of member images. Every synset has a unique WordNet ID (*wnid*) and $20,000$ such exist at time of writing [31]. Although for the most part, *ImageNet* contains only a single class label per image, this hasn't hindered its use in training algorithms for computer visions tasks other than classification; such as image segmentation, action recognition, and object detection. This practice is commonly known as *transfer learning* and involves training a network on a primary task and data set (e.g. classifying *ImageNet* images), preserving the trained variables and possibly extending this network, and subsequently training on a secondary task and data set (e.g. object detection on *KITTI*). If the secondary data-set is relatively small and the number of trainable parameters high, it is advised to *freeze* some, or all, the layers which have been transferred from the primary to the secondary network to prevent over-fitting i.e. allow no further optimisation of the parameters in these layers [32].

Regardless of the task or training data set, the weights learned by the first layer of a CNN tend to simulate colour blobs or Gabor filters [33]. With increasing network depth however, the specificity of the features to the the training environment tends to increase. Furthermore, research has suggested that training a network on data with objects and scenes similar to those found in the secondary task can improve performance on the secondary task. Reducing the number of classes, but increasing the number of images per class, has also been shown to improve the performance of the network in both the secondary task in some instances [33]. Thus, the similarity between the primary and secondary data and task, along with the issue of *fragile co-adaption* of successive layers, plays an important role in transfer learning. However, in order to make a fair comparison with the results outlined in [21], the *ImageNet* 2012 Large Scale Recognition Challenge data-set was used,

Figure 3.2: Top: Training image with minimum saturation alteration (factor of 0.75). Bottom: The same image with maximum saturation alteration (factor of 1.25).

consisting of approximately 1.3 million training and 50,000 validation images from 1,000 synsets.

3.1.3 Data augmentation

As mentioned in **section 2.2.4**, artificially expanding data is common practice in deep learning applications as a means of preventing over-fitting. This is particularly important when a limited number of training samples is available. In this study, colour distortions are applied to the input image after reading from disk. Such a procedure has the benefit of preventing inflation of memory requirements. Explicitly, alterations to the brightness and saturation of the image are exercised in random order and to a random extent, with the alteration factors sampled from a uniform distribution between a minimum and maximum value (see figure 3.2). For primary training of the network, horizontal mirroring of images is also applied randomly.

3.2 TensorFlow

TensorFlow is an open source software library for machine learning applications created by *Google's Brain Team*. It was developed to satisfy the company's need for a flexible, scalable, and portable platform for machine learning at both research and production level. It is currently employed within the company in several products (e.g. *Gmail*) and by third parties such as *Snapchat*, *Uber*, and *Twitter*. Data flow graphs are a central element of *TensorFlow*, where nodes in the graph represent

mathematical operations (like convolution and pooling) and graph edges serve as multidimensional data arrays (tensors) communicated between them. TensorFlow can map such graphs onto a variety of devices, such as mobile devices, or CPU/GPU clusters. The library can be accessed using API like Python, C and C++ [34].

3.3 Network Architecture

In this section, we describe the model chosen following the state-of-the-art review summarised in **section 2.3**. An outline of the novel modifications to this architecture which are made in this thesis is also given.

3.3.1 SqueezeDet

High accuracy and recall across a variety of environments, high inference speed, small model size and energy efficiency are important considerations in the context of object detection in embedded systems. Belonging to the *Single Shot Detector* family, *SqueezeDet* is a fully convolutional neural network for object detection which was developed to address these concerns [21]. The *SqueezeDet* architecture is composed of a convolutional layer, ten sequential *fire modules*, and a final convolutional layer which outputs predictions of object locations and class labels for a given input image.

SqueezeDet is small in its model size. The main contribution to this is its feature extractor (*SqueezeNet*) which is composed of sequential *fire modules*. As observed in figure 3.3, this module consists of a 1×1 convolutional layer that compresses the feature volume without decreasing the spatial resolution. This first operation is known as the *squeeze layer*. To the output of this layer, two parallel convolution operations are applied, one layer with 1×1 convolutions and another with 3×3 convolutions. This parallel procedure is called the *expand layer*, as the depth of the feature volume is increased at this stage [21]. The reasoning behind the *fire module* is that the *expand layer* incur a high computational cost which can be relieved by first compressing the feature representations. Along with all other *Single Shot Detectors*, the benefit of architectures such as *SqueezeDet* over *R-CNN* based approaches is the complete sharing of computations between all ROI, which has repercussions for both the inference speed and the integration of global contextual information into each prediction.

3.3.2 Residual layer connections

As outlined in **section 2.1.2.1**, CNN are composed of stacked convolutional layers which are capable of extracting features which increase in complexity with network depth. Deeper networks have given rise to state-of-the-art results in many domains. However significant complications have come with these advancements; most notably the problem of performance saturation whereby accuracy begins to degrade beyond a certain network depth. He et al. [35] (among many others) have shown comprehensive empirical evidence of this phenomenon. In particular, a CNN was trained

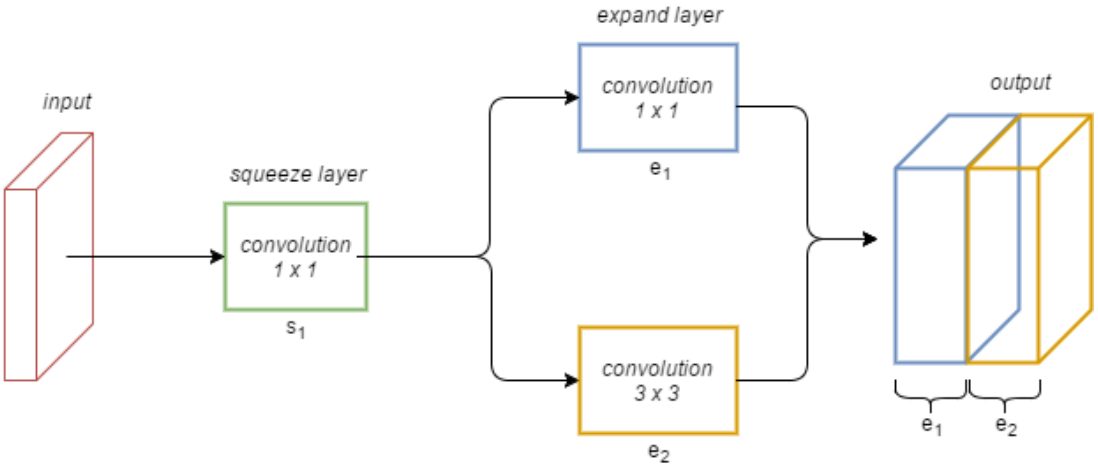


Figure 3.3: *Fire module*: first the input feature volume is passed to the *squeeze layer* where 1×1 convolutions are applied, with s_1 representing the number of filters. The output of this operation is passed through an *expand layer*, composed of two parallel convolutional layers. One is a 1×1 layer with e_1 filters and the other is a 3×3 layer with e_2 filters. The output of both is concatenated along the depth to generate the *fire module* output.

for classification on the *ImageNet* data set. Subsequently, several convolutional layers were added, and the network trained again. Counter-intuitively, the resulting accuracy was lower than that achieved with the shallower network. Evidently, the additional layers were not able to mimic identity mappings i.e. reproduce the input in each layer, and output predictions similar to those generated by the shallower network. Inspired by *shortcut* or *skip* connections seen in feed-forward neural network, residual connection were introduced as a way of hard-coding the identity mapping. Essentially, the feature maps from different layers are added element-wise, with the feature maps from shallower layers thus *'skipping'* several layers of linear mappings and non-linear activation function. Further to increases in accuracy achieved with this architecture, faster training convergence has been observed in many applications [35].

With *highway networks*, residual connections are compounded with a gating mechanism. For an input to a network layer x_i, the output y_i is typically an affine transform of the input $(H(x_i, W_{a,i}))$ with parameters $W_{a,i}$, passed through some non-linear activation function. In residual connections, the input to the following layer x_{i+1} is simply the sum of x_i and y_i. In highway networks, the transformation undertaken in a single layer is given by:

$$x_{i+1} = H(x_i, W_{a,i}) \cdot T(x_i, W_{h,i}) + x_i \cdot (1 - T(x_i, W_{h,i})) \tag{3.1}$$

Where $T(a, b)$ is a transformation with the same dimensionality as x and y, and is a set of linear transformations (a convolution layer) followed by a sigmoid function in application. Such connections allow for more complex regulation of information preservation over the depth of the network. This approach takes inspiration from

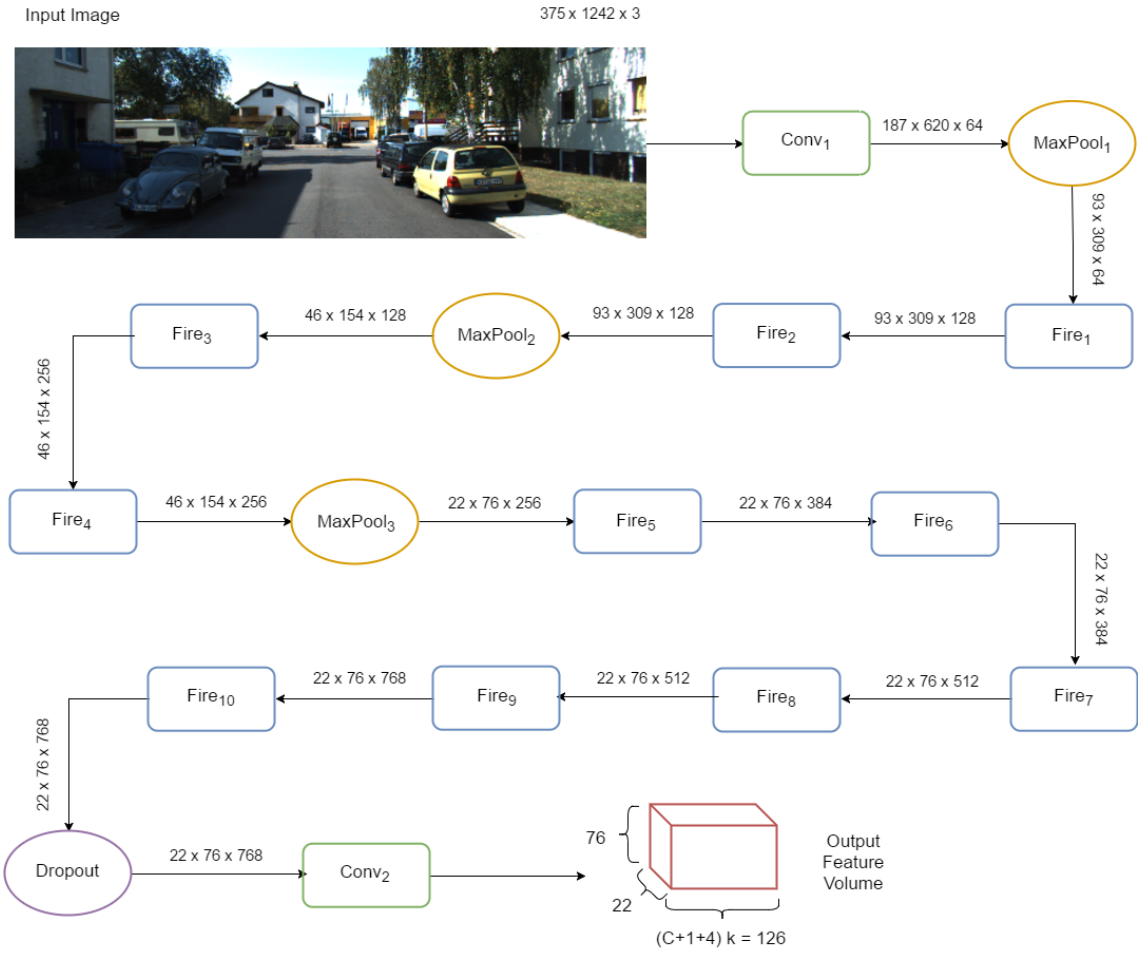


Figure 3.4: *SqueezeDet* structure with feature map dimensions after each operation.

the adaptive informational flow of long short-term memory (LSTM) networks, although the routing occurs across different feature representation spaces rather than a temporal dimension [36].

3.3.3 Novel modifications

In this study, three different network architectures are analysed. The first is the *SqueezeDet* network illustrated in figure 3.4 and described in detail in [21]. This network is an extension to the *SqueezeNet* model, developed as a compact, high-performance solution to object classification tasks [37]. In total, the model has approximately 2.5 million trainable parameters.

Despite empirical evidence suggesting that the incorporation of residual connections into many different architectures can ameliorate performance, this addition to the *SqueezeDet* network seems not yet to have been reported. Residual connections were however analysed in the context of *SqueezeNet* performance on the *ImageNet* classification task (where Top-5 accuracy improved by over 2%) Here, our second network is constructed by adding residual connections to bypass the second, fourth, sixth, eighth, and tenth fire modules (figure 3.5). In this case, the residual connections are not parameterised and thus there is no inflation of the model size compared to the original architecture.

Taking inspiration from *highway networks*, we also integrate gated residual connections into fire modules, as illustrated in figure 3.5, to generate the third architecture. Here, the input (x_i) and output ($H(x_i, W_{a,i})$) feature volumes are added elementwise as in plain residual connections. Next, however, the resulting feature volume is transformed by a single 3×3 convolutional filter with zero-padding to preserve the resolution. The output of this step is an activation map with the same width and height as the input and output of the fire module, but with a depth of one. This map is then passed through a sigmoid activation function and the output ($T(x_i + H(x_i, W_{a,i}), W_{h,i})$), which we refer to as the *gate mask*, contains values bounded between 0 and 1. The final output of the gated residual fire module is given by:

$$x_{i+1} = H(x_i, W_{a,i}) \lozenge T(x_i + H(x_i, W_{a,i}), W_{h,i}) + x_i \lozenge (1 - T(x_i + H(x_i, W_{a,i}), W_{h,i})) \tag{3.2}$$

where $\lozenge$ represents element-wise multiplication by the mask *broadcasted* across the depth of the feature volume. Thus, this extension to residual connections facilitates explicit spatial discrimination with regard to the residual routing of feature information. The adaptive mechanism for controlling residual connections may help preserve activations from shallow layers which can coupled to small objects. Another interesting property of this approach is that the mask is two-dimensional with bounded values, and thus the flow of information can be easily visualised and interpreted for a given input image. This may help alleviate the notion that the network is a *'black box'* which defies our understanding.

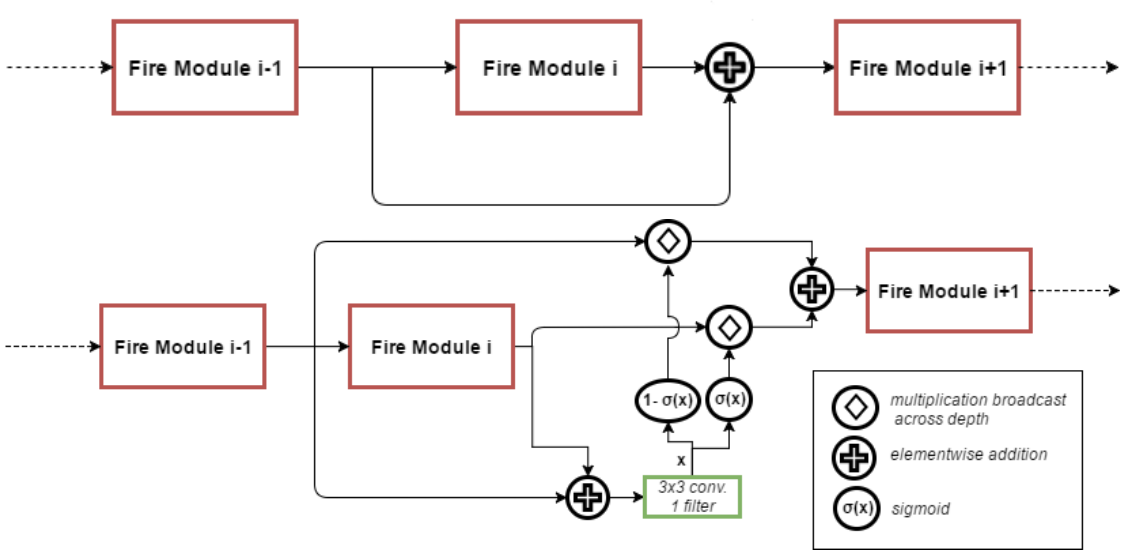


Figure 3.5: Top: Residual fire module. Bottom: Gated residual fire module.

In designing the gated residual fire module, we take careful consideration in the number of supplementary parameterised operations in order to retain to high inference speed of the original network. For this reason T is an affine transformation by a single convolution with the output passed through a non-linear function. Thus, for an input feature volume of size $W \times H \times D$, a further $3 \times 3 \times D$ trainable parameters are added. *Highway networks* do not feature the broadcast multiplication operation, instead opting for element-wise multiplication of the mask and feature volumes. To achieve this, one would require $3 \times 3 \times D \times D$ parameters for the gate mechanism.

3.4 Training

Here, we breakdown the multi-task loss function used to train our models. Furthermore, we descrbe the choice of hyperparameters and how these networks are intialised.

3.4.1 Loss Functions

Loss functions are an important consideration in the design of a CNN, as they are the sole metric that CNN are optimised for. In object detection, three sets of outputs are produced which are compared against ground-truths to generate a loss for a given parameter setting. The total loss for a single image is the sum of three separate losses, which are replicated from [21]:

$$L_{total} = L_{bbox} + L_{conf} + L_{class} \tag{3.3}$$

One set of network outputs is the bounding box offsets (or *deltas*), which encode the translation of the center of an anchor and the deformation of its width and height to generate a bounding box prediction. In this case, we compare the deltas predicted

by the network, $(\delta x_{i,j,k}, \delta y_{i,j,k}, \delta w_{i,j,k}, \delta h_{i,j,k})$ with the ground truth bounding box deltas ($\delta x^G_{i,j,k}$, $\delta y^G_{i,j,k}$, $\delta w^G_{i,j,k}$, $\delta h^G_{i,j,k}$). Here, k refers to the anchor index centered at the grid intersection point (i, j). To calculate the ground truth deltas, we apply the following formula:

$$\begin{aligned}
\delta x^G_{i,j,k} &= (x^G - \hat{x}_i)/\hat{w}_k \\
\delta y^G_{i,j,k} &= (y^G - \hat{y}_j)/\hat{h}_k \\
\delta w^G_{i,j,k} &= log(w^G/\hat{w}_k) \\
\delta h^G_{i,j,k} &= log(h^G/\hat{h}_k)
\end{aligned} \tag{3.4}$$

Where (x^G, y^G, w^G, h^G) denotes the coordinates of the ground truth bounding box. These deltas are calculated in a pre-processing stage to improve training speed, with each ground truth assigned to an anchor based on having the highest IOU. $(\hat{x}_i, \hat{y}_j, \hat{w}_k, \hat{h}_k)$ are the centre coordinates, width, and height of the anchor k centered at grid point (i, j). The bounding box regression loss is calculated as follows:

$$\begin{aligned}
L_{bbox} = \frac{\lambda_{bbox}}{N_{obj}} \sum_{i=1}^{W_o} \sum_{j=1}^{H_o} \sum_{k=1}^{K} I_{i,j,k} [(\delta x_{i,j,k} - \delta x^G_{i,j,k})^2 + (\delta y_{i,j,k} - \delta y^G_{i,j,k})^2 \\
+ (\delta w_{i,j,k} - \delta w^G_{i,j,k})^2 + (\delta h_{i,j,k} - \delta h^G_{i,j,k})^2]
\end{aligned} \tag{3.5}$$

$I_{i,j,k}$ is a mask calculated in the pre-processing stage, with values 1 if anchor k centered at grid point (i, j) has the largest overlap (IOU) with a ground truth, and 0 otherwise. λ_{bbox} is a hyperparameter to regulate the contribution of the bounding box regression loss to the overall loss. N_{obj} is the number of objects per image, and is used to normalize the loss.

The second network output is the object confidence score, which is interpreted as an estimate of the probability that an object lies in that anchor multiplied by the IOU between the predicted and ground truth bounding boxes:

$$L_{conf} = \sum_{i=1}^{W_o} \sum_{j=1}^{H_o} \sum_{k=1}^{K} \left[\frac{\lambda^+_{conf}}{N_{obj}} I_{i,j,k} (\gamma_{i,j,k} - \gamma^G_{i,j,k})^2 + \frac{\lambda^-_{conf}}{W_o H_o K - N_{obj}} (1 - I_{i,j,k}) \gamma^2_{i,j,k} \right] \tag{3.6}$$

$\gamma_{i,j,k}$ is the predicted object confidence score returned by the network and $\gamma^G_{i,j,k}$ simplifies to IOU^{pred}_G. The term on the left is the contribution of the positive anchors to the loss (anchors which have been assigned to ground truths), while the term on the right is to penalize negative anchors (anchors that should return zero object confidence). Again, λ^+_{conf} and λ^-_{conf} are hyperparameters used to regulate the contribution of these losses.

The last network output is the classification score, that is interpreted as an estimate of the probability that the object inside a bounding box belongs to one of the C

classes. The classification loss between ground truth labels (l_c^G) and the classes predicted by the network (p_c) is given by the cross-entropy:

$$L_{class} = \frac{-1}{N_{obj}} \sum_{i=1}^{W_o} \sum_{j=1}^{H_o} \sum_{k=1}^{K} \sum_{c=1}^{C} I_{i,j,k} \left(l_c^G(-log(p_c)) + (1 - l_c^G)(-log(1 - p_c)) \right) \quad (3.7)$$

Note that the classification loss is different from that described in the original paper which is clearly erroneous in its omission of the minus sign [21]. In training, the total loss is calculated for each image, and the average value is used in optimising the network.

3.4.2 Hyperparameter selection

The majority of hyperparameters used are set to values described in [21]. The mini-batch size used is 20. In order to adjust the influence of the bounding box and confidence score loss functions, three weighting hyperparameters are set as $\lambda_{bbox} = 5$, $\lambda_{conf}^{+} = 75$, $\lambda_{conf}^{-} = 100$. Weight decay is also added to the loss equation with a factor of 10^{-4}. The parameter $\epsilon = 0.0001$ is used to avoid dividing by zero. The model is trained to recognize three categories of objects (cars, pedestrians, and cyclists) and for each grid intersection point in the input image, predictions for 9 anchors are made. Thus approximately $15,000$ detection proposals are made for each image. Unlike the original implementation described by [21], the *Adam* optimizer is chosen as the updating algorithm with an initial learning rate of 0.0001. Learning rate decay is implicit within *TensorFlow's Adam* implementation.

3.4.3 Parameter initialization

Initialising a network's parameters to zero lead to issues with the backpropogation of error. It is therefore common to sample initial values for parameters from a Gaussian distribution with zero mean and a standard deviation smaller than one. Low initial values can introduce the problem of gradients that prohibit quick learning. On the other hand, large initial values can lead to unstable training. With Xavier initialization [38], the weights are initialised using a Gaussian distribution with zero mean and a variance specified by the following formula:

$$Var(W) = \frac{1}{n_{in}} \quad (3.8)$$

Where n_{in} is the number of input neurons. It also ensures that all neurons have the same output distribution initially and it has shown to improve convergence [38]. The above formula follows from a series of assumptions and variance properties. First, the output of a linear network is given by $s = \sum_{i=1}^{n} W_i x_i$, then assuming that W_i and x_i are independent, the variance of s is:

$$Var(s) = Var(\sum_{i=1}^{n} W_i x_i) = \sum_{i=1}^{n} Var(W_i x_i)$$
$$= \sum_{i=1}^{n} [E(W_i)]^2 Var(x_i) + [E(x_i)]^2 Var(W_i) + Var(W_i)Var(x_i) \tag{3.9}$$

If both input x_i and weights W_i have zero mean:

$$Var(s) = \sum_{i=1}^{n} Var(W_i)Var(x_i) \tag{3.10}$$

Finally, assume that all x_i are identically distributed and that all W_i are identically distributed. Then if we want s to have the same variance as all inputs x, the variance of every weight W has to be $1/n$. This means that the variance of the output is the same as the input but scaled by $Var(W)/n$, such that $Var(W)/n = 1 \Leftrightarrow Var(W) = 1/n = 1/n_{in}$ [38]. All biases are initialised to zero except for those in the gated residual connection (where -1 is instead taken). This strategy is employed to encourage the flow of information in the residual channels as outlined in [36].

3.5 Testing

While training, checkpoints storing all the weights and bias values are periodically saved to disk. To test a network, variables are restored from a checkpoint, testing images are fed to the network and a filtering algorithm is applied to the network's output. The first step in filtering is to calculate the confidence of each detection, which is given by the product of the object confidence and classification scores. The top k predictions with the highest confidences in each image are retained. NMS is then applied to these detections within each image and class with a threshold of 0.4 (refer to figure 3.6). This filtering algorithm returns the bounding boxes, confidences and classes for the final detections in each image.

3.6 Conclusion to Data & Methods

Artificial augmentation of the data-set is advised as a way to prevent over-fitting of the network, particularly for environments like the *KITTI* object detection set which are limited in size compared to large-scale sets like that available from the *ImageNet* 2012 classification challenge. The *SqueezeDet* architecture is chosen as the main focus of this work for its promising performance and inference time. We suspect that the addition of residual and gated residual connections to this network may improve its detection capabilities. The training and testing procedures for these networks are replicated from [21] for comparison purposes.

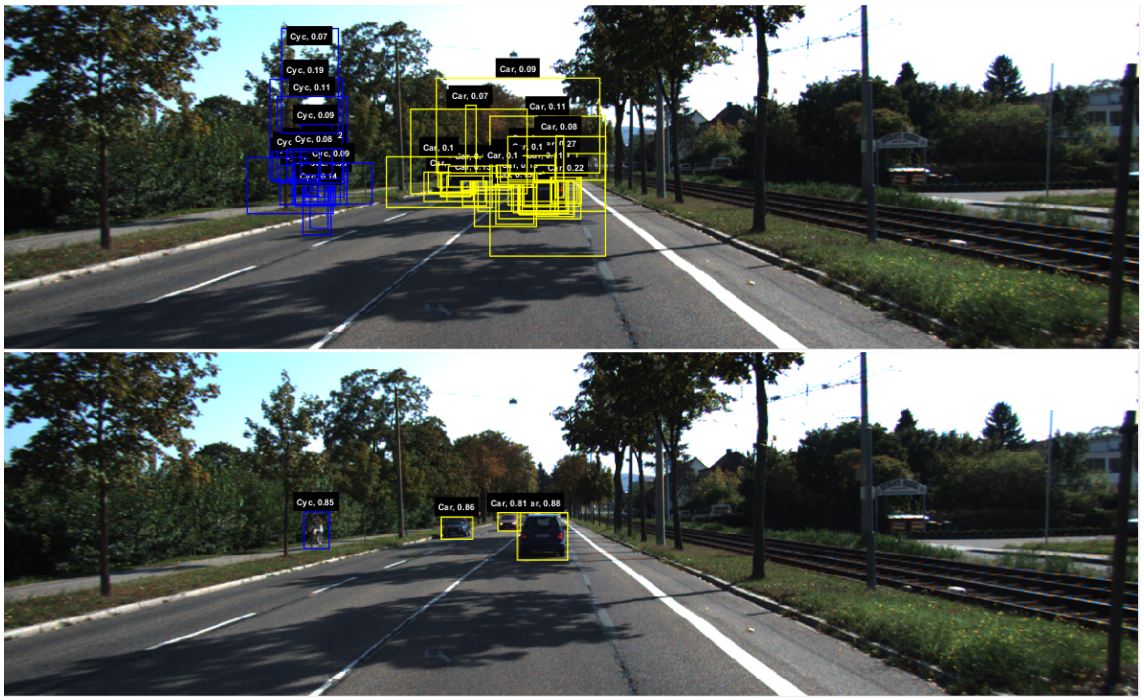


Figure 3.6: Top: 64 detections with the highest confidence scores outputted from a residual squeeze network trained from scratch. Bottom: The same detections filtered by NMS and a confidence threshold of 0.5.

4

Results

In this thesis, two approaches to network training have been implemented. The first involves training solely on the *KITTI* object detection data-set, while the other also includes a pre-training stage of optimising the network for image classification. The first training procedure is used to optimise all three networks (which we refer to as *SqueezeDet*, *Residual SqueezeDet*, and *Gated SqueezeDet*) and this is followed by a comparative evaluation. The second training procedure is then applied to *SqueezeDet* and *Gated SqueezeDet* and its performance is contrasted against the results reported in [21].

4.1 Training strategy I: training solely on KITTI

The first training strategy involves optimising the three different network architectures on 6981 images from the *KITTI* object detection training set for $100,000$ iterations (approximately 286 epochs), with the parameters outlined in **section 3.4.2**. Each run took approximately 27 hours to complete. Every 200 iterations, the bounding box, object confidence, and classification losses were recorded for mini-batches from the training set. As shown in figure 4.1 (a), the total of the three losses over the training data is near indistinguishable between the different models. To monitor over-fitting, a validation set was constructed with the other 500 pictures from the training set. The losses on mini-batches from the validation set were recorded at the same frequency (illustrated in figure 4.1 (b)). For all networks, we see a rapid decrease in validation loss within the first quarter of training. Beyond this point, the validation loss gradually increases, more significantly for *Residual SqueezeDet* than the other two models. These results suggest that some of features learned by the networks beyond after $20,000$ iterations were specific to the training set (i.e. over-fitting on the training data had occurred).

Counterintuitively, performance improves substantially between 20 and 100 thousand iterations in terms of *KITTI*'s evaluation metrics on the validation set (figure 4.2), despite the increase in validation loss. Examining the contribution of each of the individual losses (figure 4.1 (d)), we see that the object confidence and classification losses continue to decrease while the bounding box loss increases after approximately $20,000$ iterations. Clearly, the total loss function does not accurately represent the end performance of an algorithm. It could be the case that the error incurred by the localisation of objects is over-weighted compared to the other losses. This seems appropriate when we recall that detections are recognised as true

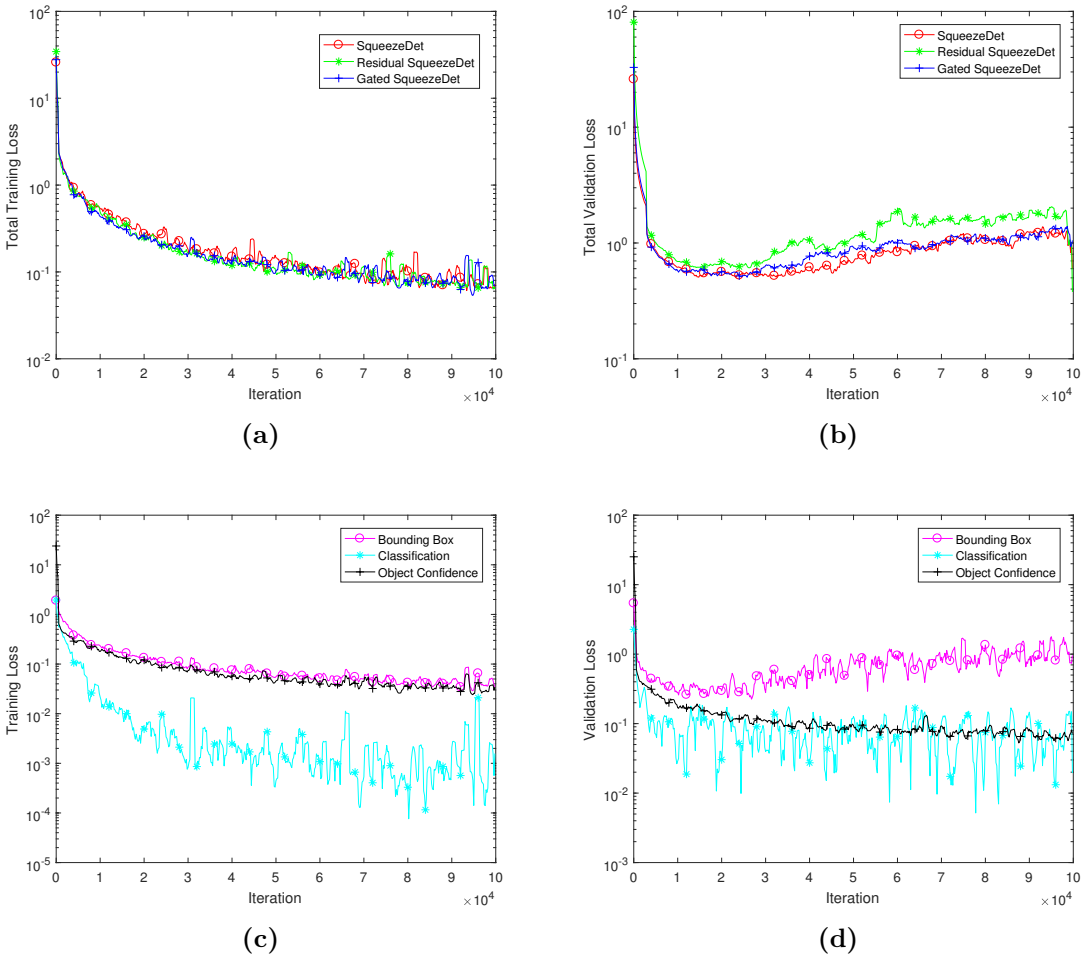


Figure 4.1: Loss curves for the three models trained solely on *KITTI* data. (a) Total training loss. (b) Total validation loss. (c) Individual training losses for *Gated SqueezeDet*. (d) Individual validation losses for *Gated SqueezeDet*

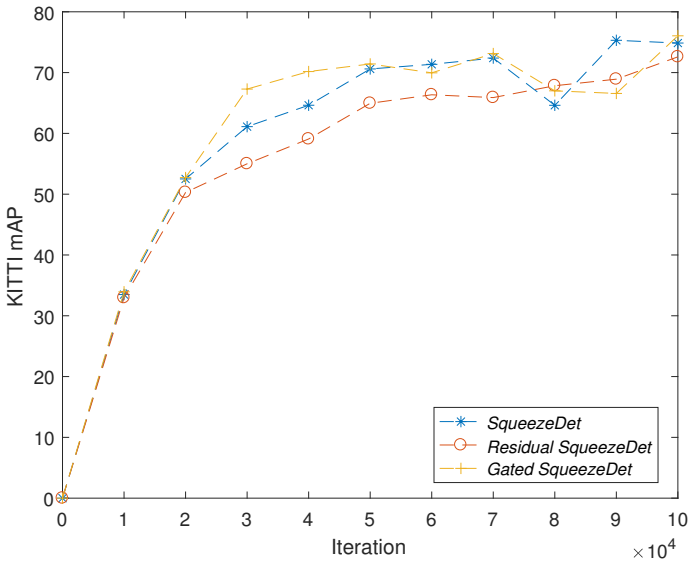


Figure 4.2: mAP over iterations of training on the *KITTI* validation set of 500 images.

positives only if they surpass a fixed IOU threshold with a ground-truth, while the object confidence and classification scores play an important role in the filtering of detections, and the generation of precision/recall curves for example. One possible cause of the localisation over-fitting could be the fact that the anchors are designed specifically for the data-set.

	car			pedestrian			cyclist			mAP
Network	E	M	H	E	M	H	E	M	H	
SqueezeDet	89.1	85.2	77.0	73.3	69.1	61.6	76.0	76.4	70.2	75.3
Residual SqueezeDet	84.5	83.9	74.6	79.1	74.0	65.1	61.3	64.9	65.7	72.6
Gated SqueezeDet	82.9	84.0	73.3	78.4	72.3	63.7	76.4	76.9	76.5	76.1

Table 4.1: Average precision for each object class in the different difficulty categories (E: easy, M: moderate, H: hard) measured on the validation set for each network trained solely on *KITTI*. Results are reported for the networks at their highest mAP.

Table 4.1 shows the final performance of each network on the validation set quantified using *KITTI*'s evaluation metrics (see **section 3.1.1** for more details). Here we see that the addition of gated residual connections to *SqueezeDet* have led to an mAP increase of 0.8%. Surprisingly, these results suggest that without gating mechanisms, standard residual connection lead to decrease in mAP (nearly 3%). At more fine-grained level, *SqueezeDet*, *Residual SqueezeDet*, and *Gated SqueezeDet* perform best in one category across all difficulty levels. It is important to state here that a relatively small number of examples are used to do this comparative study, a result of the limited amount of labelled training data compared to other popular data-sets. Samples from the test set with predictions generated by the 3 networks are shown in figure 4.3 and more detailed performance metrics can be found in **appendix A**.

(a) *SqueezeDet* (b) *Residual SqueezeDet* (c) *Gated SqueezeDet*

Figure 4.3: Cropped images from the *KITTI* test dataset with detections (over a confidence threshold of 0.5) generated by the 3 networks trained from scratch.

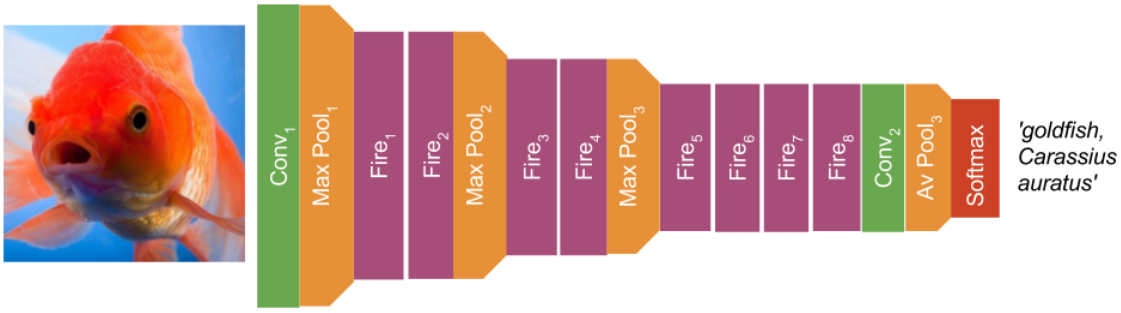


Figure 4.4: *SqueezeNet* model used in the pre-training stage, optimisation for *ImageNet* classification.

4.2 Training strategy II: training on ImageNet and KITTI

Here we describe the second training strategy undertaken in this work: pre-training on *ImageNet* and then applying transfer learning for object detection on *KITTI* data.

4.2.1 Pre-training

In this section we describe the pre-training applied to the *SqueezeNet* architecture. This network is composed of a convolutional layer, 8 fire modules, a dropout layer, and a second convolutional layer, with the final classification scores given by global averaging pooling of the output feature maps passed through a *softmax* activation function (figure 4.4). Here, fire modules 2, 4, 6, 8 are replaced by *gated fire modules* to construct *Gated SqueezeNet*. Due to the protracted training time, and the model's poor performance when trained from scratch (as reported in **section 4.1**), *Residual SqueezeNet* was not considered for pre-training. As described in **section 3.4.2**, the 2012 *ImageNet* Large Scale Visual Recognition Challenge data-set is used for pre-training, composed of approximately 1.3 million training images and 50,000 validation images. Each sample focuses on an instance of one of 1,000 classes. The networks was trained for $1,000,000$ iterations with a mini-batch size of 64 images, using the *Adam* optimizer and an initial learning rate of 0.0001. The cross-entropy between the predictions and one-hot labels is taken as the loss function.

In figure 4.5, we see cross-entropy and top-1 accuracy for the *SqueezeNet* and *Gated SqueezeNet* models over iterations for mini-batches from the training and validation data-sets. Top-1 accuracy refers to the percentage of samples whereby the network's highest scoring class prediction matches the label. This metric is often used in conjunction with top-5 accuracy (the percentage of samples whereby one of the network's top 5 highest scoring class matches the label). After approximately 49 epochs of training, both networks reach a top-1 accuracy of close to 50% and 45% for the training and validation data respectively. That being, the rate at which accuracy increases is greater for *Gated Squeeze* in the first quarter of training. These

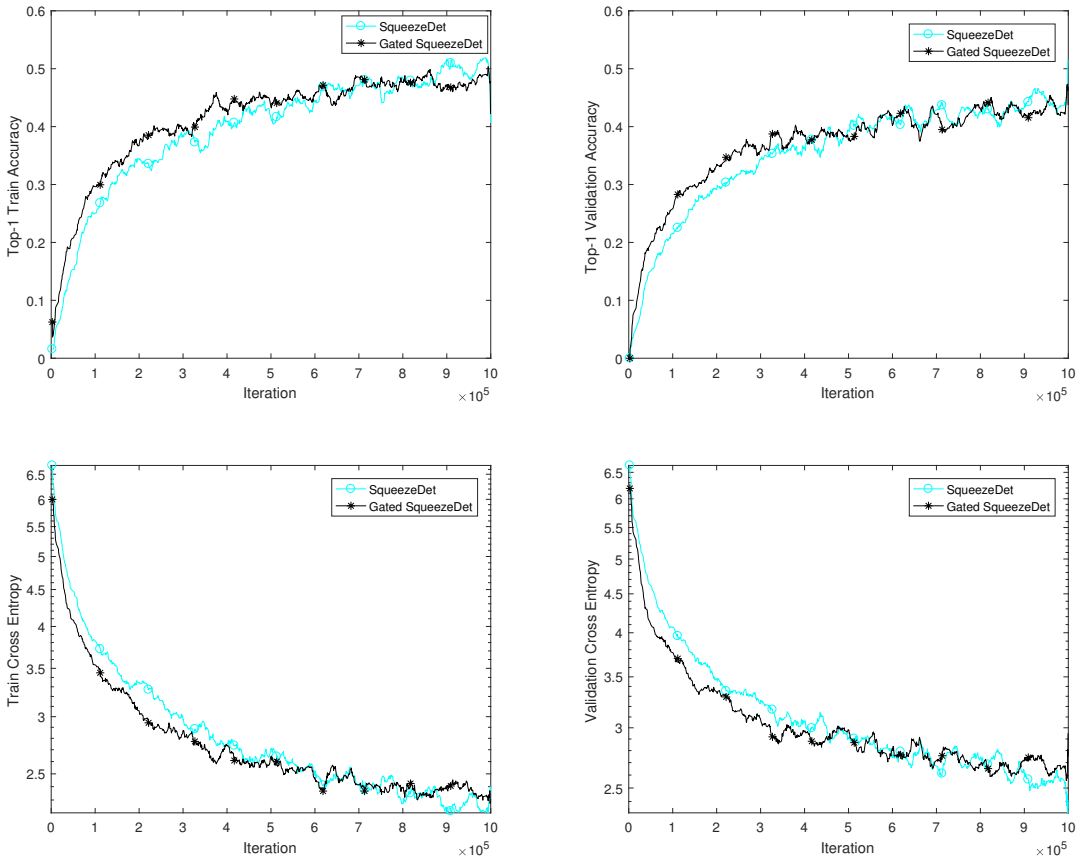


Figure 4.5: Pre-training the *SqueezeNet* and *Gated SqueezeNet* models on the *ImageNet* 2012 classification challenge data-set.

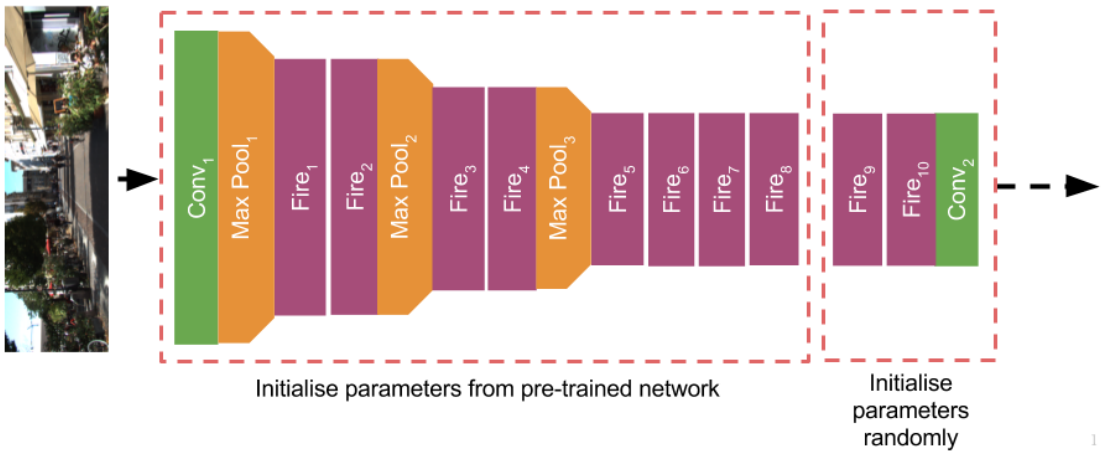


Figure 4.6: *SqueezeDet* model used in secondary training: object detection on *KITTI*.

results can be compared with the 57.5% top-1 validation accuracy outlined in the original *SqueezeNet* publication, although this model was trained for approximately 70 epochs with a batch size of 512 using the *Momentum* optimiser [37]. In neural network training, the usual procedure is to cease training when the validation loss begins to consistently rise above a certain acceptable threshold. Due to the lengthy training time (approximately one week), network optimisation was not continued after 1 million iterations despite the fact that there were no signs of the network over-fitting to the training data.

4.2.2 Transfer Learning

In figure 4.6, we see how transfer learning is realised using a pre-trained *SqueezeNet* network. Explicitly, the parameters learned for the first convolutional layer ($Conv_1$) and the first eight *fire modules* are used to initialise the same layers in the *SqueezeDet* model. A further two *fire modules* and a convolutional layer ($Conv_2$) are added to complete the network, with parameters initialised using the Xavier method (see **section 3.4.3**). The same procedure is used in applying transfer learning to the *Gated SqueezeDet* model. It is important to note that only half of the $7,480$ training images are used to optimise the *SqueezeDet* and *Gated SqueezeDet* networks in secondary training, with the other half utilised as a validation set. There are several ways to now approach secondary training. Depending of the size of the data-set, it is advised to *freeze* a subset of layers [39]. By freezing, we refer to fixing the parameters of a layer i.e. the weights and biases are not updated during secondary training. In theory, as *KITTI* is considered a relatively small data-set, it should be beneficial to freeze all, or most, layers which are initialised for the pre-trained model.

As advised in the literature, the first convolution layer and the first eight *fire modules* were frozen during secondary training on the *KITTI* data-set. The resulting performance for the *SqueezeDet* and *Gated SqueezeDet* models can be observed in table 4.2 and figure 4.7. Here, we see that pre-training the networks on *ImageNet*

and subsequently training on a smaller *KITTI* data-set produces poorer mAP than training solely on a larger set of *KITTI* image for both models. Following this, it was decided to undertake secondary training with only $Conv_1$ frozen, a technique used by the original author of *SqueezeDet*. This approach leads to a 30 % increase in mAP as shown in table 4.3 and figure 4.7. However, the performance does not match that achieved in [21]. It is difficult to compare these results, as the top-1 validation accuracy reached by [21] in *ImageNet* is 12.2 % greater than ours. Relatively speaking however, it can be seen that our implementation of *SqueezeDet* performs better than *Gated SqueezeDet* with a 5 % difference in validation mAP.

	car			pedestrian			cyclist			mAP
Method	E	M	H	E	M	H	E	M	H	
Gated SqueezeDet	49.84	41.65	36.1	45.01	39.18	35.61	25.2	27.18	26.75	36.28
SqueezeDet	50.8	42.98	36.99	49.47	41.46	37.76	25.66	28.93	26.81	37.87

Table 4.2: Average precision for each object class in the different difficulty categories. The models are measured on the validation set for the two pre-trained networks where the three last layer were trained and the rest were retrieved from *ImageNet*.

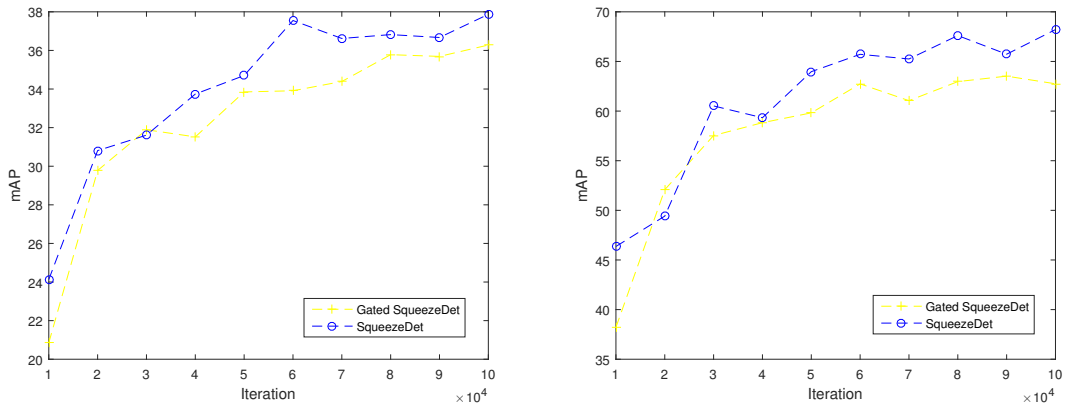


Figure 4.7: mAP over iterations of training on the *KITTI* validation set of 3,740 images. Left: both networks have the first convolution layer and the first 8 *fire modules* frozen from the networks trained on *ImageNet*. Right: both networks have just the first convolution layer frozen and the first 8 fire modules initialized from the network trained on *ImageNet*

4.3 KITTI evaluation submission

Following both training procedures, it was decided to further evaluate the novel *Gated SqueezeDet* model which was not pre-trained. Performance on the *KITTI* test data can be observed in table 4.4. As can be seen, its detection capacity is far from the state-of-the-art results, with a mAP of just 13.2%. Considering the performance gap with the results in the previous sections, *Gated SqueezeDet* has

Method	car			pedestrian			cyclist			mAP	FPS^2
	E	M	H	E	M	H	E	M	H		
SqueezeDet[1]	90.2	84.7	73.9	82.9	75.4	72.1	77.1	68.3	65.8	76.7	57.2
Gated SqueezeDet	81.37	75.8	67.29	67.95	59.28	54.5	55.82	55.72	53.83	63.52	58
SqueezeDet (ours)	82.26	80.34	68.42	73.63	64.03	57.36	67.17	62.18	58.56	68.21	63

Table 4.3: Average precision for each object class in the different difficulty categories. The three models are measured on the validation set for the two pre-trained networks when the first convolution layer is frozen and the 8 following fire modules were initialized with the network trained in *ImageNet*. 1. Taken from [21]. 2. Inference speed was recorded on a unit with a Titan X GPU running Windows 10.

clearly overfitted to type of data observed in both the training and validation sets to a considerable degree. It's important to note that test results are not given in [21] and evaluation can only be completed once, so no comparison can be made with the original architecture.

Method	car			pedestrian			cyclist			mAP	FPS
	E	M	H	E	M	H	E	M	H		
iDST-VC	90.88	90.55	81.04	-	-	-	-	-	-		0.25
Tu Simple	90.77	90.33	82.86	86.78	77.04	72.40	81.38	74.26	64.88	80.07	0.625
RRC	90.61	90.22	87.44	84.14	75.33	70.39	84.96	76.47	65.46	80.55	0.27
Gated SqueezeDet	17.44	17.46	15.05	16.83	14.37	13.65	9.49	7.46	7.04	13.2	58

Table 4.4: Methods with the state-of-the-art performance in each of the object categories on *KITTI* object detection testing data-set by June 2017.

4.4 Conclusion

When trained solely on the *KITTI* data-set, the addition of novel *gated connections* improves performance on the validation set by close 1% mAP. However, with inclusion of a pre-training stage (image classification on *ImageNet*), object detection performance is considerable depreciated compared to the original architecture. Furthermore, performance on the *KITTI* test data indicatse that *Gated SqueezeDet* has experienced considerable overfitting on the training data.

5
Discussion

In this section, we interpret the results outlined in **section 4**. The work done in this thesis highlights several interesting research questions in terms of network structure, data, task, and extensions, which are discussed here. It is important to preface this section by stating that a major limiting factor to the scope of this project was the computational complexity of training and evaluating deep learning algorithms.

5.1 Analysis of Results

As described in **section 4.1**, our *Gated SqueezeDet* network outperforms both *SqueezeDet* and *Residual SqueezeDet* in terms of mAP on the validation set of 500 images. This is achieved with just a 0.7% inflation in the number of parameters compared to the original architecture. One theory as to why the addition of gated residual connections creates this performance disparity relates to the size of objects when projected onto the image plane. Cyclist and pedestrian detection made a major contribution to *Gated SqueezeDet's* increased performance. These object classes are generally much smaller than cars but one must also take into account that instances of the cyclist class in the *KITTI* data-set are much more uniformly distributed over distance from the ego-vehicle (see figure 5.1). Across all object categories, *Gated SqueezeDet* shows a greater capacity to recall objects more than 60 metres away from the ego-vehicle. *Gated SqueezeDet* is the only model to recall instances further than 80 metres from the ego-vehicle as shown in figure 5.2, although there are very few ground-truths at this range. Many authors have suggested that smaller objects in the input image result in high activations in shallower layers of the network. They have attempted to tackle the issue by applying classifiers directly on intermediate feature maps [23] [25] [40]. As shown in the left column of figure 5.3, the *Gated SqueezeDet* network has learned to discard a large portion of the information from two of the fire modules in response to the given input image, instead opting to route information through the residual connections. In effect, activations coupled to small objects may be preserved to a greater extent by the final layer of the network. It must be stated that this line of thinking is highly speculative however, and requires more stringent investigation. It does raise the idea that pedestrian and cyclists detection may be improved for a *SqueezeDet* model with fewer *fire modules*.

Applying some sort of pre-training step is the default approach used in CNN-based object detectors. This technique usually leads to an increase in performance on the secondary task. *SqueezeDet* achieves a higher mAP than *Gated SqueezeDet* on

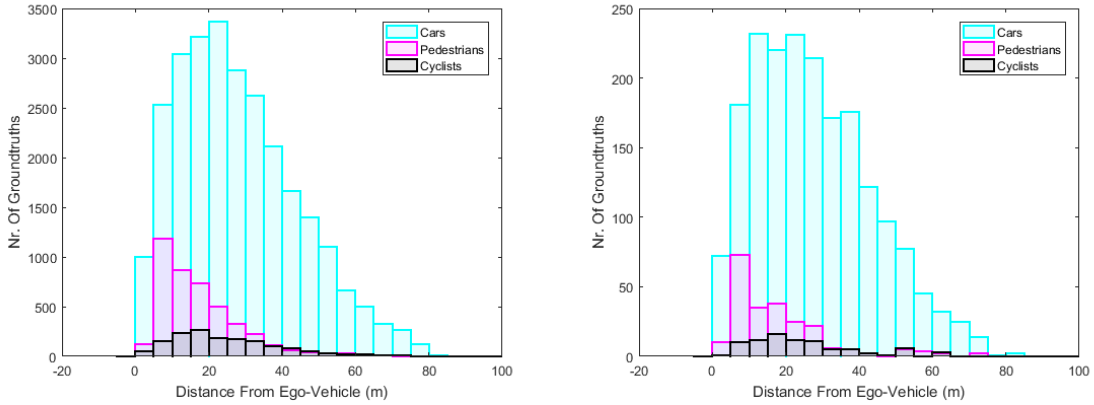


Figure 5.1: Distribution of ground-truths over distance from ego-vehicle for the training set of 6981 images (left) and the validation set of 500 images (right).

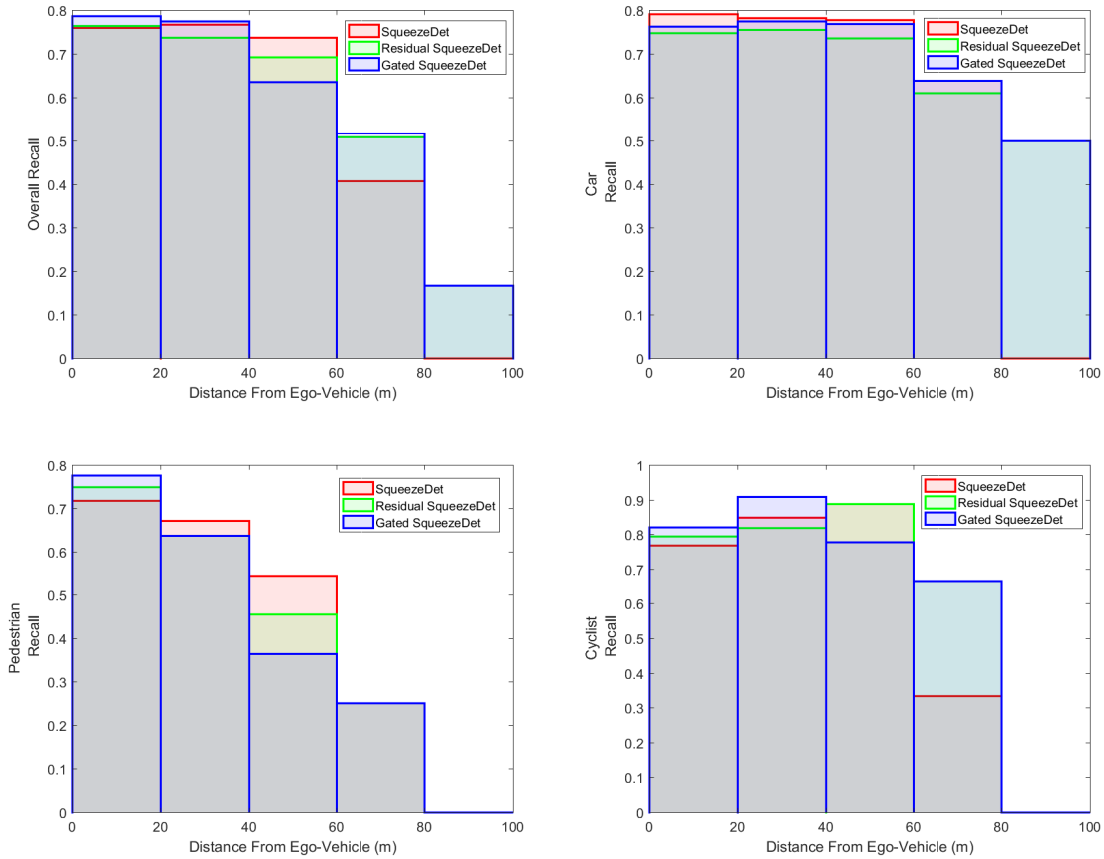


Figure 5.2: Distribution of recall over distance from ego-vehicle in each object category for the validation set of 500 images.

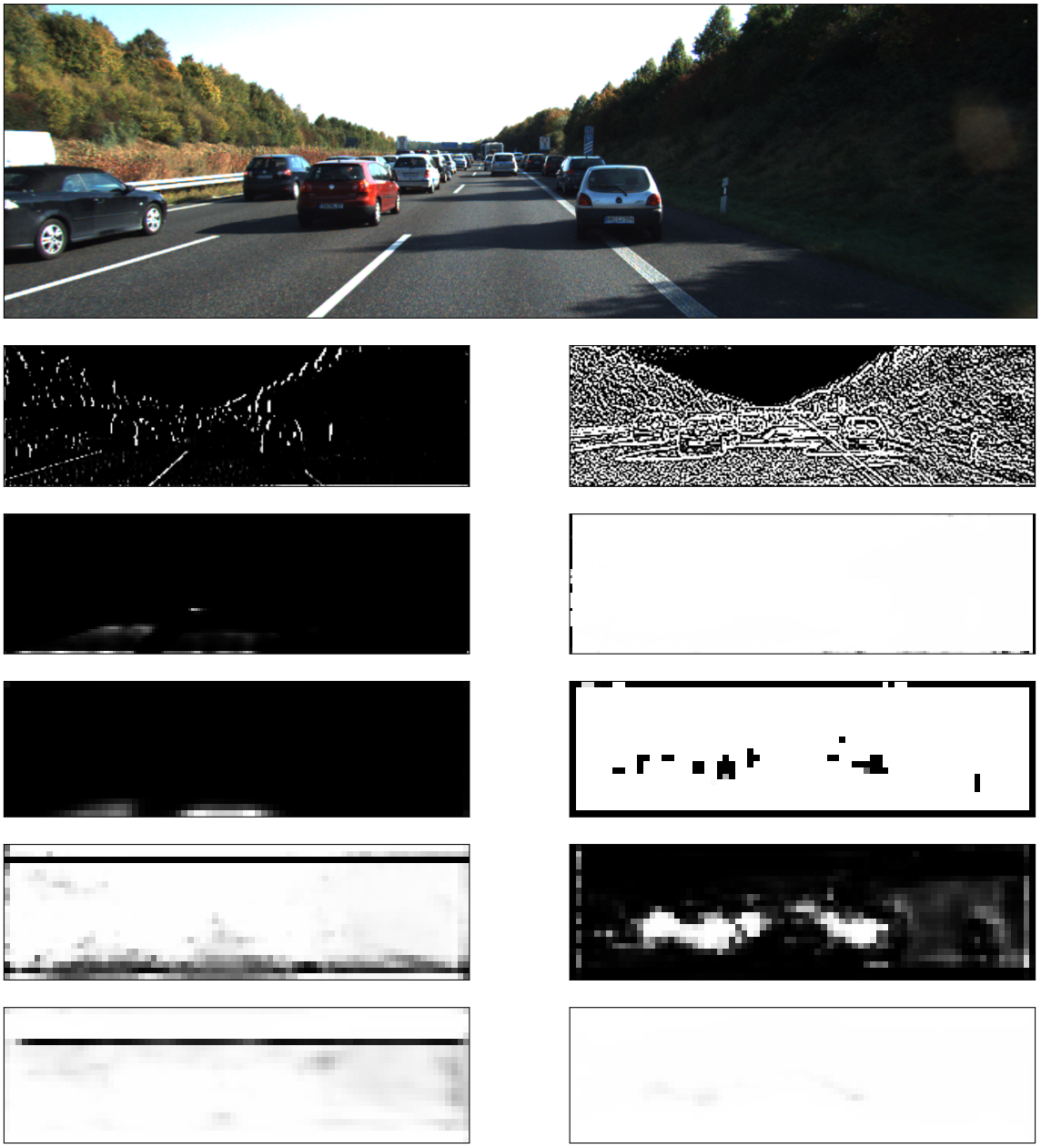


Figure 5.3: A sample test image (top) with the associated *gate masks* generated in each of the *Gated Fire Modules* for the network trained solely on *KITTI* (left column) and the network which was trained both on *ImageNet* and *KITTI*. Black pixels illustrates areas of the feature maps which have been completely routed through the residual channel, whereas white pixels described regions where information has come solely from the previous fire module.

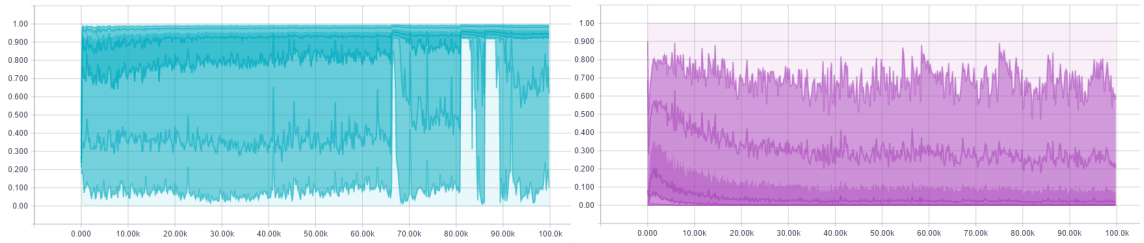


Figure 5.4: Distribution of the values in the *gated mask* in a mini-batch (y-axis) over training iterations (x-axis). Each line represents a percentile in the distribution, from top to bottom: maximum $\mu + 1.5\sigma$, $\mu + \sigma$, $\mu + 0.5$, μ, $\mu - 0.5$, $\mu - \sigma$, $\mu - 1.5$, minimum, where μ, are the mean and standard deviation respectively. Left: network which has not been pre-trained. Right: network which has been pre-trained.

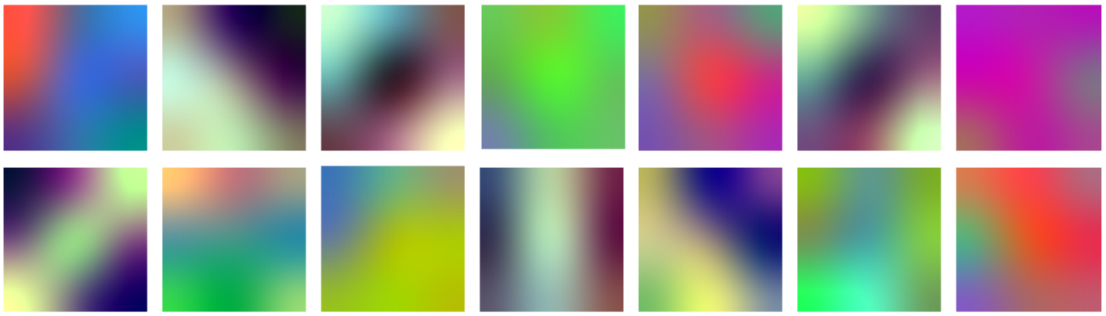


Figure 5.5: Visualisation of some of the filters in the first layer of our trained *Gated SqueezeDet* model. Note that the filter are of size $3 \times 3 \times 3$ but have been smoothed for illustrative purposes.

KITTI using this approach. This runs contrary to the results observed when the models are trained solely on *KITTI*. One contribution to *Gated SqueezeDet's* underperformance may be the sigmoid activation function which is used to generate the mask in the gating mechanisms. As shown in figure 5.3, masks generated for the pre-trained and *KITTI*-trained networks are mostly saturated i.e. the pixels take values of either 0 or 1 and thus the partial derivative tends to 0. As shown in 5.4, the gradient through the sigmoid function appears to have vanished during primary training and thus the gating mechanisms do not adapt as much to the secondary task. Clearly further evaluation and consideration of the design this unit is required. The lack of time prevented further training of *SqueezeDet* and *Gated SqueezeDet* on *ImageNet* which may explain the performance disparity when compared to the results outlined in [21].

Despite the advancements in detection performance achieved by CNN, these methods are often seen as *'black boxes'*. There remains much to be understood about how these networks reason in generating predictions e.g. what features of a pedestrian are most important in discriminating it from the background or other objects of interest. A rudimentary way of illustrating the features extracted by our CNN

can be achieved by visualising the filters of the first convolution layer. These filters are applied directly to the input image so they can be can be depicted in the same colour space. As shown in figure 5.5, local combinations of colours seem to play an important role in the detection process, and not just edges of specific orientations. It is no surprise then that our network trained on colour images performs significantly poorer on the same images in grayscale (explicitly, a drop in mAP from 76 to 41% on the validation set is observed). Other investigative tools include occlusion test, whereby we would block certain parts of an object from view and see how it affects performance, and abstraction test, in which we perform testing on synthetic data. Finally, while the focus of this study was the *SqueezeDet* architecture, an invetigation of the contemporary CNN-based object detection methods should be undertaken. An implementation in *Tensorflow* of the most important networks should lead to a better understanding of the contemporary architectures and their main characteristics. This will allow us to carry out a fair comparison between *SqueezeDet*, and its modifications, and other models such as *YOLO* or *R-FCN*. Another interesting research question is how the integration of the novel *gated connections* developed in this work would effect the performance of these models.

5.2 Network Structure

Humans often rely on a binocular view of an object over time in characterising it. It follows from this that an interesting extension to our model would be the inclusion of some notion of temporality. There are several ways this could be incorporated; possibly the simplest being to feed the network several sequential images in one forward pass and expand the network to incorporate 3D convolutions. Recurrent neural units offer a more complex way for CNN to process multiple temporally correlated images. *Long short-term memory* (LSTM) is a type of recurrent neural network unit which has been shown to better capture long-term temporal dependencies in data. This structure features input, forget, and output gates analogous to reading, deleting, and writing memory operations respectively [41]. One issue with this approach is the significant inflation in training time though. Of course, extraneous input images do not have to come from distinct instances of time, but can come from spatially displaced cameras analogous to the stereo vision apparatus of many animals. Images from two horizontally removed cameras are available in the *KITTI* object detection data-set, however data from only one of them was used in this study.

The main motivation of this work was to analyze the *SqueezeDet* model described in [21], and evaluate the effect on performance of two structural modifications. For this reason, it was important to replicate the original model as close as possible, and the majority of the hyperparameters were duplicated from the original paper. That being said, only a limited model design space exploration was undertaken in [21]. Here *KITTI* object detection mAP was reported for finer image resolution, increased number of anchors per grid point, and a greater number of trainable parameters. For the *SqueezeNet* model in the context of image classification, the change in performance related to altering the squeeze ratio (the extent to which the feature representation volume is compressed by the squeeze layer in the fire modules) and

Figure 5.6: Examples of redundant detections not removed through filtering by NMS.

the number of filters in the 3×3 expand layers was also studied in [37]. As with all deep learning networks, the hyperparameter space is very high dimensional and significant tracts of it are unexplored, with design decisions often boiling down to heuristics.

While significant focus has been placed on the input and structure of the CNN, post-processing of the over 15,000 detections generated per image presumably play an important role. Despite this, non-maximum surpression (NMS) has remained the default filtering algorithm for most contemporary CNN-based object detection systems. As shown in 5.6, redundant detections still contribute to precision degradation even after NMS. *Soft NMS* differs from the original algorithm by only a single line of code, yet delivers close to a 2% improvement in mAP in the PASCAL VOC 2007 detection challenge when appended to both *Faster R-CNN* and *R-FCN*. This is done while retaining the computational complexity of standard NMS [42]. The distinction with this algorithm is that a detection which has an IOU with another detection of the same class greater than some threshold has its confidence score penalised. Work has also been invested into developing a CNN-based NMS replacement, allowing for complete end-to-end training of the detection and filtering stages. With *T-Net*, a 4-layered CNN, detections are re-scored rather than discarded, with experiments showing improved precision and recall on a pedestrian detection data-set [43].

5.3 Data

The *KITTI* object detection data-set consists of a training set of 7481 images, which is quite limited in size compared to the training sets found for other image recognition challenges. This is one of the motivations behind the use of transfer learning and data augmentation in this study. Despite its size, *KITTI* presents a variability in driving environments with depictions of motorways, cities and secondaries roads. That being said, the performance of our networks trained solely on *KITTI* data does not generalise well to new data. In figure 5.7, we see that our *Gated SqueezeDet* model trained solely on *KITTI* data gives very inconsistent predictions on *Udacity* images, even when those images are downsampled to match *KITTI* res-

olution. Understandably, performance appears to be better on images which liken those found in *KITTI* (e.g. the middle and bottom images). For the top image, the network generates only pedestrian false positives. Here, the poor recall may come from the fact that the difference in camera placement is more pronounced for nearby objects (*Udacity* camera is fixed to the car's dashboard while the *KITTI* camera is roof-mounted). There is a clear need for more work in quantifying this phenomenom.

Another issue is that of *KITTI*'s bounding boxes, which were annotated through crowd-sourcing. In the case of occluded objects, the ground truth labels are designed to encapsulate all parts of the article. Thus the annotator must estimate the position of object parts blocked from view. This leaves room for annotation error. It also implies that networks trained on *KITTI* can not be tested on other data-sets such as those available from *Udacity*, as their bounding boxes cover only what is visible of an object (see figure 5.8) [44]. Moreover, inconsistencies in the image labelling can found throughout the *KITTI* data-set as seen in figure 5.9.

As explained in **section 3**, the *ImageNet* 2012 classification challenge data set was chosen for network primary training, which is the current default training procedure for detection networks. The study made by [33] in the context of PASCAL image recognition challenges suggests that a smaller set containing objects and scenes similar to the secondary task (object detection) could improve performance. Similarly, reducing the number of classes but increasing the number of images per class may ameliorate detection accuracy. Due to the lack of time, these theories were not explored in this study.

5.4 Task

Predicting the location of cars, pedestrians, and cyclists in a 2 dimensional view plane should be an important step in developing AD and ADAS functionalities and was the output of models studied in this work. That being said, road environments typically contain a much broader array of object classes which could be integrated into the decision processes. *KITTI* training data contains ground-truth annotations for 8 classes of object, 5 of which were ignored by our models. Rather than increasing the number of classes, discriminating objects into more fine-grained categories could also be of interest. For example, it may be of more use to classify an object as a specific type of car. *SubCNN* received competitive results in the *KITTI* object detection challenge. One interesting facet of this approach is classification of objects into subcategories. Here, subcategories are groupings of objects with similar pose and shape. Thus using 2D bounding box and subcategory predictions, one can estimate the occupancy of that object in 3D space [45]. One drawback of this method is the low inference speed resulting from it's *R-CNN* type architecture with a much more extensive region proposal network. Other works have focused on explicitly utilising the 2D properties outputted by an object detector network in regressing the pose and 3D dimensions of objects of interest [46]. The benefit of such an approach is that it could be more easily integrated into our model.

Figure 5.7: Detection predictions on *Udacity* data generated by our *Gated SqueezeDet* network trained solely on *KITTI* data (confidence threshold of 0.5).

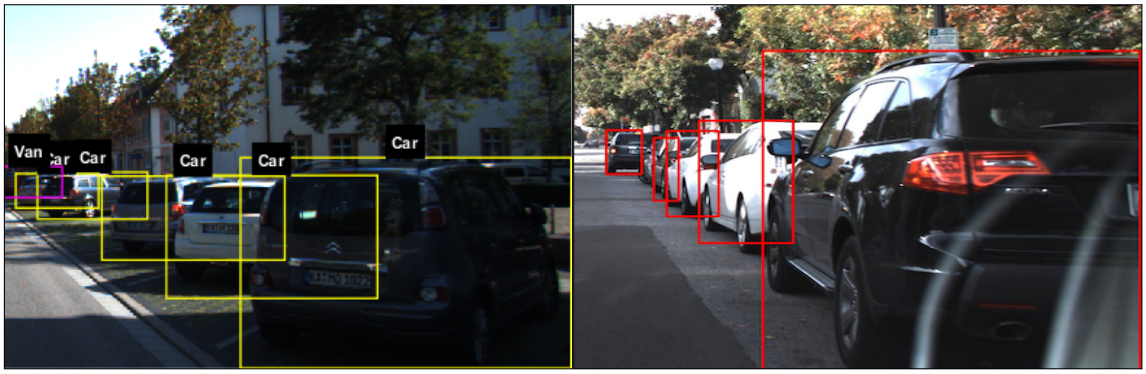


Figure 5.8: Examples of ground truth labels for occluded objects in the *KITTI* (left) and *Udacity* (right) data-sets.

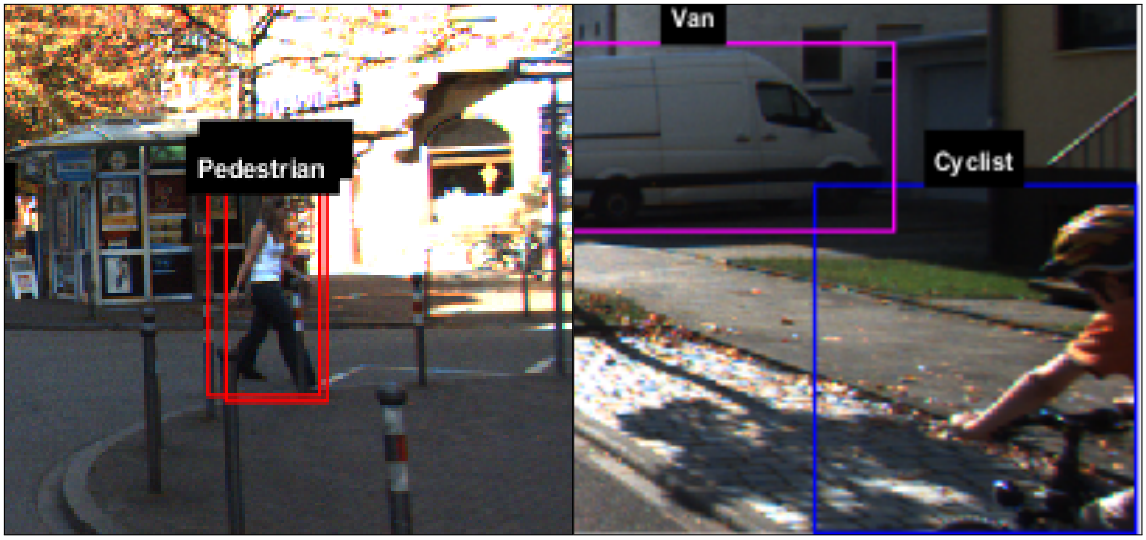


Figure 5.9: Examples of inaccurate ground-truth labels in the *KITTI* training data-set. Left: a single pedestrian has be labelled as two pedestrians. Right: the bounding box for the cyclist includes significant redundant space.

As discussed in **section 5.2**, the effect modifying the network to utilise information from several instances of time on detection performance is an outstanding question. While this may aid in giving instantaneous predictions, it could also be exploited as part of some tracking system where the goal is continually detect specific object instances over time. Several recent works have looked at integrating neural networks in object tracking pipelines. For example, [47] employ a fully connected feed forward network to predict the confidences associated with image-patch samples close to the object's location in the previous frame. Tracklets are produced by using these confidence scores as inputs to a particle filter. Recently, efforts have focused on end-to-end deep learning algorithms which map directly from image pixels to object tracklets. [48] has demonstrated single object tracking at 100 fps with a recurrent CNN. Here predicting the movement of an object from one frame to the next is treated solely as a regression problem. The object-containing crop predicted by the previous time step is encoding using a CNN. This area is expanded, extracted from the current frame, and passed though the same network. The encodings from the previous and current time-steps are amalgamated, and the offset between the frames are regressed through fully connected layers. Both the aforementioned methods require initialisation with bounding boxes which could be provided by our model. Traditionally object detection and tracking have been treated as two distinct problem, and integrating them in an efficient way is an open question.

Action classification is another task which would benefit from sequences of data. Generally, this involves not only estimating the location of objects, but also classifying the type of activity they are undertaking. This field is becoming of particular interest in applications such as sport and retail space analysis. In the same light, discerning whether a pedestrian is standing still, on their mobile phone, jogging etc. could give greater insight into their expected behaviours. Several works have

explored the application of DNN in this domain [49] [50]. As with all the tasks discussed in in this section, there is a significant lack of annotated data which specifically depicts driving environments.

6

Conclusion

Recent breakthroughs in object detection have come predominately from CNN-based methods. These models can be coarsely divided into two groups. On the one hand, you find a lineage that began with *R-CNN* [10] concerned with classifying and refining regions-of-interest (ROI). With *Fast R-CNN* [18], greater sharing of convolutional features between ROI was facilitated by the novel ROI pooling layer. Despite the improvement, both methods rely on computationally expensive external ROI proposal algorithms. Through the introduction of anchors (reference shapes in the input image), *Faster R-CNN* [14] integrated ROI-proposal into the network structure incurring a significant decrease in inference time. By replacing the *Fast R-CNN* module with fully convolutional layers and near complete sharing of convolutional features, *R-FCN* [19] took a step closer to commercialisation of this technology. On the other hand, there is a family of approaches that have been recently termed as *Single Shot Detectors* (SSD). Such approaches share a more direct mapping from image pixels to detections, bypassing the need for class-agnostic region proposing parallel to more general feature extraction for detection. Instead, the image is processed by a single-track network once, and predictions are generated for evenly-spaced locations in the input. *You Only Look Once* (YOLO) [22] was one of the first detection methods to surpass processing rates of 50 frame per second, however strong spatial constraints contributed to poor accuracy across many detection challenges.

Traditionally, the latter group held an advantage in terms of processing speed but lacked in detection precision and recall. Further to reliable performance, low memory and energy requirements are important considerations in integrating this technology into AD systems, considerations which have largely been unaddressed. To this end, *SqueezeDet* [21] was developed as a light-weight SSD with a model size of approximately 8 MB, one or two orders of magnitude smaller than many of it counterparts. The main contribution to its modest memory requirement is the *fire module*, composed of a *squeeze layer* which significantly compresses intermediate image representations, followed by an *expand layer* which applies computationally expensive convolution operations. In this study, two structural modification to the *SqueezeDet* architecture were proposed and analysed; the introduction of residual connection (*Residual SqueezeDet*) and novel gated connections (*Gated SqueezeDet*). Two separate training procedures were used; one in which training was carried out using only the *KITTI* object detection data and another which also included a pre-training stage on the *ImageNet* 2012 classification challenge data.

The original network, along with the two modified networks, were trained soley on

KITTI, with the majority of the hyperparameters replicated from [21]. After 100, 000 iterations of training using the *Adam* optimiser, it emerged that *Gated SqueezeDet* held a 0.8% improvement in validation mAP compared to *SqueezeDet*, with noticeably increased recall for distant objects. This is achieved with only a small inflation in the number of parameters (0.7%). On the other hand, adding standard residual connections lead to significant performance depreciation (2%). Due to *Residual SqueezeDet's* poor mAP, only *SqueezeDet* and *Gated SqueezeDet* were optimised with the inclusion of a pre-training stage. Both networks were trained for approximately 49 epochs on the *ImageNet* data-set, reaching a top-1 validation accuracy of 45%. The first convolutional layer and following 8 *fire modules* were next initialised using the parameters learned in pre-training. Two additional *fire modules* and a final convolutional layer were appended to the network and were subsequently trained for object detection on *KITTI* with the first convolutional layer frozen. After 100, 000 iterations of training, *SqueezeDet* exhibited substantially improved validation mAP compared to *Gated SqueezeDet* (5%).

One reason as to why *Gated SqueezeDet* exhibits poor performance through transfer learning may be the sigmoid activation function used in the gating mechanism. The saturation of this function leads to a vanishing gradient in the primary task which appears to halt learning in the gating mechanism during the secondary task. Thus, further investigation of the gating mechanism design must be undertaken to better facilitate transfer learning with this network. Furthermore, training on 49 epochs of *ImageNet* data were not sufficient to reach a 60% top-1 accuracy and therefore performance cannot be compared to the results outlined in [21]. The results outlined in **section 4.3** suggest that the *Gated SqueezeDet* is susceptible to over-fitting, an issue that may be resolved with more variability in the training and validation data.

There were several limitations to the scope of the project. One significant hurdle was restrictions in available data. *KITTI* presents a relatively good variability in driving environments, but not in weather and light conditions. Also, *KITTI*'s small data-set size may prevent the networks from learning more general representations of objects. Another major limitation was the available time and computational power for the project. Granted more resources, several extensions could be implemented to the networks. Examples include exploring other tasks so as action classification, object tracking and 3D bounding box prediction.

To summarize, architectures based on the *SqueezeDet* model have shown promising capacity to be commercialised due to their high inference rates. Further investigation is, however, required in order to rectify their detection performance.

Bibliography

[1] L. Isik, E. M. Meyers, J. Z. Leibo, T. Poggio, *The dynamics of invariant object recognition in the human visual system*, Journal of Neurophysiology, Vol. 111 no. 1, 91-102, 2014.

[2] Barber, D, *Bayesian Reasoning and Machine Learning. Cambridge University Press*, 2011.

[3] Silver et al. *Mastering the game of Go with deep neural networks and tree search*, Nature 529.7587, 2016.

[4] Gulshan et al. *Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs*, JAMA, 2016.

[5] K. Fukushima, *Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position*, Biological cybernetics, 36(4):193–202, 1980.

[6] D. E. Rumelhart, G. E. Hinton, and R. J. Williams *Learning internal representations by error propagation*. Parallel Distributed Processing, 1:318–362, 1986.

[7] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. *Backpropagation applied to handwritten zip code recognition*. Neural Computation 1, 541-551, 1989.

[8] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. *Gradient-based learning applied to document recognition*. Proc. of the IEEE, 1998.

[9] A. Krizhevsky, I. Sutskever, and G. Hinton. *ImageNet classification with deep convolutional neural networks*. In NIPS,2012.

[10] R. Girshick, J. Donahue, T. Darrell, and J. Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation*, in IEEE Conference on Computer Vision and Pattern Recognition, 2014.

[11] Karpathy A. *CS231n Convolutional Neural Networks for Visual Recognition*. Convolutional Neurons. `http://cs231n.github.io/convolutional-networks/`. Accessed 24 January 2017.

[12] Karpathy A. *CS231n Convolutional Neural Networks for Visual Recognition*. Neural Networks Part 1. `http://cs231n.github.io/neural-networks-1/`. Accessed 24 January 2017.

[13] K. He, X. Zhang, S. Ren, and J. Sun, *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, in IEEE International Conference on Computer Vision (ICCV), 2015.

[14] S. Ren, K. He, R. Girshick, and J. Sun, *Faster R-CNN: Towards realtime object detection with region proposal networks*, in Advances in Neural Information Processing Systems (NIPS), 2015

[15] P. Felzenszwalb, R. Girshick, D. McAllester, D. Ramanan, *Object Detection with Discriminatively Trained Part Based Models*, PAMI 32(9), 2010.

[16] D. Lowe. Distinctive image features from scale-invariant keypoints. IJCV, 2004.

[17] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. In ICLR, 2014.

[18] R. Girshick, *Fast R-CNN*, in IEEE Conference on Computer Vision and Pattern Recognition, 2015.

[19] J. Dai, Y. Li, K. He, and J. Sun, *Object Detection via Region-based Fully Convolutional Networks*, in ECCV, 2016.

[20] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.- Y. F. Fu, A. C. Berg, *SSD: Single Shot MultiBox Detector*, arXiv preprint arXiv:1512.02325, 2016.

[21] B. Wu, F. Iandola, P. H. Jin, K. Keutzer, *SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving*, in CVPR, 2017.

[22] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, *You Only Look Once: Unified, Real-Time Object Detection*, in CVPR, 2016.

[23] Z. Cai, Q. Fan, R. S. Feris, and N. Vasconcelos, *A Unified Multi-scale Deep Convolutional Neural Network for Fast Object Detection*, Computer Vision – ECCV 2016, pp.354-370.

[24] K. Kim, Y. Cheon, S. Hong, B. Roh, M. Park, *PVANET: Deep but Lightweight Neural Networks for Real-time Object Detection*, in Advances in Neural Information Processing Systems (NIPS), 2016.

[25] F. Yang, W. Choi, and Y. Lin, *Exploit All the Layers: Fast and Accurate CNN Object Detector with Scale Dependent Pooling and Cascaded Rejection Classifiers* in IEEE International Conference on Computer Vision and Pattern Recognition, 2016.

[26] R. N. Rajaram, E. Ohn-Bar, and M. M. Trivedi, *RefineNet: Iterative Refinement for Accurate Object Localization* in IEEE 19th International Conference on Intelligent Transportation Systems, 2016.

[27] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press, 2016.

[28] B. Singh, S. De, Y. Zhang, T. Goldstein, and G. Taylor, *Layer-Specific Adaptive Learning Rates for Deep Networks* in International Conference on Machine Learning and Applications (IEEE ICMLA), 2016.

[29] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, Journal of Machine Learning Research 15, 1929-1958. 2014.

[30] A. Geiger, P. Lenz and R. Urtasun, *Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite*, Conference on Computer Vision and Pattern Recognition (CVPR), 2012.

[31] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, *ImageNet Large Scale Visual Recognition Challenge.* International Journal of Computer Vision (IJCV), 2015.

[32] Yosinski J, Clune J, Bengio Y, and Lipson H, *How transferable are features in deep neural networks?* In Advances in Neural Information Processing Systems 27, NIPS Foundation, 2014.

[33] M.-Y. Huh, P. Agrawal, A. A. Efros, *What makes ImageNet good for transfer learning?* arXiv:1608.08614, 2016.

[34] TensorFlow. `https://www.tensorflow.org/`. Accessed 01 February 2017.

[35] K. He, X. Zhang, S. Ren, J. Sun, *Deep Residual Learning for Image Recognition*, arXiv:1512.03385, 2015.

[36] R. K. Srivastava, K. Greff, J. Schmidhuber, *Training Very Deep Networks*, arXiv:1507.06228, 2015.

[37] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, *SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and <0.5MB model size*. arXiv:1602.07360, 2016.

[38] A. Karpathy. *CS231n Convolutional Neural Networks for Visual Recognition*. Setting up the data and the model. `http://cs231n.github.io/neural-networks-2/#init`. Accessed 28 March 2017.

[39] Karpathy A. *CS231n Convolutional Neural Networks for Visual Recognition*. Transfer Learning. `http://cs231n.github.io/transfer-learning/`. Accessed 24 May 2017. MNISTY. LeCun, L. Bottou, Y. Bengio, and P. Haffner. *Gradient-based learning applied to document recognition.* Proceedings of the IEEE, 86(11):2278-2324, November 1998.

[40] C. Y. Fu, W. Liu, A. Ranga, A. Tyagi, A. C. Berg. *DSSD: Deconvolutional Single Shot Detector*. arXiv preprint arXiv:1701.06659, 2017.

[41] F. A. Gers, J. Schmidhuber, and F. Cummins. *Learning to forget: Continual prediction with LSTM*. In International Conference on Artificial Neural Networks, volume 2, pp. 850–855, 1999.

[42] N. Bodla, B. Singh, R. Chellappa, L. S. Davis. *Improving Object Detection With One Line of Code*. arXiv:1704.04503. 2017.

[43] J. Hosang, R. Benenson, and B. Schiele. *A Convnet for Non-maximum Suppression*. In German Conference on Pattern Recognition (GCPR), pp 192-204, 2016.

[44] Udacity *Annotated Driving Dataset*. `https://github.com/udacity/self-driving-car/tree/master/annotations`. Accessed 16 May 2017.

[45] Y. Xiang, W. Choi, Y. Lin, and S. Savarese. *Subcategory-aware convolutional neural networks for object proposals and detection*. In arXiv:1604.04693, 2016.

[46] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka. *3d bounding box estimation using deep learning and geometry*. In CVPR, 2017.

[47] N. Wang and D.-Y. Yeung. *Learning a deep compact image representation for visual tracking*. In Advances in Neural Information Processing Systems, pages 809–817, 2013

[48] D. Held, S. Thrun, and S. Savarese. *Learning to track at 100 fps with deep regression networks*. In ECCV, 2016

[49] S. Yeung, O. Russakovsky, G. Mori, L. Fei-Fei. *End-to-End Learning of Action Detection from Frame Glimpses in Videos*. In CVPR, 2016.

[50] G. Gkioxari B. Hariharan R. Girshick J. Malik. *R-CNNs for pose estimation and action detection*. arXiv:1406.5212, 2014.

A

Appendix 1

Method	car			pedestrian			cyclist			mAP
	E	M	H	E	M	H	E	M	H	
SqueezeDet	81.5	82.0	74.4	74.3	70.6	63.6	72.7	74.7	72.2	74.0
Residual SqueezeDet	82.8	83.0	74.0	67.5	65.8	57.4	53.9	61.2	60.8	67.3
Gated SqueezeDet	82.3	80.0	72.6	79.9	74.7	65.8	70.3	74.8	74.2	75.0

Table A.1: Average precision for each object class in the different difficulty categories (E: easy, M: moderate, H: hard) measured on the validation set for a second run.

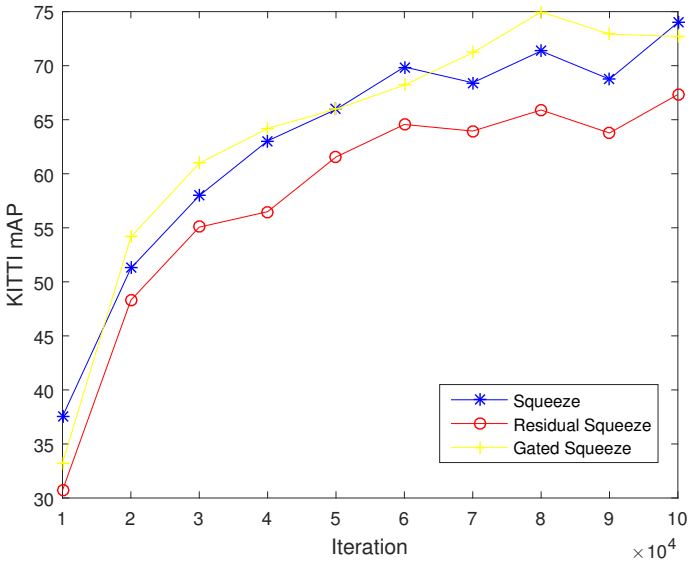
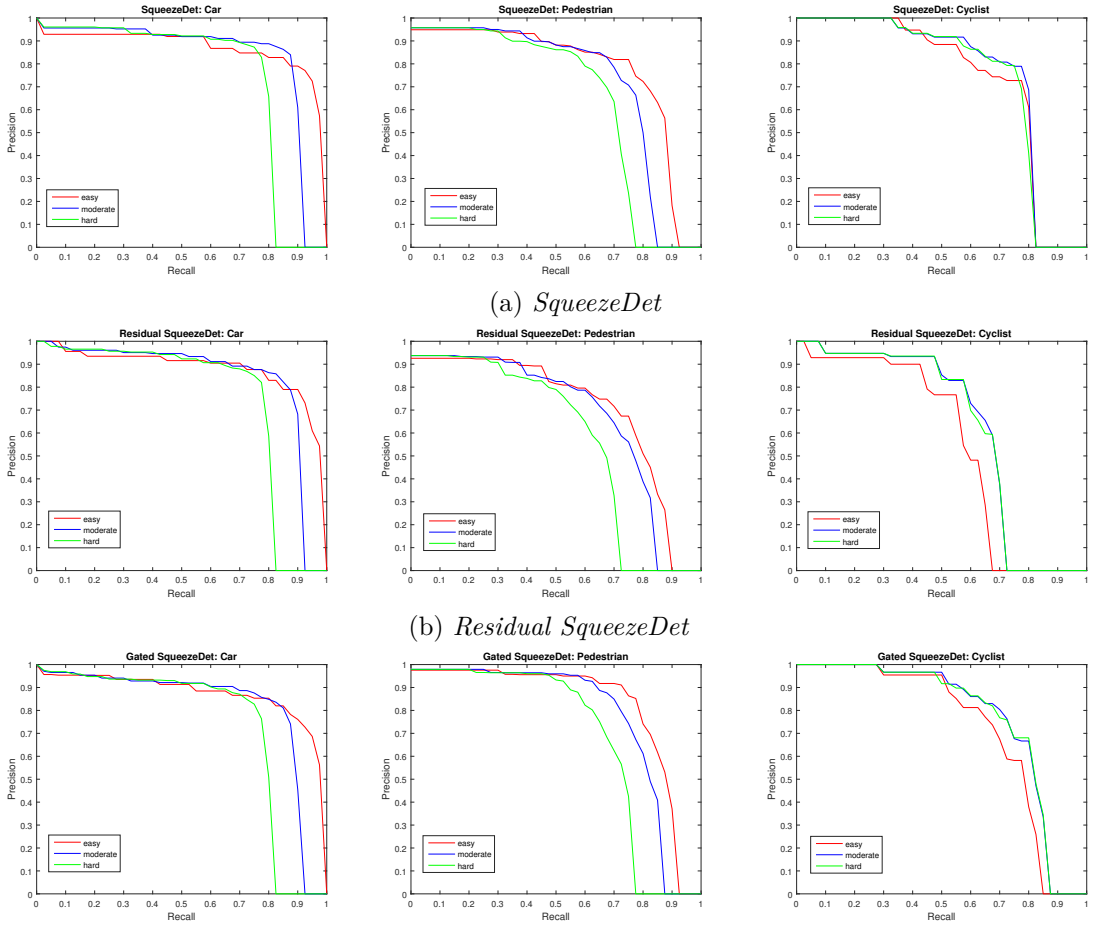


Figure A.1: mAP over iterations of training on the *KITTI* validation set of 500 images on a second run. We observe that *Gated SqueezeDet* overfit after 80000 iterations, being *SqueezeDet* better at the end, but slightly worst before.



(a) *SqueezeDet*

(b) *Residual SqueezeDet*

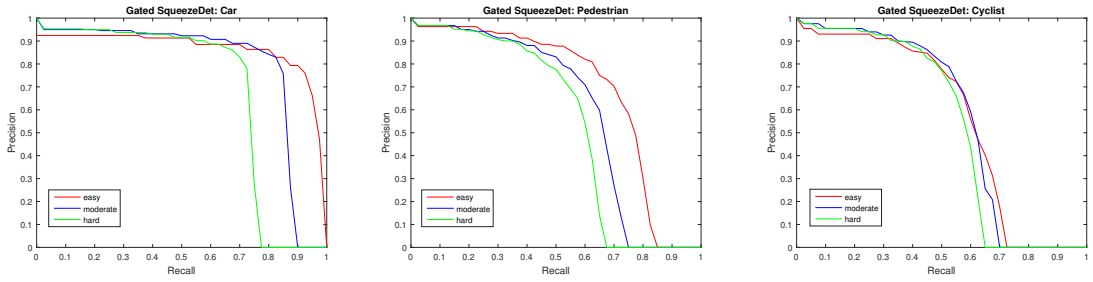
(c) *Gated SqueezeDet*

Figure A.2: Precision/recall curves for the networks on the different object categories (left column: car, middle column: pedestrian, right column: cyclist) evaluated on the validation set.

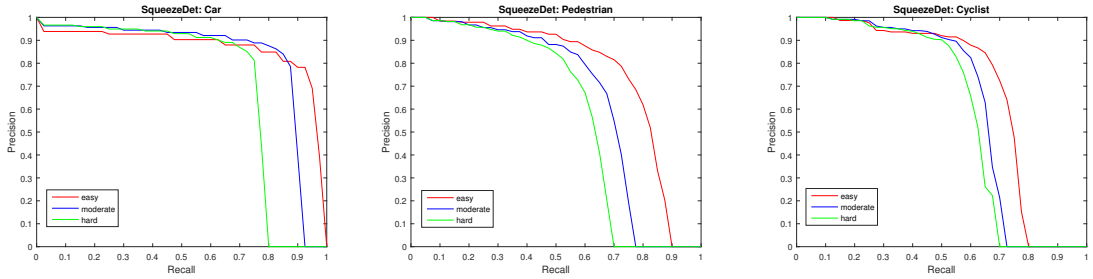
Figure A.3: Sample images from the *KITTI* testing data with detections generated by the *SqueezeDet* trained from scratch (confidence threshold of 0.5.

Figure A.4: Sample images from the *KITTI* testing data with detections generated by the *Residual SqueezeDet* trained from scratch (confidence threshold of 0.5.

Figure A.5: Sample images from the *KITTI* testing data with detections generated by the *Gated SqueezeDet* trained from scratch (confidence threshold of 0.5.



(a) *Gated SqueezeDet*



(b) *SqueezeDet*

Figure A.6: Precision/recall curves for the networks on the different object categories (left column: car, middle column: pedestrian, right column: cyclist) evaluated on the validation set composed of 3740 images. Both networks have the first convolution layer frozen in the training and the first 8 fire modules initialized from the network trained in *ImageNet* .