Aalto University
School of Science
Degree Programme in Computer Science and Engineering

Joonas Haapala

Recurrent neural networks for object detection in video sequences

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science (Technology)

Espoo, March 21, 2017

Supervisor:	Prof. Jaakko Lehtinen
Advisor:	D.Sc. Pekka Jänis

Aalto University
School of Science

Aalto University
School of Science
Degree Programme in Computer Science and Engineering

ABSTRACT OF MASTER'S THESIS

Author: Joonas Haapala

Title:
Recurrent neural networks for object detection in video sequences

Date:	March 21, 2017	**Pages:**	58
Major:	Machine Learning and Data Mining	**Code:**	SCI3044
Supervisor:	Prof. Jaakko Lehtinen		
Advisor:	D.Sc. Pekka Jänis		

This thesis explores recurrent neural network based methods for object detection in video sequences. Several models for object recognition are compared by using the KITTI object tracking dataset containing photos taken in an urban traffic environment. Metrics such as robustness to noise and object velocity prediction error are used to analyze the results. Neural networks and their training methodology is described in depth and recent models from the literature are reviewed.

Several novel convolutional neural network architectures are introduced for the problem. The VGG-19 deep neural network is enhanced with convolutive recurrent layers to make it suitable for video analysis. Additionally a temporal coherency loss term is introduced to guide the learning process. Velocity estimation has not been studied in the literature and the velocity estimation performance was compared against a baseline frame-by-frame object detector neural network.

The results from the experiments show that the recurrent architectures operating on video sequences consistently outperform an object detector that only perceives one frame of video at once. The recurrent models are more resilient to noise and produce more confident object detections as measured by the standard deviation of the predicted bounding boxes. The recurrent models are able to predict object velocity more accurately from video than the baseline frame-by-frame model.

Keywords:	neural network, object detection, video processing, recurrent, deep learning
Language:	English

Aalto-yliopisto
Perustieteiden korkeakoulu

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Joonas Haapala		
Työn nimi: Rekurrenttien neuroverkkojen käyttäminen kohteiden tunnistamiseen videoissa			
Päiväys:	21. maaliskuuta 2017	**Sivumäärä:**	58
Pääaine:	Koneoppiminen ja tiedon louhinta	**Koodi:**	SCI3044
Valvoja:	Prof. Jaakko Lehtinen		
Ohjaaja:	TkT Pekka Jänis		

Diplomityössä tutkitaan rekurrentteihin neuroverkkoihin perustuvia menetelmiä kohteiden tunnistamiseen videosekvensseissä. Useita kohdetunnistuksen malleja verrataan käyttäen kaupunkiliikennekuvia sisältävää KITTI tracking -kuvakantaa. Mallien vertailun metriikkoina käytetään kohdetunnistuksien kohinasietoisuutta ja arvioitujen nopeusvektorien virhettä. Työssä esitellään neuroverkkojen teoriaa ja niiden kouluttamiseen liittyviä tekniikoita ja arvioidaan viimeaikaisia neuroverkkomalleja kirjallisuudesta.

Työssä esitellään useita uusia konvolutiivisia neuroverkkoarkkitehtuureja henkilöautojen tunnistamiseen kuvista. VGG-19-neuroverkkoa muokataan lisäämällä siihen konvolutiivinen rekurrentti neuroverkkokerros, jolloin sitä voidaan käyttää videosekvenssien analyysiin. Lisäksi tutkitaan uuden aikajatkuvuuskannustetermin vaikutusta verkon oppimisen ohjaamisessa. Kohteiden nopeuden arviointia kuvista ei ole tutkittu kirjallisuudessa ja nopeusvektoriarvioiden virhettä verrattiin perusmalliin, joka tunnisti kohteet nopeuksineen yksittäisistä kuvista.

Tulokset osoittavat, että videosekvenssejä tulkitsevat rekurrentit neuroverkkiarkkitehtuurit tunnistavat kohteita paremmin kuin neuroverkot, jotka tulkitsevat kohteita kuvista yksi kerrallaan. Rekurrentit mallit sietävät kuviin lisättyä kohinaa paremmin ja tuottavat tilastollisesti varmempia tunnistuksia. Myös rekurrenttien mallien tekemät nopeusvektoriarviot ovat tarkempia kuin vertaillun perusmallin.

Asiasanat:	neuroverkko, hahmontunnistus, videoprosessointi, rekurrentti, syvä oppiminen
Kieli:	Englanti

Acknowledgements

Foremost, I would like to thank my advisor Pekka Jänis, who has been very supportive in this research. I am grateful for the many interesting discussions we had together and for his persistence in guiding me forwards with this work. I also send thanks to my supervisor Jaakko Lehtinen for his interesting research ideas and professional advice.

I would like to thank my team in NVIDIA for their help and support with the tools and hardware required for this research. I would also like to express my gratitude to Timo Roman for the great experience I had working in his team and my employer NVIDIA for allowing me to work on this thesis full time.

I also want to thank my parents Minna and Kari, my girlfriend Nadja and my whole family for their continuous support and care throughout my study path.

Espoo, March 21, 2017

Joonas Haapala

Abbreviations and Acronyms

ADAS	Advanced Driver Assistance System
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
FPS	Frames Per Second
GPU	Graphics Processing Unit
mAP	mean Average Precision
MLP	Multilayer Perceptron
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent

Contents

Abbreviations and Acronyms 4

1 Introduction 8
1.1 Structure of the thesis 9

2 Background 10
2.1 Artificial Neural Networks 10
2.1.1 Perceptron 11
2.1.2 Multilayer Perceptron 12
2.1.3 Activation functions 14
2.1.4 Training procedure 15
2.1.4.1 Model selection 15
2.1.4.2 Initialization 16
2.1.4.3 Optimization 16
2.1.4.4 Learning rate schedule 17
2.1.4.5 Momentum 18
2.1.4.6 Adam 18
2.1.4.7 Overfitting and regularization 18
2.1.4.8 Dropout 19
2.1.4.9 Dataset augmentation 20
2.1.5 Convolutional Neural Networks 20
2.1.5.1 Pooling 21
2.1.5.2 VGG-19 22
2.1.6 Recurrent Neural Networks 22
2.1.6.1 LSTM 23
2.1.6.2 GRU 23
2.2 Object detection with neural networks 24
2.2.1 Gridbox model 24
2.2.1.1 Velocity estimation 26
2.2.2 Video processing 26
2.2.3 Related work 27

3 Methods 30
3.1 Object detection using recurrent networks 30
3.2 Recurrent forecast layer 31
3.3 Experiments 33
3.4 Performance metrics 34

4 Results 37
4.1 Bounding box standard deviation 37
4.2 Object detection average precision 37
4.3 Velocity prediction performance 39

5 Discussion 44
5.1 Result analysis 44
5.2 Future work 46

6 Conclusions 50

Bibliography 52

A Learning curves 56

Chapter 1

Introduction

The recent development of deep learning has created many new applications for high performance computing. Deep learning, the craft of creating hierarchical models for function approximation, is being used for many tasks that were previously seen as unapproachable by computer algorithms. Nowadays these machine learning algorithms are used in, for example, image search services to look up images based on their content, not just textual metadata.

Meanwhile many companies are seeking to create an artificial intelligence for steering autonomous vehicles. The human driver is difficult to replace, but the potential benefits are remarkable. WHO estimates [1] that globally there are 1.25 million road traffic related deaths each year. Many lives could potentially be saved if superhuman driving capabilities and reaction times were achieved.

Deep convolutional neural networks have been used successfully in many object detection and recognition tasks [2–5]. One application is the detection of different entities such as cars and pedestrians in photos taken in an urban traffic environment. Reliable detection of such objects is an important component of an Advanced Driver Assistance System (ADAS). Large models involving millions of artificial neurons have been shown to be accurate enough to both recognize and localize cars, pedestrians and other objects of interest [6].

By extending car detection from pictures to videos the models are capable of detecting important temporal features such as velocity as well as predicting object movement behind temporary occlusions. Object detection may be performed fast enough to be run on each frame of a video sequence to localize objects present in the video.

Recurrent neural network architectures contain a memory component that can be used to keep track of object movement across multiple frames of video. The model outputs a stable movement trajectory for the objects

without a separate tracker. Following objects's trajectories is automatically learned during the model training phase. This is the approach researched in this thesis.

This thesis explores the current state of object detection using neural networks and proposes novel models for object detection in video sequences. The models are applied to the real-time car detection and localization problem to be used as a component in an Advanced Driver Assistance System.

1.1 Structure of the thesis

This thesis is structured as follows. Chapter 2 describes neural networks and their training procedure in the section 2.1 and how to apply them to object detection in section 2.2. Experimental models for object detection in video sequences and their evaluation methods are described in Chapter 3 and the results are shown in Chapter 4. Chapter 5 provides some thoughts on the results. Finally, Chapter 6 gives a recap of the main themes discussed in this thesis.

Chapter 2

Background

This chapter describes artificial neural networks as a mathematical model and then discusses how they can be used for object detection. Existing work is briefly reviewed at the end of the chapter.

2.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) have rapidly gained popularity in the recent years [7]. Even though the algorithms were formulated already in the 1960s, many problems have become tractable with the development of high-throughput computing via Graphics Processing Units (GPU). Modern neural networks are used for solving problems that were previously only solvable by humans, such as recognizing handwriting, sentiment analysis or classifying pictures by their content. In some problems recent models are able to surpass human performance [8].

Computation in an artificial neural network is distributed across small units called neurons, which are simple decision makers inspired by the biological brain cell. They receive stimuli from other neurons and in some conditions the neuron can activate and in turn pass stimuli to other neurons. The connection between two neurons can either enhance or inhibit the activation. Complex behaviours can be learned by adjusting this property of the connection in a large population of connected neurons and it has been shown that given enough neurons neural networks are able to approximate any nonlinear function [9]. This property is exploited to learn a complex nonlinear function that maps given training data to given ground truth labels.

Neural network training involves solving an optimization problem. The function to be optimized is an error metric where lower values indicate more accurate predictions that match the ground truth labels more closely. The

parameters to be optimized are the connection weights between neurons. The training is computationally very intensive and large models with millions of parameters can easily take days or even weeks of GPU time to train.

This is in contrast to traditional feature engineering, where human experts fine-tune explicit rules to match interesting or useful traits in the data. Neural networks find the interesting features themselves as a result of the training and their complexity easily exceeds human comprehension. A common critique is that the ANN is in a sense a black-box model, since it's inner workings are very difficult to understand.

Training a neural network model usually requires a huge set of training data. The training set consists of labeled observations and the task of the model is to learn to connect the observations to the corresponding labels. An observation could be a photograph of a busy street and the label could be the bounding box coordinates of each car, pedestrian, bicycle or other object of interest in the photograph. With enough training examples and repetition the network will slowly learn to understand the objects' appearence and the individual components help it identify them. While training a network is time-consuming, the time it takes to detect objects in an image can be made short enough to make the method suitable for real-time applications by powerful hardware and techniques such as compression [10] and pruning [11].

2.1.1 Perceptron

In machine learning and data mining, a common task is to classify samples of data as belonging into one of a set of classes. The samples are typically represented as vectors, where each element describes a particular feature of the sample. In a binary classification problem, such a vector is classified as belonging to one of two classes.

The perceptron algorithm [12] was one of the first algorithms for training artificial neural networks. The algorithm describes a learning rule for updating the parameters of a simple binary classifier so that it forms a linear decision boundary in the input space.

Given the observation vector $x \in R^N$, where N is the number of features, the classification rule is

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0, \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

where the vector w and scalar b are parameters that are learned by the perceptron algorithm. The learning rule for the perceptron algorithm is

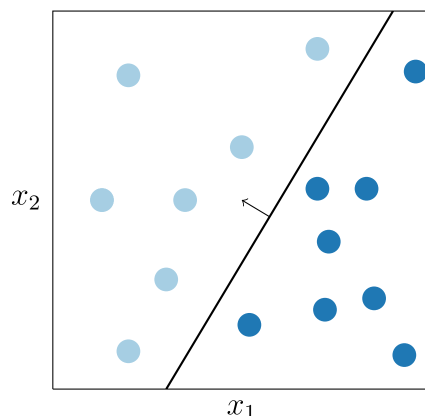

Figure 2.1: A decision boundary learned by the perceptron algorithm. The learned slope perfectly separates all samples of the two classes. Classification is done by $w_1x_1 + w_2x_2 + b > 0 \implies$ Class 1, otherwise Class 2.

$$\begin{aligned} w &\leftarrow w + \lambda(y - f(x))x \\ b &\leftarrow b + \lambda(y - f(x)), \end{aligned} \tag{2.2}$$

where y is the real binary class of observation x, $f(x)$ is the classification of x made by the model, and λ is a scalar learning rate for the algorithm. The classification error term $y - f(x)$ guides learning so that learning only happens when a training sample was misclassified ($y - f(x) \neq 0$). The learned linear decision boundary for a two-dimensional classification problem is visualized in Figure 2.1.

2.1.2 Multilayer Perceptron

The simple classifier in the perceptron algorithm is unable to model complex nonlinear relationships between variables. More complex models can be created by stacking simple linear classifiers in consecutive layers, where each neuron in a layer receives a connection from each neuron in a previous layer. Each connection can either enhance or inhibit the activation from the sending neuron. However, two consecutive affine transformations still yield an affine transformation and the model becomes powerful only once the output of each layer is transformed by a non-linear activation function.

Large models can contain tens or even hundreds of layers. Care must be taken that as the neural activations propagate through the network their norm does not grow exponentially or approach zero, both of which are common phenomenona when working with consecutive matrix multiplications.

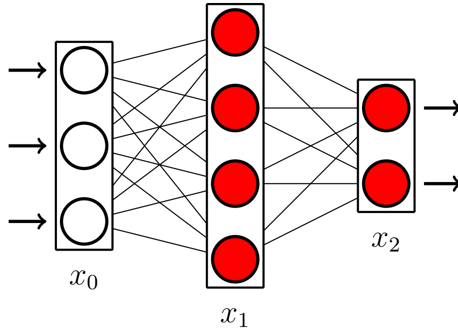

Figure 2.2: A multilayer neural network with two layers. Each colored circle represents a single neuron that receives activations from the previous layer. Input data is placed in x_0 after which information passes to the right. The activation vectors x_1 and x_2 are a function of the previous activations, and the network output is read from $x_2 = \widetilde{y}_{x,\theta}$.

Parameter initialization is discussed in more detail in Section 2.1.4.2.

The connection between two neurons is parameterised by a scalar weight value, that is a multiplier for activation of the neuron in the previous layer. The activation of a neuron is also a scalar, formed by accumulating the activations of connected neurons weighted by the weights associated by the connections and adding a scalar bias associated with that neuron. The resulting value is transformed by a non-linear activation function. Activation functions are discussed in more depth in Section 2.1.3.

Formally, if the first layer l_1 contains n_1 neurons, and the second layer l_2 contains n_2 neurons, and each neuron in l_2 receives input from l_1, the activation of the second layer of neurons can be written as,

$$x_2 = \sigma(W_1 x_1 + b_1), \tag{2.3}$$

where x_2 is a vector of activations in the second layer, W_1 is a $n_2 \times n_1$ matrix of weights between the neurons in l_1 and l_2. b_1 is a trainable bias vector that together with the weight matrix W_1 parameterize an affine transformation. The $\sigma(x)$ is an activation function, such as $\sigma(x) = \tanh(x)$.

The outputs of the neural network are the neuron activations on the last layer. The activation vector $\widetilde{y}_{x,\theta}$ depends on all parameters $\theta = \{W_i, b_i \ \forall i\}$ present in the network and the observations x_1. A multilayer perceptron with two layers is visualized in Figure 2.2.

The learning rule of the perceptron algorithm (Eq. 2.2) cannot be used for the multilayer network. To find the optimal parameters $\theta^* = \{W_i^*, b_i^* \ \forall i\}$ an

objective function has to be defined. One common objective is to minimize the squared prediction error, which is defined as

$$L_{x,y,\theta} = (\widetilde{y}_{x,\theta} - y)^2, \tag{2.4}$$

where $L_{x,y,\theta}$ is the loss function defined in terms of the training data x, corresponding ground truth labels y and the network parameters θ. The output of the network is denoted by $\widetilde{y}_{x,\theta}$. Minimizing the loss involves finding the optimal parameters θ^* that best map the network input x to the labels y. The terms objective function and loss function are used interchangeably in this thesis.

2.1.3 Activation functions

To learn complex patterns present in training data, neural networks need to be able to learn nonlinear mappings between inputs and outputs. The role of the activation function is to introduce nonlinearities in the network. They are used on each layer following the affine transformation: $x_{n+1} = \sigma(Wx_n + b)$.

The most common activation function is the rectified linear unit (ReLU) $\sigma(x) = \max(0, x)$. Linear functions are easy to optimize and being a piecewise linear the ReLU preserves many favorable features of linear functions [13]. The ReLU is illustrated in Figure 2.3. Other often used activation functions include the hyperbolic tangent $\tanh(x)$, sigmoid $(1 + e^{-x})^{-1}$ and the softmax function

$$\sigma(x_i) = \frac{\mathrm{e}^{x_i}}{\sum_{j=1}^{n} \mathrm{e}^{x_j}}, \tag{2.5}$$

where x_i denote the individual elements of the vector x.

If no activation function is used (it is replaced with the identity $\sigma(x) = x$) two consecutive layers can be merged and the second layer cannot provide any additional representational capacity in the model, since two composed affine transformations is just another affine transformation: $W_2(W_1x + b_1) + b_2 = (W_2W_1)x + (W_2b_1 + b_2)$.

Continuity is a desirable property for the activation function in order to give better convergence guarantees for the model as a whole, but there may exist points where the function is not differentiable: for example the ReLU function is not differentiable at $x = 0$. In practice this is not an issue since hitting the exact value $x = 0$ is rare, and the decision to choose either the left or right limit of the gradient makes very little difference in the results.

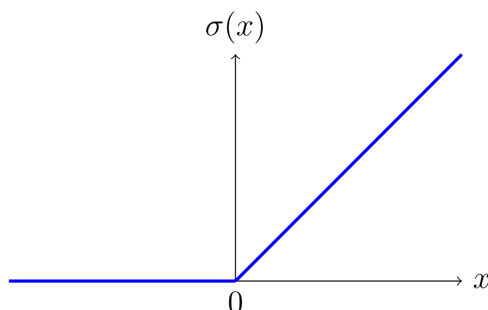

Figure 2.3: The rectified linear unit activation function (ReLU). It is defined as $\sigma(x) = \max(0, x)$.

2.1.4 Training procedure

Neural networks are very complex models and obtaining good results can be difficult. This chapter describes many practical methods for neural network training.

2.1.4.1 Model selection

Models in machine learning contain many hyperparameters. In neural networks the network size, learning rate, choice of activation function, etc. are all hyperparameters and their effect on the final model performance can be difficult to predict. The process of finding a good combination of model hyperparameters is called model selection.

Machine learning methods use the concepts training data, validation data and test data to differentiate between different disjoint sets of data that are used in different phases of training. The samples in each set are assumed to be **i.i.d.**, i.e. independent and identically distributed.

First, a model is trained using the training data. The model is suspectible to overfitting against this particular subset. The Section 2.1.4.7 describes overfitting in more depth.

After and during training the model is validated against the validation set by comparing the models predictions against the labels in the validation set. The model has not been trained on the samples in this set and thus the validation loss can be used to measure the models capacity to generalize beyond the training set. If the model overfits to the training set its validation loss increases, and the model hyperparameters should be chosen so that the validation set loss is close to the training set loss.

During hyperparameter optimization it is possible that the model indirectly overfits to the particular validation set, and thus the model is finally

tested against the test set. If the test set loss is also close to the training set loss the model can be said to generalize well.

Cross-validation can also be used to obtain a robust loss metric for hyperparameter tuning. In K-fold cross validation [14], the training set is split into K sets. One of the sets is used for validation and the rest are used as training data for a model. This process is repeated for K models so that each of K sets is used once as a validation set. Each model is validated on their corresponding validation set and the final cross validation loss is the average of the K losses. The value of K is a tradeoff between training time and variance of the loss averaged across folds. In practice K is often chosen to be 5–10.

2.1.4.2 Initialization

Before training can begin, the initial values for the parameters of the network, the weights and biases, have to be selected. Proper initialization can have a great impact on both the speed of the training and the evaluation accuracy of the model. Poorly initialized models can even fail to converge.

If one would initialize all network parameters to zero, each neuron would receive the same learning signal from the optimizer that is minimizing the loss, thus making all but one of them completely redundant. As such, one goal of neural network initialization is to break symmetry between neurons.

The values of W and b should be chosen so that the activations neither vanish (approach zero) or explode (approach infinity) as they pass through all layers in the network. Glorot and Bengio propose [15] to initialize the neuron biases to zero and sample initial weights from a random distribution such as the normal distribution or the uniform distribution with variance

$$\mathrm{Var}[W_i] = \frac{2}{n_i + n_{i-1}}, \tag{2.6}$$

where n_j is the number of neurons on the layer j. The justification for this initialization is that it approximately preserves the norms of the activation vector and the error gradient.

In the presence of a rectified linear activation function, He et al. show [8] that by taking the activation into account learning can be made even faster than in Glorot's method [15]. They propose initializing the weights by

$$\mathrm{Var}[W_i] = \frac{2}{n_i}, \tag{2.7}$$

where the motivation is that since the rectifier halves the variance of the data the weights should compensate for it. Similarly to Glorot and Bengio, He et al. initialize neuron biases to zero.

2.1.4.3 Optimization

Neural network training methods boil down to minimizing an objective function with respect to the network parameters. Even though the networks form highly non-linear non-convex minimization problems simple first-order optimization methods have proven to be effective in practice [16]. The simplest first-order method is the gradient descent method. It repeatedly calculates the gradient of the objective with respect to all parameters given the training data and updates the parameters in the direction that gives the greatest decrease in the value of the objective function. Its update rule is

$$\theta \leftarrow \theta - \lambda \nabla_\theta L_{x,y,\theta}, \tag{2.8}$$

where θ is a vector consisting of all the network parameters, λ is an adjustable learning rate or step size and $L_{x,y,\theta}$ is the loss function. The rule is applied elementwise for each scalar element of θ.

Calculating the gradient for the whole training data set $(x; y)$ can be costly, and in practice the gradient can be approximated by randomly sampling small subsets of $(x; y)$. The subsets are called minibatches and this form of training is called minibatch gradient descent or stochastic gradient descent (SGD).

With SGD, the gradient vectors become stochastic and fluctuate around the real gradient. The variance of the gradients can be decreased by increasing the number of samples in the minibatch. Larger minibatches also make better use of the parallel computation resources provided by modern GPUs. However, models trained with batch size 1 can generalize better due to the noise of the gradients [17].

2.1.4.4 Learning rate schedule

The learning rate λ is a multiplier in SGD, that controls the size of the optimization step. If the step size is too large the optimization overshoots and can end up oscillating around a local minimum in the optimization space. If it is too small the method will take long to converge to optimal parameters.

The learning rate is often adjusted so that in the beginning of the training it is set to a large value and it is let slowly decay over time, so that it doesn't

overshoot minima. Two favorable properties for the learning rate schedule are [13]

$$\sum_{i=1}^{\infty} \lambda_i = \infty, \quad \text{and} \tag{2.9}$$

$$\sum_{i=1}^{\infty} \lambda_i^2 < \infty, \tag{2.10}$$

where i is a scalar that grows linearly during the learning process, such as the number of gradient updates applied thus far. The first equation dictates that the optimizer is able to travel arbitrarily long paths in the optimization space while the second guarantees that the learning rate is decreasing at a sufficient rate. Setting $\lambda_i = \frac{\Lambda}{i}$, where Λ is an initial learning rate, is one way to satisfy the constraints. Methods such as Adam, described in section 2.1.4.6, use more advanced techniques to evolve the learning rate.

2.1.4.5 Momentum

Methods based on the gradient descent suffer from stalling around nearly flat areas in the objective function. As the gradient approaches zero the step size also approaches zero. One way to prevent stalling in these situations without overshooting in areas of higher curvature is to add a momentum term to the update rule [18, 19]. The momentum vector also helps the optimizer maintain a general direction across randomly chosen minibatches.

The momentum is an exponentially decaying vector where gradients from previous steps are accumulated. If the gradient of the objective function is close to zero or noisy the momentum term will help the optimizer continue in the same direction. A downside of this method is that it requires one more hyperparameter to tune, the rate of the exponential decay of the momentum vector. The update rule for the momentum method is

$$v_{t+1} = \mu v_t - \lambda \nabla_\theta L_{x,y,\theta_t}, \tag{2.11}$$

$$\theta_{t+1} = \theta_t + v_{t+1}, \tag{2.12}$$

where v_t is the momentum vector at step t and μ is the decay factor of the momentum.

2.1.4.6 Adam

Adam [20] is an algorithm that adjusts the learning rate for each variable separately. Built on the SGD, it is a first-order optimization method, but by

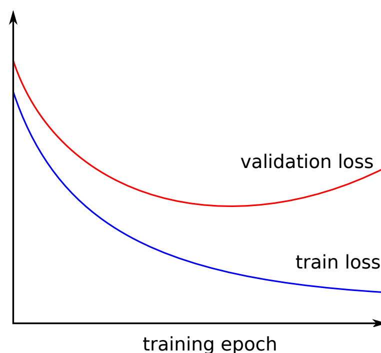

Figure 2.4: Overfitting leads to failing to generalize. Training for too long on too small a training set can lead to poor performance on the validation set.

building estimates on the first and second moment of the gradients it is able to take larger and more accurate steps in the optimization space than a basic SGD. It often outperforms other first-order methods of gradient descent by achieving lower loss values in fewer evaluations of the gradient.

2.1.4.7 Overfitting and regularization

Neural networks are powerful models and are very prone to overfitting, which refers to the models tendency to memorize the training data and not generalize to new data. The phenomenon can be seen by looking at evolution of the validation loss during training as shown in Figure 2.4. At some point the validation error will start increasing even though the network is still improving at the training set.

Overfitting is usually the result of having an excessively complex model compared to the amount of training data. Thus, one way to combat overfitting is to reduce the number of parameters in the model. In the context of neural networks, one commonly used method is weight decay regularization, which stands for penalizing the growth of the parameters of the network. Two commonly used forms of weight decay are L1 and L2 regularization, where the objective function of the network contains an additional L^p-norm term to be minimized,

$$L = L_{x,y,\theta} + \alpha \|\theta\|_p, \tag{2.13}$$

where θ are the parameters of the network and $\|.\|_p$ is the type of norm to be minimized. Regularization introduces a new hyperparameter α, which controls the amount of regularization applied.

2.1.4.8 Dropout

A recent method used to regularize deep neural networks is called dropout [21]. In dropout, the output of a random subset of the neurons on a layer are randomly set to zero, right after the activation function. The dropout effect is parameterized by p, the probability of preserving the value of an individual neuron. The stochastic dropping is done only during training time and during evaluation the output of each neuron is multiplied by p, making the expected value of the neuron activation equal in both phases.

Dropout prevents neuron activation co-adaptations and prevents overfitting [21]. By randomly dropping neurons, it implicitly trains a large family of neural networks that share many of their parameters. In inference the multiplication by p equates to sampling all of the networks together and ensembling their results reducing the overall bias of the model.

2.1.4.9 Dataset augmentation

Dataset augmentation can be used to combat overfitting. Augmentation artificially creates new training data by stochastically modifying the existing data, for example by adding noise. Augmentations are dataset dependent and care has to be taken to ensure that the augmentations are meaningful in the context of the problem. An object detection task should be robust against different levels of brightness, so the network is trained with dataset where the brightness of individual images is artificially modified. Likewise, object detectors should be invariant to object location, scale and orientation within the image, so images are often cropped, flipped and scaled randomly during training.

2.1.5 Convolutional Neural Networks

Neural networks are regularily used in image processing. Images are effectively 3D tensors with three dimensions: height, width and color depth. To be used as an input to a conventional neural network the tensor has to be flattened to a long feature vector.

There are two downsides to this kind of approach. The input dimensionality is the product of the three dimensions. With a 800×600 RGB image the size of the input vector the network is already more than million scalars long. Secondly, the features learned by the network are dependent on the position of the features, i.e. detecting an object at some position (x1, y1) has to be learned separately from detecting the same object at position (x2, y2).

Convolutional layers [13] introduce position invariance to neural networks.

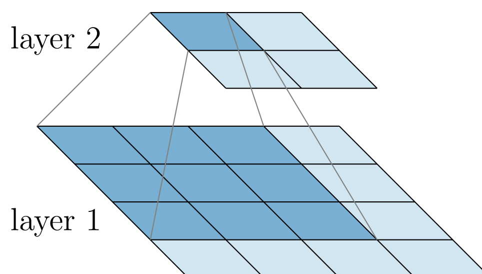

Figure 2.5: A convolutional layer. In this example each neuron in layer 2 is influenced by an overlapping 3×3 region of neurons in layer 1.

Neurons on a convolutional layer perceive only a small rectangular area in the input image. The same parameters are swept across the image making the detections the same irrespective of the location. The motivation for such a model is to decrease the number of trainable parameters and is justified by the fact that interesting features in natural images are local and can be captured by small, local windows. The weight sharing and local connectivity features both drastically reduce the number of learnable parameters.

The calculation is carried out by the mathematical convolution operation, which is defined [13] for the discrete 2D case as

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n), \tag{2.14}$$

where I is the input image and K is the convolution kernel. The kernel K is optimized during training.

Like in fully connected networks, the output of the convolution operation is then transformed by an activation function. Stacking multiple convolutional layers causes the first layers to focus on natural features such as edges and the layer layers detect more complex shapes by composing the detections of the earlier layers. With a suitable hierarchy, the neurons in the highest layer activate when they detect high level features such as object presence in an image.

Deep convolutional neural networks created the field of deep learning. Deep models contain multiple convolutional layers where neurons higher in the hierarchy are influenced by larger areas of the input images. This area is known as the receptive field.

2.1.5.1 Pooling

Pooling layers split their input into $N \times N$ non-overlapping windows, and reduce the values within each window to one at their output. Two commonly used pooling operations are max pooling where the maximum value of each window is preserved and average pooling where the values are averaged together.

The pooling operation is carried out separately for each channel of the feature map so that the output of pooling a $H \times W \times C$ map, where H is the height, W is the width and C the number of channels, with a 2×2 pooling operation results in a tensor with the shape $W/2 \times H/2 \times C$. Pooling layers are parameterised by the window size N and contain no learnable parameters.

The motivation behind pooling layers is twofold. First, they introduce position invariance to the detections as the pooling operation discards information on the location of the values. Pooling also decreases the size of the output feature map, which grows the total receptive field of the network and is beneficial in enabling faster inference and training, albeit at the expense of spatial accuracy.

2.1.5.2 VGG-19

One commonly used model is the 19-layer VGG-19 [22] developed by the Visual Geometry Group at the University of Oxford. The network is used for the classification of 224×224 RGB images into one of 1000 categories. The network is shown in Figure 2.6. The VGG team achieved the first and second places in the localisation and classification categories of the ILSVRC-2014 competition [23].

The VGG-19 network contains 19 layers with learnable parameters in total. On high level it contains five stacked modules that each contain several convolutional layers and finally a max pooling layer to reduce the size of the feature maps. To be useful for classification, the output of the final module is then flattened to a single vector, which after three fully-connected layers is finally transformed into a probability distribution across 1000 classes by using a soft-max activation function [13].

2.1.6 Recurrent Neural Networks

Recurrent neural networks (RNN) [13] contain feedback loops, where some neurons receive their previous state in time as an input. They are often used in tasks that involve time series modeling, such as speech recognition [7]. A neural network with a recurrent layer is shown in Figure 2.7.

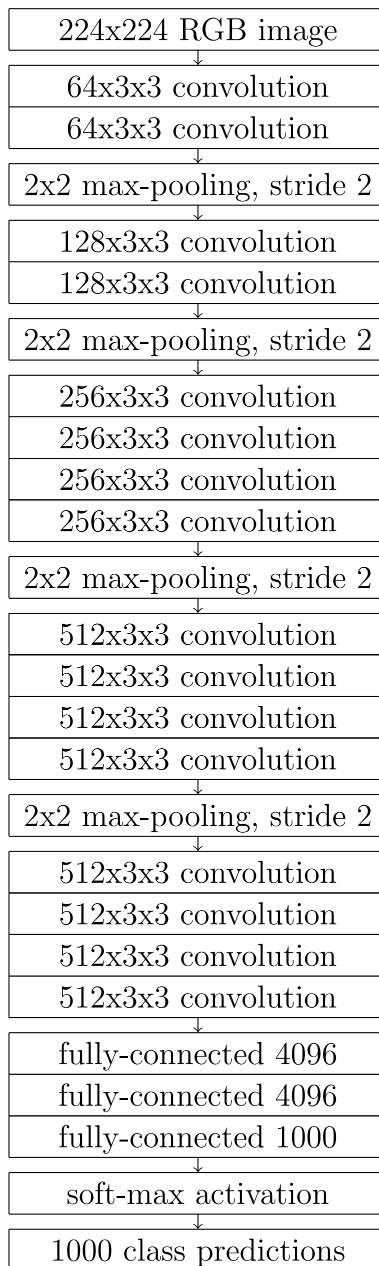

Figure 2.6: The VGG-19 network for image classification [22]. The input to the network is a 224×224 RGB image and the network classifies it to one of 1000 classes. All convolutional layers have a 3×3 receptive field with a varying depth. The network uses 2×2 max-pooling with stride 2 to reduce the resolution of the feature maps as they transform through the network.

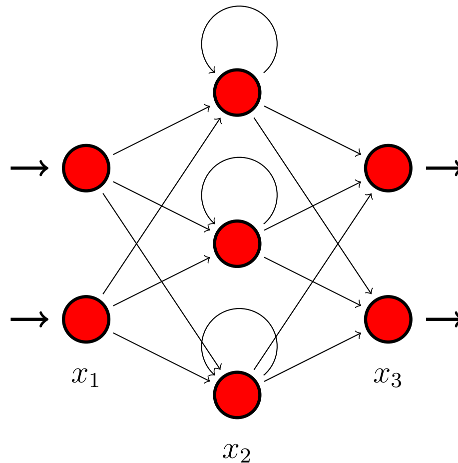

Figure 2.7: A recurrent neural network (RNN). Neurons on the second layer have a feedback loop so that their next activation depends on their previous activation in time.

The activation of a recurrent layer is formalized as

$$h_t = \sigma(Wx + Uh_{t-1} + b), \tag{2.15}$$

where x is the activation of the previous layer in the network and W is the associated connection weight matrix. The activation on the previous time-step of the recurrent layer h_{t-1} is multiplied by the weight matrix U. Finally, σ is the activation function and b the bias vector of the layer.

RNNs are particularly susceptible to vanishing gradients, where the neural activations saturate after multiple steps and the magnitude of the gradient vector ends up very close or equal to zero, making further learning halt. The following subsections describe extensions to the basic RNN that are more robust against this problem.

2.1.6.1 LSTM

Sepp Hochreiter and Jürgen Schmidhuber developed an extension to recurrent neural networks called the Long Short-Term Memory (LSTM) [24]. It combats the vanishing gradients problem by explicitly learning when to modify the hidden state via gates. There are three gates, forget gate, input gate and output gate, each of which acts as a one layer neural network on its own. The output of each gate is limited to $[0, 1]$ via a sigmoid activation and all gates observe both the previous hidden state and the current LSTM layer input.

The forget gate outputs a multiplier vector for the hidden state, where zero output causes the hidden state to be discarded and one to be kept. The input gate controls when to use information present in the input, and the output gate controls how the hidden state influences the network output.

All gates are learned simultaneously with the neural network, as all the internal operations of the layer are differentiable and their effect to the final loss function can be inferred. It can be said that during optimization the LSTM learns which features of the input are worth keeping in the hidden state and which features of the hidden state are not useful for prediction and can be discarded.

By explicitly modifying the hidden state via learnable gates, the LSTM sidesteps the vanishing gradients problem in simpler RNNs where long term dependencies are easily lost after repeated transformations through a hidden layer. In LSTM, the hidden state is modified only via the gates and it is very easy for the state to flow through multiple time steps unchanged.

2.1.6.2 GRU

The Gated Recurrent Unit [25] or GRU is a simpler variant of the LSTM recurrent layer. By omitting the output gate the GRU has fewer parameters to learn and the output of the layer is the same as the hidden state. Despite the difference in complexity the GRU has been shown to have a performance comparable to the LSTM [26].

The GRU is defined as the set of equations

$$z_t = \sigma(W_z x_t + U_z h_{t-1}) \tag{2.16}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1}) \tag{2.17}$$

$$\widetilde{h_t} = \tanh(W x_t + U(r_t * h_{t-1})) \tag{2.18}$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \widetilde{h_t}, \tag{2.19}$$

where z_t and r_t are respectively the learnable update and reset gates that use the sigmoid activation function σ. The activation of the gates depends on the current input to the GRU layer x_t and the previous hidden state of the layer h_{t-1}. The arguments U, U_r, U_z, W, W_r, W_z are learnable weight matrices. Here $*$ is used to denote elementwise multiplication. The output of the GRU layer is the same as the hidden state, and thus it can be defined that

$$h_t \triangleq \mathrm{GRU}(x_t, h_{t-1}), \tag{2.20}$$

for some set of parameters U, U_r, U_z, W, W_r, W_z.

2.2 Object detection with neural networks

Convolutional neural networks have been successfully applied to detecting and localizing objects in still images [2, 3, 5, 27]. This section describes the gridbox model for object detection and reviews existing work in object detection with neural networks.

2.2.1 Gridbox model

The gridbox model is a deep convolutional neural network for detecting bounding boxes of objects of interest in an image. The detections are a set of rectangles in the image space and associated confidence values where higher confidences denote higher certainty. The model can be used to both localize and classify objects as was done in e.g. the Single Shot MultiBox Detector (SSD) [2].

The model is built on a stack of convolutional layers. The final layer neurons have large, overlapping receptive fields over the input image and are very effective feature detectors. These 'superpixels' are forwarded to a final gridbox layer, that outputs the object bounding boxes and confidences based on the output of the convolutional layers. The gridbox layer is a convolutional layer with a 1x1 kernel, that outputs multiple channels for each superpixel. These channels include the bounding box coordinates (left, top, right, bottom) in pixel space. The output size of the final layer grows with the size of the images shown to the first layer of the network, because the network is fully convolutional.

Because the receptive fields of the superpixels overlap heavily there will be multiple bounding box predictions per object present in the image. The bounding boxes are noisy and fluctuate around the actual object in the image. Thus, to form the final bounding box prediction list the predictions are clustered using a clustering algorithm and the bounding boxes of each cluster are averaged together to obtain the final predictions of the network.

The gridbox layer also outputs a coverage value for each superpixel. The coverage value is used to measure the models certainty that the superpixel has detected a car. In this work, each final bounding box is a weighted sum of the raw bounding boxes predicted by the network, where the associated coverage in each bounding box is used to control the amount of contribution to the final bounding box.

The clustering phase is a separate post-processing step of the object detector framework and not a part of the computational graph. This means that the parameters of the clustering algorithm cannot be optimized by the

Figure 2.8: One validation frame from the KITTI tracking dataset [6]. Ground truth bounding boxes and velocity vectors are drawn in red and the detected objects and predicted velocity vectors are in green. The model has correctly predicted the velocity vectors for the cars based on their locations in the previous frames.

network training process but instead have to be found by cross-validation.

In order to form the loss function, a ground truth bounding box and a coverage value is generated for each superpixel. The coverage $\in [0, 1]$ denotes whether the superpixel is inside of an object or not. The loss function used to train the gridbox model is

$$L_{\text{cov}} = \|c_{\text{true}} - c_{\text{predicted}}\|_2, \tag{2.21}$$

$$L_{\text{bbox}} = c_{\text{true}}\|b_{\text{true}} - b_{\text{predicted}}\|_1, \tag{2.22}$$

$$L = \alpha L_{\text{cov}} + \beta L_{\text{bbox}}, \tag{2.23}$$

where c are the coverage values and b are the bounding box coordinates. The constant scalars α and β control the weight given to each objective. Note that c_{true} is used to multiply the latter term so that the bounding box predictions do not affect the loss in places where the ground truth coverage is zero.

2.2.1.1 Velocity estimation

The gridbox layer can be used to regress other object features apart from the bounding boxes. In this work the gridbox layer also outputs an approximate image plane object velocity vector for each object detection. During clustering the velocity vectors are averaged together by weighting them with their associated coverage values as is done to the bounding boxes. Figure 2.8 visualizes some velocity predictions made by the model.

The ground truth velocity vectors for each frame are formed by subtracting the bounding box center position from the previous frame from the corresponding box at the current frame. The mean squared velocity L2 norm between the predicted velocity and the ground truth velocity is added to the loss function of the network:

$$L_{\text{velocity}} = c_{\text{true}} \|v_{\text{predicted}} - v_{\text{true}}\|_2, \tag{2.24}$$

$$L = \alpha L_{\text{cov}} + \beta L_{\text{bbox}} + \gamma L_{\text{velocity}}, \tag{2.25}$$

where γ is a constant scalar multiplier that controls the weight of the velocity error term.

2.2.2 Video processing

To process videos, a convolutional network can be extended to three dimensions (width, height and time) so that the convolution kernel observes several consecutive frames at once and outputs features that describe local object motion. However, this method can have issues with scaling: A 3D convolution layer that observes N time steps and is W features wide, H features high and D features deep while outputting O features has $N \times W \times H \times D \times O$ parameters. The size of the filter grows linearly with number of time steps, causing the number of parameters to rapidly grow as longer sequences are modeled. Moreover, during evaluation of the model the user has to supply all frames of the video sequence to the model at once to create a prediction again increasing memory usage.

Recurrent layers can be combined with 2D convolution so that the output of the convolution at each frame is used as input to the recurrent layer. The recurrent layer can be built on top of 2D convolutional layers so that both the input and the hidden state of the recurrent layer is a tensor describing the whole image. This kind of a layer requires $2 \times W \times H \times D \times O$ parameters and is independent of the length of the sequence N. Evaluating such a model only requires one to store the previous hidden state of the recurrent layer, since the prediction is only dependent on the hidden state and the input image of the current time step.

2.2.3 Related work

Object detection using convolutional neural networks is a well-researched topic [2–5]. Easy access to GPUs has led to many people using neural networks in their object detection research.

Existing approaches to object detection from still images using neural networks can be roughly put into two categories. Object proposal methods use a multi-phase pipeline, where some simpler method is first used to extract coarse proposal bounding boxes or regions of interest from an image and the boxes are then refined or discarded by a convolutional neural network. Others use a single deep neural network that predicts object bounding boxes directly from a given image.

DeepBox [27] is an object proposal method. They define a deep neural network based model that detects whether raw bounding boxes contain an object or not. Their method is independent on the source of the raw bounding boxes. Disentangling the two components means that they have to be trained separately and the neural network cannot communicate to the object detector how it could improve its detections to increase the accuracy of the whole model.

OverFeat [5], YOLO [3] and SSD [2] are end-to-end trainable neural network approaches. They are also called sliding window detectors since the neural convolution layer can be interpreted as a pattern recognition layer that is slided across the picture. There is no separate proposal generation phase and the neural network predicts object bounding boxes directly from images, making the training process considerable simpler.

There is a clear performance difference between object proposal methods and direct neural networks. DeepBox reports 4 frames per second (FPS) inference rate for single images whereas one version of the SSD can detect objects at 58 frames per second, and Fast YOLO achieves 155 FPS, making them suitable for real-time applications.

The gridbox method outputs a large number of raw bounding boxes and one is required to group the ones corresponding to the same object using a clustering algorithm. The clustering phase is separate from the neural network training and thus the hyperparameters of the clustering algorithm will have to be tuned by hand. A way around this was described by Stewart and Andriluka [28], who put a recurrent layer on top of a convolutional neural network to analyze the outputs. The recurrent layer is not unfolded in time and the network still processes still images, but the recurrent layer is allowed to observe the output of the convolutional network multiple times in succession, each time outputting one new bounding box detection. This makes the whole model end-to-end trainable using neural network optimization methods such as Adam [20]. The recurrence poses new challenges in the formulation of the loss function as the order in which predictions are output from the network can be arbitrary, but they still have to be matched to the ground truth labeling order to allow gradient propagation. To match the generated bounding boxes to ground truths they define a distance measure

between pairs of boxes between the sets and use a polynomial-time algorithm to find the minimal cost matching.

Video processing using recurrent neural networks was explored by Tripathi et al. [29]. They built a two-phase system using two specialized neural networks. First, a YOLO [3] network pretrained on the PASCAL VOC [30] is trained on the YouTube Objects [31] dataset. The object detector is used to detect objects frame by frame and the obtained bounding box predictions are then used as training data for a second neural network, which consists of a recurrent GRU layer. The GRU layer predicts temporally coherent bounding boxes each frame by taking into account the previous frames in the sequence. The authors implement an objective function that contains a term that encourages smoothness across consecutive predictions, allowing the network to learn object detection from weakly-labeled data. The training sequence could contain only ground truth labels for some of the images and the temporal smoothness objective guides the object detector's learning across frames, even learning from frames without labels. While their convolutional YOLO network alone achieves 61.66 mean Average Precision (mAP) score in the Youtube Objects dataset, adding the recurrent component and the associated regularization objectives improves it to 68.73 mAP.

In this work a single-phase end-to-end trainable deep neural network model is used. The model receives full images as an input and produces bounding box coordinates and coverage values as an output. Unlike in DeepBox [27], no separate component is used to first extract region proposals from images. A recurrent layer is installed on top of a stack of convolutional layers, which differs from YOLO [3], where no recurrent layers were used. Tripathi et al. [29] described a model where the recurrent layers received bounding box coordinate predictions from the below convolutional layer, whereas in this work the recurrent layer receives arbitrarily complex neural activations from the convolutional layers.

Chapter 3

Methods

An effective ADAS should be able to predict the movement of other vehicles and objects in the traffic. The object detection module should be able to sense the movement of the surrounding vehicles and provide this information to the actual driving system. The predictions should be accurate even in the presence of temporary occlusions.

This chapter describes several models for object detection in video sequences and describes the experimental methods and evaluation metrics used to compare their performance.

3.1 Object detection using recurrent networks

While deep convolutional neural networks are effective at finding objects in single independent frames, a recurrent layer excels at integrating temporal context. Tripathi et al. defined [29] a neural network architecture where the frame-by-frame object predictions of a pre-trained convolutional neural network where used as inputs to a recurrent layer.

A similar approach is taken in these experiments, with the exception that the convolutional network is fine-tuned jointly with the training of the recurrent layer. The communication between the CNN and the RNN is learned concurrently with the optimization and is not limited to passing actual object bounding box predictions. The RNN outputs the final predictions for each frame while taking previous frames into account.

In addition to detecting objects, the models are capable of predicting objects' velocity vectors in the image plane. By allowing detections from previous frames to flow to the current frame via recurrent connections in the RNN layers, the network can keep track of moving objects across time and give estimates for the velocity. This capability will also be tested for the

non-recurrent models, where the models have to use visual cues such as the object's orientation to predict movement.

3.2 Recurrent forecast layer

Good performance can be achieved completely without a recurrent component by making predictions frame-by-frame. Temporary visual obstructions and unevenly lit frames are rare in the training data and thus there is little incentive for the network to learn temporal dependencies. To encourage the use of temporal context an extension on top of a recurrent layer such as RNN or GRU is developed and a new regularization term is added to the loss function.

The extension uses either the RNN or the GRU as a component. Like before, the output of the convolutional neural network is passed on to the recurrent layer and the recurrent layer outputs a prediction based on it and the layers previous activation. However, the same recurrent layer then calculates another output from a 'blank input' vector in place of the CNN output while observing the same temporal state. The blank input is a learnable vector that is constant across the feature map size.

The new regularization is added to measure the distance between the outputs of the two predictions made by the recurrent layer, the one made after observing a frame and the one made after observing blank input. By minimizing this distance, the recurrent layer is forced to predict what is visible in the image at frame $t = n$ while only indirectly observing the frame $t = n-1$. The regularization is weighted by a new hyperparameter δ that is optimized by grid-search. This new layer with a temporal prediction coherency loss is called the recurrent forecast layer and is illustrated in Figure 3.2. Formally, the regularization term is defined as

$$L_{\text{regularization}} = \|\text{GRU}(x_t, h_{t-1}) - \text{GRU}(B, h_{t-1})\|_2, \tag{3.1}$$

where x_t is the output from the convolutional network at time t, h_t is the hidden state (temporal context) at time t, and B is the trainable blank input vector. The learnable parameters inside the two recurrent layers are shared. The $\text{GRU}(x_t, h_{t-1})$ function is defined in Section 2.1.6.2.

Now, the final loss function of the recurrent forecast model is

$$L = \alpha L_{\text{cov}} + \beta L_{\text{bbox}} + \gamma L_{\text{velocity}} + \delta L_{\text{regularization}}, \tag{3.2}$$

where the scalar δ is used to control the amount of regularization applied during training.

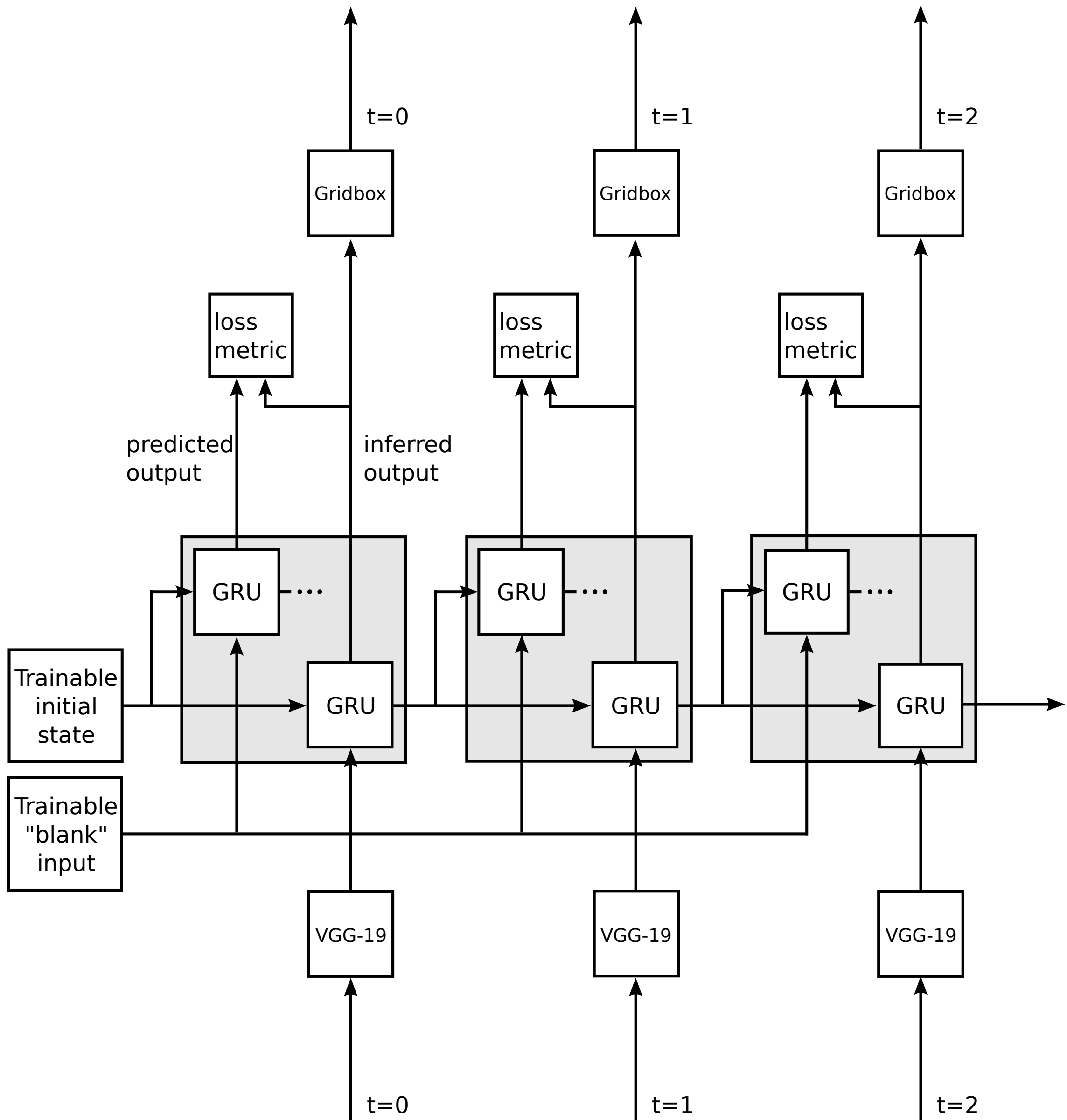

Figure 3.1: The recurrent forecast layer. The layer is expanded in the time dimension so that horizontal arrows denote time progress within the model. The network accepts multiple consecutive frames of video at once and forms predictions similarly to the other recurrent models discussed in this thesis by transforming the output of the VGG-19 convolutional neural network through a GRU layer, that is able to transfer features from previous frames via the hidden state. The same recurrent layer is then executed again but this time a blank feature map is shown to it instead of the VGG-19 output. The distance between the outputs of the invocations of the recurrent layer is then minimized by adding it as a regularization term in the objective function.

3.3 Experiments

The goal of this set of experiments is to compare the performance of different types of recurrent networks to the pure CNN model, which acts as a baseline. Five different gridbox models are evaluated in the KITTI [6] tracking dataset benchmark. Each model is a deep stack of convolutional layers followed by a specialized layer, which is either the RNN layer, GRU layer, the recurrent gridbox layer described in the Section 3.2, or a simple convolutional layer. A multi-frame CNN is added to observe multiple frames at once. The model (a) functions as a frame-per-frame object detector baseline.

The VGG-19 convolutional network described in Section 2.1.5.2 is chosen as the CNN component for the experiments. To enable the processing of images of arbitrary size, the fully connected layers are replaced with a gridbox-layer allowing the network to predict object bounding boxes, coverages and velocity vectors. The network is then pre-trained for object detection until convergence on the Cityscapes dataset [32], which consists of still images and a detection ground truth. After training, the gridbox layer of the resulting model is removed. This pre-trained and truncated VGG-19 is used as a building block for the models.

To measure the object detection performance of the studied models they are trained using the KITTI tracking dataset [6]. The dataset consists of video sequences from a camera placed in a car where the car instances are labeled each frame. The labels denote the bounding box coordinates (position and size) of the objects. The dataset also defines an error metric, which is based on the overlap of the predicted and ground truth bounding boxes. The velocity ground truth is a 2D vector generated by taking the difference between two bounding box centers in consecutive frames.

The computational complexity of the models can be compared by approximating the number of multiply-add operations required to produce a set of bounding box predictions for one image. For convolutional layers this can be calculated directly from the kernel size and input image resolution. The complexities of the (d) GRU and (e) forecast models are equal, since the extra temporal regularization term is only used during training. The models are summarized in Table 3.1 and illustrated in Figure 3.2.

Each model is trained for 90 epochs of the training set using the Adam (Subsection 2.1.4.6) algorithm and a base learning rate of 5×10^{-5}. The learning rate is evolved so that it increases linearly from 1% to the base rate during the first 10 epochs. This is used to prevent numerical instabilities in the beginning of the training. The learning rate is then kept at base rate until 45 epochs after which it is slowly annealed back to 1% of the base rate.

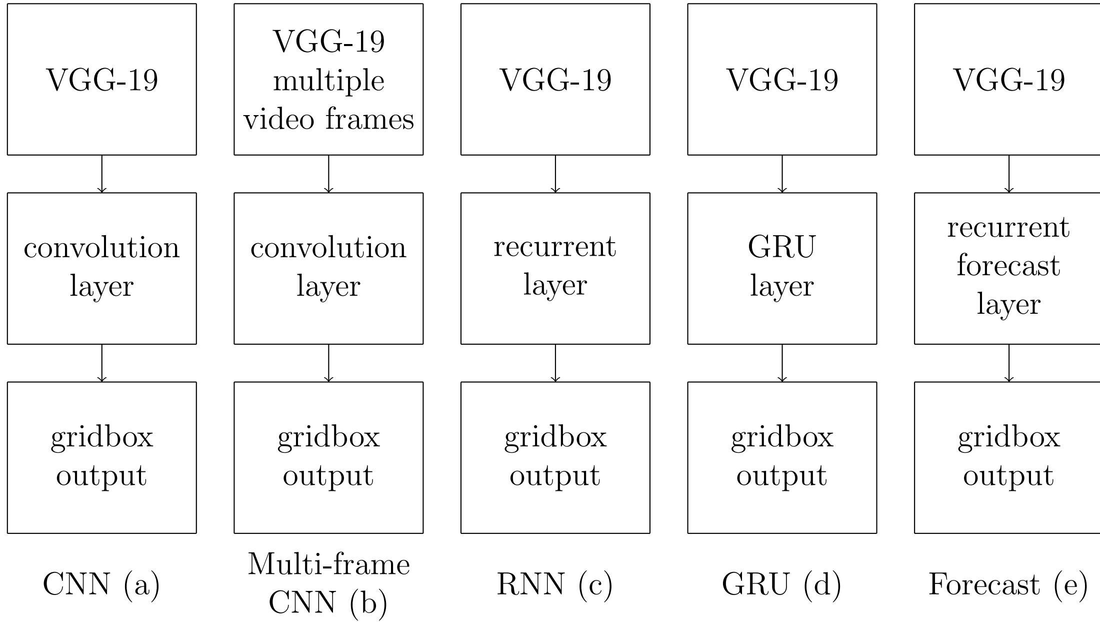

Figure 3.2: Five different models used for the comparison. Each network uses the same pre-trained VGG-19 network as its basis with the exception of (b), for which the first layer was retrofitted to accept several frames of video at once.

Model	Complexity	Parameters	Recurrent
CNN (a) (baseline)	1.89×10^{11}	2.12×10^{7}	No
M-CNN (b)	1.91×10^{11}	2.12×10^{7}	No
RNN (c)	1.90×10^{11}	2.24×10^{7}	Yes
GRU (d)	1.96×10^{11}	2.47×10^{7}	Yes
Forecast (e)	1.96×10^{11}	2.47×10^{7}	Yes

Table 3.1: Summary of the models. Complexity is calculated by counting the number of multiply-add operations required for producing bounding box predictions for one 1248x384 resolution RGB image. The third column counts the number of trainable parameters in each model.

Batch size of four is used in training. Thus, each model observes four frames of video at a time. Data augmentation by random cropping, horizontal flipping, zooming and gaussian RGB noise is used to prevent overfitting to the training dataset. To make the results more comparable across models the random number generator that selects the 5-fold cross-validation splits is initialized with a common random seed.

3.4 Performance metrics

The following metrics are used to compare the performance of the models.

1. **Object detection performance**

   Pascal VOC Challenge [30] defines a metric called Average Precision (AP), which is the area of the Precision-Recall curve of the detections. In addition, the KITTI [6] benchmark suite definition of object instance matching is used, where the predicted bounding box is considered to match the ground truth bounding box when their intersection-over-union (IOU) is over 0.7. The IOU metric for two bounding boxes is defined as

$$\mathrm{IOU}(b_1, b_2) = \frac{\mathrm{Area}(\mathrm{Intersection}(b_1, b_2))}{\mathrm{Area}(\mathrm{Union}(b_1, b_2))}. \tag{3.3}$$

2. **Weighted standard deviation of clusters**

   The gridbox layer gives multiple object bounding boxes and associated coverages for each detection and the final detection is obtained by averaging them. The weighted standard deviation of the bounding boxes before averaging can be interpreted as confidence, where more confident predictions have a smaller deviation. The bounding boxes b_i are represented as 4-element vectors where each element corresponds to one of the image coordinates $\{\mathrm{left}, \mathrm{top}, \mathrm{right}, \mathrm{bottom}\}$. The weighted standard deviation of the clusters is defined as

$$\sigma = \sqrt{\frac{1}{nA} \frac{\sum_{i=1}^{n} w_i (b_i - \mu)^2}{\sum_{i=1}^{n} w_i}}, \tag{3.4}$$

   where n is the number of bounding boxes in the cluster, w_i is the associated coverage of the ith bounding box b_i,

$$\mu = \frac{\sum_{i=1}^{n} w_i b_i}{\sum_{i=1}^{n} w_i} \tag{3.5}$$

is the weighted mean bounding box in the cluster and

$$A = \frac{\sum_{i=1}^{n} w_i \text{Area}(b_i)}{\sum_{i=1}^{n} w_i} \tag{3.6}$$

is the weighted mean bounding box area.

3. **Velocity estimation performance**

   Car velocity can be approximated in image space as the difference of the bounding box centers in consecutive frames. To predict object velocity in each frame, the network is tasked to output a two-element vector for each detection. Absolute velocity error is measured as the root-mean-square error (RMSE) difference between predicted velocity vector and the ground truth velocity vector, which is in pixels per frame. Additionally, the relative velocity error is defined as

$$\delta_v = \frac{\|v_{\text{predicted}} - v_{\text{true}}\|_2}{\|v_{\text{true}}\|_2}. \tag{3.7}$$

   Velocity estimation is discussed in more detail Section 2.2.1.1.

4. **Robustness to noise**

   Different levels of gaussian noise are added to the validation set images. The hypothesis is that recurrent networks should be able to integrate temporal information from previous frames in the detections and produce a higher average precision under noise than a baseline CNN.

   The amount of noise is described via the signal-to-noise ratio (SNR) measured in decibels, which is a logaritmic unit used to measure ratios between two units. x dB is equal to the ratio $10^{x/10}$. When used to represent the SNR, 10 dB is equal to 0.1 standard deviation of noise per standard deviation of signal. Infinite dB SNR is equivalent to zero amplitude noise.

Chapter 4

Results

Recurrent neural network architectures were tested against a baseline frame-by-frame feedforward convolutional neural network and its variant that observes several frames at once. The recurrent models are expected to be able to build a temporal context across several frames so that observations made in previous frames influence observations in later frames. The learning curves of the five fold cross-validation runs can be found in the appendix A.

Each model is trained with various levels of additive gaussian noise applied to them. The noise acts as a regularizer and improves the generalization performance. Noise is required to be able to infer on noisy input.

4.1 Bounding box standard deviation

The final predictions made by the gridbox model are weighted averages of bounding boxes predicted by the network. The network also predicts a coverage for each bounding box and higher coverages have higher influence in the final predictions. A clustering algorithm determines which bounding boxes are averaged together. The average weighted standard deviation of each model measured on the validation set is shown in Figure 4.1.

4.2 Object detection average precision

Each car in the KITTI tracking dataset is put into one category based on their detection difficulty, easy, moderate or hard. Cars in the hard category are usually small or highly occluded. In this test, the models' performance are compared in varying levels of gaussian noise applied to the input pixels, where infinite signal-to-noise ratio represents noise-free input. The mAP scores are stochastic due to the added noise during validation and thus the

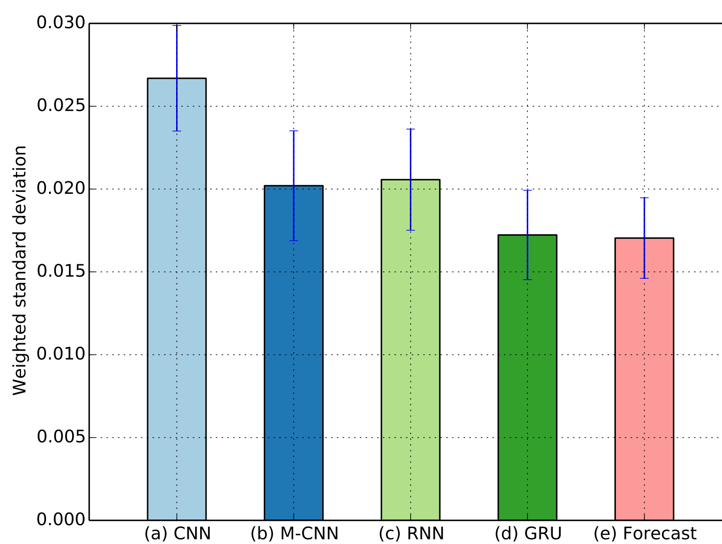

Figure 4.1: Model performances in the bounding box standard deviation test. Lower deviation can be interpreted as a higher confidence in the predictions and that the bounding box predictions made by the model have less jitter. The results are averages over 5-fold cross validation and the error bars show the standard deviation across different folds. Top-100 most confident bounding box clusters in the validation set of each model are considered, since the total number of clusters and predictions can vary across models and random seeds.

results are averaged over three tests with different random seeds. Figure 4.2 illustrates the results for the KITTI moderate category.

4.3 Velocity prediction performance

Each model is tasked to predict the velocity of the car in pixel space. Velocity is defined as the screen-space 2D pixel difference of the car bounding box centroid from the previous frame. The learning curve of the velocity error for each model is shown in Figure 4.4 and the relative velocity prediction error for different velocity ranges is shown in the Figure 4.5. To make the measurements comparable across velocity ranges that consist of different numbers of objects, the results only consider the top-10 smallest velocity error true positive detections.

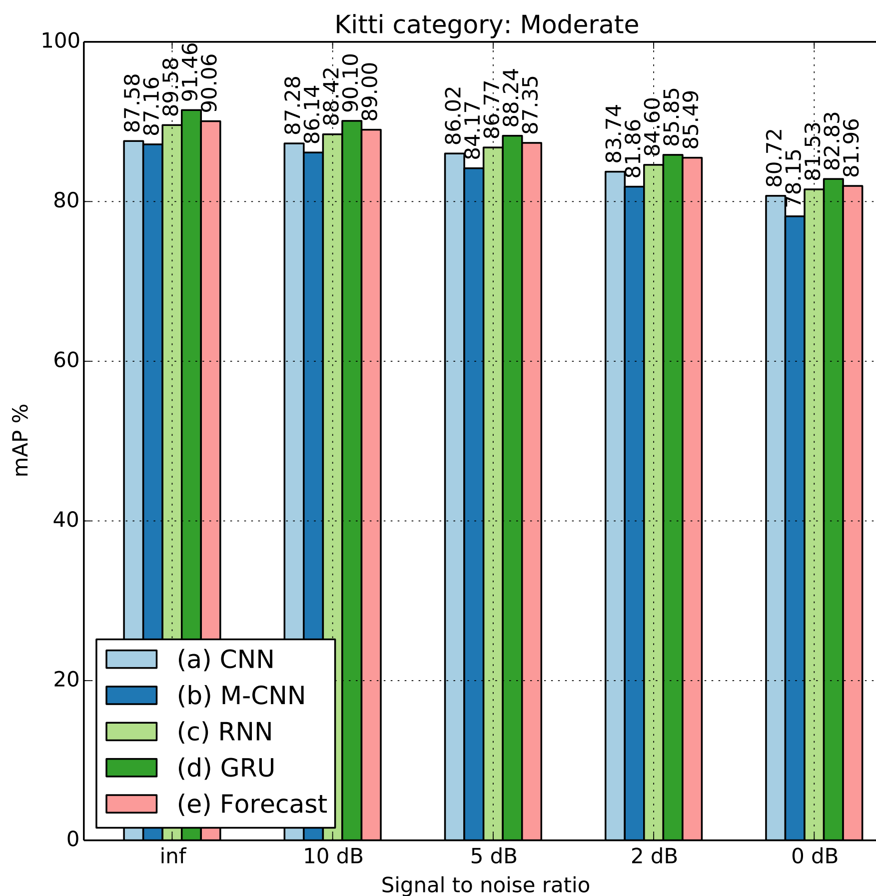

Figure 4.2: Mean average precision of each model in the KITTI tracking car detection dataset under varying levels of noise. The results are averaged over 5-fold cross validation and due to the stochasticity of the mAP calculation each fold is averaged over three random seeds. This figure shows the results of the KITTI moderate category.

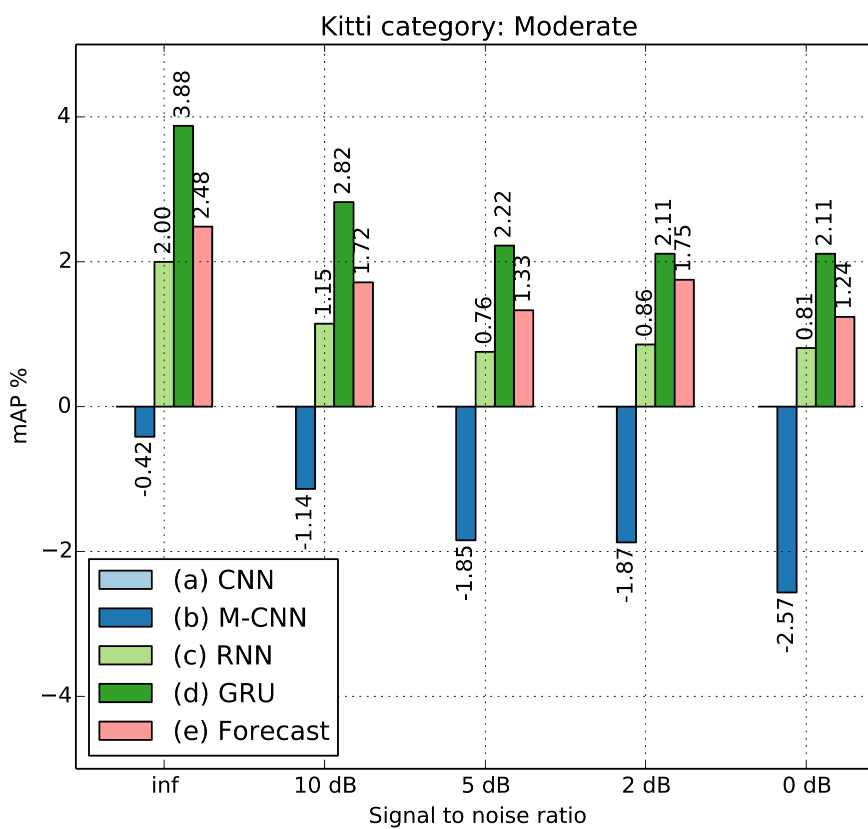

Figure 4.3: Difference in cross-validated mAP score in different models compared to the baseline CNN model under varying levels of noise.

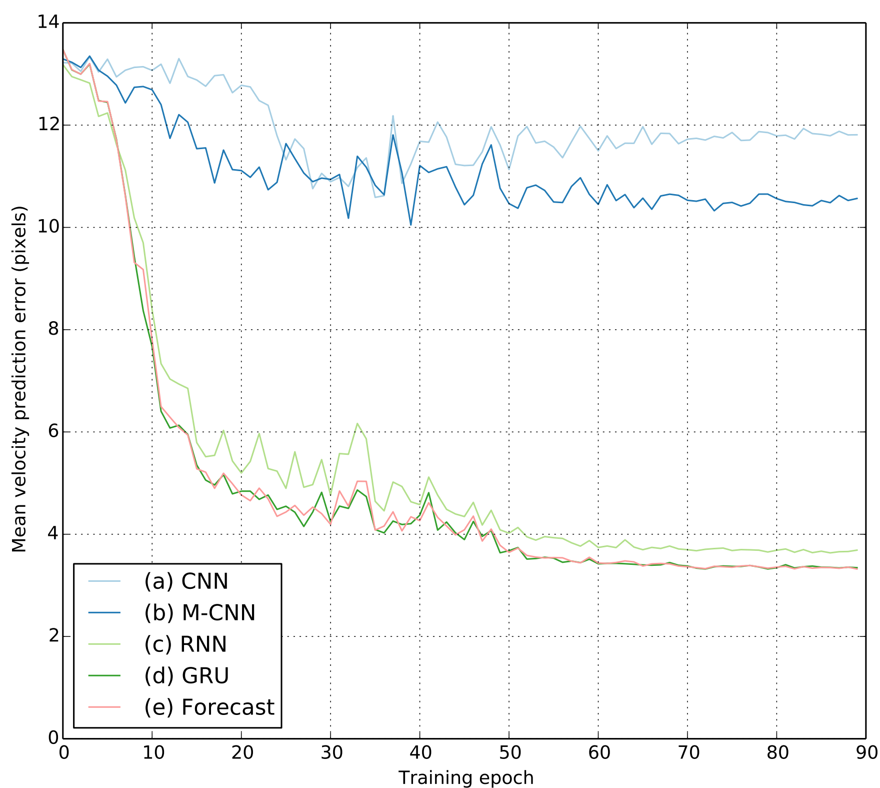

Figure 4.4: Evolution of velocity error loss in the validation set plotted against the training epoch number. The error is calculated as the mean pixel distance between the predicted velocity vector and the ground truth velocity. The curve is obtained by averaging across five folds of the training data. The co-occurring spikes in error are due to a common random seed used in training.

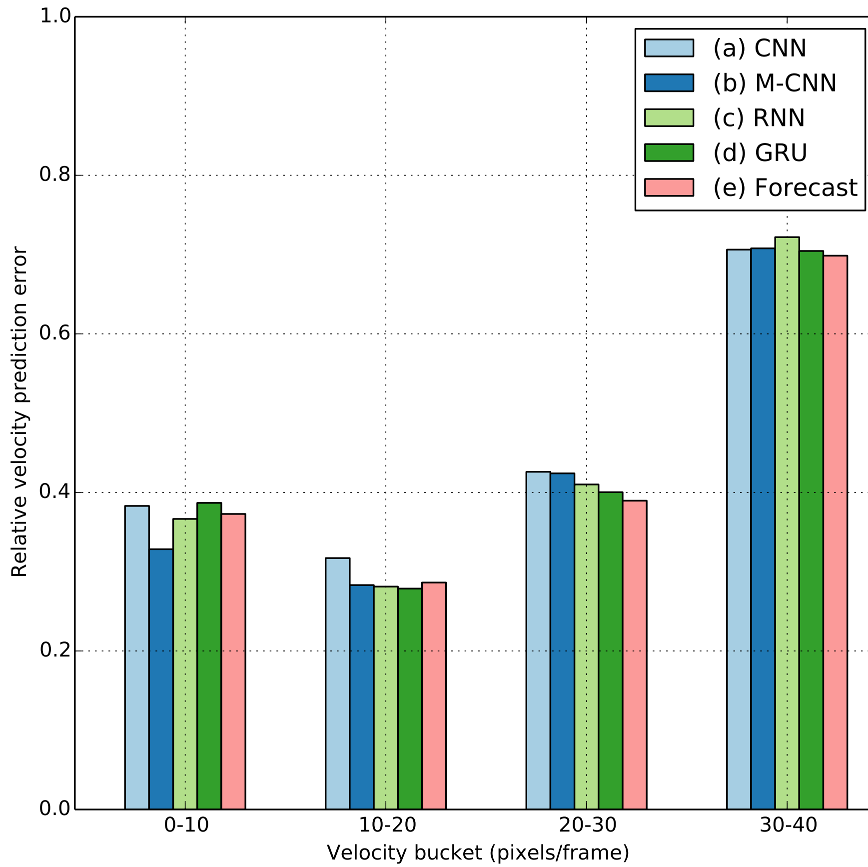

Figure 4.5: The relative velocity prediction error for different velocity ranges is shown. The velocity unit is pixels per frame. The error is defined as the ratio of the norm of the difference between the actual velocity and the predicted velocity over the norm of the predicted velocity. To make the results comparable across velocity ranges that consist of different numbers of predictions the relative error is measured only for top 10 true positive predictions made by each network in each bucket. The result is averaged across five folds of validation data, SNR levels $\{\infty, 10, 5, 2, 0\}$ dB and three random seeds.

Chapter 5

Discussion

In this chapter the results obtained in the previous chapter are discussed. Future research directions are considered at the end of the chapter.

5.1 Result analysis

The Figure 4.1 shows the standard deviation of the predicted bounding box clusters, where the GRU and the forecast model achieve the best results. The variance of the bounding boxes within a cluster can be interpreted as a confidence value, and both GRU forecast made the tighest clusters. Their temporal context evidently helps them achieve more confident results by combining detections across consecutive frames.

Looking at the KITTI car detection mAP scores in Figure 4.2, an expected trend can be seen in all categories: as the noise increases the performance of each model decreases. Also unsurprisingly, every model achieves their best scores in the noise-free category.

Interestingly the multi-frame feedforward network (M-CNN) performs comparatively poorly compared to the other models. One hypothesis for this behaviour is that the network is stuck in a bad local minimum due to the way it is initialized. The first layer kernel of the M-CNN network was initialized by repeating the pre-trained kernel four times in the channel dimension. This initialization makes each frame in the short sequence cause equal activations in the later layers and it might be difficult for the optimizer to adjust it so that it pay most attention to the last image in the batch, which is the one that corresponds to the ground truth labeling. In sequences with little movement the other frames are still roughly aligned with the ground truth labels.

The recurrent models (RNN, GRU, forecast) have a slight edge over the

Figure 5.1: CNN model velocity prediction performance shown against a validation frame from the KITTI tracking dataset [6]. The predicted vectors are red and ground truth vectors are green. Interestingly, even though the CNN model has no temporal context across frames, it is able to make statistical guesses about car movement based on their orientation. The velocity of the car on the front is incorrectly marked by a vector pointing to the left even though in reality it is close to stationary. The model may have learned that cars driving on the next lane usually move towards the viewer.

baseline CNN in all categories and noise levels. The difference is small and it is possible that the result would be different with a larger dataset.

The forecast model is slightly worse than the GRU model, which it is built on, suggesting that the added regularization term is limiting its object detection performance. By enforcing temporal continuity, it could have issues with frames where objects appear or disappear abruptly.

Velocity prediction performance varies significantly across models. The baseline CNN model has no concept of car movement across frames and has to make statistical guesses, such as that cars usually move forward. In fact, Figure 4.4 shows that the validation loss curve reaches its minimum in earlier phases of the training at around 35 epochs and after that the loss starts increasing. This is a sign of overfitting, meaning that despite the augmentations used during training the CNN has started to memorize the desired velocity vectors on the training set, which does not generalize well to new data. The Figure 5.1 shows some velocity predictions made by the CNN model.

The M-CNN achieves a slightly lower velocity error, showing that it is able to predict object movement somewhat from just a few consecutive frames.

It is also likely to overfit to the data, but it is not as evident in the curve. Interestingly it achieves the lowest relative velocity error in the 0-10 pixels per frame range, shown in Figure 4.5. The M-CNN architecture begins with one convolutional layer that combines four frames of detections to one. Since the merging happens early in the beginning of the model it has to use very low level image features such as edges to learn object movement, which could explain its ability to predict very slow movement. The merging also helps it differentiate from the baseline CNN in the 10-20 pixels per frame range.

Despite performing slightly worse in the mAP metric, the forecast model achieves the lowest relative velocity error in the velocity range 20-40 pixels per frame. The additional temporal continuity regression term allows it to keep track of object movement more accurately than the GRU model. However, the difference in error is quite small.

The RNN layer has a very simple time dependency parametrised by just a single weight matrix. Despite its simplicity the RNN model is much more effective at velocity prediction than the baseline feedforward network. In mAP scores the RNN doesn't show much improvement over the CNN. This is presumably because the CNN is already producing good object bounding boxes and it often works fairly well even without the temporal context provided by a recurrent layer. However, the recurrent layer does help when gaussian noise is added to the images.

The KITTI tracking dataset was used for these tests due to the requirement of consecutive video frames for object tracking. The achieved mAP scores are fairly good considering that the current best KITTI object detection challenge scores are slightly above 90 percent mAP[6]. However, the results are not fully comparable since the KITTI object detection challenge uses a slightly different dataset.

5.2 Future work

The goal of this work was to explore models for detecting objects in video sequences. The studied neural network models predicted only bounding boxes of cars, whereas an ADAS requires detecting many other types of objects, such as pedestrians and bicycles. In the future, the models could be extended to do classification simultaneously with detection.

It is straightforward to extend the the gridbox layer to predict many other object properties than just position and velocity in the pixel plane. By also predicting the object's distance to the camera one could also model velocity in three dimensions. An ADAS could also benefit from knowing the car's orientation.

Figure 5.2: An example of a missing label in the KITTI dataset [6]. The bounding box prediction made by the network is counted as a false positive as the occluded car has not been labeled.

Consisting of only 8000 frames of video, the KITTI tracking dataset is relatively small and despite the augmentations applied during training it is possible to overfit to the training data, as was observed with the learning curve of the velocity objective. The labels are also not perfect, as can be seen in Figure 5.2, making learning difficult.

Obtaining a large and accurately labeled training set is expensive, but as Tripathi et al. show [29], it is possible to formulate the problem in a semi-supervised manner, where the model can learn from sparsely labeled data. With additional priors on temporal consistency, the model can learn from frames with no bounding box targets as long as the object movement is predictable between consecutive frames. In the future, such loss terms could be integrated to this work as well.

Several neural network architectures were tested, but many other models exist in the literature. While the VGG-19 was chosen as the convolutional neural network component for this work it is not the only architecture that has proved to be useful for object recognition [2, 3, 33].

The convolutive recurrent layers in all of the tests used a convolutional filter with the size 3×3, which limits the mixing of the temporal state to the immediate neighbour neurons and possibly limits the highest object velocity that can be modeled. However, there is a dependency between object size and velocity as visualized in Figure 5.3. Small cars are located further away from the camera and thus move more slowly in the pixel plane. Larger objects are

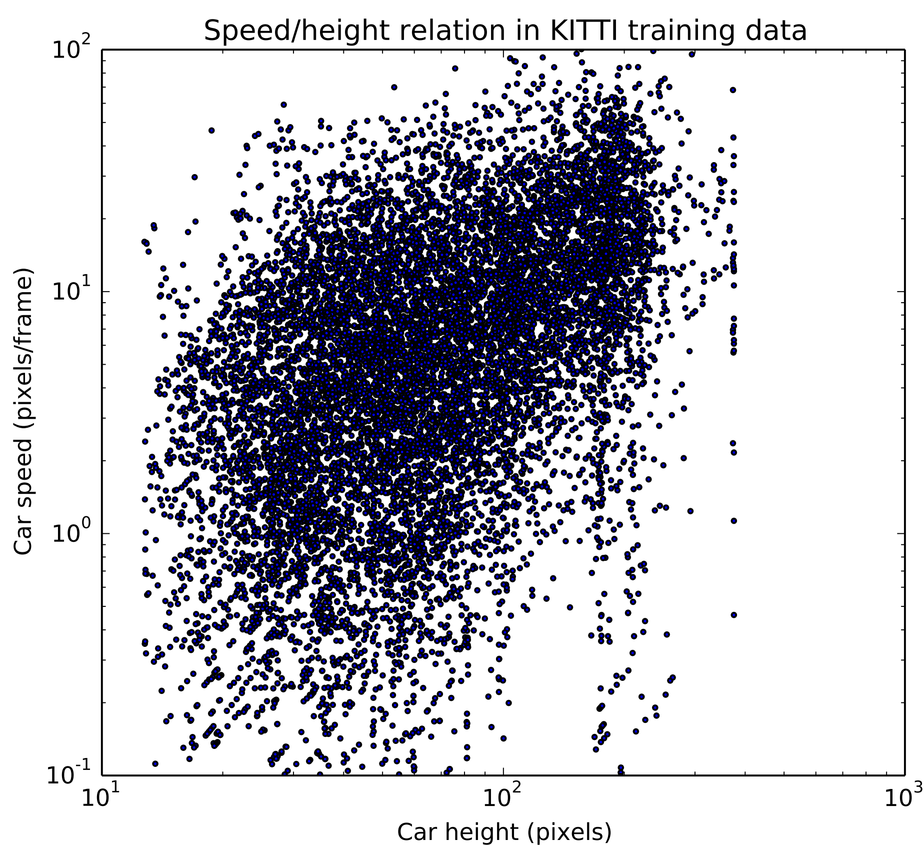

Figure 5.3: The distribution of car sizes and their velocities in the KITTI tracking dataset. A trend can be seen where larger cars often have a larger velocity than slow cars, which is expected as the size of the bounding box correlates negatively with the object's distance to the camera. Note the logarithmic axes.

easier to detect, since they cover a larger number of superpixels capable of assigning a bounding box to the object. The effect of the filter size to the car size and velocity detection performance could be examined in the future.

However, increasing the size of the filter leads to higher computational costs. In the future, the size of the filter and the position of the recurrent layers in the network architecture could be investigated more thoroughly also from the performance perspective. It is also worth considering if it helps to add more than one recurrent layer to the model.

Chapter 6

Conclusions

With the development of GPUs for high performance computing, neural networks are rapidly gaining popularity in computer vision tasks. This thesis showed how they can be used for object detection in video sequences and explored existing work in this area.

Several architectures for object detection in video streams were examined. The widely known VGG-19 object detector network was enhanced with different recurrent layers to create object detectors that make use of several frames of video for their detections. The motivation is that when a single-frame feedforward neural network is used to detect objects in videos the resulting object trajectory is noisy and requires postprocessing. A recurrent neural network can learn to track objects across multiple frames as a result of the learning and produce smoother detections. A convolutional single-frame object detector neural network acted as a baseline model for all comparisons.

The experiments measured the models' object detection performance under varying levels of gaussian noise applied to the images. The object detectors produce multiple bounding boxes for each object in the scene and the standard deviation of the boxes was used as a metric for confidence in predictions. Velocity prediction performance was also considered. Object velocity is defined as the pixel plane difference between the object centers in consecutive frames.

The results showed that the recurrent architectures outperform the frame-by-frame object detector and are more robust to noise. The models are also more confident in their predictions. The recurrent architectures also outperform the single frame baseline model in velocity detection tasks as they form a temporal context across several frames of video. Without knowledge of the previous frames the baseline model has to guess object velocity from cues such as object orientation, location and size.

Only a small set of recurrent models was examined and it is likely that the choices of learning rate, activation functions, kernel sizes, layer count, etc. were not optimal for this problem. The interactions between hyperparameters can be difficult to predict and the difficulty increases with every new hyperparameter. Finding the optimal architecture and parametrization for this problem was left for future research.

Neural networks are complex nonlinear models and creating them requires intuitive high-level understanding of the learning dynamics. This thesis gave some practical insights into choosing the network structure, learning rate and other hyperparameters. However, the field of deep learning is developing rapidly and more concrete guidelines for neural network design are likely to arrive in the future. In the end it is important that the experimenter evaluates many models suitable for their particular problem.

Bibliography

[1] World Health Organization. Global status report on road safety 2015. URL http://www.who.int/violence_injury_prevention/road_safety_status/2015/en/.

[2] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single Shot MultiBox Detector, 2015, arXiv:1512.02325.

[3] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection, 2015, arXiv:1506.02640.

[4] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Pranav Rajpurkar, Toki Migimatsu, Royce Cheng-Yue, Fernando Mujica, Adam Coates, and Andrew Y. Ng. An Empirical Evaluation of Deep Learning on Highway Driving, 2015, arXiv:1504.01716.

[5] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks, 2013, arXiv:1312.6229.

[6] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[7] Jürgen Schmidhuber. Deep Learning in Neural Networks: An Overview. *CoRR*, abs/1404.7828, 2014. URL http://arxiv.org/abs/1404.7828.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[9] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. URL http://neuralnetworksanddeeplearning.com/.

[10] Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. *CoRR*, abs/1510.00149, 2015. URL http://arxiv.org/abs/1510.00149.

[11] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks, 2015. URL https://arxiv.org/abs/1506.02626.

[12] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. ISSN 19391471.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. URL http://www.deeplearningbook.org. Book in preparation for MIT Press, 2016.

[14] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, Berlin, 2001.

[15] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, volume 9, pages 249–256, 2010.

[16] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[17] D Randall Wilson and Tony R Martinez. The general inefficiency of batch training for gradient descent learning. *Neural Networks*, 16(10): 1429–1451, 2003.

[18] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. 4(5):791–803, 1964.

[19] Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 2013.

[20] Diederik Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, 2014, arXiv:1412.6980.

[21] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014. ISSN 1532-4435.

[22] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*, abs/1409.1556, 2014.

[23] Large Scale Visual Recognition Challenge 2014 (ILSVRC2014). URL `http://www.image-net.org/challenges/LSVRC/2014/`.

[24] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667.

[25] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, 2014, arXiv:1406.1078.

[26] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, 2014, arXiv:1412.3555.

[27] Weicheng Kuo, Bharath Hariharan, and Jitendra Malik. DeepBox: Learning Objectness with Convolutional Networks, 2015, arXiv:1505.02146.

[28] Russell Stewart and Mykhaylo Andriluka. End-to-end people detection in crowded scenes, 2015, arXiv:1506.04878.

[29] Subarna Tripathi, Zachary C. Lipton, Serge Belongie, and Truong Nguyen. Context Matters: Refining Object Detection in Video with Recurrent Neural Networks. In *British Machine Vision Conference (BMVC)*, 2016. URL `http://vision.cornell.edu/se3/wp-content/uploads/2016/07/video_object_detection_BMVC.pdf`.

[30] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. URL `http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html`.

[31] Alessandro Prest, Christian Leistner, Javier Civera, Cordelia Schmid, and Vittorio Ferrari. Learning object class detectors from weakly annotated video. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, pages 3282–3289. IEEE Computer Society, 2012. ISBN 978-1-4673-1226-4. URL http://dx.doi.org/10.1109/CVPR.2012.6248065.

[32] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The Cityscapes Dataset for Semantic Urban Scene Understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[33] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *CoRR*, abs/1602.07261, 2016. URL http://dblp.uni-trier.de/db/journals/corr/corr1602.html#SzegedyIV16.

Appendix A

Learning curves

The learning curves of the five tested models are shown in this appendix. For K-fold cross validation each model was trained five times using different splits of the training set so that each training sample is used once for validation. Each figure shows the value of the loss function in the training set and in the validation set during training.

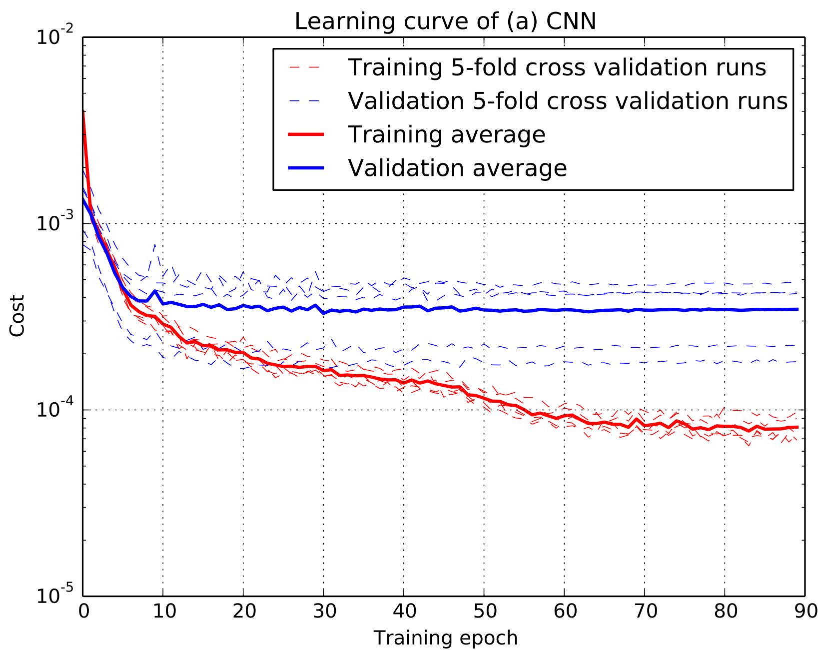

Figure A.1: The learning curves of the (a) CNN (baseline) model.

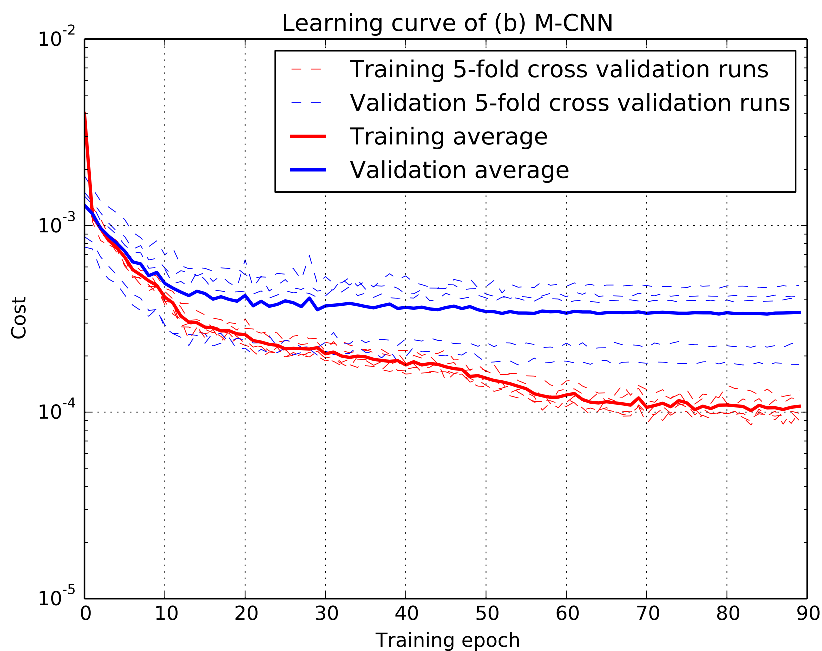

Figure A.2: The learning curves of the (b) Multi-CNN model.

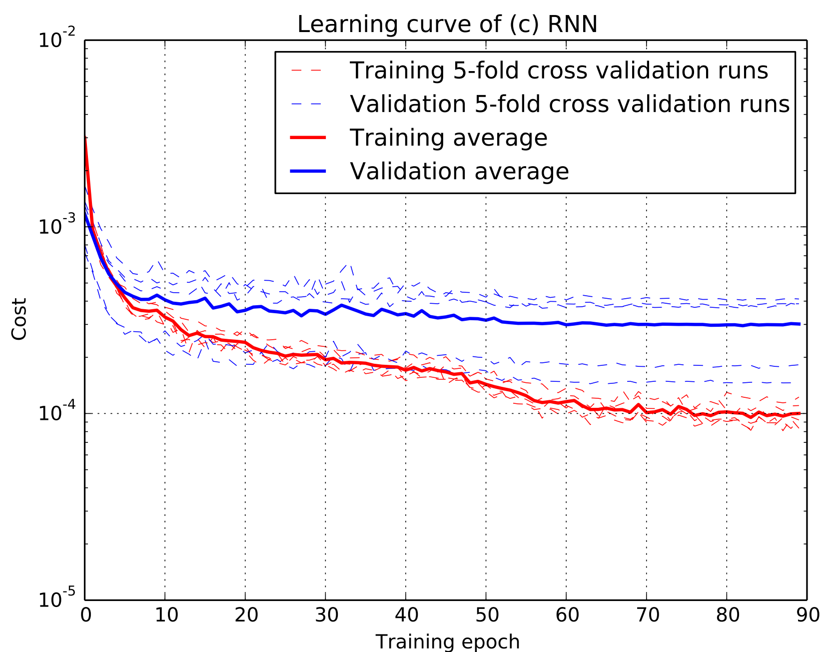

Figure A.3: The learning curves of the (c) RNN model.

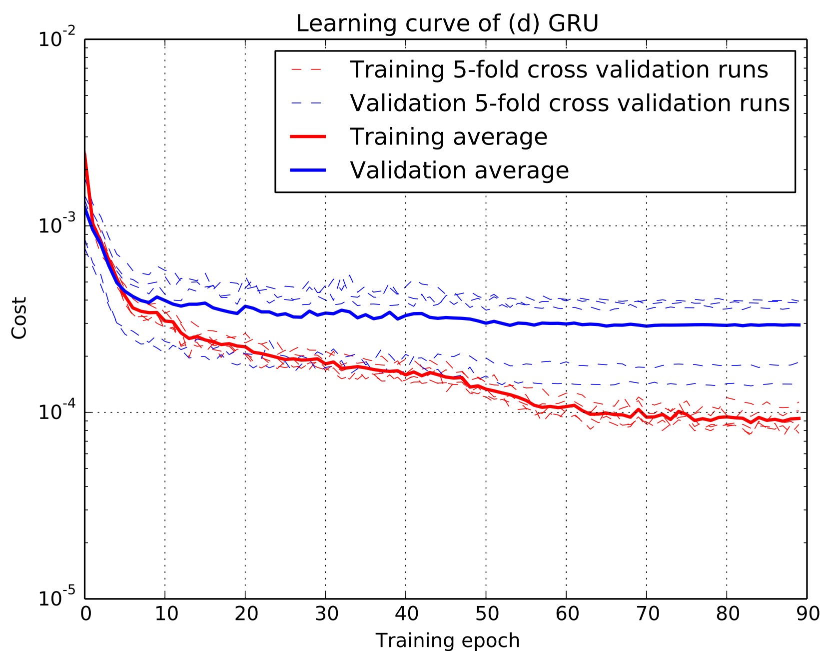

Figure A.4: The learning curves of the (d) GRU model.

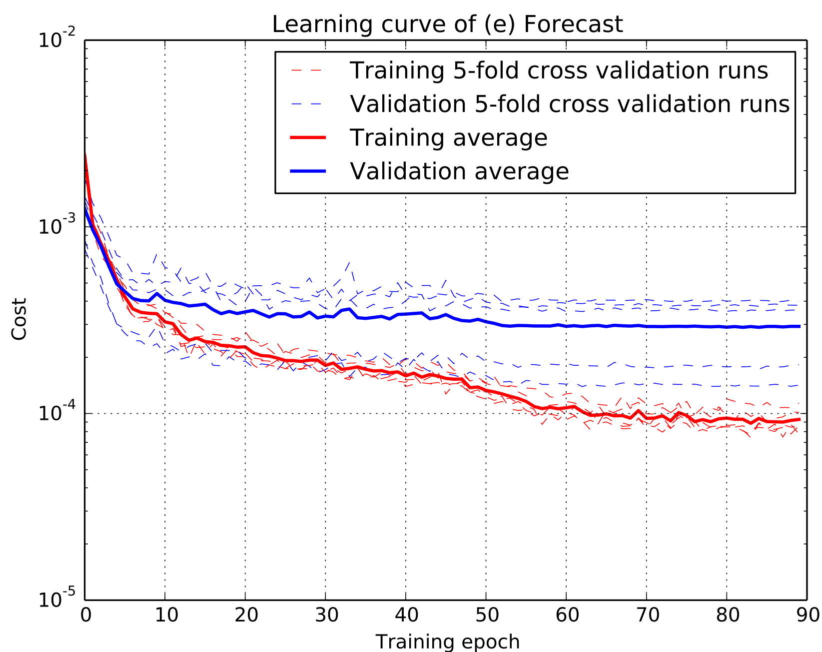

Figure A.5: The learning curves of the (e) forecast model.