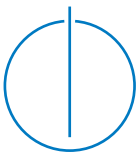
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Occupancy Grid Prediction for Highway Traffic Scenarios Using Deep Learning

Muhammad Mushfiqur Rahman



DEPARTMENT OF INFORMATICS

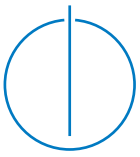
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Occupancy Grid Prediction for Highway Traffic Scenarios Using Deep Learning

Prädiktion von Belegtheitszuständen der Autobahn mittels Neuronaler Netze

Author:	Muhammad Mushfiqur Rahman
Supervisor:	Prof. Dr.-Ing. habil. Alois Christian Knoll
Advisors:	Christoph Schöller, Gereon Hinz
Submission Date:	January 15, 2019



I confirm that this master's thesis is my own work and I have documented all sources and materials used.

Munich, January 15, 2019 Muhammad Mushfiqur Rahman

Acknowledgments

I would like to express my sincere gratitude to those who offered feedback on the content or made this thesis even possible. At first, a special thanks to my supervisor Prof. Dr.-Ing. habil. Alois Christian Knoll for his outstanding lecture in Cognitive Systems: Neural Networks and Applied AI at TUM, which driven me to do a thesis in Artificial Intelligence field with Autonomous Driving. Next, my sincere thanks to my advisors Christoph Schöller and Gereon Hinz for offering this interesting topic and for providing this opportunity to work on such an interesting research project. Also, I would like to express my deepest appreciation to Christoph Schöller for his continuous support, guidance and technical help with ideas throughout the thesis work as well as our fruitful discussion at the 'Fortiss' office. Furthermore, I would like to acknowledge the Providentia team of 'Fortiss' for their continuous support and their valuable input.

Finally, I want to thank my family for always being there for me and helping me in various ways. Last but not least, a special thanks to Md Rezaur Rahman and Kazi Mahbub Mutakabbir for investing their time in proofreading this thesis.

Abstract

The ability to predict the future motion of road vehicles helps to plan safe maneuvers and avoid traffic accidents. The majority of current research focuses on prediction algorithms which handle interactions and conflicts between various driving intentions. However, they focus on the separate prediction of traffic participants instead of whole traffic scenes. In this work, we use 2D Occupancy Grid Maps (OGMs) to represent traffic scenes. And, the goal of the thesis is to predict future grid maps based on an observed history. To achieve that, three deep neural networks (DNN) - a CNN architecture, a recurrent CNN and a recurrent encoder-decoder architecture are examined. The first CNN network model is a Multi-Stream FCN(MS-FCN) model from which future single grid map can be predicted by exploiting spatial context of the past observed grid maps. The second one, ConvLSTM Encoder-Decoder (ConvLSTMED), is a RNN based encoder-decoder architecture which exploits Convolutional Long Short-Term Memory (ConvLSTM) to predict the next grid map at each time step. Finally, a sequence to sequence model was examined by implementing a Recurrent Encoder-Decoder ConvLSTM (RED-ConvLSTM) model which extrapolate the future grid maps from an input grid map sequence by discovering both spatial and temporal motion patterns. It has the advantage to predict multiple future grid maps from a single grid map input. All the proposed models' performances for grid map prediction are evaluated with established computer vision metrics and compare them extensively.

Zusammenfassung

Die Fähigkeit, die zukünftige Bewegung von Straßenfahrzeugen vorherzusagen, hilft dabei, die Sicherheit zu planen manövriert und vermeidet Verkehrsunfälle. Die Mehrzahl der aktuellen Forschungsschwerpunkte auf Vorhersagealgorithmen, die Interaktionen und Konflikte zwischen verschiedenen behandeln Fahrabsichten. Sie konzentrieren sich jedoch auf die separate Vorhersage des Verkehrs Teilnehmer statt ganze Verkehrsszenen. In dieser Arbeit verwenden wir 2D Occupancy Grid Maps (OGMs) zur Darstellung von Verkehrsszenen. Das Ziel dieser Arbeit ist es, zukünftige Grid-Karten basierend darauf vorherzusagen eine beobachtete Geschichte. Um dies zu erreichen, sind drei tiefe neuronale Netze (DNN) - ein CNN Architektur, ein wiederkehrendes CNN und eine wiederkehrende Encoder-DecoderArchitektur untersucht werden. Das erste CNN-Netzwerkmodell ist ein Multi-StreamFCN-Modell, anhand derer zukünftiger Einzelgitterkarte vorhergesagt werden kann Nutzung des räumlichen Kontextes der in der Vergangenheit beobachteten Gitterkarten. Und der zweite ist eine RNN-basierte Encoder-Decoder-Architektur welche nutzt Convolutional Long Short-Term Memory (ConvLSTM) zur Vorhersage des nächster Gitterkarten bei jedem Zeitschritt. Schließlich wurde ein Sequenz-zu-Sequenz-Modell untersucht, indem ein wiederkehrender Codierer-Decodierer implementiert wurde ConvLSTM-Modell, das die zukünftigen Gitterkarten aus einer Eingaberaster-Map extrapoliert Sequenz durch Entdecken von räumlichen und zeitlichen Bewegungsmustern. Es hat Der Vorteil, mehrere zukünftige Gitterkarten aus einer Einzelgitterkarte vorherzusagen. Alles Die Leistungen der vorgeschlagenen Modelle für die Gitterkartenvorhersage werden mit bewertet etablierte Computer-Vision-Metriken und vergleichen diese ausführlich.

Contents

1 Introduction

Progress in scientific research on artificial intelligence and robotics made automotive industry develop more “intelligent” vehicles. Already today Advanced Driver Assistance Systems (ADAS) promise to enhance road safety and the security of drivers. In the coming years, it is likely that vehicles will be able to drive completely without human intervention [1]. In order to drive safely and efficiently in highway traffic, autonomous vehicles must anticipate the motion and intentions of surrounding vehicles and adapt their own behavior accordingly. Usually, a human driver predicts surrounding vehicles' and other traffic participants' motion up to a few seconds in the future. This helps to understand the environment of the vehicle and drive in a foresighted manner.But, in contrast, most autonomous systems are unable to perform such medium-term forecasts. From an intelligent transportation systems (ITS) point of view, to be able to predict the behavior of other traffic participants, like cars and trucks, motion models of traffic participants are needed. Making reliable predictions is a difficult task due to complex interactions between traffic participants. Previous research focused on predicting the future motion of surrounding vehicles based on kinematic models or interacting multiple models [2, 3]. However, in complex driving scenarios, the interactions between two or more vehicles become important but challenging when predicting the future.

With the recent success of Deep Neural Networks (DNN) in the fields of Computer Vision, several DNNs approaches are widely used for future traffic participants motion predictions. For example, Morton et al. [4] used a Recurrent Neural Network (RNN) to model driver behaviors on highways. Their work investigated the use of different neural models to predict a distribution of the acceleration of cars in front of the ego vehicle in a car-following scenario for the next time step. In [5], a RNN Encoder-decoder framework is proposed for distant future prediction of multiple interacting agents in a complex scene. The RNN model incorporates both static and dynamic scene contexts. It predicts future locations of objects by applying the multi-modal nature of the future prediction, making a strategic decision based on that. These works inspire the future research to model a bird's eye view of future motion prediction of all the involved traffic participants using deep neural networks (DNNs).

Therefore, in this study, long-term motion prediction for all the vehicles within a traffic scene is addressed. In this work, we propose several deep learning approaches to exploit

the information of all the involved vehicles dynamics and the past traffic history to get the future state of the whole highway. To achieve this, we use a single channel occupancy grid map (OGM) to represent the current traffic scene and then feed it into deep neural networks as an input. It considers predictive motions of all the vehicles at once within a specific stretch of a highway. It means that only vehicles positions are taken into consideration while making OGM representation. With single channel OGM, all the vehicles current states are represented into grid cells as foreground value which reformed traffic scene into an image-like structure from bird's eye view.

In this thesis work, we propose three deep neural networks (DNN) - a CNN architecture, a recurrent CNN and a recurrent encoder-decoder architecture for predicting the future state of traffic scenarios as a whole. The first network is a Multi-Stream Fully Convolutional Network (MS-FCN) network which takes multiple input grid maps at various time-steps and predicts the single future grid map. The second network, ConvLSTM Encoder-Decoder (ConvLSTMED), is constructed by using a RNN with encoder and a decoder. As a RNN component, in this work, we use Convolutional Long Short-Term Memories (ConvLSTMs) [6], to exploit spatial context and capture temporal correlations between time steps. These two networks follow a many-to-one architectural approach where a single future grid map is predicted from a given multiple input grid maps. On the other hand, to predict multiple future grid maps, we developed a sequence to sequence model, named as Recurrent Encoder-Decoder ConvLSTM (RED-ConvLSTM). In the developed model, the recurrent network part is built of an Encoder-ConvLSTM, encoding a series of grid maps into a latent feature tensor, and, a Decoder-ConvLSTM to produce predictions for several time steps. The developed network architecture uses a recurrent skip architecture which helps to increase the network convergence speed and to smooth the loss landscape [7]. The network also benefited from a skip architecture because it preserves spatial information from shallow layers of an input grid map to get pixel-wise predictions at the output.

The remainder of this thesis is outlined as follows. In chapter 2, we review relevant related work. We explain necessary background information, such as artificial neural networks (ANNs), CNN, FCN, RNN, LSTM, and ConvLSTM in chapter 3. Proposed DNN architectures are discussed in Chapter 4. In addition, our spatial loss function, data structures, and the training process will be explained. How we pre-processed our data and prepared a dataset for training will be discussed in Chapter 5. Chapter 6 presents the experiments and comparison of our networks performances. Also, the evaluation metrics are described in this chapter. At the end of this work, we summarize our findings and provide perspectives for future research in Chapter 7.

2 Related Work

Numerous models for the prediction of vehicles future motion have been proposed based on different methods and different kinds of input data. Prediction of surround vehicle motion is an extremely challenging problem due to a large number of factors that affect the future trajectories of vehicles. For instance, one challenging factor is the interactions between two or more vehicles when predicting future motions of these multiple vehicles in real-time. One of the characteristics of these inter-vehicular interactions which increase the complexity of the problem is simultaneously occurring multiple interaction pairs according to the each vehicle's driving strategy. Also, interaction features are continuously changing according to the future driving strategies of each vehicle. Therefore, to solve these challenges, several deep learning based future prediction solutions [8, 9, 10, 11, 12] have been recently proposed. Both in [8], a Bayesian network based future prediction and in [9], a LSTM based future prediction, pair-wise interactions are considered. However, the models are not feasible to handle simultaneously occurring multiple interaction pairs because the relative state between only two vehicles is used. On the contrary, in our work, all the vehicles interactions are considered automatically from the observed grid maps.

Also, other deep learning approaches were introduced, such as convolutional neural networks [10], recurrent neural networks [11], and fully connected neural networks [12]. A CNN based approach is proposed in [10] to infer traffic participants' lane-change-intentions for enhancing Adaptive Cruise Control (ACC). They transformed real-world driving data into a simplified bird's-eye view representation that enables a computationally efficient CNN-based approach. However, their work only focused on the traffic participants' lane change intention without considering surrounding vehicles motion. In [11], a Long Short Term Memory (LSTM) based Recurrent Neural Network (RNN) model is proposed to predict driver intention as the vehicle enters an intersection. The LSTM model classifies which manoeuvre a driver may take through an intersection, given the particular approach path of the vehicle. Nonetheless, their work only focuses on classifying an individual participants' intention, but it also did not consider any surround vehicles motion or interactions. Multiple vehicles interaction was considered in [12]. But, the size of the proposed architecture depends on the complexity of the traffic scene, which demands high computational power for the increased number of involved traffic participants. Moreover,

a fully connected neural network is not capable of processing large state spaces like time-dependent feature. All mentioned prediction frameworks have in common that they are not able to handle simultaneous multiple interactions between vehicles. They all are limited to individual vehicles and pair-wise interactions. As already mentioned, in this work we represent the whole traffic scenes by 2D occupancy grid maps, which allows us to contain the dynamic state information about all vehicles at once. Then, proposed network models use grid maps to predict the future state of the vehicles in the highway.

In addition, the mentioned 2D grid maps are the image-like structure which is an abstract representation of the traffic participants. And each of the grid maps represents a time-dependent feature. Since the proposed RNN networks use sequences of grid maps as input to predict the future state of the highway scene predictions, the networks are similar to a future video frame prediction from past input frames. From this point of view, in recent years, many studies focus on future video frame prediction using DNN architectures. Initially, Ranzato et al. [13] introduced a generative model that uses a RNN to predict the next frame or interpolate between frames. Srivastava et al. [14] extended this work by utilizing a Long Short-Term Memory (LSTM) Encoder-Decoder architecture to reconstruct the current frame or predict future frames. Luo et al. [15] presented an unsupervised learning approach by using a convolutional LSTM architecture that compactly encodes the motion dependencies in videos to predict sequences of up to eight frames of optical flow in RGB-D videos. Villegas et al. [16] introduced a motion-content network for the task of frame prediction, which separates the information streams (motion and content) into different encoder pathways in an unsupervised learning fashion for prediction of the future. In this study, recurrent network part is inspired from the Encoder-Decoder framework introduced by Srivastava et. al in [14] based on [17]. However, they use Fully Connected LSTMs (FC-LSTMs) to predict future video frames. But, we use ConvLSTMs which outperform the FC-LSTMs in capturing spatiotemporal relationships in both recurrent models. Also, alternative neural network-based approaches for grid-based scene prediction are evaluated which are mostly differing in the way the input data is provided and how they can be applied in practice. In [18], rectangular grids are used to predict vehicle trajectories in a highway scenario using an Encoder-Decoder LSTM architecture, whereas in this study, we focus on the prediction of vehicle motion in the whole traffic scene.

Meanwhile, for single frame prediction, Jin et al. [19] trained a deep neural network model to predict the future scene parsing and motion dynamics of the immediate next frame from the previous input frames. Luc et al. [20], also worked on the problem of predicting future semantic segmentation. They also considered single-frame future prediction and designed an autoregressive method to predict next frames up to 0.5 seconds

into the future. However, both of their network models require ground-truth optical flow annotations, which are difficult to obtain, whereas in this study, the grid map dataset is formulated such as that each recorded sequence of frames is automatically labeled with a next grid map frame as ground-truth. In short, the grid map dataset is designed in such a way that each dataset row contains 4 grid map frames sequentially, where first 3 frames are used as input and the last frame is denoted as ground truth.

In this work, we aim to provide several DNN prediction models to get the future predictive motion state of all the involved participants within traffic scenarios based on already observed sequence.

3 Background

This chapter provides background material and literature review for artificial neural networks and deep learning in general. It covers the theoretical concepts, terminology, and components that are relevant to this thesis. Many of the concepts and components were used to construct the models of our approach. The chapter starts with simple feedforward networks, describes the general procedure of how neural networks are trained, discussed in Section 3.1 and continue with Convolutional Neural Networks(CNN) and Fully Convolutional Networks(FCN) in Section 3.2 . Finally, the chapter ends with a description of the Recurrent Neural Networks(RNN) and the Convolutional Long Short-Time Memory Networks(ConvLSTM) that take advantage of the data's spatial and temporal properties in Section 3.3.

3.1 Artificial Neural Networks

The term Artificial Neural Networks (ANN) – or short Neural Networks (NN) – was first introduced by Warren McCulloch and Walter Pitts while they attempted to find a mathematical representations for information processing in biological systems in the early 1950s. Inspired by this work, Frank Rosenblatt developed the perceptron which is the basis of artificial NNs to this day [21].

3.1.1 Basics

Perceptron and Neurons

The perceptron 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. It has quite a simple structure. It takes several weighted real-valued inputs $x_i \in \mathbb{R}$ and summarizes them, and if the combined input exceeds a threshold it will activate and send an output y with x_i, $y \in \{0, 1\}$ [22]. Each input is weighted by $w_i \in \mathbb{R}$ to represent the importance of each input. The output of the perceptron is determined by the activation function. It uses a Heaviside step function or the unit step function as an activation function. In the modern sense, the perceptron is an algorithm for learning a binary classifier: a function that maps its input $\boldsymbol{x}$ (a real-valued vector) to an output value $f(\boldsymbol{x})$ (a single binary value),

$$y = \begin{cases} 1, & if, \ \boldsymbol{w} \cdot \boldsymbol{x} + b \\ 0, & otherwise \end{cases}, \tag{3.1}$$

where, $\boldsymbol{w}$ is a vector of real-valued weights, $\boldsymbol{wx}$ is the dot product $\sum_{i=1}^{n} w_i x_i$, where n is the number of inputs to the perceptron, and $b \in \mathbb{R}$ is the bias. The bias shifts the decision boundary away from the origin and does not depend on any input value.

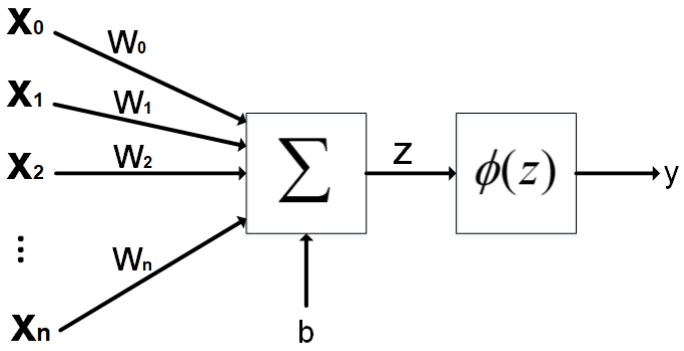


Figure 3.1: Schematic structure of a neuron with its n inputs $\boldsymbol{x_i}$, weights $\boldsymbol{w_i}$, bias b and activation function ϕ (z).

An NN comprises of simple computational units called nodes or neurons. A neuron is a more general (or advanced) version of the perceptron. The output of a single neuron is a non-linear function of the weighted sum of its inputs. The non-linearity is introduced by the activation function. This is important because most real-world data is non-linear and neurons help to learn these non-linear representations. It can be represented mathematically by,

$$y = \phi(\sum_{i=1}^{n} w_i x_i + b), \tag{3.2}$$

which allow x_i, $y \in \mathbb{R}$ by wrapping its term with a non-linear activation function ϕ(z). Frequently used activation functions are, for example, the sigmoid function, the hyperbolic tangent and the Rectified Linear Unit (ReLU). These activation functions will be described in more detail later in this section. The exemplary structure of a neuron is illustrated in Figure 3.1.

Feedforward Networks

Artificial Neural networks are modeled as a structure of neurons connected in layers. The outputs of neurons of a certain layer become the input of neurons of the next layer, these networks are called feedforward neural networks. The main property of a feedforward network is that the input is always passed forward i.e. the information flows through the network along a single direction, hence the network does not contain any feedback or self-connections. In the feedforward network, neurons are often arranged in layers. So,

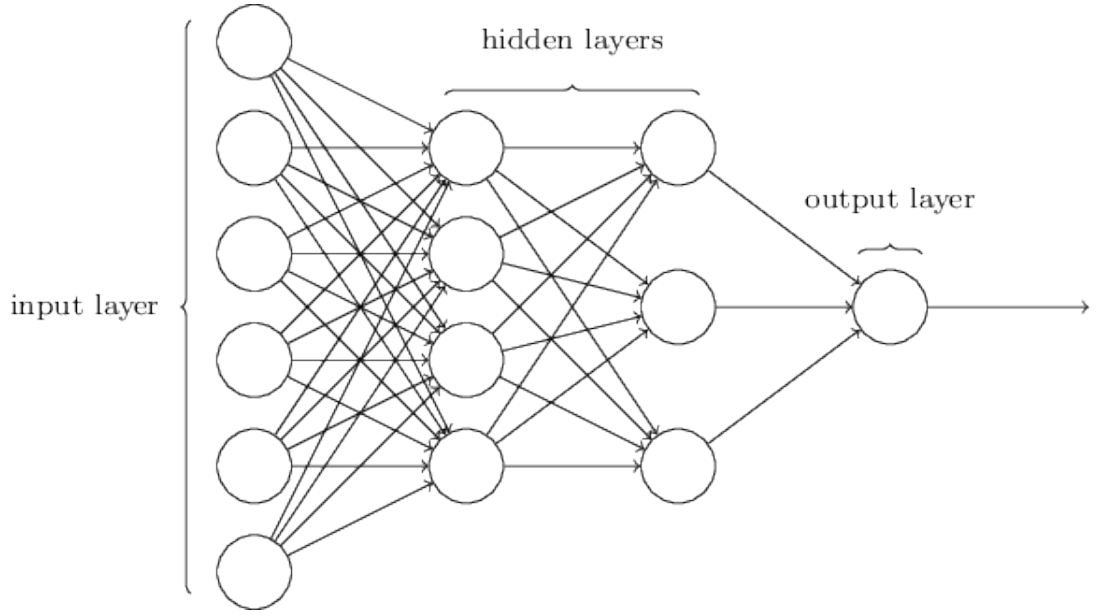


Figure 3.2: A feed-forward NN with two hidden layers. The neurons are represented by circles. Each neuron in a layer is connected to all the neurons in the previous (bottom) layer. The input layer nodes are not technically neurons as they only forward the input values without any processing [23].

the network consists of a dedicated input, output layer and potentially some hidden layers in between them. These hidden layers are providing a systematic method of calculating the activations of the output layer, given the input and the weights and the biases of all intermediate layers. When every node from one layer is connected to all nodes of its subsequent layer, then the layer is called the fully connected (FC) layers. The network depicted in Figure 3.2 is an example of a feedforward neural network with fully connected layers.

In machine learning applications, feedforward networks have proven to be very successful

in pattern recognition, classification, regression, and other machine learning tasks. Feed-forward networks are a conceptual stepping stone on the path to Recurrent Networks when it is extended to include feedback connections, which power many natural language applications.

3.1.2 Network Training

Feed-forward neural networks are mostly used for supervised learning tasks [23]. The final goal of training such a network is to end up with a model that generalizes on any kind of data of the same distribution [24]. Training data is necessary to train a neural network in a supervised manner. The network then is shown input data (or fed) and an error is computed that describes how far it is off the desired output. Since the ground truth output for each data point during the training phase is known, the distance between network's output and the desired output is measured by a loss function J(Θ) (often called cost function or objective function as well). Here, Θ represents the network parameters that are learnable during the training. A commonly used loss function for regression problems is the Mean Squared Error (MSE), which is defined as,

$$J(\Theta) = \mathcal{L}_{MSE}(w, b) = \frac{1}{N} \sum_{x=1}^{N} (y_x - \widehat{y_x})^2, \tag{3.3}$$

where, N is the number of observations, y_x denotes the true value, and the predicted value is denoted by $\widehat{y}_x$. MSE measures the averaged squared distance between the predicted values and true values for the complete training set. The difference between the true value and predicted value is also referred to as the error or residual.

There exist different loss functions and the choice of the loss function depends on the learning problem. In regression problems, the MSE loss function is a good choice. But for the problem of classification, it is not so practical. So, the cross-entropy loss (also known as log loss) is commonly used for binary or multi-class problems. It calculates a loss between two distributions with,

$$J(\Theta) = \mathcal{L}_{multiclass}(y_x, \widehat{y_x}) = -\sum_{x=1}^{n} y_x \, .log(\widehat{y_x}). \tag{3.4}$$

And for binary class problems, binary cross-entropy,

$$\begin{aligned} J(\Theta) &= \mathcal{L}_{BCE}(w,b) \\ &= -\frac{1}{N}\sum_{x=1}^{N} y_x \times log(\widehat{y_x}) \\ &= \sum_{x} -y_x log(\widehat{y_x}) + (1-y_x)\, log(1-\widehat{y_x}). \end{aligned} \tag{3.5}$$

During training of the network, the set of weights W and biases b must be found that minimizes the error,

$$arg\, min_{w,b}\mathcal{L}(w,b). \tag{3.6}$$

Initialization

Since the loss function $J(\Theta)$ is used to minimize the error between the training data and the ANN with respect to Θ, the learning is converted into a minimization problem with the goal to minimize the loss function by tuning the parameters of the NN. The optimization algorithm used to train NNs is called gradient descent. Gradient descent involves calculating the gradients of the loss function with respect to or to the network parameters Θ.

Gradient descent is an iterative optimization algorithm, which updates the weights with the help of the gradient. In the simplest version, it has the following form,

$$\Theta \leftarrow \Theta - \eta.\frac{\delta J(\Theta)}{\delta \Theta}, \tag{3.7}$$

where η describes the learning rate.

Meanwhile, deep neural networks are often suffering greatly during the training period, from vanishing gradients or exploding gradients problems. Therefore, it must be necessary to choose a good initialization parameter in order to avoid vanishing or exploding gradients [24]. Hence, a good initialization of Θ is important and have a radical effect on how fast the network will learn generalized patterns. One solution to this problem is Xavier Initialization proposed by Glorot and Bengio [25]. The Xavier Initialization formula is based on the number of input and output neurons and uses sampling from a uniform distribution with zero mean and all biases set to zero,

$$w \sim U\left[-\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}}\right], \tag{3.8}$$

where $U[-a, a]$ is the uniform distribution in the interval $(-a, a)$, n is the size of the previous layer, w is the weight matrix at any network layer, n_{in} is the number of incoming connections and n_{out} is the number of outgoing connections to the next layer. This initialization is designed to keep the gradients in all layers within approximately the same scale.

Backpropagation

Backpropagation is an algorithm used in ANNs to calculate a gradient that is needed in the calculation of the weights to be used in the network. Backpropagation is shorthand for "the backward propagation of errors," since an error is computed at the output and distributed backwards throughout the network's layers. The algorithm is based on gradient descent, which iteratively tries to find the minima of a function by doing small steps towards the negative gradient. The gradient is a measure of the change in the loss value corresponding to a small change in a network parameter. And it computes the gradients based on the chain rule of derivatives [26]. By applying backpropagation to the given loss function results in the update rule for trainable weight and bias parameter, such as,

$$w_i{}^{(l+1)} = w_i{}^{(l)} - \eta . \frac{\delta J(\Theta)}{\delta w_i{}^{(l)}}, \tag{3.9}$$

$$b_i{}^{(l+1)} = b_i{}^{(l)} - \eta . \frac{\delta J(\Theta)}{\delta b_i{}^{(l)}}, \tag{3.10}$$

where $\eta > 0$ is the learning rate that determines the step size that is done along the slope in each iteration [21].

Activation Functions

The activation functions are the essential components of ANNs, which helps the network to solve non-linear problems. Also, these functions must be continuously differentiable. Otherwise, the backpropagation algorithm will not work, because it cannot compute the gradient. The activation functions commonly used in the intermediate layers of ANNs are presented in below.

Sigmoid

The sigmoid activation function (Figure 3.3) is defined as,

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{3.11}$$

Its range is between 0 and 1. It has a "squashing" property such that large positive or negative input values result in values close to 1 or 0, respectively.

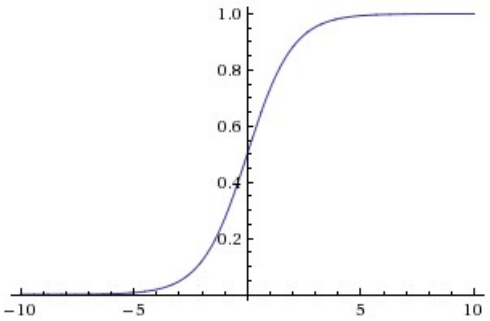


Figure 3.3: Sigmoid activation function [27].

Rectified Linear Unit

The Rectified Linear Unit (ReLU) is the most used activation function for neural networks as of right now. The formula for this function is simply given by:

$$f(x) = \max(x, 0). \tag{3.12}$$

In Figure 3.4, ReLU activation function is shown. Compared to the sigmoid activation function, the ReLU has the advantage of not saturating for the large inputs. Since sigmoids have only a small range of value but have a sufficiently high derivative, can produce the exploding gradients problem. To address this issue, ReLUs are taken into consideration in most cases due to its large range of values i.e. $(0, \alpha)$.

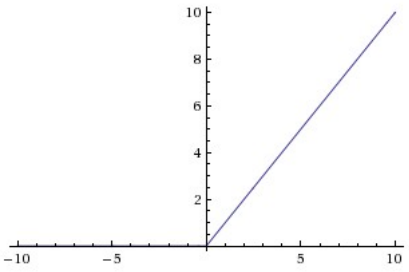
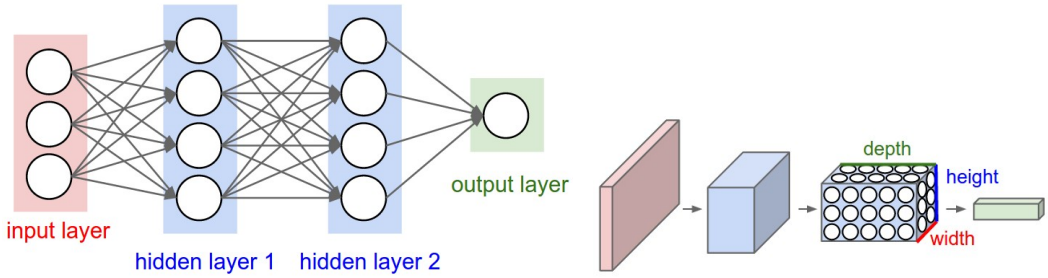


Figure 3.4: ReLU activation function [27].

3.2 Convolutional Neural Networks

In mathematics convolution is a mathematical operation on two functions (f and g) to produce a third function that expresses how the shape of one is modified by the other. The term convolution refers to both the result function and to the process of computing it. And, in DNN, Convolutional Neural Networks (CNNs/ConvNets) employs this convolution mathematical operation. ConvNets have a different architecture than regular feedforward NNs. Regular neural networks transform an input by putting it through a series of hidden layers. Each neuron of a hidden layer is fully connected to all neurons in the previous layer, where neurons in a single layer function completely independently and do not share any connections. Finally, there is a last fully-connected layer — the output layer — that represent the predictions distribution. While this allows learning complex representations, on the one hand, it comes with a couple of drawbacks on the other hand as well. For example, data such as images would require the layers of the network to become very large, especially in consideration of the input layer. Consequently, the number of connections between these layers would increase exponentially and thus the number of trainable parameters as well. clearly, these large number of parameters would quickly lead to overfitting [23].



(a) (b)

Figure 3.5: (a): A regular 3-layer Neural Network. (b): A ConvNet arranges its neurons in 3D (width, height, depth), as visualized in one of the layers. In this example, the red input layer represents the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels). By using activation functions, each ConvNet layer transforms its 3D input tensor to 3D output tensor to the next layer [28].

On the contrary, ConvNets take advantage of the fact that the input consists of images. So, it shows a different arrangement and a different connection mechanism than a regular neural network. First of all, the ConvNets layers are organized in 3 dimensions: width,

height, and depth. The three-dimensional arrangement comes from the image. A colored image has normally three channels (red, green, blue), and every channel is described by a two-dimensional matrix. Therefore, the input of the ConvNet is a three-dimensional matrix. Further, ConvNets use shared weights. The weight sharing means, that the same weights are used for multiple output units. Lastly, the neurons in one layer of the ConvNets do not connect to all the neurons in the next layer but only to a small region of it which refers to local connectivity. The size of the local connectivity is described by the kernel size [29]. Figure 3.5 depicted the advantage of CNN over the regular ANN. The CNN received its main attention after beating proven methods in the ImageNet competition by a large margin [30]. The structure of a ConvNet, the detailed advantages, and its mathematical formulation are described in the following sections.

3.2.1 CNNs Architecture

ConvNets is translation equivariant. It means, by its definition, the translation operated on the input tensor is still detectable in the output fetaure set. ConvNets, like any other ANN, are composed of neurons with learnable weights and biases. A ConvNet architecture is formed by a stack of distinct layers that transform the input tensor into an output tensor (e.g. holding the class scores in case of classifying image) through a differentiable function. A few distinct types of layers are commonly used. Among them, the convolutional layer is the core building block of a ConvNet. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field but extend through the full depth of the input tensor (Figure 3.6).

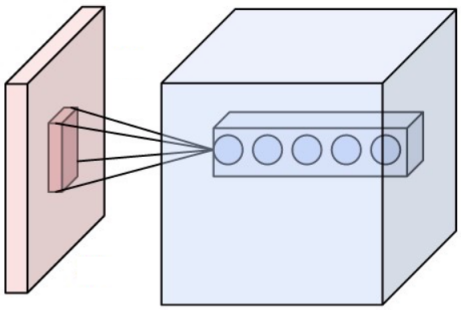


Figure 3.6: Neurons of a convolutional layer (blue), connected to their receptive field (red) [28].

Each filter is convolved across the width and height of the input tensor, computing the dot product between the entries of the filter and the input and producing a 2D activation

map of that filter during the forward pass. As a result, the network learns filters that activate when it detects different types of features at some spatial position in the input. Figure 3.7 visualizes the LeNet-5 architecture, a typical CNN architecture constructed to recognize hand-written characters.

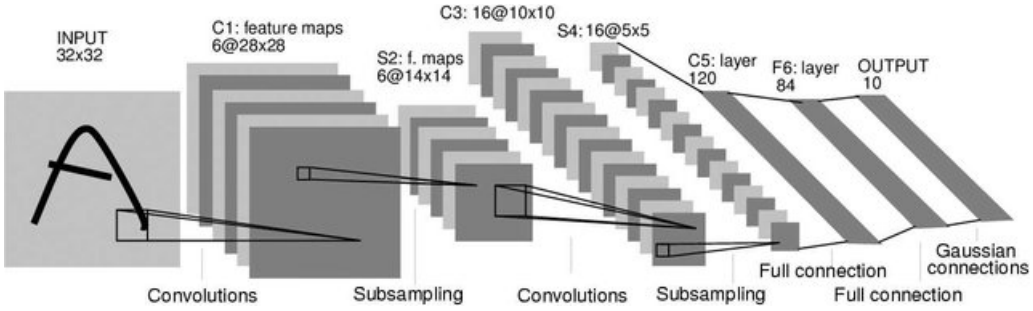


Figure 3.7: Architecture of LeNet-5, one of the first Convolutional Neural Networks trained in a supervised manner to recognize handwritten characters [29].

Stacking the activation maps for all filters along the depth dimension forms the full output tensor of the convolution layer. Three hyperparameters control the size of the output tensor of the convolutional layer: the depth, stride, and zero-padding. The depth corresponds to the number of filters are used to extract different features in the input. Strides refer to how a filter will be moved in each dimension per step. If the stride is 1 then a filter slide only one pixel at a time. The third hyperparameter zero-padding means to pad the input with zeros on the border of the input tensor. The Benefit of using zero padding is that it will allow controlling the spatial size of the output tensor. The spatial size of the output tensor can be computed as a function of the input size $W_{in} \times H_{in}$, the kernel size K, the stride with which they are applied S, and the amount of zero padding P used on the border. If input tensor size is $W_{in} \times H_{in} \times$, then the output tensor size will be $W_{out} \times H_{out} \times_{out}$, where, W_{out}, H_{out} and D_{out} will be computed as,

$$\begin{aligned} W_{out} &= \frac{W_{in} - K + 2P}{S} + 1, \\ H_{out} &= \frac{H_{in} - K + 2P}{S} + 1, \\ D_{out} &= K. \end{aligned} \tag{3.13}$$

Each convolutional layer is usually followed by a non-linear activation function, preferably a rectifier. ReLU layer will apply an elementwise activation function leaves the size of the tensor unchanged.

Convolution Operation

A convolution is a mathematical operation on two functions *f(x)* and *g(x)* to produce a third function that expresses how the shape of one is modified by the other. The convolution of f and g is written $f * g$, [23] and is defined as,

$$(f * g)(x) = \int f(\tau)g(x-\tau)d\tau. \tag{3.14}$$

In the terminology of ConvNets, the function f is termed as the input function and the filter g is referred to as the kernel function. Since the 2D grid maps are image-like structure and used in this thesis as input, the formulation of Equation 3.14 can be reformulated as for the 2D grid maps as,

$$(I * K)(x,y) = \sum_{x=1}^{h}\sum_{j=1}^{w} I_{i,j}.K_{x-i,y-j}, \tag{3.15}$$

with an input I of size $w \times h$ and a two-dimensional kernel K. Input images are represented by matrices by size $w \times h \times d$, where d denotes depth of the color channel Red-Green-Blue (RGB) of the image. Depending on kernel size of $k \times k$ and the chosen stride s, the shape of the convolved output changes. Here, the stride is the number of pixels by which the kernel shifts at a time. A larger stride will result in a smaller sized output. Figure 3.8 depicted this convolutional operation.

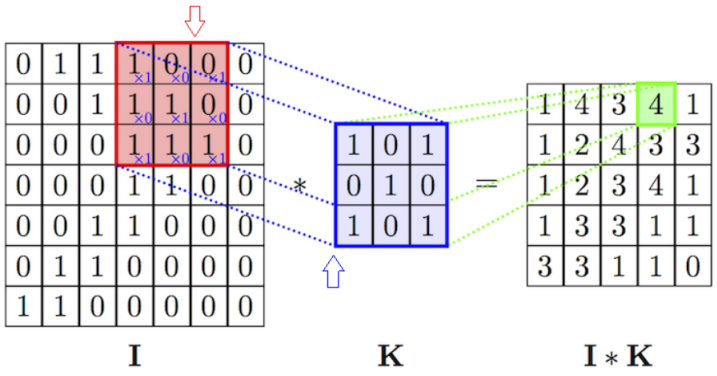


Figure 3.8: Convolution operation on the image I $(7 \times 7 \times 1)$ with a kernel K $(3 \times 3 \times 1)$ and with stride 1. The weights in the kernel are the parameters to be trained. This is an illustration of example with one color channel [31].

Pooling Layer

In image classification, a pooling layer is applied after one or multiple convolution layers that perform a subsampling onto the feature maps or tensors. The pooling layer reduces the spatial size of these tensors by a downsampling operation. This helps to reduce the number of parameters and computation in the network. Hence, it controls the overfitting. ConvNet achieves translation invariance by performing pooling operation i.e. dimensionality reduction. The pooling layer performs a local dimensionality reduction removing the spatial information, i.e. removing the small (compared to the pooling operator size) translations.

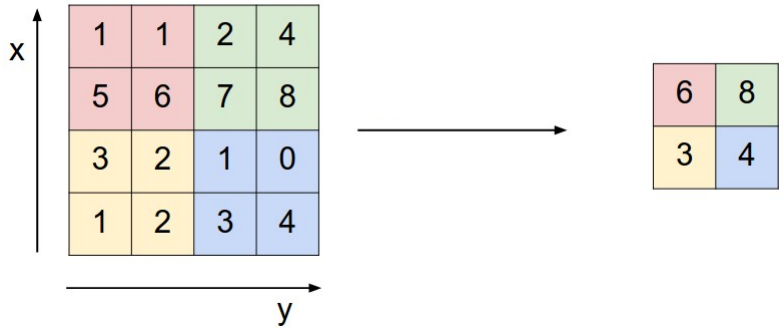


Figure 3.9: Maxpooling with a (2×2) kernel and stride $s = 2$. Maxpooling layers reduce spatial dimension of the input tensor [28].

The most frequent type of pooling is Max Pooling (Figure 3.9), which takes the maximum value from each filter. Typical Max Pooling values are 2×2. However, choosing larger shapes will dramatically reduce the dimension of the input tensor, and may result in excess information loss. There are several other methods which are commonly used for pooling in neural networks, such as average pooling and L2-norm pooling.

Transposed Convolution Layer

The need for transposed convolutions generally refers to use a transformation going in the opposite direction of a normal convolution operation [32]. A transposed convolutional layer carries out a regular convolution but reverts its spatial transformation. For instance, the decoding layer of a convolutional autoencoder or to project feature maps to a higher-dimensional space, one might use such a transformation. It is also called fractionally strided convolutions or deconvolutions because it can be emulated with a direct convolution using a zero-spaced input [32]. Figure 3.10 shows an example of a transposed convolution.

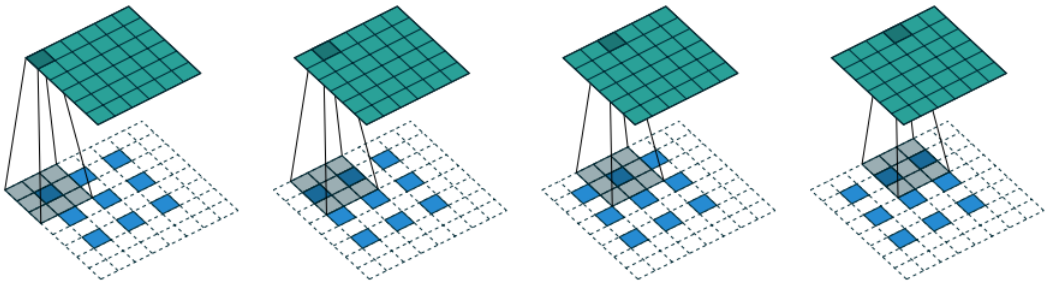


Figure 3.10: The transposed convolution of input $i = 6$, kernel $k = 3$, stride $s = 2$ and padding $p = 1$. This is equivalent to performing a convolution using a zero-spaced 3×3 input with $p = 1$ and $s = 1$.

3.2.2 Fully Convolutional Networks

Long et al. [33] introduced the Fully Convolutional Network (FCN) for image segmentation in 2015. FCN networks are able to produce dense pixel-wise predictions of arbitrary sized inputs and can be trained end-to-end, pixels-to-pixels (Figure 3.11). The idea is to capture the global context of the input scene. It means that segmentation needs to recover not only what is in the input, but also where.

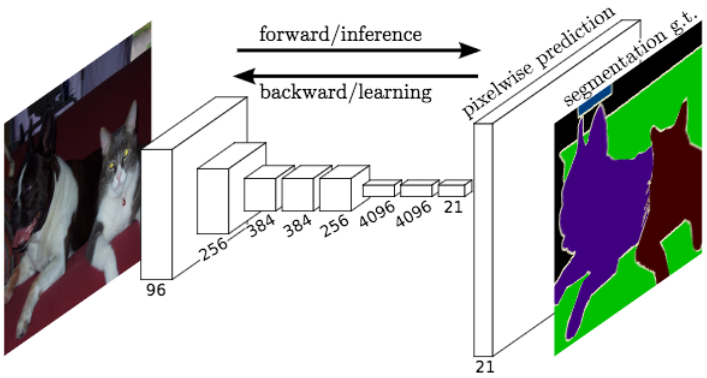


Figure 3.11: A Fully Convolutional Network (FCN) processes the entire image at once to make pixel-wise predictions in the same resolution as the original input image. The network is trained end-to-end by backpropagation and a pixel-wise loss. This requires densely labeled ground truth for supervised training [33].

A FCN is a normal CNN, where the last fully connected layer is substituted by another convolution layer with a large "receptive field". Since the kernel size in convolution layer is independent regarding its input, the network can be fed with different size of the input. Whereas this is not possible for FC-layer when it is used at inference because its fixed-sized weight matrix has to be applied to the entire input, not just to a local region. However, this does not imply that no FC-layer can be used at all when training the network. Depending on the architecture, FC-layers can be used in those parts of the network that are only used during training, but not when a prediction is performed.

During classification, a regular pre-trained classification networks actually do contain information about the location of classes (Figure 3.12). When a pre-trained classification network has been transformed into a FCN , it is able to predict location and class label simultaneously.

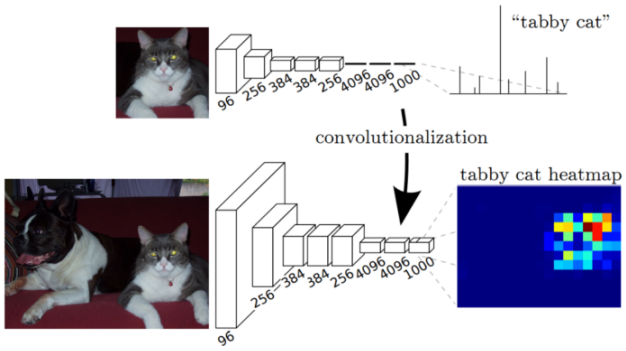


Figure 3.12: Transforming a classification network into a FCN for segmentation shows that classification networks contain information about location [33].

For segmentation task, a FCN have 2 parts. The first part is the process of compression of the intermediate layers through convolution operation. During this time striding and pooling operation reduce the height and width dimensions of the tensors. The second part is the 'deconvolution' or essentially backward convolutions, which upsample the intermediate reduced tensors so that they match the width and height of the original input image. This is also called, up convolution, and transposed convolution. Spatial location information can be lost when going deeper into the networks during the compression process. That means output from shallower layers has more location information. During the upsampling process, in segmentation, by fusing these both layers through a skip connection can recovery the fine-grained spatial information which is lost as output.

A skip connection is a connection that bypasses at least one layer. It is often used to transfer local information by concatenating or summing feature maps from the downsampling path with feature maps from the upsampling path. That is, it additionally provide the information from before compression (middle of the network) to restore the fine-grained localization which is lost in the down-sampling process.

3.3 Recurrent Neural Networks

Regular ANNs and other machine learning models are mostly working with the assumptions of independence among data samples. But various dataset like speech, language, time series, video, genomes, handwriting, the spoken word, or numerical times series data emanating from sensors, stock markets and government agencies etc., all exhibit dependence between individual elements across time. Since ANNs always handle each sample individually, therefore, it is hard to apply regular feedforward network architectures on these sequential datasets problems or exploits its sequential characteristics. To address this issue, Recurrent Neural Networks(RNNs), a type of artificial neural network, are designed to recognize patterns in sequences of data. RNN algorithms take time and sequence into account, they have a temporal dimension. In this work, the history of previous grid map frame is the most important requirement to predict the future grid map frames that certainly match to the given previous sequence. RNNs address this issue by processing the input sequence once at a time by maintaining a hidden state which acts as a memory for past information. In this section, an overview of most common RNN, a formal description of its structure and later succession model of RNN which solves its fundamental problem is given.

RNN Structure

RNN is a special type of NNs that allow its models to form a directed cyclic graph. Thereby, they can hold a hidden state that represents the sequential dynamics of the past. The decision a RNN reached at time step $t-1$ affects the decision it will reach one moment later at time step t. So RNNs have two sources of input, the present, and the recent past. These two inputs together determine how RNN respond to new data. RNNs are distinguished from feedforward networks by that feedback loop connected to their past decisions, ingesting their own outputs time after time as input. RNNs can often have memory. There is always information, or a pattern exists in the sequence itself, and RNNs use it to perform tasks that feedforward networks cannot do. That sequence pattern is preserved in the RNN's hidden state, which manages to span many time steps as it cascades forward to affect the processing of each new example. Due to this characteristic, RNNs can be think of as a way to share weights over time [23].

A standard RNN is shown in the Figure 3.13. A RNN has a feedback connection (Figure 3.13) which connects the hidden neurons across time. Given an input sequence $X = x_1, x_2, \ldots, x_{t-1}, x_t, x_{t+1}$, the t^{th} recurrent building block gets x_t as its input of this sequence, as well as the hidden state s_{t-1} of the previous one. These building blocks are typically referred to as a cell. Because the first cell has no predecessor, its hidden state input s_0 is usually manually fed with a zero-initialized state vector. Next, the hidden state is updated to s_t and finally the output of the network o_t is calculated. The current output o_t depends on all the previous inputs $x_t^{'}$ (for $t^{'} \leq t$).

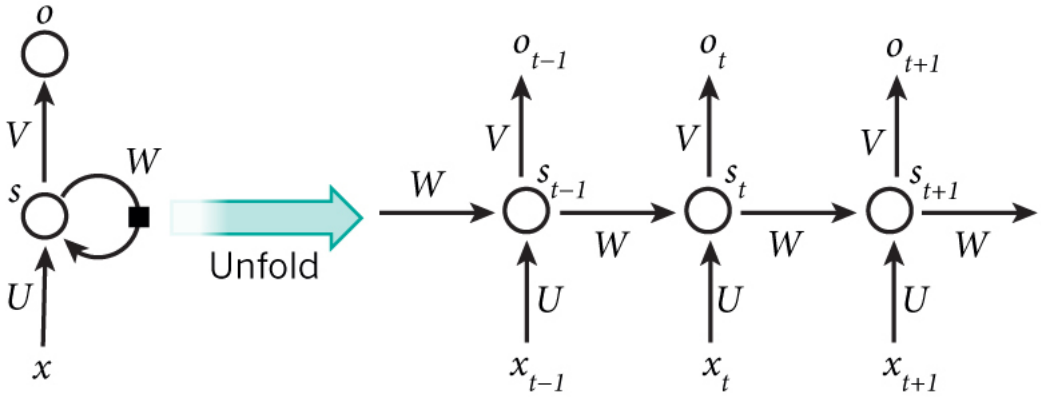


Figure 3.13: A standard RNN. The left-hand side of the figure is a standard RNN. The state vector in the hidden units is denoted by s. On the right-hand side is the same network unfolded in time to depict how the state is built over time [23].

Similar to a conventional ANN, the RNN has input to hidden connections parametrized by a weight matrix U, hidden-to-hidden recurrent connections parametrized by a weight matrix W, and hidden-to-output connections parametrized by a weight matrix V. Equations 3.16 and 3.17 summarize all the computations carried out at each time step,

$$s_t = \phi(Ux_t + Ws_{t-1} + b_s), \tag{3.16}$$

$$o_t = (Vs_t + b_o), \tag{3.17}$$

where $U \in \mathbb{R}^{d_x \times d_s}$ are the weights of the input-to-hidden connections, $W \in \mathbb{R}^{d_s \times d_s}$ are the weights of the hidden-to-hidden transitions, $V \in \mathbb{R}^{d_s \times d_x}$ are the weights of the hidden-to-output transitions and s_o, x_t, o_t as well as b_s, b_o are initial state, input, output

and the biases, respectively [34], [23]. The activation function $\phi(z)$ is usually chosen to be a hyperbolic tangent.

As shown on the right side of the Figure 3.13, once the network is unfolded in time, it can be considered a deep network with the number of layers equivalent to the number of time steps in the input sequence. Since RNNs are sharing weights, the same parameters U, V, W will be sharing across all times steps [23]. At each time step, whenever a new input is given, hidden state s_t is updated using Equation 3.16.

RNNs are more exciting due to its flexible nature that it can operate over sequences of vectors, whether It's the input sequence or the output sequence or in the most general case both. Figure 3.14 depicted few examples of input-output sequences of RNNs.

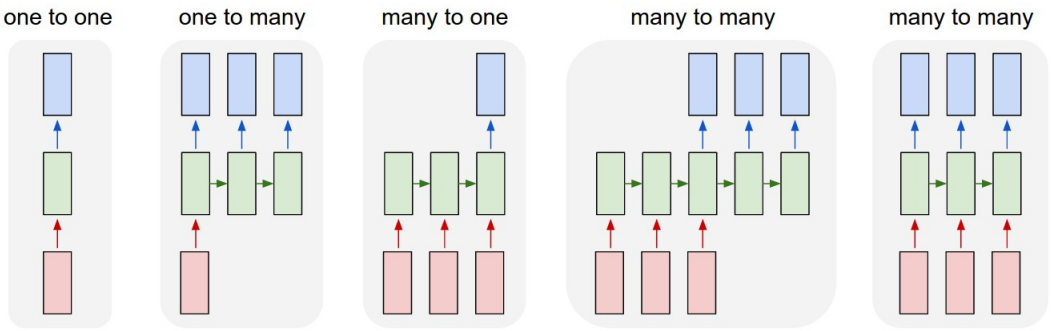


Figure 3.14: Visualization of different recurrent network input-output modes: (a) one-to-many, (b) many-to-one, (c) many-to-many. Input vectors are depicted in red, output vectors are depicted in blue and the green vectors hold the RNN's state [35].

Backpropagation Through Time

Computing the gradient through a RNN is straightforward. Backpropagation algorithm in feedforward networks can easily be applied to the unrolled computational graph. No specialized algorithms are necessary. The use of backpropagation on the unrolled graph is called the Back-Propagation Through Time (BPTT) algorithm [36]. Time is simply expressed by a well-defined, ordered series of calculations linking one time step to the next, which is all backpropagation needs to work. Gradients obtained by backpropagation may then be used with any general-purpose gradient-based techniques to train an RNN. The network is then rolled back to update the weights. The unfolded RNN depicted on the right side of Figure 3.13. Starting from the last time step t, the error propagated backward like in standard backpropagation. In general, NNs are nested composite functions like

$f(g(h(x)))$, whether they are recurrent or not. So, adding a time element only extends the series of functions for which derivatives with the chain rule are calculated.

RNN Disadvantage

Training an RNN with BPTT requires back-propagating the error gradients across several time steps and perform a several correlated steps which lead to the update of the same model parameters at once. Thus back-propagating the error involves multiplying the error gradient with the same shared weights which cause mathematical instability like the gradients will either become too large or decay to zero. These problems are referred to as exploding gradients and vanishing gradients respectively. As a result, the network does not converge at all or may take an indefinite amount of time. To be more specific, the exact problem depends on the magnitude of the recurrent edge weight and the specific activation function used. If the magnitude of weight is less than 1 and sigmoid activation (Equation 3.11) is used, vanishing gradients is more likely occurred. On the other hand, if the magnitude is greater than 1 and ReLU activation (Equation 3.12) is used exploding gradients is more likely happens [37]. Fortunately, other RNN variants exist that can deal with long-term dependencies. In one attempt, the long short-term memory (LSTM) architecture was introduced in [38] to counter the vanishing gradients problem. LSTM networks have proven to be very useful in learning long-term dependencies as compared to standard RNNs and have become the most popular variant of RNNs.

LSTM

LSTM networks, a specialized type of RNN was first introduced by Hochreiter and Schmidhuber (1997) [39]. Due to its capability of retaining information for longer periods of time, LSTM has seen tremendous growth in popularity while dealing with sequential problems in recent years. Standard RNNs have the form of a chain of repeating modules of neural network and repeating module will have a single tanh layer. On the other hand, LSTMs are a chain like structure, but have different repeating module structure than traditional RNNs. LSTMs repeating module has four neural network layers, interacting in a very special way. Figure 3.15a and Figure 3.15b shows this difference.

The advantage of LSTMs over traditional RNNs is that it has so-called memory cell state $C(t)$ (Figure 3.15). A simple RNN cell overrides its state at each time step, whereas, the LSTM's memory cell update shows only minor linear interaction, so that information could flow through very easily.

In Figure 3.15a, shows a full LSTM cell that takes the current input of the x_t, and outputs the current hidden state, h_t, passing this to the next LSTM cell for our input

sequence. The rest of the LSTM cell structure is described below (Figure 3.15b). As the traditional RNN cell has only a single "internal layer" acting on the current state h_{t-1} and input x_t, the LSTM cell has other three gates apart from the input gate. Among these gates, first, forget gate layer (f_t) decide what information should be maintained or discarded from previous state (h_{t-1}). Forget gate layer takes previous cell output h_{t-1} and current input x_t and outputs a value between 0 and 1 for each hidden unit by applying a sigmoid activation (σ). This is followed by element-wise multiplication with the previous cell state (C_{t-1}) in the Figure 3.15b.

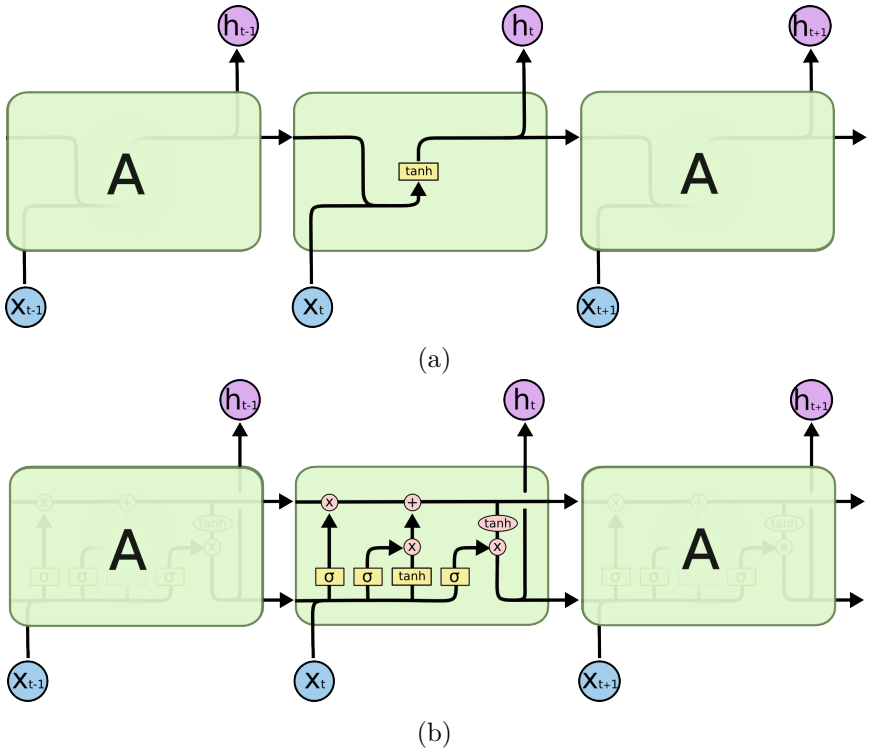


Figure 3.15: Unrolled RNN cell structure (a) vs. LSTM cell structure (b)- the repeating module in an LSTM contains four interacting layers [40].

The second gate refers to an "update gate" that decide what new information will be going to store in the cell state. It has two parts. The first part decides which values need to be updated through a sigmoid layer called the "input gate layer". A tanh activation layer creates a vector of new candidate values, ($\widehat{C}_t$), that could be added to the cell state at the next step. To update the old cell state, C_{t-1}, into the new cell C_t, these two values will be combined. So, the old state is multiplied by f_t and, then added with $i_t \odot \widehat{C}_t$

where the operator $\odot$ denotes the Hadamard product . This is the new candidate values or current cell state with new scaled information.

Finally, the "output gate", which controls what information gets passed to the next state based on the current cell state. Like the previous layer, a sigmoid layer decides what parts of the cell state are going to output. Then, the current cell state is running through a *tanh* activation layer and multiply it by the output of the sigmoid gate, in order to decide what information to be passed as output. The functioning of the LSTM unit can be represented by the following set of equations,

$$i_t = \sigma(W_h h_{t-1} + W_x x_t + b_i), \tag{3.18}$$

$$f_t = \sigma(W_h h_{t-1} + W_x x_t + b_f), \tag{3.19}$$

$$\widehat{C}_t = \tanh(W_h h_{t-1} + W_x x_t + b_c), \tag{3.20}$$

$$C_t = f_t \bigodot C_{t-1} + i_t \bigodot \widehat{C}_t, \tag{3.21}$$

$$O_t = \sigma(W_h h_{t-1} + W_x x_t + b_o), \tag{3.22}$$

$$h_t = O_t \bigodot \tanh(C_t), \tag{3.23}$$

where W_x is the shared weights for the input to hidden connections, W_h is the hidden to hidden transitions at time step t, b is the biases, and $C^{(0)}, h^{(0)}$ are the initial states of the memory cell and the hidden state, respectively [39]. The input gate, forget gate and output gate are labeled as *i,f,and o*. Furthermore, the operator $\odot$ denotes the Hadamard product [41].

ConvLSTM

Fully Connected LSTM (FC-LSTM) layer has been proven powerful for handling temporal correlation, but it contains too much redundancy for spatial data. The major disadvantage of FC-LSTM in handling spatiotemporal data is its usage of full connections in input-to-state and state-to-state transitions in which no spatial information is encoded. To address this problem, the authors of [6] proposed an extension of FC-LSTM which has convolutional structures in both the input-to-state and state-to-state transitions. That means the proposed FC-LSTM extenstion can handle all the inputs, hidden states, cell

or gate outputs as 3D tensors to preserve the spatial data properties. This is realized by exchanging the internal matrix products by convolution operations, hence, called ConvLSTM. In this work, all the recurrent models are developed by exploiting ConvLSTM modules due to the nature of image-like grid map dataset. Here, all the inputs $\chi_1, \ldots, \chi_t$, cell outputs $C_1, \ldots, C_t$, hidden states $H_1, \ldots, H_t$ and gates $i_t; f_t; o_t$ of the ConvLSTM are 3D tensors whose last two dimensions are spatial dimensions (rows and columns). The ConvLSTM determines the future state of a certain cell in the grid by the inputs and past states of its local neighbors. This can easily be achieved by using a convolution operator in the state-to-state and input-to-state transitions (Figure 3.16). The key equations (Equations 3.24 to 3.28) of ConvLSTM are shown below, where $*$ denotes the convolution operator and $\odot$ denotes the Hadamard product,

$$i_t = \sigma\left(W_{xi} \star \chi_t + W_{hi} \star \mathcal{H}_{t-1} + W_{ci} \odot C_{t-1} + b_i\right), \tag{3.24}$$

$$f_t = \sigma\left(W_{xf} \star \chi_t + W_{hf} \star \mathcal{H}_{t-1} + W_{cf} \odot C_{t-1} + b_f\right), \tag{3.25}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot tanh\left(W_{xc} \star \chi_t + W_{hc} \star \mathcal{H}_{t-1} + b_c\right), \tag{3.26}$$

$$o_t = \sigma\left(W_{xo} \star \chi_t + W_{ho} \star \mathcal{H}_{t-1} + W_{co} \odot C_{t-1} + b_o\right), \tag{3.27}$$

$$\mathcal{H}_t = o_t \odot tanh\left(C_t\right), \tag{3.28}$$

where W_x and W_{xc} are the shared weights for the input to hidden connections, W_h, W_{hc} are the hidden to hidden transitions at time step t, b_i, b_f, b_c are the biases, and $C^(0), h^(0)$ are the initial states of the memory cell and the hidden state, respectively.

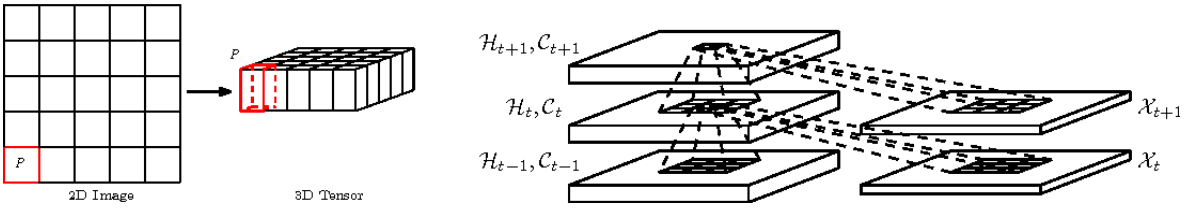


Figure 3.16: Transforming 2D image into 3D tensor (left). Inner structure of ConvLSTM (right) [6].

Recurrent Encoder-Decoder Architectures

The Recurrent encoder-decoder is a neural network model that directly computes the conditional probability of the output sequence given the input sequence without assuming a fixed alignment, i.e. $P(y_1, \ldots, y_o | x_1, \ldots, x_i)$ where the lengths of the output (o) and the input (i), may be different [23]. The core idea of encoder-decoder or sequence to sequence architecture is very simple: (1) an encoder processes the input sequence. The encoder maps the input sequence into a fixed length hidden representation C. This is known as the context. This context C might be a vector that summarize the input sequence $X = (x_1, \ldots, x_n)$. (2) a decoder is conditioned on that fixed-length vector (see Figure 3.17) to generate the output sequence $Y = y_1, \ldots, y_n$.

The encoder RNN builds the representation based on all elements of the sequence and therefore takes advantage of its temporal or ordinal structure. Afterwards, the learned representation is used to initialize the hidden state h_{dec} of the decoder, in opposite to the otherwise customary zero initialization. The decoder RNN takes then over and outputs the prediction of the target sequence. Obviously, both the encoder and decoder networks could be extended to consist multiple layers or bidirectional connections as well.

The decoder can be of two kinds – conditional or unconditional. A conditional decoder receives the last generated output as input. In the Figure 3.17, the first output $y^{(1)}$ is used as input for the next output $y^{(2)}$. On the contrary, an unconditioned decoder does not receive that previous output as input. Also, in the context of machine translation, speech recognition etc., if the context C is a vector, then the decoder RNN is simply a vector-to-sequence RNN [23]. And, there are at least two ways for a vector-to-sequence RNN i.e. decoder RNN to receive input. The input can be provided as the initial state of the RNN, or the input can be connected to the hidden units at each time step. These two ways can also be combined. In the Figure 3.17, it can be seen that decoder RNN receives two input from context C.

There is a clear limitation exists in this Recurrent Encoder-Decoder architecture. It arises when the encoder RNN output, context C, has a dimension that is too small to properly summarize a long sequence. This phenomenon was observed by Bahdanau et al. [42] in the context of machine translation. Their solution was to make C a variable-length sequence rather than a fixed-size vector. Additionally, they introduced an attention mechanism that learns to associate elements of the sequence C to elements of the output sequence which is depicted in Figure 3.17.

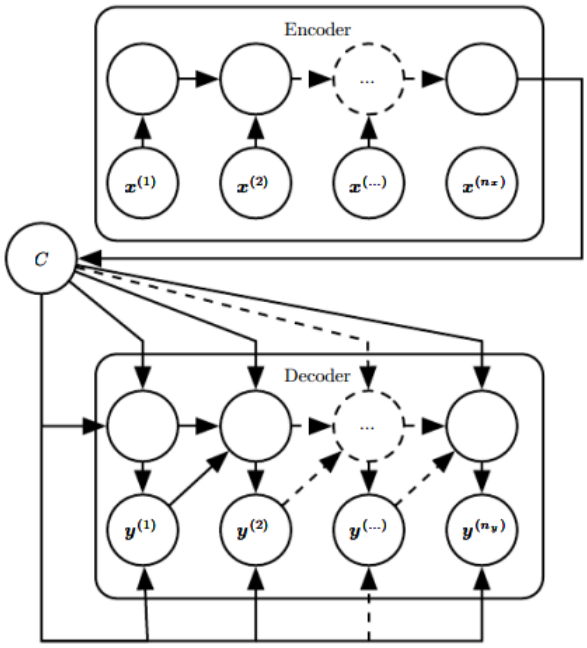


Figure 3.17: An example of a Recurrent Encoder-Decoder RNN architecture [23]. It is used for learning to generate an output sequence $(y^{(1)}, \ldots, y^{(n_y)})$ given an input sequence $(x^{(1)}, \ldots, x^{(n_x)})$. The encoder RNN reads the input sequence and generates a fixed-size context variable C. The context C is given as input to the decoder RNN to generate the output sequence. The decoder can be conditional, i.e., it receives the last output as input, or unconditional, i.e., it does not receive that as input. The figure also depicted an attention mechanism where elements of context C is associated with the output sequence [42] in the context of machine translation, speech recognition etc.

4 Methodology

In order to drive more carefully and to avoid collisions, the ability to perceive surrounding situations are necessary. From an intelligent transportation system (ITS) perspective, a real-time perception of the other traffic participants' and prediction of traffic scenes will lead to more reliable and safe driving for autonomous vehicles. Instead of considering two or more vehicles interactions, a predictive motion model of the whole traffic scenario, where all the vehicles future dynamic motions are captured, will be beneficial to an ITS. Also, the progress in the field of machine learning and Deep Neural Network (DNN) has a profound effect in the self-driving applications to overcome the challenges.

Therefore, in this thesis, we focus on learning a predictive motion model of all the vehicles at once within a certain traffic highway. In order to do that, we represent the traffic scenarios combining with the vehicle position into 2D Occupancy Grid Maps (OGMs) which discussed in Section 4.1 with problem formulation. As for learning the future motion models that predict the sequence of basic motions of future grid map vehicles from the past grid map's input, we propose three deep neural networks (DNN). First, a CNN based architecture, Multi-Stream FCN (MS-FCN) is proposed to predict a single future grid map, which is described in Section 4.2. A recurrent CNN architecture, ConvLSTM Encoder-Decoder (ConvLSTMED) later proposed to address the limitation of the MS-FCN model discussed in Section 4.3. To achieve longer time motion prediction, lastly, a sequence-to-sequence model, Recurrent Encoder-Decoder ConvLSTM (RED-ConvLSTM) is proposed for predicting the future state of traffic scenarios as a whole for several time steps. Section 4.4 gives a detail description of this model. All the components of the three models, their advantages, and disadvantages are discussed in the respective model's Sections. A common loss function is used while training the three network models. Section 4.5 covers this loss function.

4.1 Problem Statement

When predicting future motions of surrounding vehicles for autonomous vehicles, the inter-vehicular interaction must be considered in order to predict future risks and to make safe and intelligent decisions. But traffic scenes are complex due to the fact that the behavior of the traffic participants is highly correlated among each other which increases

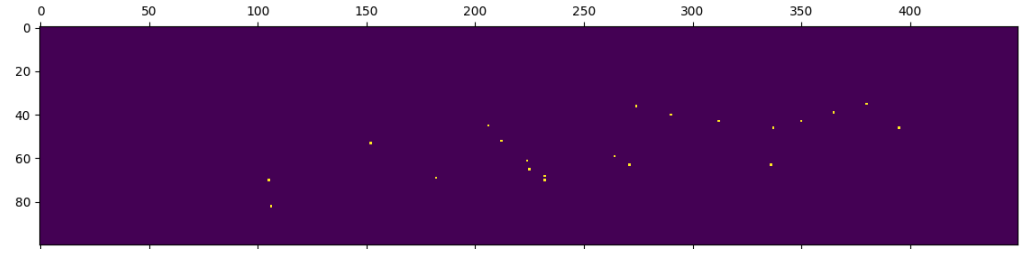
the motion prediction challenge harder cause model needs to predict the upcoming actions in a feasible time. Previously developed future prediction models have limited performance when handling simultaneous interactions and conflicts between vehicles because they focused on predicting individual vehicle motion or interaction between a single pair of vehicles rather than the entire traffic scene [2, 3, 8, 10, 11]. In this thesis, we address the problem of future predictive motion of all surrounding vehicles at once. Instead of considering the interaction between two vehicles, we concentrate on the traffic scenario as a whole to predict all the involved traffic participants dynamic motion all together within a stretch of a highway. If the predictive motion model can infer where the surrounding vehicles are heading in the near future, the vehicle can plan its driving path in response to the future situation such that the probability of collision is minimized which will be beneficial for an ITS.

In order to predict future traffic scenes based on past traffic scenes, we use DNN to learn the future motion models. We use single channel 2D Occupancy Grid Maps (OGMs), to represent the entire traffic scenario. In below, an introduction to OGMs, and how they are constructed to represent the traffic scenario with vehicles motions, are given.

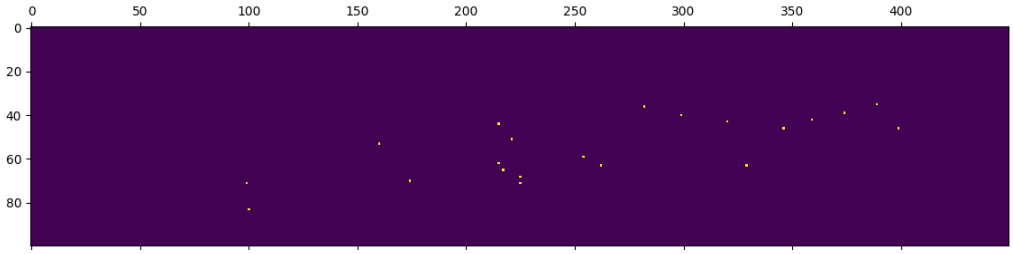
Occupancy Grid Maps (OGMs)

The occupancy grid maps (OGMs) [43], [44] are one of the most popular environmental representation tool. OGMs are a location based representation of the environment. They divide the environment into an evenly spaced grid of cells and estimate the probability of each cell for being occupied based on the sensor readings. It divides the environment or in this thesis highway traffic scenarios into an evenly spaced grid of cells and estimate the probability of each cell for being occupied based on the sensor readings.

In this work, to generate the grid maps, at first, vehicles motions are recorded for a certain period of time by using different sensors (radar etc.) in certain highway traffic. The radar sensors served as the base coordinate system. So, all other sensor detections of vehicles were transformed into that frame and projected on the XY-plane. Later, from this XY-plane, grid maps are generated. Each of the grid maps represented the number of vehicles position for a specific time step. So, the next grid map represents vehicles position for the next time steps. Therefore, vehicles motion can be easily seen at once by using current and past grid maps for several time steps. An example of vehicles motion into the grid maps are shown in Figure 4.1 where the yellow colors are depicting the position of the vehicle. That is why grid maps representation are used to feed as input to the proposed neural networks for future traffic scene predictions.



(a) An OGM represented vehicles in previous time step.



(b) An OGM represented vehicles in present time step.

Figure 4.1: Occupancy Grid Maps (OGMs) depicting vehicles motion between previous OGM (a) and current OGM (b).

4.2 Multi-Stream FCN

CNNs are a natural choice to predict grid-maps as they are 2D structures like images. As the inputs are processed through the CNN layers, the level of abstraction of the resulting features increases. FCNs are a special case of CNN which train end-to-end, pixel-to-pixel and have much fewer network parameters. Both learning and inference are performed whole-image-at-a-time by feedforward computation and backpropagation. The goal of the first approach was to predict the next grid map frame after observing a limited history of past grid maps. Since grid maps are 2D structures, this can leverage the power of a FCN, trained end-to-end, pixels-to-pixels on the future grid map predictions. While upsampling in the FCN network, transposed convolution was used. The transposed convolution operation forms the same connectivity as the normal convolution but in the backward direction. The reason to use transpose convolution is that it does not use a predefined interpolation method while upsampling. It has learnable parameters.

The proposed network, MS-FCN model, is depicted in Figure 4.2. MS-FCNs takes sequence of past grid maps to predict future frames, and learn powerful spatiotemporal features that implicitly capture grid maps dynamics as well as motions of vehicles. MS-FCN has two components - an encoder and a decoder. The left part of the network

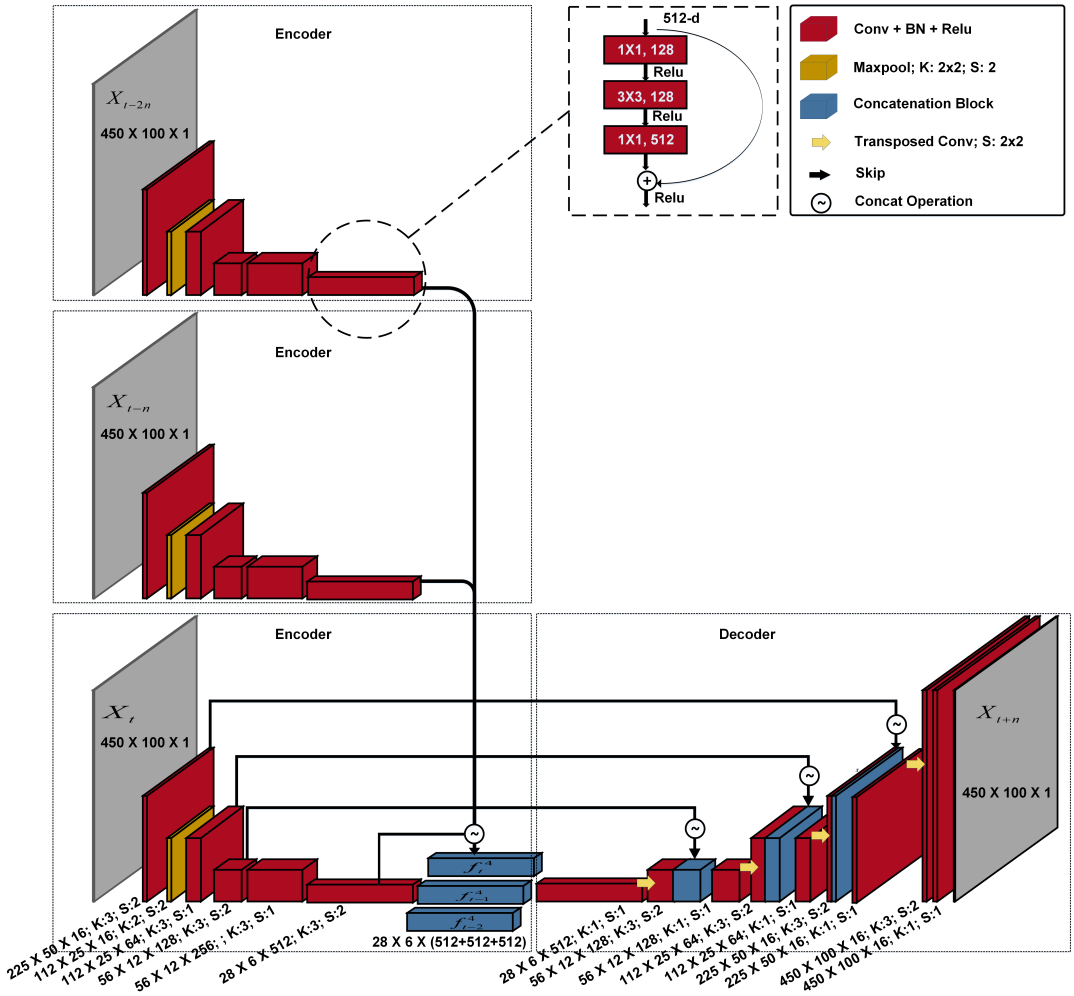


Figure 4.2: The detailed architecture of the proposed MS-FCN for predicting next single frame. The network consists of encoder (left-side) and decoder (right-side). The encoder takes past grid map frames $X = X_{t-2}, X_{t-1}, X_t$ at $t-2, t-1, t$ as input, and generates the future frame $t+1$ as output. The encoder output contains a reduced feature representation and is the output of the ResNet-101 from the last block of ResNet-101. Later, these 3 latent tensors are concatenated and transferred to the decoding path. At the decoder, this concatenated feature tensor is then upsampled by using transposed convolutional layers and concatenated with three skip connections from last input to generating a prediction for the next future occupancy grid map.

consists of a compression path, while the right part decompresses the until its original size is reached. Encoder and decoder are described in below.

Encoder

The encoder path follows the typical architecture of a convolutional network. A modified ResNet-101[45] is used at the encoder. The reason behind using ResNet-101 is, that usually deep CNNs are hard to train due to vanishing gradients in the long forward feed and backward propagate process. ResNet addresses this problem by introducing a residual block (Figure 4.3). A residual neural network has shortcut connections, parallel to the normal convolutional layers. Mathematically, A ResNet layer calculates,

$$y = f(x) + id(x) = f(x) + x, \tag{4.1}$$

where *id* means identity mapping. Those shortcuts act like highways and the gradients can easily flow back, resulting in faster training and makes a higher number of layers feasible. We removed the fully connected layers of the ResNet-101 which are used for classification in the original architecture.

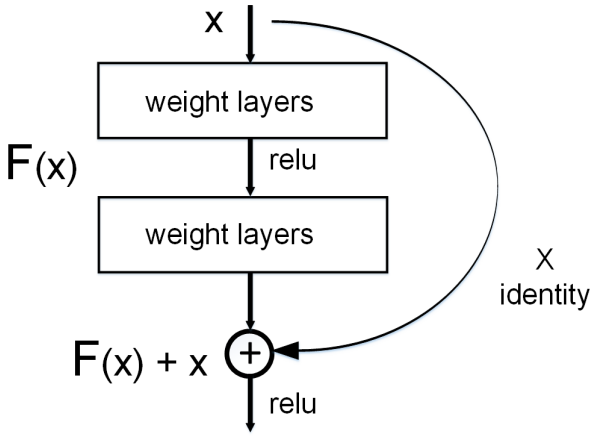


Figure 4.3: Residual Block

The encoder takes three past grid maps, X_{t-2}, X_{t-1}, X_t, at time $t-2, t-1, t$, to predict the future grid map frame X_{t+1}. In the end, the encoder produces lower resolution feature maps for each frame. Features at four different layers of the ResNet-101 ($conv1, conv2\text{-}3, conv3\text{-}3, conv5\text{-}3$) are used for skip connections at the decoder. All the three feature maps from the last convolutional layer, $conv5\text{-}3$ of the encoder are then concatenated and passed to the decoder as inputs. The reduced dimensions of the features

at different layers are $225 \times 50, 112 \times 25, 56 \times 12, 28 \times 6$, when the input grid maps has resolution 450×100. For convolutions, the kernel size is 3×3 and stride is 2×2. For the only pooling layer, 2×2 max-pooling is used. The residual block consists of convolutional layers with kernels $(1 \times 1, 3 \times 3, 1 \times 1)$. Each convolutional layer is followed by a Batch Normalization layer and a ReLu activation function. The last convolutional layer of the residual block is added (element-wise addition) with the residual block's shortcut connections (identity connections) followed by a ReLu activation. Note that, only the last grid map frames at different time-steps are preserved for skip connections. This allows the model to make use of temporal information, such that decoder learns to produce sequences that are temporally coherent with its input.

Decoder

The decoder network extracts features by taking encoder output that is concatenated features and expands the spatial support of the lower resolution feature maps in order to gather and assemble the necessary information to produces the future feature mask. The decoder works as follows: firstly, a 1×1 convolution is applied, followed by transposed convolution on the concatenated features to match the spatial and channel dimensions. Each frame is upsampled by using transposed convolution with stride 2×2. The result is then concatenated with the skip connection from the encoder part followed by a 1×1 convolution. The same sequence of operation is subsequently carried out with the same number of feature size and channels as their encoder inputs. The last feature map at the output of the final decoder layer is fed to a sigmoid activation layer. This sigmoid layer classifies each pixel independently. The output of the sigmoid classifier is a 1 channel grid map of probabilities where 1 represents the occupancy or not occupancy (1 or 0, respectively) binary decision.

Model Advantages and Disadvantages

The main benefit of MS-FCN is that it can capture the spatiotemporal information to predict the next future grid map from the past input grid maps. By concatenation operation with the last input frame the decoder learns the temporal motion for the next frame prediction. Also, since the model always predicts the future locations of the vehicles from the grid maps, it is possible to get multiple predicted locations for each vehicle. However, MS-FCN always requires three past input grid maps to predict the next single future frame. To use this model, three consecutive input frames are then required to predict the next future frame to generate an output. This brings the disadvantage of this model that it can only consider a limited history and predict one step ahead. That is, the network could not learn the future frames motion X_{t+2} from only it's previous frames X_{t+1}, which captures the latest temporal motion until now. In the next Section

4.3, this drawback is handled by introducing RNNs architecture for the single frame prediction model.

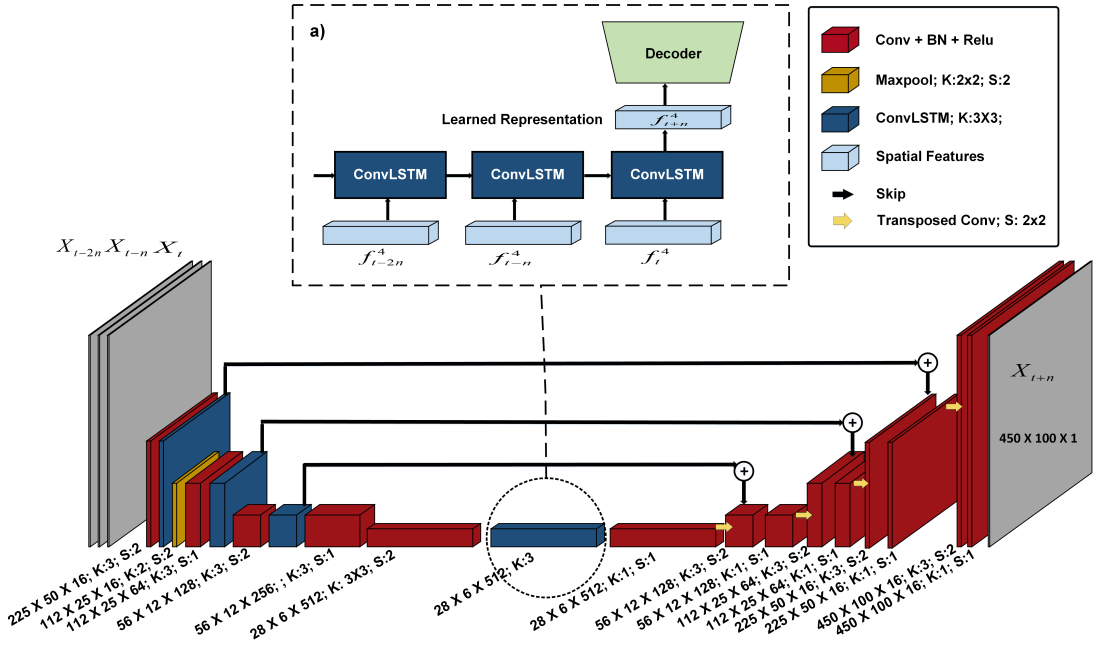


Figure 4.4: The detailed overview of the proposed ConvLSTM Encoder-Decoder (ConvLSTMED) model for predicting the next single frame. The network consists of 2 components: ConvLSTM encoder, (left-side) and decoder (right-side). The encoder produces feature maps of different reduced resolutions in various layers which are being used as inputs to the encoder ConvLSTMs outlined in a). The encoder ConvLSTM learns a encoded temporal representation from these lower dimensional features. Finally, the decoder which consists of four upsampling layers combines the outputs of different ConvLSTM modules and generate a feature map for the next time-step X_{t+1}.

4.3 ConvLSTM Encoder-Decoder

The goal for the next model is to reduce the number of inputs which otherwise must be processed in a redundant manner in the previous MS-FCN model. And as also for the related task of video frame prediction RNNs and particularly ConvLSTMs proved to be working. So, due to the success of RNNs to predict sequences of images [17], we use Convolutional LSTM (ConvLSTM) in the second model to predict the next frame.

The ConvLSTM is used to encode spatiotemporal information of observed frames for future prediction. Figure 4.4 shows the overall architecture of the proposed second model, ConvLSTM Encoder-Decoder (ConvLSTMED). The proposed network can be divided into an encoding and decoding steps. During encoding, a downsampling network (referred to as ConvLSTM encoder) extracts a low-dimensional feature from the input grid maps. Then, the ConvLTSM runs through the sequence of extracted features to learn a temporal representation. This representation is then decoded with the upsampling network (decoder) by transposed convolution to output the next grid map. The detail description of both components is given next.

ConvLSTM Encoder

The ConvLSTM encoder uses modified ResNet-101 like the previous model. But, the encoder part is different from the previous model replacing the concatenation block by one encoder ConvLSTM module after the modified ResNet-101's *conv*5-3 layer. In addition, three ConvLSTM modules are also inserted into three different layers ($conv1, conv2\text{-}3, conv3\text{-}3$) of modified ResNet-101. Each of these three intermediate ConvLSTM modules takes the feature map at a specific scale from these three layers as its input and captures the sequential correlations. Later, encoder transfers these correlated feature maps for element-wise addition over the skip connections to the decoder.

Initially, the encoder takes three consecutive grid maps at time $t-2n, t-n, t$, and extracts three lower-dimensional spatial features at each time-step. The encoder ConvLSTM at the end of the convolutional layer *conv*5-3 extracted these features of the encoder and produces an encoded vector. This encoded vector representation is then fed to the decoder for future grid map predictions. The encoded vector representation is global, meaning that the whole input sequence is encoded into a fixed vector representation. Consequently, the model can take advantage of making a future grid map prediction from the previous single grid map. Thus, the encoder is recurrent and can collect relevant information from multiple sequentially fed input grid maps.

All the ConvLSTM modules used in this model consist of a one-layer ConvLSTM with kernels of the size 3×3. And, it's cell states c, and hidden state h are initialized to zero by default. The shape of the encoded vector of encoder ConvLSTM is $\mathbb{R}^{28\times6\times512}$ for both internal states h, c. The dimension size for the kernel, stride, and max-pooling is as same as for the previous model in the ResNet-101 convolutional layers.

Decoder

The decoder uses the encoded vector representation generated from the input grid maps sequence by ConvLSTM encoder for upsampling. The upsampling steps are like the

previous MS-FCN model except that instead of concatenation operation, an element-wise addition is performed between the result of the transposed convolution layer and feature maps transferred from encoder over the skip connections. Rest of the decoder part is as same as the first model.

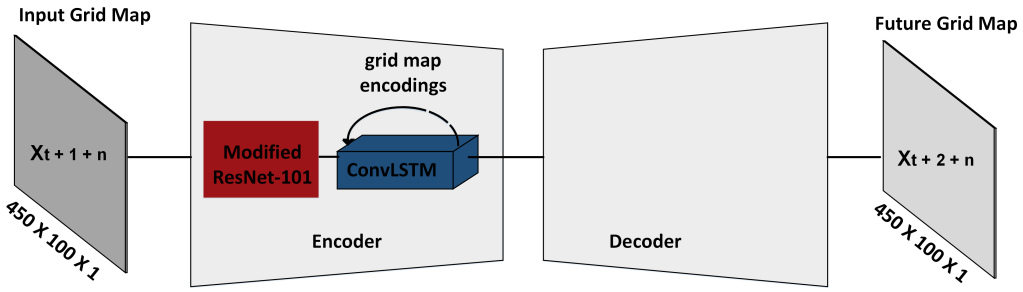


Figure 4.5: ConvLSTMED model showing One-to-One approach. After ConvLSTMED model is running with multiple input grid maps, the ConvLSTM encoder memorizes the grid map encodings and predicts the next future grid map. Since the network has already learned the current dynamics of the traffic, it is possible for ConvLSTMED model to predict the immediate future traffic scene from the present traffic scene.

Model Advantages and Disadvantages

Initially, the ConvLSTMED model follows the many-to-one approach. The model is at first trained by three consecutive input grid maps and predict the next grid map. While training, the ConvLSTM encoder stores information about the spatial and temporal dynamics of the traffic and the decoder extrapolates it one time-step in the future. The stored encoded information or the grid map encodings later helps the network to predict the next scene by only taking present grid map input which makes the network a one-to-one architecture (Figure 4.5). Due to this recurrent encoder, we can reduce the number of input to only one instead of using redundant input sequence used in the MS-FCN model.

Moreover, if it would be a usual feed-forward network like the first model and only 1 grid map is provided as input, then it would just learn the "average", expected motion sort of because it doesn't know any history. On the contrary, the advantage of using ConvLSTM encoder is that it is able to remember important things from the input grid maps and also capable of relating these input sequence due to its recurrent connections. As a result,

this recurrent nature enables ConvLSTM encoder to be very precise in predicting what's coming next.

On the other hand, there are uses cases where a single future time-steps is not sufficient to make use of the predictions. For example, for collision avoidance, multiple time-steps scene predictions are needed. Current ConvLSTMED model is not a feasible option for the sequence of grid maps prediction. So, for a longer time prediction, we examined a sequence to sequence model which is described in the next Section.

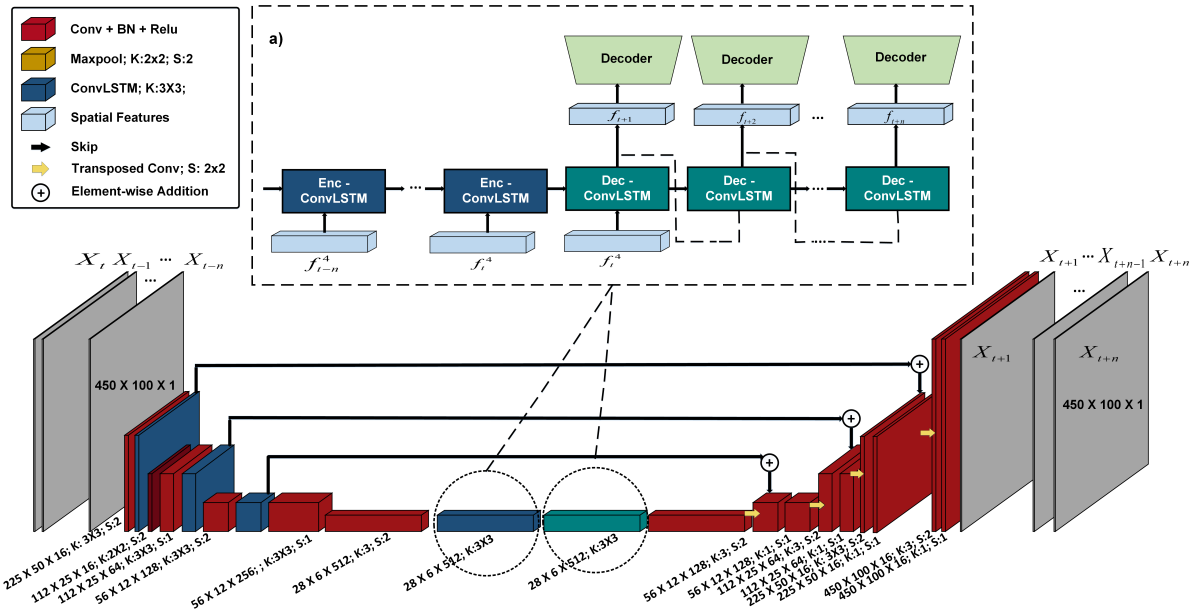


Figure 4.6: Overview of the developed Recurrent Encoder-Decoder ConvLSTM (RED-ConvLSTM) model. The main parts are the downsampling (left side) and upsampling (right side) structure consisting of feedforward neural networks (dark red) and the Recurrent Encoder-Decoder model, outlined in a). The Encoder-ConvLSTM part transforms a sequence of input grid maps into context vector, from which the Decoder-ConvLSTM recovers the corresponding output sequence of future grid maps.

4.4 Recurrent Encoder-Decoder ConvLSTM

Autonomous vehicles need to constantly assess the risk of accidents or collisions. In any traffic scenarios with different participants like, pedestrians, cyclists, and vehicles,

visual representations for several time-steps that foreshadow impending accidents, can be useful to judge such risks which in turn affect the driving patterns and level of alertness. Therefore, traffic scenario predictions for multiple time-steps eventually will help to make a safety decision beforehand. This gives us the motivation to examine the next RNN based network model. The developed network is a sequence-to-sequence model. Since the LSTM Encoder-Decoder framework [14, 17] provides a general framework for sequence-to-sequence learning problems due to its ability to capture the long-term temporal dependencies, we choose this framework to effectively learn these grid maps (vehicles motion) dependencies for future time steps. Theoretically, the RNN Encoder-Decoder is a neural network model that directly computes the conditional probability of the output sequence given the input sequence without assuming a fixed alignment, that is, $P(y_1, \ldots, y_O | X_1, \ldots, X_T)$, where the dimension of the output and the input, O and T respectively, may be different. The idea of the encoder-decoder approach is that for each output y_o, the encoder maps the input sequence into a fixed length hidden representation c_o, to which we refer as context vector. From the previous output symbols and the context vector, the decoder computes,

$$P(y_1, ..., y_o | \mathbf{x}_1, ..., \mathbf{x}_T) = \prod_{o=1}^{O} P(y_o | y_1, ..., y_{o-1}, \mathbf{c}_o). \tag{4.2}$$

Since the probability $P(y_o | y_1, ..., y_{o-1}, \mathbf{c}_o)$ is conditioned on the previous outputs as well as the context vector, a RNN can be used to compute this probability which implicitly remembers the history using a recurrent layer.

The design of this Recurrent Encoder-Decoder ConvLSTM (RED-ConvLSTM) model is the same as that of the previous ConvLSTMED RNN model, except that there is another ConvLSTM layer added as Decoder-ConvLSTM. The function of this Decoder-ConvLSTM is to predict grid map frames sequence, based on context information generated by ConvLSTM-Encoder, that comes after the input grid map sequence. The developed network architecture is shown in Figure 4.6. The model has in general two types of components. The first components are feedforward neural networks that are responsible for downsampling and upsampling. And the second part is the Recurrent Encoder-Decoder model, outlined in Figure 4.6 a). The RED-ConvLSTM model with its components are described below.

Feedforward Networks Components

The feedforward networks that have been used in this model are already used in the previous models. As already mentioned that a modified ResNet-101 is used in the downsampling path. In general, the CNNs in the downsampling path compresses the

spatial size of the input grid maps sequence, provided in $\mathbb{R}^{450\times100\times1}$, to a tensor with the size $\mathbb{R}^{28\times6\times512}$. This tensor is then fed to the Recurrent Encoder-Decoder layer, where predictions are made with the same spatial sizes as the input tensors. In the upsampling path, these predictions are upscale to the input dimension by using several transposed convolutions operations and adding skip connections.

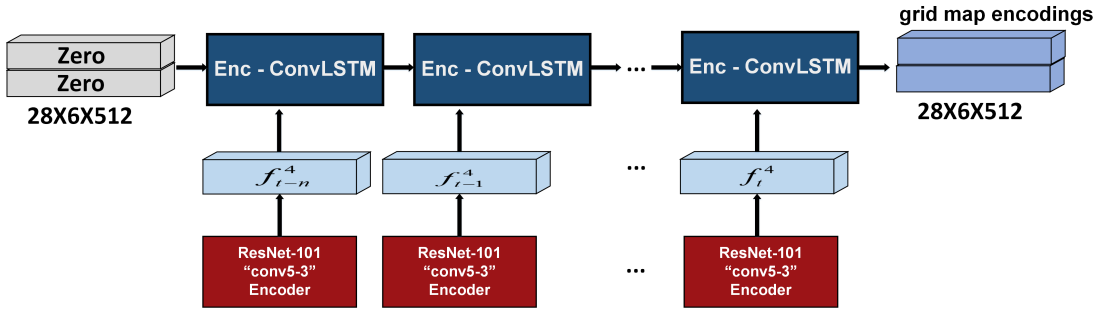


Figure 4.7: The unrolled version of the ConvLSTM-Encoder network during training. The one-layer ConvLSTM-Encoder network encodes a sequence of grid maps (light blue) and provides a context vector representation $\mathbb{R}^{28\times6\times512}$. The same ConvLSTM-Encoder is also used in the previous ConvLSTMED model. Only here, instead of handling one or at most three input grid maps, this recurrent encoder handles a sequence of input grid maps. ConvLSTM's cell state and hidden state are initialized to zero by default.

Recurrent Encoder-Decoder

The architecture of the one-layer Recurrent Encoder-Decoder is outlined in Figure 4.6 a), with the unrolled structure during training. This Recurrent Encoder-Decoder is divided into two parts - ConvLSTM-Encoder and Decoder-ConvLSTM. They have kernels 3×3 and their internal states tensor are in $\mathbb{R}^{28\times6\times512}$. Both parts unrolled version are shown separately, later. The first part, ConvLSTM-Encoder, depicted in Figure 4.7 which is also used in the previous RNN model. In Figure 4.7, the ConvLSTM-Encoder is shown with unrolled RNN structure during training. The difference with the previous model for this component is only in the usability of input grid maps. The ConvLSTM cell of the ConvLSTM-Encoder model at first extract information of the dynamic foreground objects and static background from the input grid maps sequence. And, in order to accurately predict the future grid map frames multiple time-steps, the ConvLSTM-Encoder needs to encode the dynamic motions of the objects and understand how the objects are moving so that the motion can be extrapolated later. The hidden state coming out from the encoder will try to capture this information [14]. Therefore, this motion state can be

seen as a grid map encodings of the input sequence with dimension $\mathbb{R}^{28\times6\times512}$, termed as context vector.

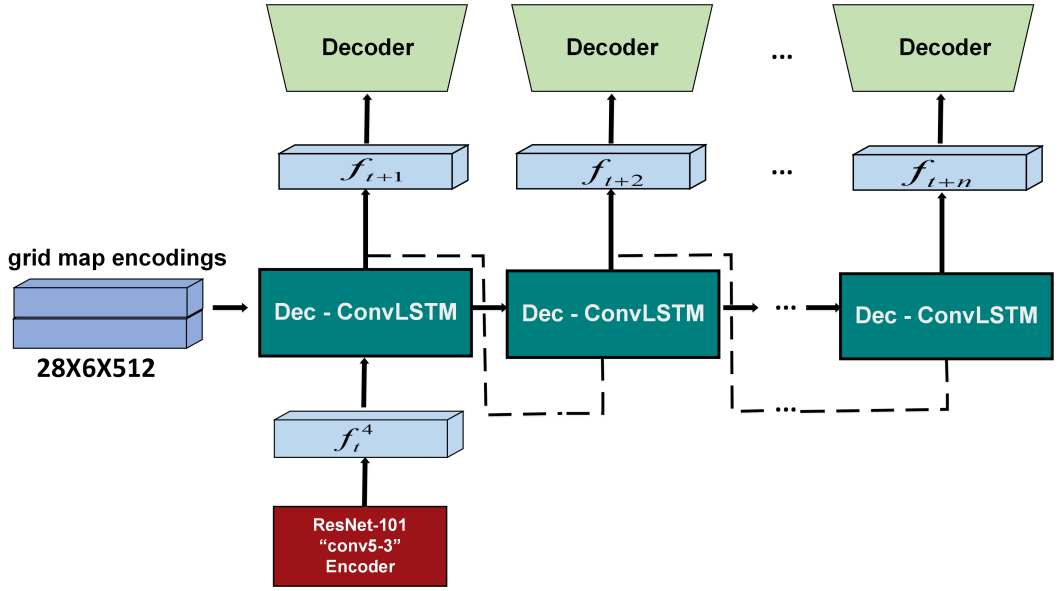


Figure 4.8: The unrolled version of Decoder-ConvLSTM during training. The Decoder-ConvLSTM's cell state and hidden state are initialized with the grid map encodings representation (light blue) produced by the ConvLSTM-Encoder. The Decoder-ConvLSTM's cells are all conditional decoder. A conditional decoder receives the last generated output frame as it's input. The first Decoder-ConvLSTM cell will take the *conv*5-3 layers spatial output stakes from ResNet-101. Then, the rest of the Decoder-ConvLSTM cells receives the output of the previous Decoder-ConvLSTM's cell as its input while training. The output of the Decoder-ConvLSTM cell is passed to the upsampling parts of the network and generate the future grid maps for the next time-steps.

Decoder-ConvLSTM depicted in Figure 4.8 is the second RNN part of the Recurrent Encoder-Decoder. This recurrent decoder module is the key difference to the previous RNN model to attain the multiple grid maps for several times-steps. At each time step, the Decoder-ConvLSTM will be initialized with the current internal states of the ConvLSTM-Encoder and produces a sequence of future motion predictions. Here, Decoder-ConvLSTM is a conditional decoder. A conditional decoder receives the last generated output frame as it's input. In the Decoder-ConvLSTM, the first decoder cell will take the *conv*5-3 layers spatial output stakes from ResNet-101. Then, the rest of the Decoder-ConvLSTM cells receives the output of the previous Decoder-ConvLSTM's

cell as its input while training. This mechanism helps to achieve the future grid maps sequence for several time-steps. The output of the Decoder-ConvLSTM cell is passed to the upsampling parts of the network and generate the future grid maps for the next time-steps. During upsampling, while skip connections are added (element-wise addition) with the result of transposed convolutions, the resultant tensors value is used to update the current skip connections tensors value in different scales for the next time-steps processing.

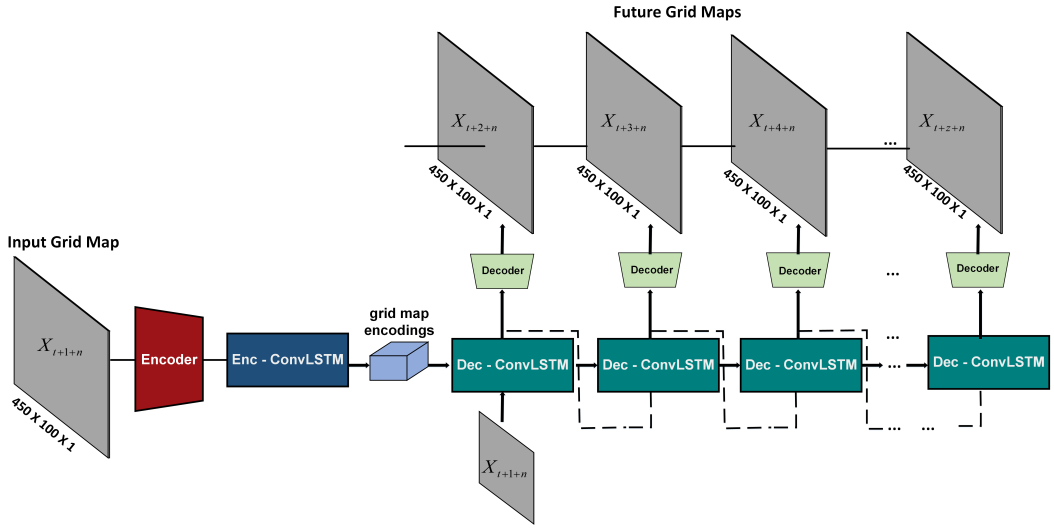


Figure 4.9: RED-ConvLSTM model showing One-to-Many approach. After RED-ConvLSTM model is running with a sequence of input grid maps, the ConvLSTM-Encoder memorizes the grid map encodings and generates a context vector. The Decoder-ConvLSTM uses this context vector and predicts the next sequence of future grid maps. Since the network has already learned the current dynamics of the traffic, it is possible for RED-ConvLSTM model to predict the future sequence of traffic scenes from the present traffic scene.

Model Advantages and Disadvantages

The developed RED-ConvLSTM model showing the Many-to-Many approach. As an application to this RED-ConvLSTM model, we use this model for One-to-Many approach. After RED-ConvLSTM model is running with a sequence of input grid maps, the context vector generated from ConvLSTM-Encoder represents the current motion dynamics of the traffic scenes. Later, the Decoder-ConvLSTM predicts the next sequence grid maps

by using this context vector. Since the network's ConvLSTM-Encoder already has the updated history of the traffic scenarios, it can easily predict the future state of the traffic scenarios for several time-steps based on only the present grid map (Figure 4.9). This has real-time use cases in autonomous driving like future collision avoidance or any kind of risk assessment through the future visual representation of the traffic scenarios. It also solves the previous ConvLSTMED model's drawbacks.

Also, we can infer all the cars possible locations by using all three developed models in this thesis, cause every model infers multiple probabilities of the car locations whether it is a single grid map prediction or sequence of future grid map predictions. For single frame prediction, any individual car has only shorter time location predictions based on the car motion. But, due to the ability of RED-ConvLSTM model to predict multiple grid maps in longer time horizon from the present state of the traffic scenario gives the car the ability to path planning or decision making based on any foreseeable risks.

4.5 Loss Function

Most deep neural network-based image segmentation methods, where each pixel i of a given image must be classified into an object class $c \in C$, rely on logistic regression, optimizing the cross-entropy loss [45]. Pixel-wise cross entropy loss examines each pixel individually, comparing the class predictions to the target image pixels. In this work, the input grid maps represented the traffic scenario with a per-pixel value for occupancy or non-occupancy (1 or 0, respectively). So, this work uses a modified cross-entropy loss for all the proposed models to measure the grid maps per-pixel prediction for the mentioned occupancy or non-occupancy classes. Since cross-entropy loss or log loss measures the performance of a classification model whose output is a probability (p) value between $[0, 1]$. For a binary (foreground vs. background) classification problem, cross entropy is,

$$\begin{aligned} \mathcal{L}_{CE} &= -\frac{1}{N}\sum_{i=1}^{N} y_i \times log(p_i) \\ &= \sum_{i} -y_i log(p_i) + (1 - y_i)\, log(1 - p_i) \end{aligned} \tag{4.3}$$

where $N = pixelnumberoftheimage$ and y_i is the ground truth value (pixel value) of the grid map.

However, when the image prediction methods target rare observations or encountered small foreground objects surrounded by an abundance of background pixels, a severe class imbalance is likely to occur. It is highly likely that this class imbalance causes the

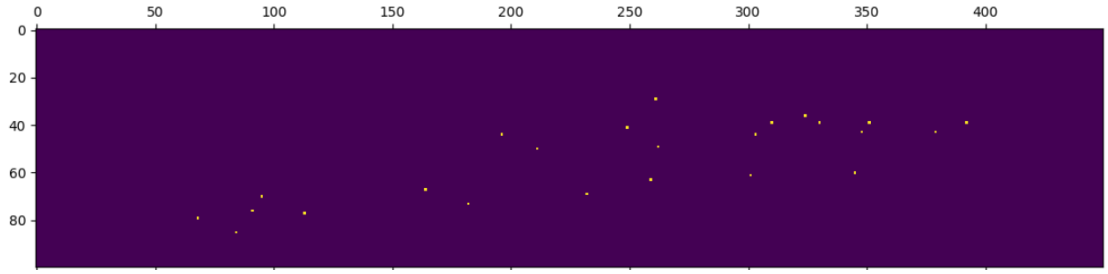


Figure 4.10: A grid-map representation with vehicles (yellow dots).

learning process to get trapped in the local minima of the loss function yielding a network whose predictions are strongly biased towards only to the background class. From the training perspective, this biased is caused by imbalanced data distribution, because semantic segmentation requires pixel-wise labeling and a small number of occupied grid cells contribute less to the loss. As a consequence, the foreground class or region is often missing or only partially detected. In this work, the percentage of the input grid map's foreground pixel to the background pixel is about $\frac{1}{1900} = 0.0526\%$.

Figure 4.10 shows one example of input grid map where vehicle cells are depicted with yellow and it has only 24 vehicles out of 45000 pixels (i.e. resolution 450×00). To handle this class imbalance, we use a weighted version of cross-entropy loss. The weighted cross-entropy for binary classification is defined as,

$$\begin{aligned} \mathcal{L}_{WCE} &= -\frac{1}{N}\sum_{i=1}^{N} w_i \times y_i \times log(p_i), \\ &= \sum_{i} -w_i y_i log(p_i) + (1 - w_i)\,(1 - y_i)\,log(1 - p_i), \end{aligned} \tag{4.4}$$

where $w \in [0, 1]$, is the weighted factor whose value controls the positive and negative probability. In this work, we chose $w = 0.99$ as it empirically led to the best results.

5 Dataset

In this chapter, we explain the dataset that is used in the following work. This grid map dataset was chosen to be used in this thesis work in order to be able to compare the results and analyze the strengths and weaknesses of different network models to evaluate the future motion prediction of vehicles. The chapter discusses this grid map dataset generation with vehicles. Later, dataset preprocessing and formulations are presented. Additionally, samples from the dataset are shown to get a better idea of how the data looks like that is fed to the network.

5.1 Dataset Generation

The data we used in this work was gathered as part of the Providentia project. The Providentia project aims to detect, localize and predict traffic objects and their movements, and make corresponding information available to traffic participants, both to drivers and to self-driving cars [46]. Also, it consists of sensors mounted on a German highway. For future deep learning-based prediction task, a grid map based dataset is generated from the Providentia project. In below, this dataset generation is discussed.

Providentia Dataset

The Providentia system facilitates the integration of self-driving and manual cars, which is a major challenge and the dominant traffic scenario of the coming decades. It provides measurement point of vehicles states and their immediate environment by combining several sensors with redundant measurement principles (e.g. cameras and radars). Based on these sensors fused information, long- and short-term traffic movements are predicted. Various vehicle sensor information is supported for communication and fusion. Supported sensor types include radars, lidars, in-vehicle cameras, and ultrasonic sensors.

In order to detect traffic participants reliably, multiple measurement principles are combined. The importance of measurement points is that it captures the traffic flow in their local regions and extract useful information. For example, cameras (both far and near cameras) have high accuracy in angular resolution and image processing algorithms allow to perform accurate object classification. Multiple cameras with optimized lenses

increase the camera detection range. On the other hand, radar sensor readings are very accurate in object distance and speed, also especially in bad lighting conditions, while angular accuracy is relatively poor and provide good performance during bad weather conditions. Since the Providentia system aims to provide a far-reaching view both in distance and time, a traffic scene prediction with a bird's eye view on the highway, based on radar detections, will provide predictive motion information of all the highway vehicles at once, to illustrate the concept. And to achieve that, a proper vehicles state representation is necessary.

Meanwhile, to get an easy and clear feature representation and constant performance independent of the number of cars, grid map representation approach is suggested, which allows the inclusion of an arbitrary number of sensors, which is a preferred approach for the Providentia system and can be combined with grid map based event detection, as shown in [46].

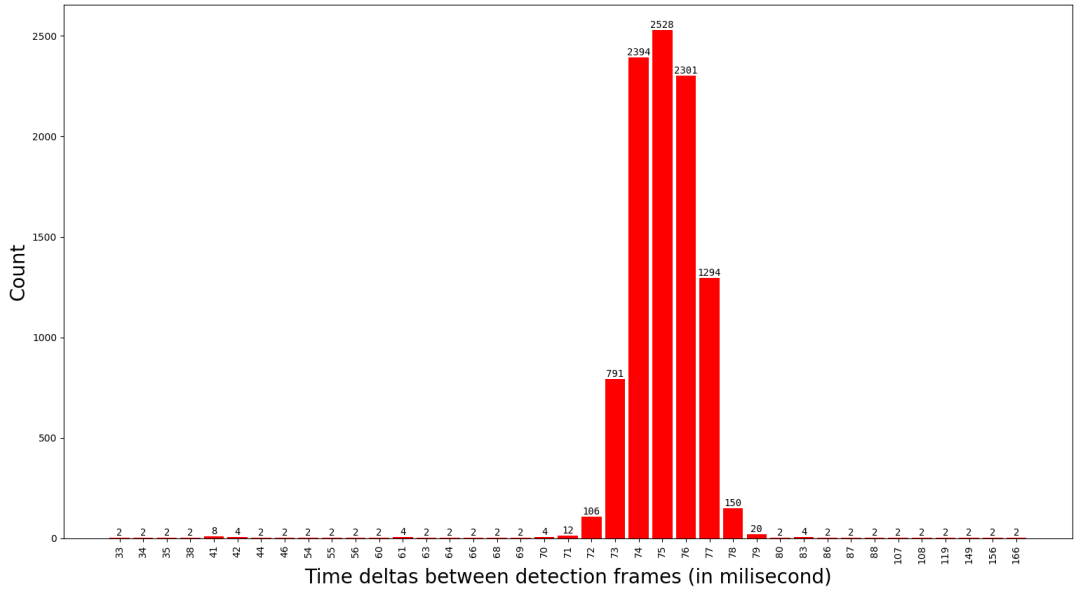


Figure 5.1: Histogram of time deltas between 5000 detection frames. It shows the time difference between adjacent grid maps in milliseconds. The plot confirms that the sampling frequency is regular which is important for the prediction. It also gives an idea for further pre-processing of the dataset. From the plot, it can be seen that 95% of the frames are in between 71ms to 79ms time window.

To get the predictive motion of all vehicles at once by implementing traffic scene prediction using deep learning techniques, grid map representation of vehicles motion states dataset created. The dataset generation procedure initially started by recording various sensors data. The recording time of the recorded data is about 377 seconds which covers approximately 60m long highway lanes. Then from the recorded data, camera image frames are retrieved with approximately 13 frames per second which has the resolution of 1920×1200, and also corresponding vehicle object's radar sensors data are extracted. Later, from the radar sensor data, each frame's vehicles motion characteristics like position is deduced and radar coordinates of the vehicle objects are projected on to the corresponding image frames, subsequently mapped this vehicle objects (of corresponding image frame's) radar properties to a downscaled grid map frame. Dataset is organized with these grid map frames with approximately 13 frames per sec. Each frame is a single channel grid, i.e. a matrix of size $450 \times 100 \times 1$ with binary entries 0 (blue background) and 1 (yellow dotted colored cells) for free and occupied (by vehicle), respectively.

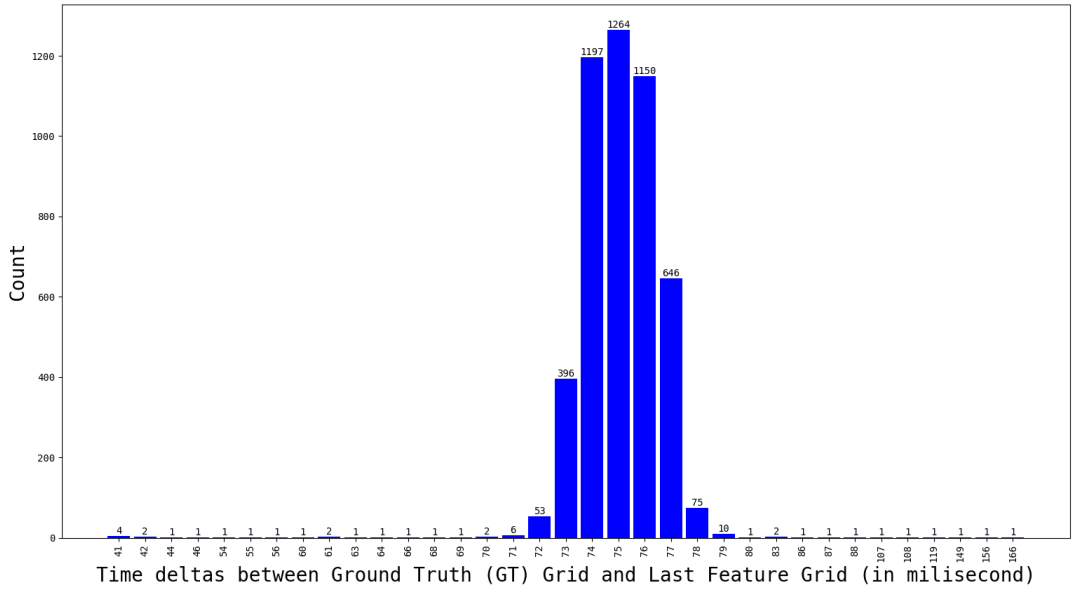


Figure 5.2: The histogram represents the time difference between the last Ground Truth (GT) frame and the last feature grid map frame for all the rows in the dataset. Each row in the dataset contains 4 frames where first 3 frames are feature grid maps and the last one is ground truth (GT).

5.2 Dataset Preprocessing

Providentia grid map dataset initially contains variable length frames per seconds. Due to the absence of constant interval gap between the frames, the learning could be noisy and ambiguous. As the first step of dataset cleaning, a histogram was produced to see the interval values of the two adjacent frames. Figure 5.1 shows the histogram with various range of values from 16ms to 1317ms range between the number of feature frames. It has been clear from the figure that almost 95% of the frames are in between 71ms - 79ms interval. So, the rest of the frames are discarded, which left 4887 frames to use for training, validation and testing processes.

Additionally, instead of feeding the model with continuous values in the normalized range, binary pixel values are used only. This decision results from the use of binary cross-entropy as the main loss function for this dataset, which has shown to be the suitable choice for the image generating models [14].

In this thesis work, to feed the network models, two types of input and output sequence are followed. While creating the dataset, only those rows are taken from the grid map sequence whose ground truth grid map frame and last feature frame difference are in between 71ms to 79ms (Figure 5.2). For the many-to-one prediction models, the dataset is created using 4 frames that is each row contain 4 frames (Figure 5.3). Of the 4 frames, the input sequence has 3 corresponding frames to feed the network, and the last frame is for output sequence as ground truth (GT). In this way, three datasets are prepared for many-to-one prediction models by formulating 1, 2, and 3 time-step ahead frame interval.

On the other hand, for the many-to-many prediction model i.e. for the sequence to sequence model, the input and output sequence length has been constructed to 5 frames each (for 1 time-step ahead frame interval) shown in Figure 5.4. In both cases, the input and output sequence are sequentially formatted with a row and compressed. In this way, for example, for the many-to-one model, the dataset generated 4776 rows (each row contains 4 sequential frames) by using 1 time-step ahead grid map frames. For each model, the dataset feeding mechanism is like below: the input sequence filenames queue, which is filled with references to the rows sequence files generated before, randomly loaded into the memory. Afterward, from randomly shuffled queued input filenames, the training data example are pushed to the batch queue, from which the model loads its batches in every iteration.

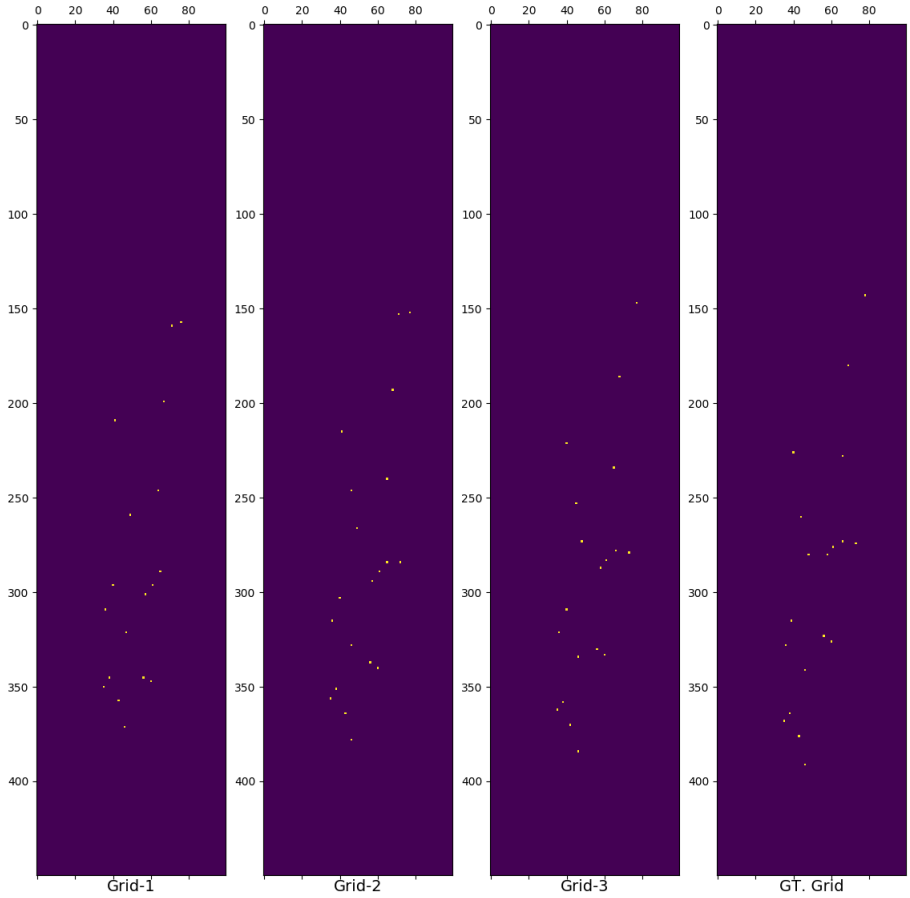


Figure 5.3: Dataset row of four grid map frames for 3 time-step ahead frame. First three frames are feature grid maps and the last grid map shows the future grid map or in this case the ground truth (GT) of the frame.

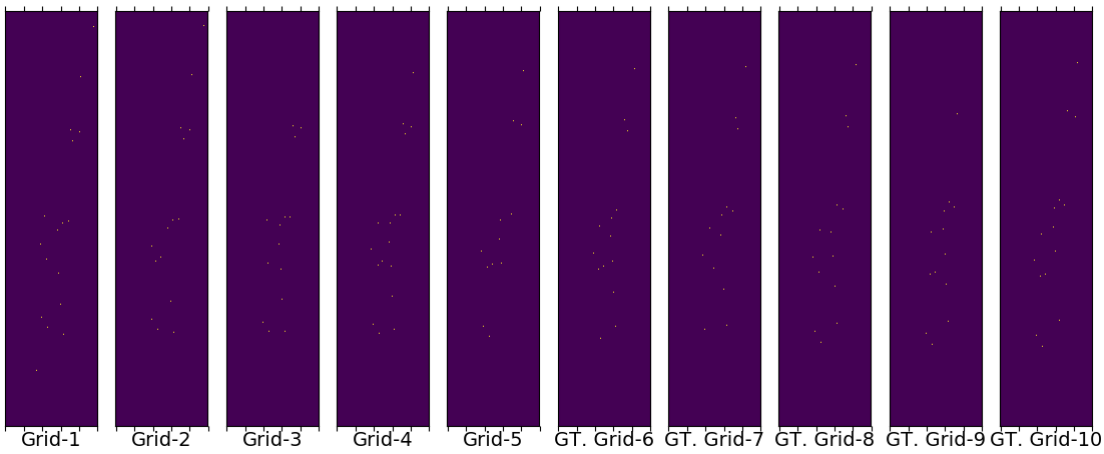


Figure 5.4: For many-to-many prediction model, a sequence dataset row is represented with 10 frames. Of these 10 frames, 5 frames are used as input and the same number of frames are act as output sequence (GT frames).

6 Experiments

In this chapter, we present the evaluation results on the proposed three models - MS-FCN, ConvLSTMED, and RED-ConvLSTM. This chapter is organized as follows. In Section 6.1, a summary of the experiment setup for training the three neural network models is given. The quantitative metrics which are used to evaluate the models' performance are introduced with literature in Section 6.2. All the models quantitative and qualitative results of the suggested models are shown in Section 6.3. Furthermore, the comparison results between the models are discussed in this section. In addition, samples of the predicted images are shown with a qualitative discussion.

6.1 Experiment Setup

In this work, the experiments are conducted on the Providentia dataset (described in chapter 5). In order to evaluate the highway traffic scenario predictions, different driving scenarios are recorded and then represented into a 2D Occupancy Grid Map (OGM). Grid map frames have a resolution of 450×100 pixels. The foreground represents the vehicles. Hence, the dataset contains 2-classes $[background, foreground]$. In general, to train the models the dataset was prepared by generating numbers of record sequences for various time-steps ahead interval. The dataset is split into 3 parts – training, validation, and test set. All the proposed network models training done by 4000 sequences of frames out of 4887 frames. And for validation 500 frames are used. The rest of the frames separated for the test phase. A mini-batch of size 4 is used for training. Single frame predictions models are trained for 30 epochs with learning rate, $\ell = 0.0001$. And, multi-frame prediction model i.e. Recurrent Encoder-Decoder ConvLSTM model is trained for 30 epochs with same learning rate. Evaluation is done on the predicted feature maps by using the Mean Intersection Over Union (mIoU) on the test set of the dataset. All experiments are computed on a single NVIDIA GeForce GTX 1080 Ti using the TensorFlow open source software library for machine intelligence in version r1.10 with CUDA 9.0 and cuDNN 7.2.

6.2 Metrics

In this section, performance evaluation metrics are described here. At first, Mean Intersection Over Union (mIoU) metrics is summarized. Another performance metric

confusion matrix is also described in details. We evaluate the predicted future grid maps of our proposed models using the Mean Intersection Over Union (mIoU). We also show corresponding models Area Under Curve-Receiver Operating Characteristic (AUC-ROC) curve.

Mean Intersection over Union (mIoU)

Let n_{ij} be the number of pixels of class i predicted to belong to class j, where there are n_{cl} different classes and let $t_i = \sum_i n_{ij}$ be the total number of occupied pixels of class i. The most common measure of determining the accuracy of a classified image is the overall Pixel Accuracy (oPA) computed with Equation 6.1,

$$oPA = \frac{\sum_i n_{ii}}{\sum_i t_i}. \tag{6.1}$$

But, one significant limitation of the oP measure is its bias in the presence of very imbalanced classes.

Meanwhile, the Intersection over Union (IoU) is an evaluation metric used to measure the accuracy of an image segment in a specific dataset. It works by calculating the overlapping areas of intersection between two bounding boxes, divided by the total area of both bounding boxes. This produces an "accuracy score" that can be used to measure how close two bounding boxes match. The IoU solves with imbalanced classes concerning oPA and it is nowadays the standard metric to evaluate image segmentation [33].

Mathematically, the Intersection over Union in Equation 6.2. is dividing the area of all true positives $\sum_i n_{ii}$ by the area of union. In this case, n_{ij} is the number of pixels in class i predicted to belong to class j and t_i the total number of pixels of class i.

$$IoU = \frac{\sum_i n_{ii}}{t_i + \sum_j n_{ji} - n_{ii}}, \tag{6.2}$$

where,

$$n_{ii} : \ number\ of\ correctly\ classified\ pixels,$$

$$n_{ij} : \ number\ of\ pixels\ wrongly\ classified,$$

$$t_i : \ total\ number\ of\ pixels\ of\ class\ i.$$

And, the mean IoU is the average over all classes.

$$IoU = \frac{1}{n_{cl}} . \frac{\sum_i n_{ii}}{t_i + \sum_j n_{ji} - n_{ii}}, \tag{6.3}$$

where, n_{cl} is the total number of classes.

Because of this, the IoU defines an evaluation metric that rewards predicted bounding boxes for heavily overlapping with the ground-truth as seen in figure 6.1. Predicted bounding boxes that heavily overlap with the ground-truth bounding boxes have higher scores than those with less overlap. This makes the mean IoU an excellent metric for evaluating class results. Scores above 50% can be considered moderate while scores above 60% reflected respectable results [47].

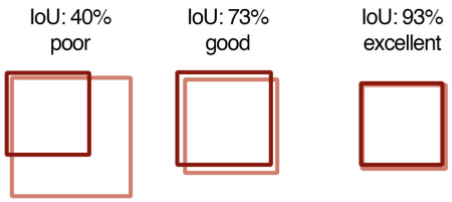


Figure 6.1: The Intersection over Union for various bounding boxes. Predicted bounding boxes that heavily overlap with the ground-truth bounding boxes have higher scores than those with less overlap [47]

Precision and Recall

Given any input, a binary classifier predicts either of two outcomes: positive, or negative. In our grid map prediction problem, vehicle pixels are considered positive, and background pixels, negative. The classifier output can be listed as:

- True Positives (TP): Vehicle pixels are classified correctly.
- True Negatives (TN): Background pixels are classified correctly.
- False Positives (FP): Background pixels are mistakenly classified as vehicles.
- False Negatives (FN): Vehicle pixels are mistakenly classified as the background.

These numbers are generally represented in a confusion matrix, as shown below, and the standard metrics used to evaluate a classifier are derived as different ratios from it. The models are then evaluated with these metrics on the initially isolated test data. This gives us a proxy measure of how well the model is able to generalize.

Two metrics relevant here are Precision and Recall, which are defined as,

$$precision = \frac{TP}{TP + FP}. \tag{6.4}$$

$$recall = \frac{TP}{TP + FN}. \tag{6.5}$$

Due to the high-class imbalance present in the grid map dataset that is vehicles occupancy are rare compared to the background, which covers a far greater spatial area per image, precision, and recall were chosen as the primary metrics.

In addition, Receiver Operating Characteristics (ROC) Curve plays an important role as a performance measurement metric for classification problem at various thresholds settings. The performance of a classifier that produces decision values (i.e., probabilities of a pixel value belonging to either class) can be interpreted more intuitively with the Precision-Recall (Pre-Rec) and Receiver Operating Characteristic (ROC) curves. The ROC curve shows how the recall vs precision relationship changes as we vary the threshold for identifying a positive in our model. A ROC curve plots the true positive rate on the y-axis versus the false positive rate on the x-axis. The true positive rate (TPR) is the recall and the false positive rate (FPR) is the probability of a false alarm. Both of these can be calculated as:

$$TPR = \frac{TP}{TP + FN} \tag{6.6}$$

$$FPR = \frac{FP}{FP + TN} \tag{6.7}$$

The TPR is the fraction of actual vehicle pixels that are correctly predicted as the occupied vehicle in the grid map cell, while the FPR is the fraction of actual background pixels incorrectly predicted as a vehicle. The area under a ROC curve provides a means of measuring the classifier's ability to discriminate between classes in the dataset. By this definition, maximizing the area (ROC-AUC) leads to better classification accuracy.

6.3 Results

In this section, the quantitative and qualitative performance of proposed 3 models, are presented. Since the Multi-Stream FCN (MS-FCN) and the ConvLSTM Encoder-Decoder (ConvLSTMED) models always predict one future single grid map, we presented both the quantitative and qualitative performance evaluation results for both of the models in single frame prediction subsection. On the other hand, proposed Recurrent Encoder-Decoder ConvLSTM predicts a sequence of grid maps and it's quantitative and qualitative

performance evaluation results are presented in multi-frame prediction subsection. All the evaluation results are done on the test dataset. Since this works solely relies on a private project dataset (Providentia dataset), there is no related work that has published any future frame prediction results based on this data.

6.3.1 Single Frame Prediction

For single frame predictions, at first we trained the Multi-Stream FCN (MS-FCN) and the ConvLSTM Encoder-Decoder (ConvLSTMED) models on 1 time-step ahead, 2 time-steps ahead, and 3 time-step ahead grid map dataset. Then we evaluated all the three trained MS-FCN models for three different time-steps ahead prediction. The same evaluation process also goes for the three trained ConvLSTMED models. Later, The quantitative results of both models' are compared to each other at different time-steps. Also, their qualitative results are compared with the ground truth (GT) of the prediction frames for 1 time-step, 2 time-steps, and 3 time-steps ahead. Later, the experiment also compared these performances with each other.

Model	**mIoU**	**AUC-ROC**
MS-FCN	57.63	96.76
ConvLSTMED	61.55	98.85
ConvLSTMED (w/o skip ConvLSTM)	58.49	97.18

Table 6.1: Comparison between MS-FCN and ConvLSTMED models for 1 time-step ahead prediction

Quantitative Results

In table 6.1, MS-FCN and ConvLSTMED models performance to predict next frame for 1 time-steps ahead are presented. In addition, the experiment also compares the performance of the modified ConvLSTMED model where simply intermediate ConvLSTMs modules are removed (w/o skip ConvLSTM) from the proposed network. Instead of using these intermediate encoder ConvLSTMs modules for skip connections, modified ResNet-101's three different layers ($conv1, conv2\text{-}3, conv3\text{-}3$) feature maps are used for skip connection and later perform element-wise addition operation with transposed convolution layers output features while upsampling the network.Table 6.1 shows that for 1 time-step ahead future frame prediction, ConvLSTMED performs better than MS-FCN and ConvLSTMED (w/o skip ConvLSTM) models. For 1 time-step ahead prediction, MS-FCN has Mean Intersection Over Union (mIoU) 57.63, and ConvLSTMED (w/o skip

ConvLSTM) has mIoU 58.49 ,whereas ConvLSTMED has mean intersection over union (mIoU) 61.55.

Model	**mIoU**	**AUC-ROC**
MS-FCN	58.09	97.41
ConvLSTMED	59.55	97.72
ConvLSTMED (w/o skip ConvLSTM)	59.01	97.50

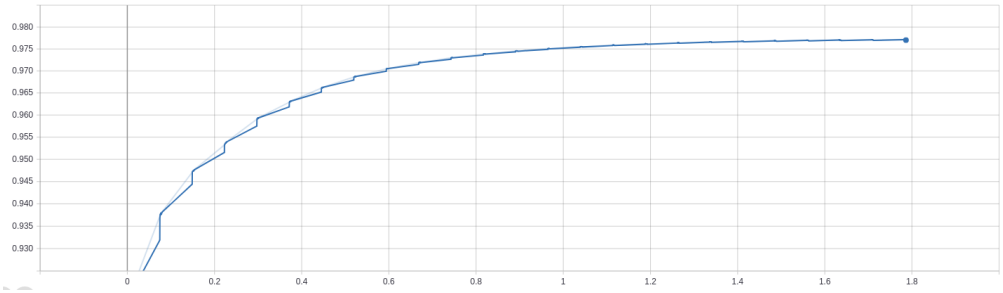
Table 6.2: Comparison between MS-FCN and ConvLSTMED models for 2 time-steps ahead prediction

Model	**mIoU**	**AUC-ROC**
MS-FCN	55.54	96.53
ConvLSTMED	58.34	96.85
ConvLSTMED (w/o skip ConvLSTM)	53.72	96.12

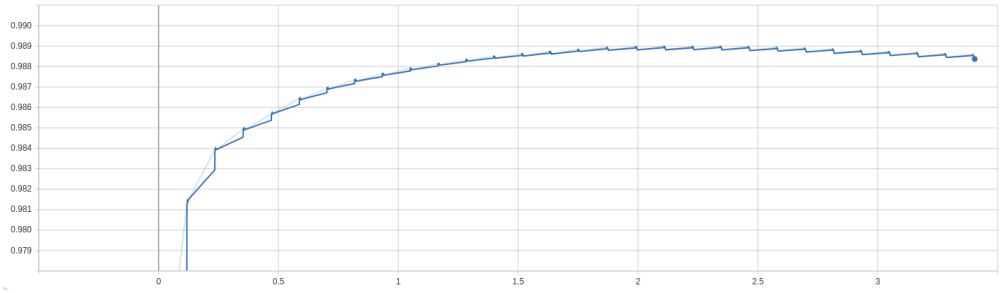
Table 6.3: Comparison between MS-FCN and ConvLSTMED models for 3 time-steps ahead prediction

Also, for 2 time-steps and 3 time-steps ahead predictions, Table 6.2 and Table 6.3 show that the ConvLSTMED model has maintained the same trend by achieving mIoU 59.55 and 58.34, respectively, which are greater than the MS-FCN model's mIoU 58.09 and 55.54 for the same steps ahead prediction, respectively. In the case of ConvLSTMED (w/o skip ConvLSTM) model, for 1 time-step and 2 time-steps ahead frame prediction, the model is performed better than MS-FCN model but has lower performance mIoU value than the ConvLSTMED model. But for the 3 time-steps ahead frame prediction, it has lowest mIoU value i.e. 53.72 compared to other 2 models.

Also, apart from mean IoU, the area under the curve (AUC) - receiver operating characteristics (ROC) curve is being used to measure the performance in between MS-FCN and ConvLSTMED models. All models show a good degree of separability (figure 6.2a and figure 6.2b) to distinguish between occupied cell (vehicles) and free cell (that is background) based on a certain threshold value. In these experiments, ground truth maps all cells with a value above a threshold of ($p_0 > 0.45$) are classified as occupied for both of the models.



(a) AUC-ROC curve for MS-FCN for 1 time-step ahead frame prediction. The classifiers performance in terms of degree of separability shows 96.76%.

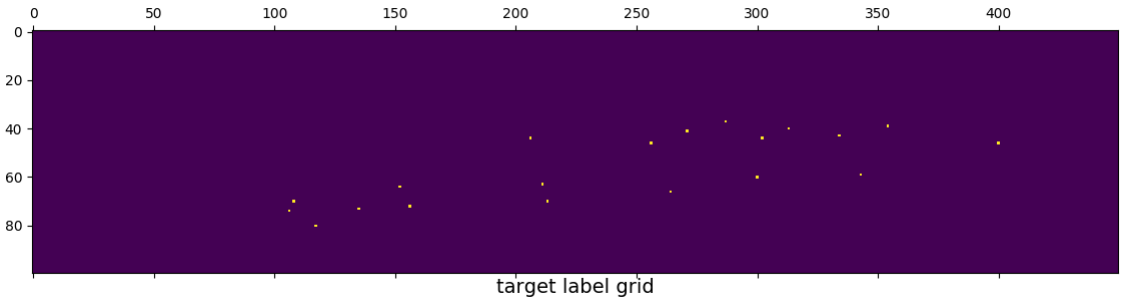


(b) AUC-ROC curve for ConvLSTMED for 1 time-step ahead frame prediction. The classifiers performance in terms of degree of separability shows 98.85%.

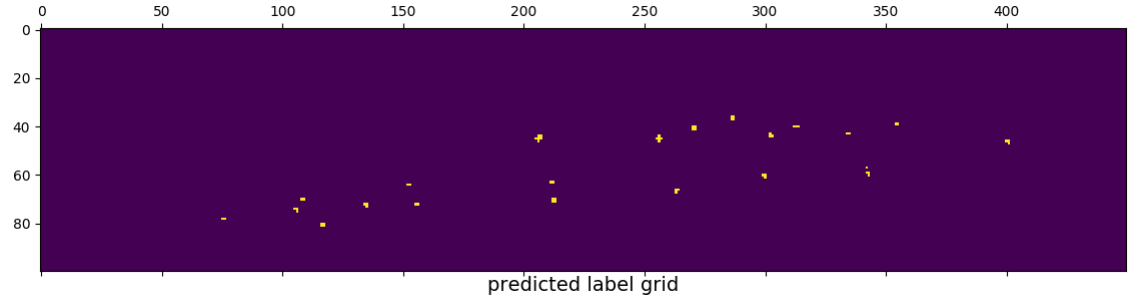
Figure 6.2: AUC-ROC curve comparison between MS-FCN and ConvLSTMED models for 1 time-step ahead prediction.

Qualitative Results

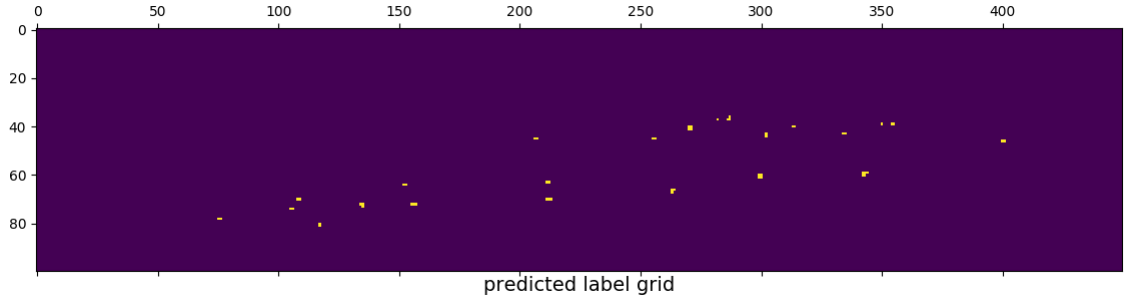
Next, several future frame prediction samples are qualitatively compared with MS-FCN and ConvLSTMED single frame prediction models. To predict the future frame motion, both of the models are able to generate persistent motion of the vehicles correctly in case of one time-step ahead prediction in Figure 6.3.



(a) Ground Truth (GT) of next frame

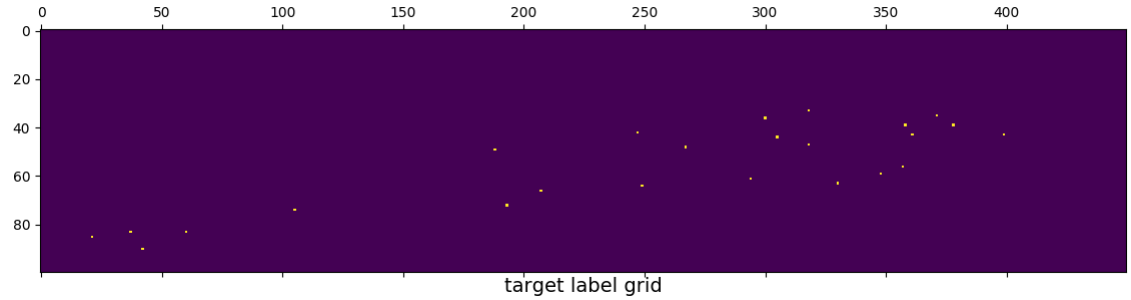


(b) Predicted frame - MS-FCN model

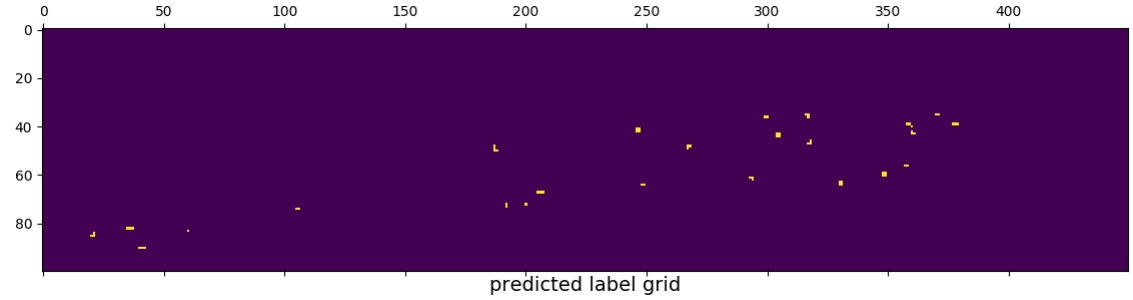


(c) Predicted frame - ConvLSTMED model

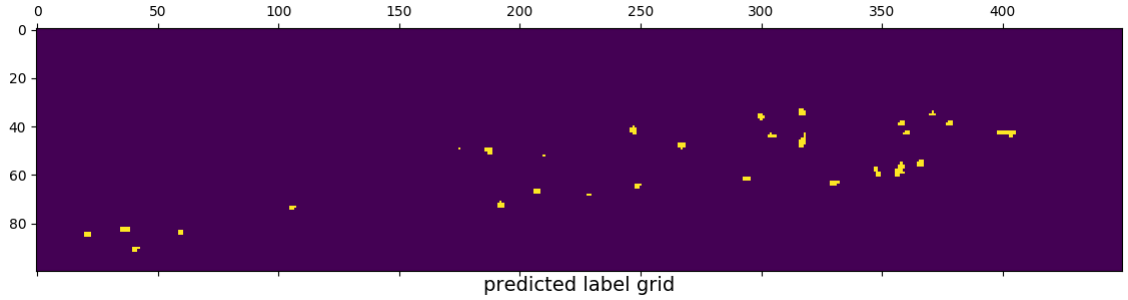
Figure 6.3: Qualitative result examples for the 1 time-step ahead prediction for MS-FCN and ConvLSTMED model



(a) Ground Truth (GT) of next frame



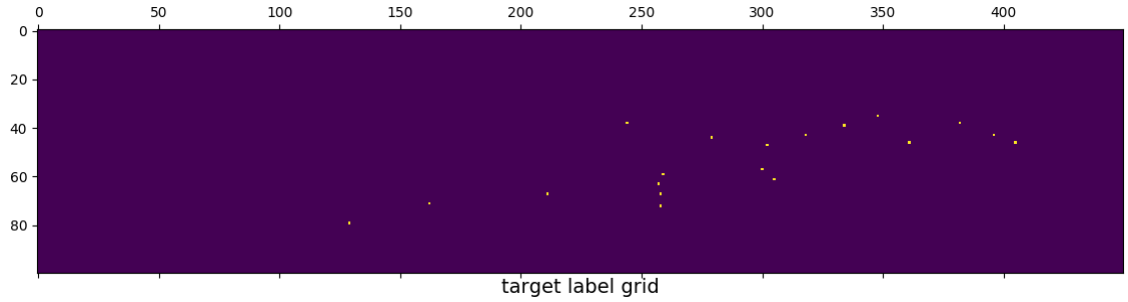
(b) Predicted frame - MS-FCN model



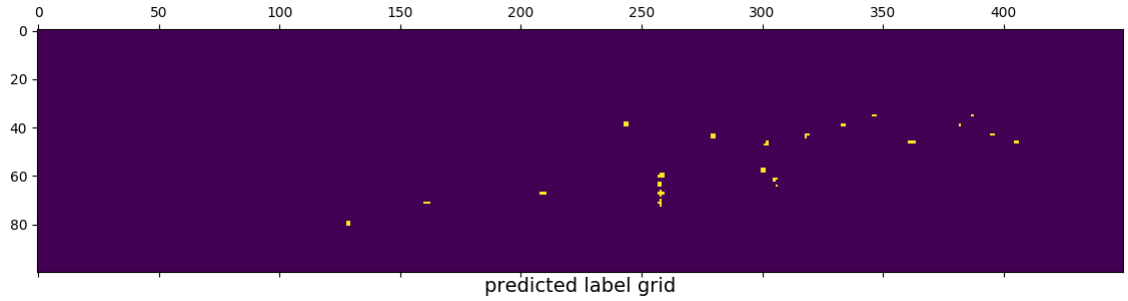
(c) Predicted frame - ConvLSTMED model

Figure 6.4: Qualitative result examples for the 2 time-step ahead prediction: (a) ground truth of next frame, (b) predicted frame from multi-stream FCN model, (c) predicted frame from Encoder-Decoder ConvLSTM model.

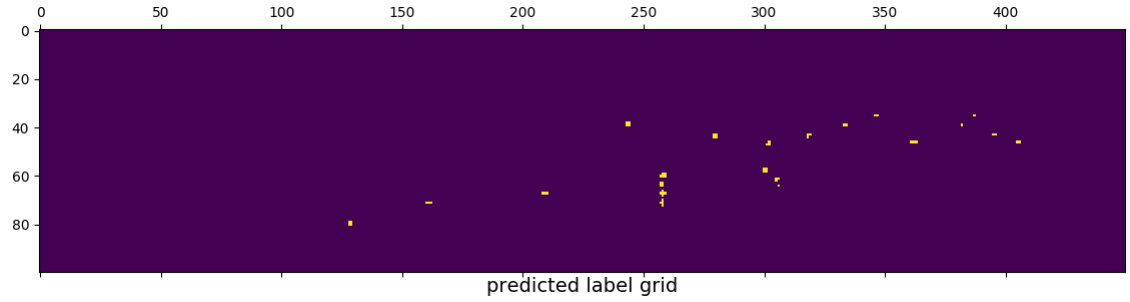
For 2 time-steps ahead frame prediction, MS-FCN model misses one occupied vehicle (Figure 6.4b) in comparison to the ground truth, shown in Figure 6.4a (in the top right of the image) and which is detected by the ConvLSTMED (Figure 6.4c) with higher confidence.



(a) Ground Truth (GT) of next frame



(b) Predicted frame - MS-FCN model



(c) Predicted frame - ConvLSTMED model

Figure 6.5: Qualitative result examples for the 3 time-steps ahead prediction: (a) ground truth of next frame, (b) predicted frame from multi-stream FCN model, (c) predicted frame from Encoder-Decoder ConvLSTM model.

In Figure 6.5, 3 time-steps ahead prediction showing no missing vehicles for both of the

models. One of the common characteristics of both models is that it suffers from false vehicles detection (false positive) to a certain degree.

6.3.2 Multi Frame Prediction

For long-term prediction, the multi-frame prediction is achieved by implementing a Recurrent Encoder-Decoder ConvLSTM (RED-ConvLSTM) model. In this work, the Decoder-ConvLSTM is applied five times to produce the predictions for $t + 75ms, t + 0.15s, t + 0.225s, t + 0.3s, t + 0.375s$, where t is the current time. In this work, the model works on the 10 grid map frames sequence. The network model is observing with 5 input grid maps by using Providentia grid map dataset. During the inference time, the network ConvLSTM-Encoder generalize the motion dynamics and generate a context vector which is then passed to ConvLSTM-Decoder. The ConvLSTM-Decoder then predicts next five sequences of frames in a recurrent way, based on already generalized conext vector and past motion patterns learned from the previous frames. Table 6.4 shows the performance of the model. The mIoU is calculated for the RED-ConvLSTM model by averaging over all the predicted frames sequence mIoU. So, the average mIoU, in this case, for RED-ConvLSTM model is found 56.84. And, the AUC-ROC shows 89.63% (Figure 6.6) to distinguish the occupied cell and free cell with the threshold value ($p_0 > 0.45$).

Frame Interval ahead	**Model**	**mIoU|**	**AUC-ROC**
1 time-step ahead	RED-ConvLSTM	56.84	89.63

Table 6.4: Table mIoU measurement of RED-ConvLSTM model.

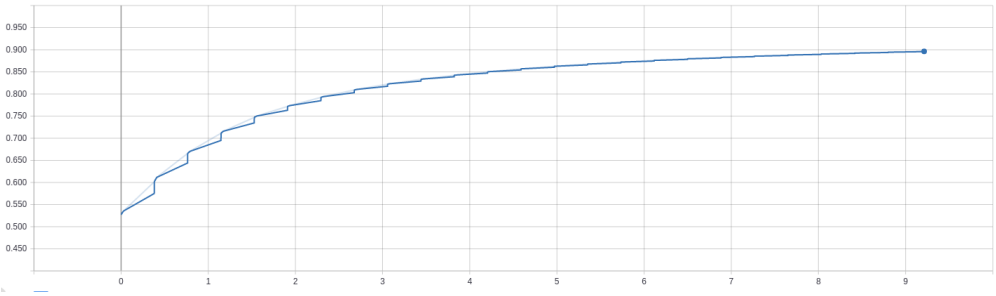
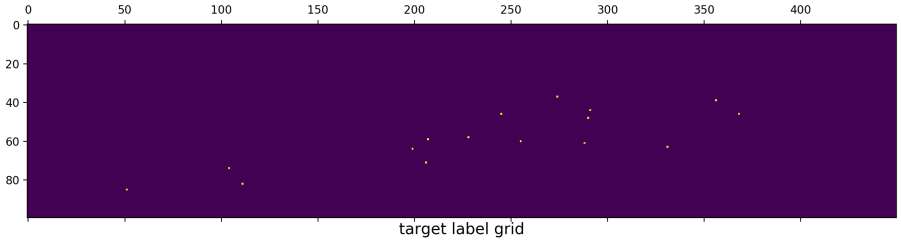


Figure 6.6: AUC-ROC curve for RED-ConvLSTM for 1 time-step ahead frame prediction for the 10 grid maps sequence (5 for input grid and 5 for ground truth grid map frames). It shows classifiers accuracy almost 90%.

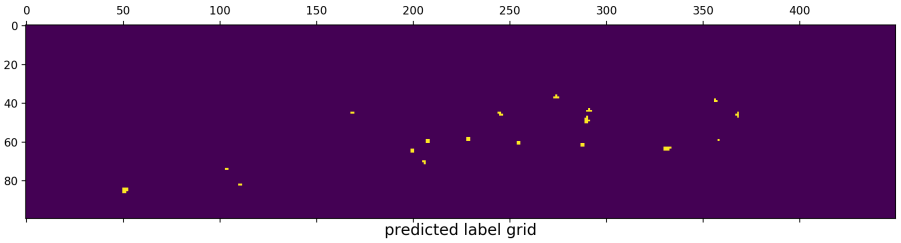
Qualitative Results

It demonstrates test results of the first predicted frame, as well as the average metric values of the forecasted 10 frames long sequence. From Figure 6.7 to Figure 6.11, the sequence model's grid map frames predictions are visualized. The prediction example in all those Figures (6.7 to 6.11) points out that the all the ground truth frames are almost correctly predicted by the model's resulted prediction frames. The prediction goes on completely with 100% confidence with all the vehicles motion detection for the first 2 frames (Figure 6.7 and Figure 6.8), but, has suffered a washed-out of one vehicle from the predicted 3rd frame (6.9). Meanwhile, the rest of the sequence (Figure 6.10 and Figure 6.11) did not suffer any missing vehicle from the ground truth.

Also, surprisingly, the prediction of the 3rd frame's probability is much lesser than the next two frames. One insight, for this reason, is that for dynamic vehicle motion prediction, in this experiment for the multi-frame predictions a static threshold value ($p_0 > 0.45$) is used like single frame prediction models. Otherwise, rest of the predictions sequence follows the expected outcomes like the false vehicle's occupancy (false positive) are presented in a lesser degree than the farthest predicted frames (frame 4 and 5). Lastly, we can make another evaluation from the Figure 6.3 where MS-FCN and ConvLSTMED models 1 time-step frame prediction is depictd, and Figures 6.7, 6.11 where frames sequence is predicted by RED-ConvLSTM model, for 1 time-step ahead prediction. For all the three models, it can seen that they hardly miss any cars for 1 time-step ahead prediction and performed approximately equal.

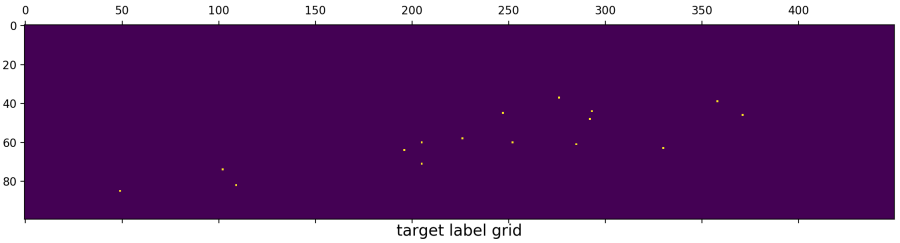


(a) Ground Truth Frame-1.

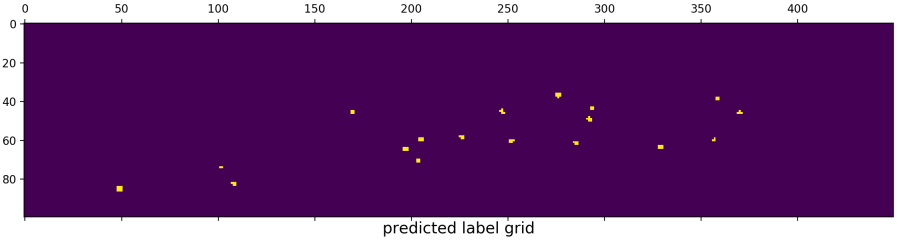


(b) Predicted Frame-1.

Figure 6.7: Qualitative examples of RED-ConvLSTM model for multi-frame prediciton.

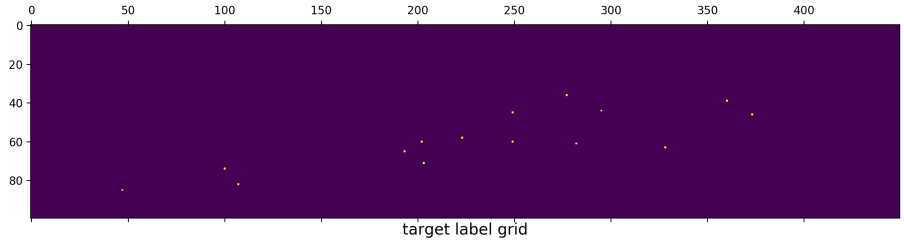


(a) Ground Truth Frame-2.

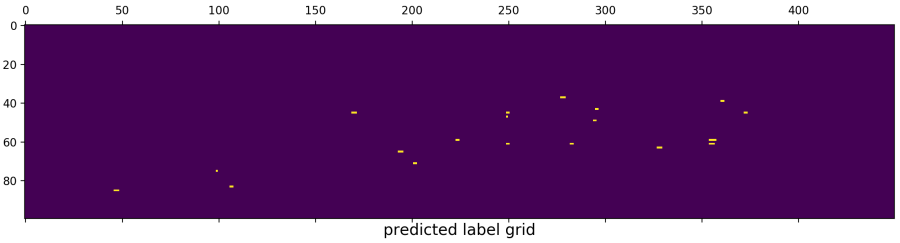


(b) Predicted Frame-2

Figure 6.8: Qualitative examples of RED-ConvLSTM model for multi-frame prediciton.

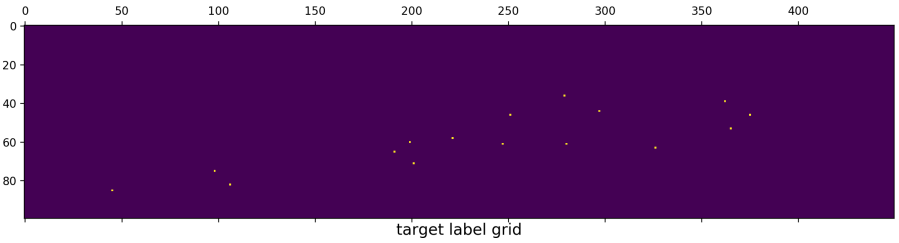


(a) Ground Truth Frame-3.

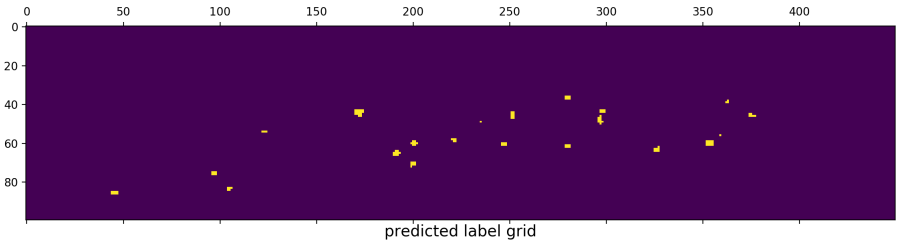


(b) Predicted Frame-3

Figure 6.9: Qualitative examples of RED-ConvLSTM model for multi-frame prediciton.

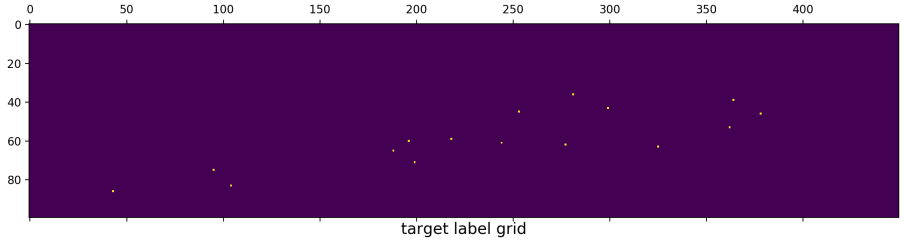


(a) Ground Truth Frame-4.

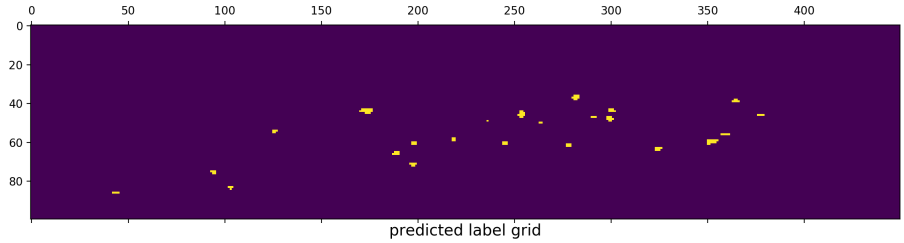


(b) Predicted Frame-4.

Figure 6.10: Qualitative examples of RED-ConvLSTM model for multi-frame prediciton.



(a) Ground Truth Frame-5.



(b) Predicted Frame-5.

Figure 6.11: Qualitative examples of RED-ConvLSTM model for multi-frame prediciton.

7 Conclusions

Autonomous driving comes with the motivation to make road traffic safer, more efficient, more economical, and more comfortable. In order to make a driving safety environment, autonomous vehicles need to understand the surrounding participants' behavior. Instead of considering single or more vehicle interactions, a predictive approach of visualizing the future motion prediction patterns of all the vehicles within traffic scenarios will lead to a more safer and collision avoidance driving environment. In this thesis work, prediction of vehicle motion within a traffic scene is studied in order to measure a predictive motion of all the vehicles from a bird's eye view presentation. As a result, an occupancy grid map (OGM) representation is made to represent simultaneous occurring vehicles motion data collected from radar sensors. The study then presented several DNN architectures by taking OGM as input and developed single frame prediction and multi-frame prediction models to predict future motion of the vehicles in between the occupied traffic scene. In the proposed single frame prediction models, MS-FCN architecture suffers a recurrent nature due to its inability to capture the past motion only from the previous frame. This drawback is overcome by introducing an ConvLSTM Encoder-Decoder model where a recurrent ConvLSTM preserves the spatiotemporal correlation of the data which in turn helps the network to predict the future motion of the vehicle only from the past grid map frame.

Furthermore, the thesis examined a RED-ConvLSTM model in order to achieve the long-term prediction mechanism. To achieve that, the study exploited one ConvLSTM-Encoder and one ConvLSTM-Decoder to generalize the sequence of motion dynamics of the grid map frames by predicting their occupancy probabilities. The encoder produced this learned representation of motion characteristics which later utilized by the conditional decoder to produce the stream of future frames which shows the future probable occupancy of vehicles within the traffic scenario for several time-steps. This motion model helps the future vehicles to plan to avoid any risk, accidents or even a path planning strategy. All the three network models are further evaluated quantitatively and qualitatively on various time-steps datasets with different settings for both single frame and multi-frame predictions.

However, the proposed DNN architectures are the primary architectures for this Providentia grid map dataset. Only radar motion sensor data is represented in the grid map.

Since the prediction of vehicle motion plays a key role such as the references for the drivers, the construction of the city road network, and the design of the vehicular network routing, thus, in future work, DNN architectures will be developed to tackle various traffic scenarios and decision-making models by using more complex data. Also, the proposed network model can be more fine-tuned in more detail, so that, in longer frame interval it would not miss the vehicle motion. Also, more real-time traffic scenarios can be adapted to these networks which will make a larget dataset to train. So, the network's predictions could be more realistic. While training these models, few scenarios could not be solved by the current technique. One scenario is when a new vehicle is entering into the traffic scene (that is present in ground truth frame) whose motion information is not known to the model; as a result, models could not predict this vehicle. Also, another scenario is while the ground truth frame does not contain the outgoing vehicle, but future predicted frame contains the motion and showed it to the output frame. All the models are suffering from these 2 scenarios. And for the sequential recurrent model, another future improvement will be to test the model with more frames for longer motion prediction.

List of Figures

List of Tables

Bibliography

[1] A. Geiger, P. Lenz, and R. Urtasun. "Are we ready for autonomous driving? The KITTI vision benchmark suite." In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. June 2012, pp. 3354–3361. DOI: 10.1109/CVPR.2012.6248074.

[2] P. Kumar, M. Perrollaz, S. Lefèvre, and C. Laugier. "Learning-based approach for online lane change intention prediction." In: *2013 IEEE Intelligent Vehicles Symposium (IV)*. June 2013, pp. 797–802. DOI: 10.1109/IVS.2013.6629564.

[3] S. Yoon and D. Kum. "The multilayer perceptron approach to lateral motion prediction of surrounding vehicles for autonomous vehicles." In: *2016 IEEE Intelligent Vehicles Symposium (IV)*. June 2016, pp. 1307–1312. DOI: 10.1109/IVS.2016.7535559.

[4] J. Morton, T. A. Wheeler, and M. J. Kochenderfer. "Analysis of Recurrent Neural Networks for Probabilistic Modeling of Driver Behavior." In: *IEEE Transactions on Intelligent Transportation Systems* 18.5 (May 2017), pp. 1289–1298. ISSN: 1524-9050. DOI: 10.1109/TITS.2016.2603007.

[5] N. Lee, W. Choi, P. Vernaza, C. B. Choy, P. H. S. Torr, and M. K. Chandraker. "DESIRE: Distant Future Prediction in Dynamic Scenes with Interacting Agents." In: *CoRR* abs/1704.04394 (2017). arXiv: 1704.04394.

[6] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo. "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting." In: *NIPS*. 2015.

[7] M. Drozdzal, E. Vorontsov, G. Chartrand, S. Kadoury, and C. J. Pal. "The Importance of Skip Connections in Biomedical Image Segmentation." In: *LABELS/DLMIA@MICCAI*. 2016.

[8] M. Schreier, V. Willert, and J. Adamy. "An Integrated Approach to Maneuver-Based Trajectory Prediction and Criticality Assessment in Arbitrary Road Environments." In: *IEEE Transactions on Intelligent Transportation Systems* 17 (2016), pp. 2751–2766.

[9] B. Kim, C. M. Kang, S. Lee, H. Chae, J. Kim, C. C. Chung, and J. W. Choi. "Probabilistic Vehicle Trajectory Prediction over Occupancy Grid Map via Recurrent Neural Network." In: *CoRR* abs/1704.07049 (2017). arXiv: 1704.07049.

[10] D. Lee, Y. P. Kwon, S. McMains, and J. K. Hedrick. "Convolution neural network-based lane change intention prediction of surrounding vehicles for ACC." In: *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)* (2017), pp. 1–6.

[11] A. Zyner, S. Worrall, J. R. Ward, and E. M. Nebot. "Long short term memory for driver intent prediction." In: *2017 IEEE Intelligent Vehicles Symposium (IV)* (2017), pp. 1484–1489.

[12] D. Lenz, F. Diehl, M. T. Le, and A. Knoll. "Deep neural networks for Markovian interactive scene prediction in highway scenarios." In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. June 2017, pp. 685–692. DOI: 10.1109/IVS.2017.7995797.

[13] M. Ranzato, A. Szlam, J. Bruna, M. Mathieu, R. Collobert, and S. Chopra. "Video (language) modeling: a baseline for generative models of natural videos." In: *CoRR* abs/1412.6604 (2014).

[14] N. Srivastava, E. Mansimov, and R. Salakhutdinov. "Unsupervised Learning of Video Representations Using LSTMs." In: *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*. ICML'15. Lille, France: JMLR.org, 2015, pp. 843–852.

[15] Z. Luo, B. Peng, D.-A. Huang, A. Alahi, and L. Fei-Fei. "Unsupervised Learning of Long-Term Motion Dynamics for Videos." In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 7101–7110.

[16] R. Villegas, J. Yang, S. Hong, X. Lin, and H. Lee. "Decomposing Motion and Content for Natural Video Sequence Prediction." In: *CoRR* abs/1706.08033 (2017). arXiv: 1706.08033.

[17] I. Sutskever, O. Vinyals, and Q. V. Le. "Sequence to Sequence Learning with Neural Networks." In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'14. Montreal, Canada: MIT Press, 2014, pp. 3104–3112.

[18] S. Park, B. Kim, C. M. Kang, C. C. Chung, and J. W. Choi. "Sequence-to-Sequence Prediction of Vehicle Trajectory via LSTM Encoder-Decoder Architecture." In: *2018 IEEE Intelligent Vehicles Symposium (IV)* (2018), pp. 1672–1678.

[19] X. Jin, H. Xiao, X. Shen, J. Yang, Z. Lin, Y. Chen, Z. Jie, J. Feng, and S. Yan. "Predicting Scene Parsing and Motion Dynamics in the Future." In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 6915–6924.

[20] P. Luc, N. Neverova, C. Couprie, J. Verbeek, and Y. Lecun. "Predicting Deeper into the Future of Semantic Segmentation." In: *ICCV 2017 - International Conference on Computer Vision*. Venise, Italy: IEEE, Oct. 2017, pp. 648–657. DOI: 10.1109/ICCV.2017.77.

[21] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.

[22] M. A. Nielsen. *Neural Networks and Deep Learning*. misc. 2018.

[23] Y. LeCun, Y. Bengio, and G. E. Hinton. "Deep learning." In: *Nature* 521.7553 (2015), pp. 436–444. DOI: 10.1038/nature14539.

[24] A. L. Jone. *An Explanation of Xavier Initialization*. http://andyljones.tumblr.com/post/110998971763/an-explanation-of-xavier-initialization/, Last accessed on 2018-11-30. 2015.

[25] X. Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks." In: *JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*. Vol. 9. Chia Laguna Resort, Sardinia, Italy, May 2010, pp. 249–256.

[26] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Neurocomputing: Foundations of Research." In: ed. by J. A. Anderson and E. Rosenfeld. Cambridge, MA, USA: MIT Press, 1988. Chap. Learning Representations by Back-propagating Errors, pp. 696–699. ISBN: 0-262-01097-6.

[27] Andrej Karpathy. *Neural Networks Part 1: Setting up the Architecture (Course Notes)*. http://cs231n.github.io//neural-networks-1//, Last accessed on 2018-12-11. 2018.

[28] Andrej Karpathy, Justin Johnson, and Fei-Fei Li. *Lecture Notes to CS231n: Convolutional Neural Networks for Visual Recognition*. http://cs231n.github.io//convolutional-networks//, Last accessed on 2018-12-10. 2015.

[29] Y. LeCun, L. Bottou, and P. Haffner. "Gradient-Based Learning Applied to Document Recognition." In: 1998.

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105.

[31] handong1587. *handong1587.github.io*. https://github.com/handong1587/handong1587.github.io/blob/master/_posts/deep_learning/2015-10-09-dl-tutorials.md, Last accessed on 2018-12-09. 2018.

[32] V. Dumoulin and F. Visin. "A guide to convolution arithmetic for deep learning." In: *CoRR* abs/1603.07285 (2016).

[33] J. Long, E. Shelhamer, and T. Darrell. "Fully convolutional networks for semantic segmentation." In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015, pp. 3431–3440. DOI: 10.1109/CVPR.2015.7298965.

[34] T. Cooijmans, N. Ballas, C. Laurent, and A. C. Courville. "Recurrent Batch Normalization." In: *CoRR* abs/1603.09025 (2016). arXiv: 1603.09025.

[35] Andrej Karpathy. *The Unreasonable Effectiveness of Recurrent Neural Networks.* http://karpathy.github.io/2015/05/21/rnn-effectiveness/, Last accessed on 2018-12-20. 2015.

[36] P. J. Werbos. "Backpropagation Through Time: What It Does and How to Do It." In: 1990.

[37] "On the difficulty of training Recurrent Neural Networks Additional material 1 Analytical analysis of the exploding and vanishing gradients problem 1 ." In: 2013.

[38] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory." In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

[39] C. Olah. *Understanding LSTM Networks.* http://colah.github.io/posts/2015-08-Understanding-LSTMs/, Last accessed on 2018-12-23. 2015.

[40] Eugine Kang. *Long Short-Term Memory (LSTM): Concept.* https://medium.com/@kangeugine/long-short-term-memory-lstm-concept-cb3283934359/, Last accessed on 2018-12-24. 2017.

[41] Wikipedia Org. *Hadamard product (matrices).* https://en.wikipedia.org/wiki/Hadamard_product_(matrices)/, Last accessed on 2018-12-25. 2018.

[42] D. Bahdanau, K. Cho, and Y. Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate." In: *CoRR* abs/1409.0473 (2014).

[43] H. P. Moravec and A. Elfes. "High resolution maps from wide angle sonar." In: *ICRA*. 1985.

[44] S. Thrun. "Learning Occupancy Grid Maps with Forward Sensor Models." In: *Auton. Robots* 15.2 (Sept. 2003), pp. 111–127. ISSN: 0929-5593. DOI: 10.1023/A:1025584807625.

[45] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition." In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778.

[46] R. Gruner, P. Henzler, G. Hinz, C. Eckstein, and A. Knoll. "Spatiotemporal representation of driving scenarios and classification using neural networks." In: *2017 IEEE Intelligent Vehicles Symposium (IV)* (2017), pp. 1782–1788.

[47] Adrian Rosebrock. *Image search image guide - Resource guide.* `https://www.pyimagesearch.com/wp-content/uploads/2014/02/ImageSearchEngineResourceGuide.pdf/`, Last accessed on 2018-12-25. 2016.