Manuel Rebol

Frame-To-Frame Consistent Semantic Segmentation

MASTER'S THESIS

to achieve the university degree of
Diplom-Ingenieur

Master's degree programme
Software Engineering and Management

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dr.techn. Dipl.-Ing. Thomas Pock
Institute for Computer Graphics and Vision

Dipl.-Ing. Patrick Knöbelreiter, BSc
Institute for Computer Graphics and Vision

Graz, Austria, Jun. 2019

Abstract

Currently, most image understanding algorithms process single image data, although in many applications video data can be produced or is already available. This individual processing leads to inconsistent scene interpretation because of the missing temporal information. Consequently, physically implausible results are produced, which can be observed when watching a predicted video sequence.

In this thesis we tackle this problem by training a convolutional neural network (CNN) to perform frame-to-frame consistent semantic segmentation. We use a prediction oracle to create missing ground-truth labels for video data together with a synthetic video data set to train our model. In order to propagate features through the different time steps in a scene, we implement recurrent convolutional layers. More precisely, we use long short term memory (LSTM) and convolutions over the time dimension. Besides the temporal feature propagation, we also add a inconsistency penalty to the loss function which enforces frame-to-frame consisted prediction. The methods are evaluated with a newly created VSSNet architecture as well as on the state-of-the art ESPNet architecture.

Results show that the performance improves for the VSSNet and for the ESPNet when utilizing video information compared to single frame prediction. We evaluate our models on the Cityscapes validation dataset. The mean intersection over union (mIoU) on the 19 classes of the Cityscapes dataset increases from 44.0 % on single frame images to 56.5 % on video data for the ESPNet. When using a LSTM to propagate features trough time mIoU raises to 57.9 % while inconsistencies decrease from 2.4 % to 1.3 % which is an improvement by 46.0 %.

The presented results suggest that the added temporal information gives frame-to-frame consistent and more accurate image understanding compared to single frame processing. This ensures that CNN architectures with few parameters and low computational effort are already able to predict scenes accurately.

Kurzfassung

Heutzutage verarbeiten die meisten Bildverarbeitungs- und Bilderkennungsalgorithmen jedes Bild einzeln, obwohl es vielen Anwendungsfällen möglich wäre Videodaten zu generieren oder diese sogar schon vorliegen. Durch die separate Verarbeitung werden Szenen inkonsistent interpretiert. Es entstehen physikalisch unplausible Ergebnisse, die beim betrachten solcher Videosequenzen mit freiem Auge erkennbar sind.

In dieser Arbeit widmen wir uns diesem Problem. Wir verwenden die Methode des überwachten Lernens mit Convolutional Neural Networks (CNNs) um innerhalb einer Szene konsistente Semantische Segmentierung zu erzeugen. Um unser neuronales Netzwerk zu trainieren produzieren wir semantisch segmentierte Videodaten, indem wir auf das bereits trainierte DeepLab Netzwerk zurückgreifen. Weiters versuchen wir auch synthetische Szenen einfließen zu lassen. Die mittels des CNNs extrahierten Features propagieren wir mithilfe von Recurrent Neural Networks (RNNs) von einem Bild zum nächsten in einer Videoszene. Eine Convolutional Long Short-Term Memory (ConvLSTM) Schicht innerhalb der Netzwerk Architektur ist hierfür verantwortlich. Um möglichst konsistente Ergebnisse zu erhalten, bestrafen wir Inkonsistenzen während des Trainings in der Zielfunktion. Wir verwenden die selbst erzeuge Architektur VSSNet und die effiziente ESPNet Architektur um die Methoden zu evaluieren.

Unsere Ergebnisse zeigen, dass wir die Semantische Segmentierung sowohl beim VSSNet als auch beim ESPNet verbessern können, wenn wir Videodaten verwenden. Wir evaluieren unsere Modelle am Cityscapes Validierungsdatensatz. Die Mean Intersection Over Union (mIoU) Metrik die wir auf den 19 Cityscapes Klassen berechnen, können wir beim ESPNet von 44,0 % bei Einzelbildern auf 56,5 % bei Videodaten verbessern. Wenn wir nun zusätzlich eine LSTM-Schicht hinzufügen, um die Feature über die Zeitachse hinweg weiterzugeben, erreichen wir 57,9 % mIoU. Im gleichen Zug vermindern wir Inkonsistenzen von 2,4 % auf 1,3 %, was einer Verbesserung von 46,0 % entspricht.

Die präsentierten Ergebnisse unterstreichen die These, dass das in Betracht ziehen von

Zeitabhängigkeiten in Videos konsistente und genaue Resultate hervorruft. Im direkten Vergleich zu Einzelbildverarbeitung konnte eine deutliche Steigerung in der Bilderkennungsplausiblilität erzielt werden.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

The text document uploaded to TUGRAZonline is identical to the presented master's thesis dissertation.

Place	Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Ort	Datum	Unterschrift

Contents

List of Figures

List of Tables

1 Introduction

Contents

In the age of smartphones it has become easier than ever before to take pictures and record videos. The ability to create memories visually is available all the time, because the smartphone is usually nearby all the time. Both simplicity and availability causes people to produce a large amount of image and video data. One problem of this large amount of data is that it can only be interpreted by humans. Therefore, images have to be labeled, sorted or put together in a photo album manually, which takes a lot of time. If computer vision algorithms would be able to understand and interpret images, they could assist us in handling the large amount of data.

Similar to smartphones, visual sensors are used for business applications as well to produce high quality images and videos. Many branches (medicine, mobility, industry, etc.) already benefit tremendously from the capabilities of imaging sensors in combination with computer vision algorithms. Products and services can be created more efficiently to enhance quality of life for humans.

The visual sense is the most important sense to humans. It generates about 70 % of information the brain collects [65]. A human can have a short look at an image and already knows which objects are present and what the scenario is about. The quote "A picture is worth a thousand words" already suggests that there is a lot of information in an image. Humans can even write stories several pages long after seeing a picture just for a few seconds. However, getting a deeper understanding of an image is very difficult for machines. Computer vision algorithms have problems identifying objects in an image correctly. Getting human level performance at understanding and describing the whole scenery is not possible at the moment, although there exists work which tries to achieve this [16]. The objective of this thesis is to improve algorithmic performance of a specific field within image understanding, semantic segmentation.

Although machines are behind humans in the discipline of image understanding, they are improving rapidly. There exist many applications which computer vision algorithms are excellent for. For example, the company Facebook needs to check 3,500 images for inappropriate content every second [80]. This is just one example of tasks which are already delegated to computer algorithms.

Furthermore, monotonous assembly line work such as sorting out defective parts can be done by robots with cameras. They can do simple tasks faster which allows humans to focus on more challenging work. Another application is medical imaging, where vision algorithms assist doctors by highlighting and detecting critical areas in patient images.

A very popular application is the development of visual algorithms for autonomous vehicles. The reason for the popularity is that this technology can have an impact on everyones daily life. On the one hand, this technology has the potential to save time. Statistics indicate that the amount of time equal to 162 lifetimes is wasted every day by commuting within the US only [36]. On the other hand, it could also improve safety. Road traffic injuries are the eighth leading cause of death and first among people aged 5 to 29 [1]. In 2018, 1.35 million people died in road traffic. Since these accidents are mainly caused by human error, machine driven vehicles have the possibility to save human lives.

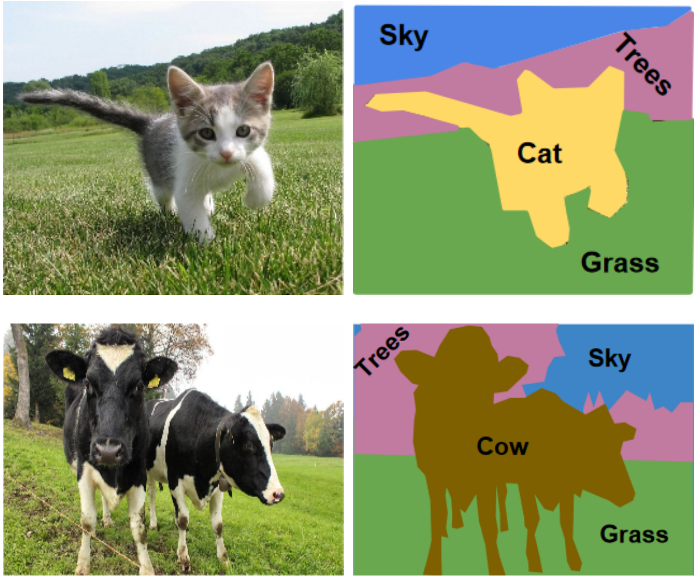
Image analysis and understanding is definitely an important research field with many interesting applications. Although many successful projects have already been developed, there is still a long path towards reaching human level performance.

1.1 Image Analysis and Understanding

It is important to understand the high amount of image data available, otherwise lots of data is uninterpretable and useless for machine processing. Image analysis and understanding algorithms extract information out of raw RGB image data. One of the most important information to extract from an image is to identify which objects are present and what it is about. These problems are faced at different granularity by the four main disciples: i) image classification, ii) object recognition, iii) semantic segmentation and iv) instance segmentation.

This thesis focuses on semantic segmentation, because it is a challenging problem which computes semantic labels of a scenery. Semantic segmentation decides for each pixel in an image to which semantic class it belongs to. An example is shown in Figure 1.1.

Semantic segmentation is very useful when the task is not only to detect the different objects, but also to identify the exact boundaries. Important applications include understanding of street scenes and medical image analysis.



RGB Image Semantic Segmentation

Figure 1.1: Semantic Segmentation. Two examples of semantic segmentation. Each pixel of an RGB image is assigned to a semantic class.

1.2 Video Data

As already pointed out, the large number of cameras and the instant possibility to take photos results in a high amount of image data. However, modern cameras can not only be used to create pictures, but can record videos in good quality as well. What in former days was only possible with two different devices is now covered by a single device. Similar to image sharing on Facebook [80], people share videos on YouTube and other platforms. About five hours of video are uploaded on YouTube every second [2].

In most applications, the camera which is used to take images can produce video data by capturing multiple consecutive images, referred to as frames, per second. The objective of this thesis is to use this additional available frames from the past in order to create a better understanding of the current image. We want to predict a semantic segmentation and keep the prediction consistent over multiple frames in a video. This task is called frame-to-frame consistent semantic segmentation.

The idea behind frame-to-frame consistent semantic segmentation is to use information from previous frames to improve semantic segmentation of the current frame. Humans know that objects do not change position over time when the light conditions differ as long as they are not moved. However, algorithms which process each frame individually are prone to those changes. Despite illumination changes, other variances that typically occur when recording natural videos are occlusions, glaring problems, mirroring effects and objects visible from different angles. An example algorithm which shows weaknesses at giving a consistent prediction of the world is shown in Figure 1.2.

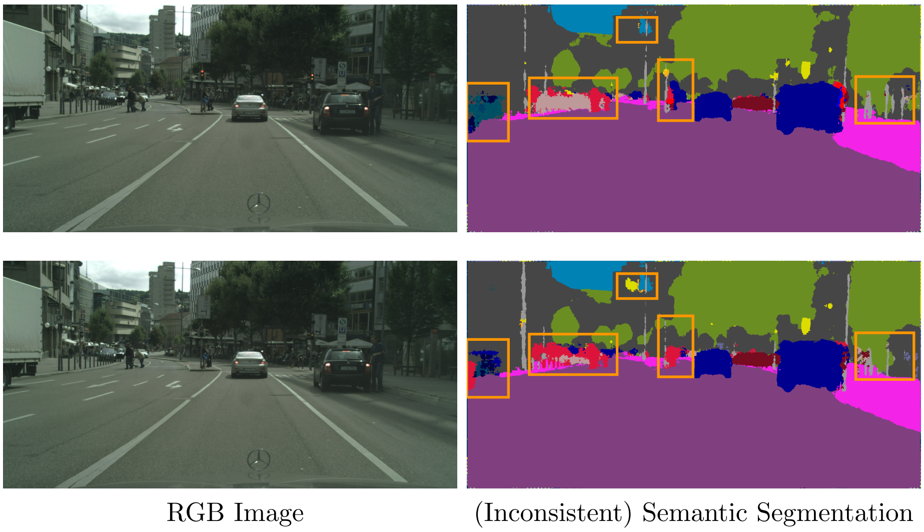


Figure 1.2: Inconsistent Semantic Segmentation. The state-of-the-art algorithm [66] has problems keeping a consistent view of the street scene. A few significant differences are highlighted by orange boxes.

In this thesis we want to utilize the behavior and movement of semantic objects in videos. The task of object tracking to get a consistent view of the world is complete in the human brain automatically. In computer vision, the information of multiple frames should assist the semantic segmentation prediction in order to give more consistent results and therefore get the correct understanding of the world.

1.3 Algorithm on Street Scenes

One of the most fascinating places to apply computer vision algorithms is on street scene data. Especially, videos from cities contain so much information that even human brains are completely overwhelmed. It takes us some time to identify all objects in a complex scene. Professional semantic segmentation annotators need about 90 minutes to label a single street scene image [13]. When designing a self driving car, the understanding of how street scenes need to be interpreted semantically is essential. The complexity of this task can be seen at the number of accidents caused by human drivers, because sometimes even the human brain makes misjudgments interpreting visual scenes.

However, machines have some advantages compared to humans. Computers can react much faster. Humans have a reaction time of about 200 milliseconds whereas a standard computer processor can perform one billion instructions in that same time interval. Nevertheless, computer algorithms needs to be correct and efficient in order to capitalize on the time advantage without the need of too much hardware resources.

The importance of true consistent information when considering street scenes is shown in Figure 1.2. The recognition of *e.g.* street boundaries or the difference between pedestrian and biker has a large impact when designing a system with the task of steering a car safely. The frame-to-frame consistent semantic segmentation algorithms developed in this thesis are tested on street scene data because street scenes contain a very high amount of information and therefore are challenging to interpret correctly.

To summarize, the objectives of this thesis is to better understand visual data by means of semantic segmentation. Since in many applications (*e.g.* autonomous vehicles [74]) video data is available, the information about the past will be used in order to predict the present. Because street scene data is very complex to interpret and substantial in the interesting field of autonomous driving, the developed algorithms are tested on this domain.

2 Machine Learning and Neural Networks

Contents

One of the methods to face scientific problems is through machine learning. It is has been used for many years in data science and computer vision [3, 5, 40]. Recently it became increasingly popular because the subfield of artificial neural networks achieved excellent results in various computer vision tasks [30, 50, 82]. Some reasons artificial neural networks beat many traditional image processing algorithms in certain benchmarks are the high amount of data available nowadays and rapid development in the hardware industry. Despite their success, artificial neural networks also have drawbacks compared to traditional approaches, because once established for a certain domain, they are limited to that domains and given scenarios.

The aim of this chapter is to explain the main principles of machine learning and neural networks in order be able to understand the basic building blocks of this thesis. Because the next Chapter 3, Semantic Segmentation and Video Processing already focuses on different types and architectures of neural networks, the following pages provide an introduction to the subject. Application and results of the methods explained are shown in Chapter 5.

2.1 Mathematical Notation and Conventions

In order to encourage mathematical understanding we introduce our notation. We use italic fonts for scalar values, *e.g.* x or x_i if we index an element i of vector $\mathbf{x}$. Matrices and vectors have bold font, *e.g.* $\mathbf{M}$ or $\mathbf{v}$. The space definition of a vector is written in

Entity	Notation
Scalar	x, x_i, M
Vector space N	$\mathbb{R}^N$
Vector in $\mathbb{R}^N$	$\mathbf{x} = (x_1, \ldots, x_N)^{\mathrm{T}}$
Matrix space $M \times N$	$\mathbb{R}^{M \times N}$
Matrix of dimension $\mathbb{R}^{M \times N}$	$\mathbf{M} = \begin{bmatrix} m_{1,1} & \cdots & m_{1,N} \\ \vdots & \ddots & \vdots \\ m_{M,1} & \cdots & m_{M,N} \end{bmatrix}$
ℓ_p norm of vector	$\|\|\mathbf{w}\|\|_p$
Cardinality of set $\mathbb{S}$	$\|\mathbb{S}\|$
Operation	**Notation**
Hadamard product	$\odot$
Convolution (in neural network)	$*$
Specific Naming	**Notation**
Set of target labels	$\mathbb{T}$
Set of target semantic labels	$\mathbb{S}$
Loss function	$\mathcal{L}$
Energy function	$\mathcal{E}$
Discrete probability distribution on random variable X	$p(X), q(X)$
Learning rate	η
Other hyper-parameters	λ
Cell state vector	$\mathcal{C}$ or $\mathbf{c}$
Hidden state vector	$\mathcal{H}$ or $\mathbf{h}$
Training data set	$\mathcal{X}$
Training target set	$\mathcal{T}$
Parameter of a model	Θ

Table 2.1: Mathematical Notation. In this table we summarize the notations used in this thesis.

double-lined upper case letters, *e.g.* $\mathbb{R}^3$ if the vector has three real valued entries. Table 2.1 shows a summary of our notation.

2.2 Machine Learning

The idea of machine learning is to use patterns, relationships and laws of data to optimize parameters of a model. To optimize the parameters, a machine learning algorithm is fed with example data from a certain domain where the source data distribution is unknown. Although the algorithm only sees example data it needs to generalize towards the real underlying distribution. This means that the machine learning algorithm needs to find a representation of the source domain which not only reflects the example data, but also new examples drawn from the same concept. An example is shown in Figure 2.1. The presented example shows a binary classification problem. It can be solved by a machine learning algorithm which fits a linear decision boundary.

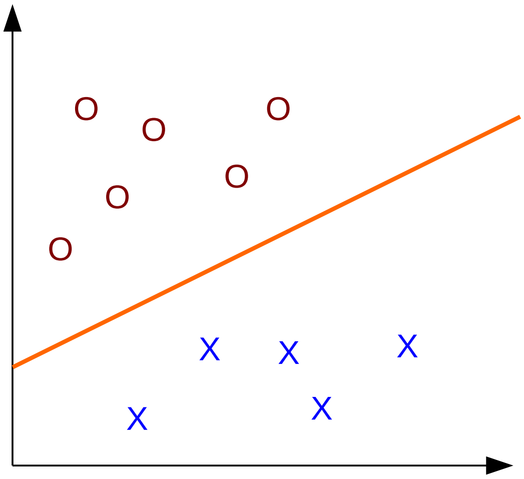


Figure 2.1: Machine Learning Example. We illustrate an example of a binary classification model. The decision boundary (orange) is trained to separate class “O” (red) from class “X” (blue).

In case of computer vision the task could be to classify whether a dog appears in an image. If we consider Figure 2.1, data points “X” represent images with a dog and data points “O” represent images without a dog. During the learning phase, many examples of images with and without a dog are given as an input to the model. This causes the algorithm to learn the concept behind a dog appearing inside an image. After the learning phase, the algorithm should be able to classify dogs which where previously unseen. Since this task is already complex a sophisticated machine learning method is required.

Other applications include financial market prediction [44], human language translation [84] and voice/text recognition [53]. In the following section three groups of machine learning methods are introduced.

2.2.1 Classification & Regression

Machine learning algorithms can be distinguished by what they are trying to predict. Tasks like semantic segmentation are classification problems since the final outputs are discrete labels. In this case a machine learning algorithm f maps any input $\mathbf{x}$ to a label $\hat{t} = f(\mathbf{x})$, where $\hat{t} \in \mathbb{T} = 0, \ldots, N-1$ and N represents the number of classes. If we consider the dog classification problem again, we would provide an input image $\mathbf{x}$ and the algorithm would return $\hat{t} = 1$ if a dog is found and $\hat{t} = 0$ otherwise. The dog classification example illustrates a binary classification problem which is a special case with $N = 2$ classes.

In contrast, regression models predict a continuous output vector $\hat{\mathbf{t}} \in \mathbb{R}^N$.An example would be to predict future sales growth of a company by providing micro and macro economic indicators as input data. However, the focus will be on classification problems in this thesis, because the task of semantic segmentation involves a classification problem for each pixel in an image. For each pixel it has to be decided to which semantic class the pixel belongs.

2.2.2 Discriminative Models & Generative Models

Machine learning models are grouped on their probability based assumptions. Classification algorithms can be created using discriminate and generative models. Discriminative approaches only consider mapping the training sample $\mathbf{X}^{(i)} \in \mathcal{X}$ to the target $t^{(i)} \in \mathcal{T}$ where $i = 1, \ldots, M$ for M training samples. This mapping is learned by considering the conditional probability $p(t^{(i)}|\mathbf{X}^{(i)})$ which is the posterior probability in Bayes' theorem

$$p(t^{(i)}|\mathbf{X}^{(i)}) = \frac{p(\mathbf{X}^{(i)}|t^{(i)}) \cdot p(t^{(i)})}{p(\mathbf{X}^{(i)})}. \tag{2.1}$$

Consequently, discriminative models learn decision boundaries between the target classes $\mathbb{T}$.

In difference to that, generative models also learn how the training data $\mathcal{X}$ is distributed by modeling the joint probability distribution $p(\mathbf{X}^{(i)}, t^{(i)})$, which are likelihood and class prior in Bayes' theorem. This allows generative models to create new data which is similar to the observed data besides classification. Both types of models use the conditional probability $p(t|\mathbf{x})$ to classify an unobserved input $\mathbf{x}$ to a new label $t \in \mathbb{T}$.

Two famous generative models are the more traditional Gaussian Mixture Model (GMM) and the Generative Adversarial Network (GAN) which was introduced recently by Goodfellow *et al.* [25]. Neural networks explained in Section 2.3 are discriminative models. Neural network models classify the output solely based on the input, not considering how the input is distributed. Other discriminative models are Logistic Regression and Support Vector Machine.

2.2.3 Supervised Learning & Unsupervised Learning

Machine learning algorithms can also be grouped by how they are trained. One possibility to perform image segmentation is through unsupervised learning. The segmentation is retrieved by only using the input data without any ground truth labels. Therefore, the algorithm needs to learn structures in data only by inspection of the input. One example of unsupervised learning is the clustering algorithm k-means [29] which can be used to segment images into foreground and background. Segmentation with k-means has been done by *e.g.* Dhanachandra *et al.* [15]. The advantage of unsupervised learning is that no human effort is necessary in order to create the ground truth for training examples. This is especially advantageous when considering a task like semantic segmentation where each pixel has to be assigned a category.

In contrast, supervised learning needs labeled training data which can be very expensive *e.g.* in case of semantic segmentation. However, if enough training data exists, supervised learning delivers good results within a specific domain, which can be seen in *e.g.* the Cityscapes benchmarks [13]. Fortunately, ground-truth data for the domain of street scene images is provided by the Cityscapes dataset and others [4, 24, 68]. This allows working on complex supervised learning tasks without labeling effort. One approach of summarizing the disciplines of machine learning introduced in the last three sections is shown in Figure 2.2.

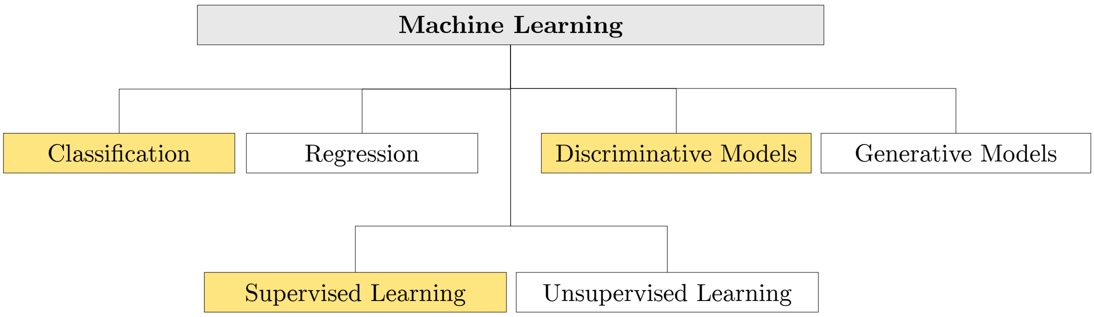


Figure 2.2: Disciplines of Machine Learning. Differentiation of machine learning methods into three groups. The parts which this thesis focuses on are highlighted in orange.

2.2.4 Training the Algorithm

Each learning algorithm f consists of parameters Θ which we want to learn from data. To perform training, an objective function which measures the error between the prediction $\hat{t}^{(i)} = f(\mathbf{x}^{(i)}; \Theta)$ of the current model and the target t has to be introduced. Using a quadratic loss for each training sample $\mathbf{x}^{(i)}$ the loss function $\mathcal{L}$ for N training samples is given by

$$\mathcal{L}(\Theta) := \sum_{i=1}^{N} (f(\mathbf{x}^{(i)}; \Theta) - t^{(i)})^2. \tag{2.2}$$

To train the algorithm we need to adjust the parameters to give better predictions. This is achieved by minimizing the loss function

$$\min_{\Theta} \mathcal{L}(\Theta) \tag{2.3}$$

After training, the learned parameters of the model are evaluated on a separate test data set. When evaluating the model on test data, one of two problems might be observable: The model might be underfitted or overfitted. An underfitted model produces high training and test error. The most probable reason is a too simple model for a complex problem. In contrast, low training and high test error are signs for overfitting. Possible reasons for this phenomenon are too many parameters inside the model and/or not enough training data. A graph which illustrates this problem is shown in Figure 2.3.

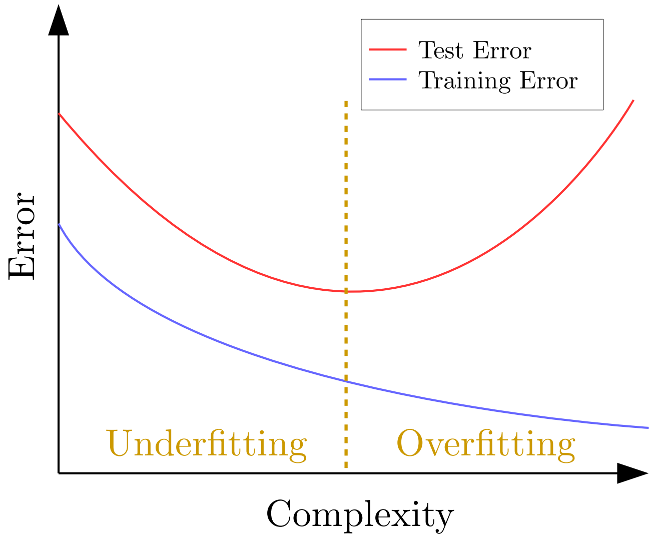


Figure 2.3: Underfitting/Overfitting. Problems which occur during training of a machine learning model are underfitting and overfitting. They are detected by comparing prediction errors of training and test data set. Underfitting might be caused by choosing a too simple model, whereas overfitting might be caused by a too complex model.

One possibility to avoid overfitting is to add a regularization term $\mathcal{R}(\Theta)$ to the cost function

$$\min_{\Theta} \mathcal{L}(\Theta) := \sum_{i=1}^{N} (f(\mathbf{x}^{(i)}; \Theta) - t^{(i)})^2 + \lambda \mathcal{R}(\Theta), \tag{2.4}$$

where the newly introduced hyper-parameter λ acts as a trade-off between and data fidelity and regularization. The task of the regularization term is to penalize complexity, *e.g.* by applying the ℓ_1-norm. In order to choose the hyper-parameter λ, data has to be split into a third part the validation set. On this validation set, several training runs are compared using different values for the hyper-parameter λ. The best value for λ is then selected for the final evaluation on the test set. This allows the model to be evaluated on the test set with all parameters fixed beforehand.

To summarize, the main goal of machine learning is to understand similarities and relations within data and to store this information in a model. In order to learn, the model must be trained using sample data set. Finally, the performance is evaluated using a test data set. Depending on the nature of the problem, different types of machine learning algorithms are more or less suitable. We will now focus on a specific machine learning model.

2.3 Neural Networks

Artificial neural networks (ANNs), referred to as neural networks (NNs) in this work, can be used to perform classification and regression tasks. Since the general machine learning principles mentioned in Section 2.2 apply, they are trained to give accurate predictions. To optimize the parameters of an ANN, a loss function measures derivations between prediction $\hat{\mathbf{t}}$ and target $\mathbf{t}$.

As the name suggests, ANNs are inspired by the human brain. The human brain consists of about 86 billion neurons [33] and over 100 trillion connections between neurons called synapses [20]. Artificial neural networks are not able to achieve these complexity so far because of computation constraints. Especially the large number of connections is impossible to model within a machine. One type of artificial neural network which tries to imitate the biological model are spiking neural networks introduced by Maas [64]. They attempt to remain closely related to the processes in human brains.

Most types of ANNs used for computer vision, speech recognition, medical diagnosis, etc. differ a lot from the biological model, while still using the most basic concepts. The next sections will focus on artificial neural networks.

2.3.1 Components of a Neural Network

To map the input of a neural network to a deterministic output, the data flows through neurons. The architecture of such a network decides how the neurons are located and connected. It can be represented using a directed graph, where the nodes correspond to neurons and directed edges model the connections. The smallest possible architecture would only contain a single neuron.

The most common example of how such a neuron works is given in Figure 2.4. Inputs $\mathbf{x}$ are multiplied with a weight vector $\mathbf{w}$ and a bias b is added. Weights and bias are called the parameters of the neuron. The result is passed through an activation function ϕ which then returns the output y of the neuron:

$$y = \phi(\mathbf{x}^T \mathbf{w} + b). \tag{2.5}$$

2.3.1.1 Activation Function

To produce a meaningful output the choice to the activation function is important. It can decide whether a neuron is active or not. To achieve this, the most simple method is to

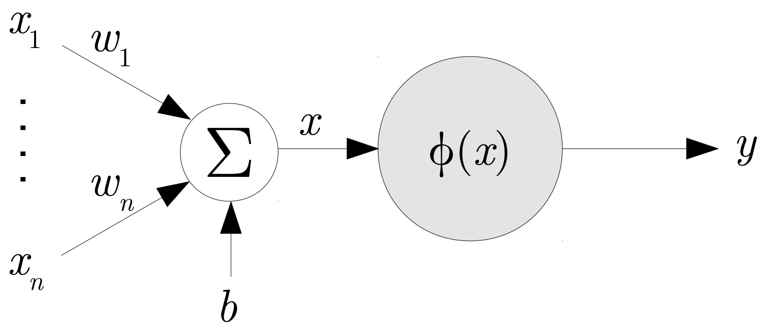


Figure 2.4: Example Neuron. The input vector is multiplied with the weight vector: $\mathbf{x}^T\mathbf{w}$. After the bias b is added, the result x is passed through an activation function $\phi(\cdot)$ to produce output y of the neuron.

use a step function which is activated if a certain threshold τ is reached. A network which only consists of such a single neuron is able to perform binary classification. It is termed *single-layer perceptron*. The activation function is given by

$$\phi(x) = \begin{cases} 0 & \text{for } x < \tau \\ 1 & \text{for } x \geq \tau \end{cases}. \tag{2.6}$$

The problem of the step function is that its gradient is zero almost everywhere. Thus, training with gradient methods is not possible. To overcome this issue, the sigmoid function provides a continuous approximation of the step function. It is continuous differentiable and the outputs remain between zero and one:

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

The tanh activation function is similar to the sigmoid function, however it creates outputs within the range from -1 to +1. This function allows data to remain zero-centered:

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

A very popular linear activation function is the Rectified linear unit (ReLU) [67]. It avoids the vanishing gradient problem of sigmoid and tanh in the positive domain by using a linear function. In the negative domain sparsity is ensured through deactivation of neurons. The ReLU is defined as

$$\phi(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}. \tag{2.9}$$

One variation of the ReLU is the Leaky rectified linear unit (LReLU) [63]. It introduces a small slope c in the negative domain:

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

where $c = 0.01$ is a common choice.

The Parametric rectified linear unit (PReLU) [31] is similar to the LReLU, except that the parameter α defining the slope, is learned during training:

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

The last important activation is the softmax function. It can be used at the output layer of a network to return probability values which are between zero and one. They also sum up to one, because softmax normalizes over all neurons at a layer:

$$\phi_i(x) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \text{ for } i = 1, \ldots, J, \tag{2.12}$$

where J is the number of neurons at that layer.

In conclusion, ReLU functions work well when considering complex problems with many consecutive neurons. The advantage over sigmoid and tanh is that gradients do not vanish during training [39]. However there are also applications for other activation functions. Sigmoid functions are used in recurrent neural networks explained in Chapter 3 where they act as one/off gates. The softmax activation function is often used at the last layer of a network to output probability values. The next section focuses on how multiple neurons are stacked together to create a network.

2.3.1.2 Layers of Neurons

Problems like image segmentation which are tackled by neural networks are complex in a sense that a single neuron cannot decide between multiple classes successfully. Therefore, multiple neurons are combined in a directed graph to allow more complex decision boundaries. Nodes at the same layer of the graph are grouped into a layer of the network. All root nodes establish the first layer (input layer). This is where input data is handed over to the network. The input to a network can either be raw data or preprocessed data termed features. On the other side of the network, leaf nodes represent the final output layer. The results of the output layer can either be used directly for predictions or they can also be processed further. Layers in between are called hidden layers.

One example of a fully connected network with three layers is shown in Figure 2.5. In most neural network architecture all directed edges point in the same direction namely from the input to the output. Consequently, no cycles are present. However, recurrent neural networks also contain neurons which have connections to themselves. They are discussed in detail in Chapter 3. If each neuron within a layer is connected to every other neuron of the previous layer, the layer is called fully connected. Similarly, if each neurons within the whole network is connected to every other neuron in the previous layer, the network is called fully connected.

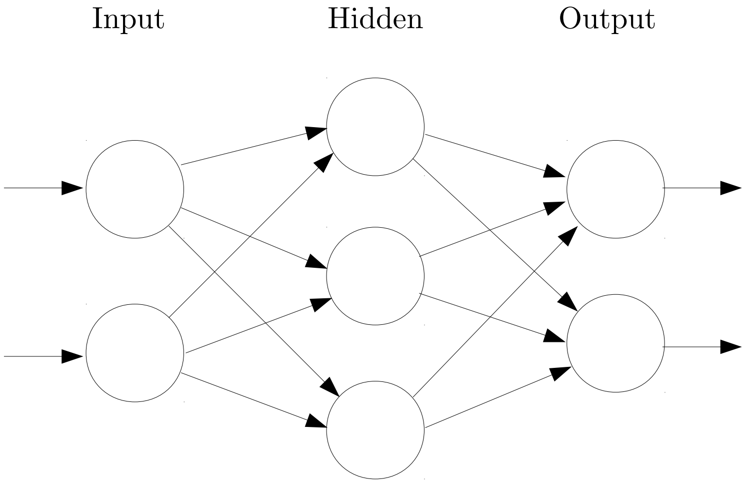


Figure 2.5: 3 Layer Network. This is an example network architecture consisting of three fully connected layers. Firstly, the left input layer (layer one) contains 2 neurons which process the input of the network. Secondly, the output of the activation function of layer one is forwarded to the three neurons in the hidden layer. Finally, the two output neurons of layer three generate the output of the network.

The design of network architectures highly depends on the application domain. In computer vision mostly convolutional neural networks (CNNs) (see Section 2.4) are used. Their architecture is characterized by sparse connections and a high number of neurons and layers. Training neural networks with many layers is also referred to as *Deep learning.*

2.3.2 Training a Neural Network

The most important part is the actual learning, where all weights and biases of the network are trained. This is achieved by assigning a loss which measures the difference between output and target. Since this loss function is not convex a good assignment of all parameters has to be found using non-convex optimization algorithms. Gradient-based methods back-propagate the loss in direction to the input which creates an update for every weight and bias in the network. One efficient optimization algorithm for this non-convex problem is stochastic gradient descent (SGD) which is applied to batches of the training data set iteratively.

To optimize the high dimensional problem, a large training data set is essential to successfully perform learning. Training with a too small data set will most likely result in learning data by hard, instead of learning patterns and relationships of the distribution from which the data is drawn from.

Choosing the loss function $\mathcal{L}$ depends on whether to face a regression or classification problem. When considering classification problems *e.g.* semantic segmentation, the cross-entropy function is often used. It treats input and target as distributions and penalizes differences exponentially. It is defined as

$$\mathcal{L}(p, q) = -\sum_{c=1}^{C} q(t_c) \log(p(t_c)), \tag{2.13}$$

where p is the distribution of the prediction, q is the distribution of the target and C is the number of classes.

The learning with SGD involves the chain rule for differentiation applied at each layer of the network. Let us consider a small binary classification example depicted in Figure 2.6. In this architecture we only use a single neuron. The loss is given by

$$\mathcal{L}(\mathbf{x}, t_0) = t_0 \log(\sigma(x_1 w_1 + x_2 w_2 + b)) + (1 - t_0)(1 - \log(\sigma(x_1 w_1 + x_2 w_2 + b))), \tag{2.14}$$

where (x_1, x_2) is a single 2D training example. The corresponding groundtruth label $t_0 = 1$ if the class is 0 and $t_0 = 0$ if the class is 1.

The loss of a weight w_i is given by the gradient

$$\frac{\partial \mathcal{L}}{\partial w_i} = \frac{\partial \mathcal{L}}{\partial \sigma} \frac{\partial \sigma}{\partial w_i} \tag{2.15}$$

$$= t_0 (\frac{1}{\sigma(x_1 w_1 + x_2 w_2 + b)}) \frac{\partial \sigma(x_1 w_1 + x_2 w_2 + b)}{\partial w_i} \tag{2.16}$$

$$- \quad (1 - t_0)(\frac{1}{\sigma(x_1 w_1 + x_2 w_2 + b)}) \frac{\partial \sigma(x_1 w_1 + x_2 w_2 + b)}{\partial w_i} \tag{2.17}$$

$$= t_0 (1 - \sigma(x_1 w_1 + x_2 w_2 + b)) x_i - (1 - t_0)(1 - \sigma(x_1 w_1 + x_2 w_2 + b)) x_i \tag{2.18}$$

$$= (2t_0 - 1)(1 - \sigma(x_1 w_1 + x_2 w_2 + b)) x_i. \tag{2.19}$$

Similarly, gradients for deeper networks can be computed using the chain rule.

During the training of neural networks with gradient descent, parameters are not updated with the full magnitude, but only with a small fraction. The reason is that problems might occur in non-convex optimization *i.e.*, nummerical problems, oscillation

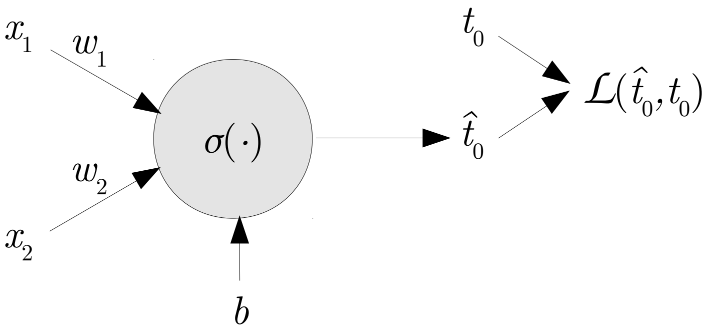


Figure 2.6: Training a single neuron. The loss function $\mathcal{L}$ is applied to the prediction $\hat{t}_0$ of a single neuron using the ground-truth label t_0.

and saturation at saddle points. To avoid these problems, a learning rate $\eta \in (0, 1)$ is introduced as a hyper parameter. The neural network optimizer Adam developed by Kingma *et al.* [45] uses additional hyper parameters to support fast training. The optimizer calculates an exponential moving average of the gradient and the quadratic gradient over the last iterations. This momentum is applied as a weight update. Adam allows for faster training than Adagrad [21] or SGD with momentum of many convolutional neural networks [45]. Therefore, Adam is used as optimizer in Chapter 5.

To summarize, the smallest unit in a neural network is a neuron. It takes an input vector and outputs a scalar value. Within a neuron, the input vector is multiplied element wise with a weight vector, which is a linear operation. Then, the sum of this multiplication together with a bias are passed to an activation function, which represent the non-linear part, to produce the output of the neuron.

Multiple neurons are stacked together in a neural network to learn complex patterns. These networks are structured in layers where a layer is composed of neurons with the same distance to the input.

The actual learning occurs when training the network using example data. An objective function needs to evaluate the deviation of the network prediction from the desired output. The loss measured between prediction and target is back-propagated to every parameter of the network in order to improve prediction accuracy iteratively. The next section focuses on a special linear operation which is used to tackle tasks in computer vision.

2.4 Convolutional Neural Networks

To give an overview, Convolutional neural networks (CNNs) are an architecture of neural network which apply the convolution operation on some layers. When used for complex applications, they are characterized by many layers but few connections. The main applications include imaging problems such as *e.g.* image classification [50], semantic segmentation [56] and human pose estimation [83]. Their network structure is hierarchical, because the graph representing the CNN is directed from the input to the output layer. Lower layers which are close to the input of the network, retrieve information about colors and gradients in an image. Higher layers operate close to the output and learn complex structures like shapes of objects.

CNNs became popular in 2012 when they first beat traditional approaches on the ImageNet classification challenge introduced by Russakovsky *et al.* [75]. The winning CNN was AlexNet by Krizhevsky *et al.* [50] with eight layers and over 600,000 neurons. Although CNNs where already used in 1998 by LeCun *et al.* [53] for handwritten character recognition, they were not able perform image classification at that time. The advances in computational power and availability of larger data sets enabled them to deliver good results in recent years.

The following sections focus on how different types convolution are applied to images and how different channels can be used to transfer information from lower layers to the output of the network. Furthermore, the receptive field of a neuron at a given layer will be discussed. These operations are the basis of the experiments conducted in Chapter 5.

2.4.1 The Convolution Operation

The basic operation of a CNN is the discrete convolution operation. Let us consider a grayscale image $\mathbf{X} \in \mathbb{R}^{M \times N}$ which we convolve with a filter $\mathbf{W} \in \mathbb{R}^{P \times Q}$. Then, the pre-activated output $\mathbf{A} \in \mathbb{R}^{U \times V}$ has the dimension U × V, where

$$U = M - P + 1 \text{ and} \tag{2.20}$$

$$V = N - Q + 1. \tag{2.21}$$

The 2D convolution is given by

$$a_{u,v} = (\mathbf{X} * \mathbf{W})_{u,v} = \sum_{p=1}^{P} \sum_{q=1}^{Q} x_{u+p-1,v+q-1} w_{p,q}, \tag{2.22}$$

where $u = 1, \ldots, U$ and $v = 1, \ldots, V$.

Similarly to the fully connected layer, the convolution can also be calculated using the matrix-vector product $\mathbf{W}'\mathbf{x}' = \mathbf{a}'$. Therefore we must reshape and reorder as follows:

$\mathbf{x}' \in \mathbb{R}^{MN}$, $\mathbf{W}' \in \mathbb{R}^{UV \times MN}$ and $\mathbf{a}' \in \mathbb{R}^{UV}$. The matrix multiplication is now given by

$$\begin{bmatrix}
\mathbf{w}_1^T, \overbrace{0\cdots0}^{N-Q} & \mathbf{w}_2^T, \overbrace{0\cdots0}^{N-Q} & \cdots & \mathbf{w}_P^T, \overbrace{0\cdots0}^{N-Q} & 0\cdots & \overbrace{\cdots0}^{(M-P)N} \\
0, \mathbf{w}_1^T, 0\cdots0 & 0, \mathbf{w}_2^T, 0\cdots0 & \cdots & 0, \mathbf{w}_P^T, 0\cdots0 & 0\cdots & \cdots0 \\
\vdots & & & & & \\
0\cdots0, \mathbf{w}_1^T & 0\cdots0, \mathbf{w}_2^T & \cdots & 0\cdots0, \mathbf{w}_P^T & 0\cdots & \cdots0 \\
\overbrace{0\cdots0}^{N} & \mathbf{w}_1^T, 0\cdots0 & \mathbf{w}_2^T, 0\cdots0 & \cdots & \mathbf{w}_P^T, 0\cdots0 & \overbrace{\cdots0}^{(M-P)N-N} \\
\vdots & & & & & \\
0\cdots0 & 0\cdots0, \mathbf{w}_1^T & 0\cdots0, \mathbf{w}_2^T & \cdots & 0\cdots0, \mathbf{w}_P^T, & \cdots0 \\
\vdots & & & & & \\
\vdots & & & & & \\
\overbrace{0\cdots}^{(M-P)N} & \cdots0 & \mathbf{w}_1^T, 0\cdots0 & \mathbf{w}_2^T, 0\cdots0 & \cdots & \mathbf{w}_P^T, 0\cdots0 \\
\vdots & & & & & \\
0\cdots & \cdots0 & 0\cdots0, \mathbf{w}_1^T & 0\cdots0, \mathbf{w}_2^T & \cdots & 0\cdots0, \mathbf{w}_P^T
\end{bmatrix}
\begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_M \end{pmatrix}
=
\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_V \\ a_{V+1} \\ \vdots \\ a_{2V} \\ \vdots \\ \vdots \\ a_{((U-1)V)+1} \\ \vdots \\ a_{UV} \end{pmatrix},$$

where $\mathbf{x}_m$ is row m of image $\mathbf{X}$ and $\mathbf{w}_p$ is row p of filter $\mathbf{W}$ both represented as column vectors. The sparse matrix $\mathbf{W}'$ indicates that the model has few connections. Furthermore, the weights are shared across neurons. Both aspects allow an efficient implementation of the convolution operation.

Since RGB images provide much more information than grayscale images, we can also extent definition of the convolution to multiple-channel inputs C. This is achieved by the neuron having separate weights for each channel and summation over all channels:

$$a_{u,v} = (\mathbf{X} * \mathbf{W})_{u,v} = \sum_{c=1}^{C} \sum_{p=1}^{P} \sum_{q=1}^{Q} x_{u+p-1,v+q-1,c} w_{p,q,c}, \tag{2.23}$$

where $\mathbf{X} \in \mathbb{R}^{M,N,C}$ and $\mathbf{W} \in \mathbb{R}^{P,Q,C}$. Changes compared to the definition Equation (2.22) are highlighted in green. An example of a 5 × 5 × 3 convolution with a 64 × 64 × 3 input is shown in Figure 2.7.

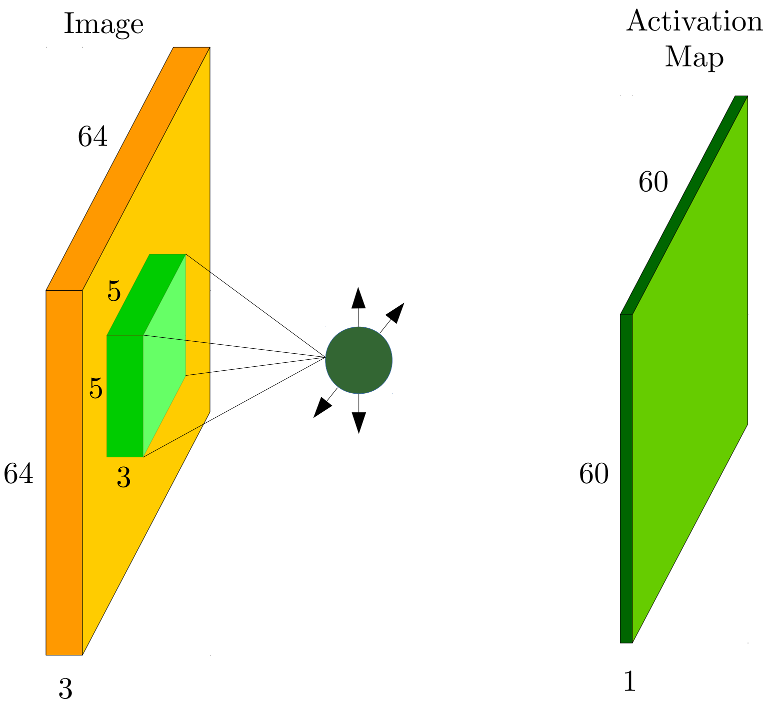


Figure 2.7: Convolution Operation. A $64 \times 64 \times 3$ dimensional input image (orange) is convolved with a $5 \times 5 \times 3$ filter (light green). The filter is moved to every possible location within the image. Results of the convolution at each position are stored in a $60 \times 60 \times 1$ dimensional output, which is called "activation map" (dark green). The receptive field of every neuron is 5×5, this means that each entry in the activation map is affected by 5×5 pixels in the input image.

2.4.2 The Convolutional Layer

The activation map computed by the convolution operation, is processed further within a convolutional layer. After the convolution is computed for every neuron, a bias b is added to influence the firing behavior of the neuron. Then, the activation function $\phi(\cdot)$ appends the non-linearity to allow learning of complex data dependencies through multiple layers of neurons. The output y of neuron (u, v) is given by

$$y_{u,v} = \phi(a_{u,v} + b). \tag{2.24}$$

As we have already seen in Equation (2.22) a single neuron in a CNN is only connected to $P \times Q$ pixels in the input channel. This causes the neuron to search for local patterns in the image. Sharing weights across the spatial dimensions M, N enforces searching the same pattern in the whole input channel.

However, it is essential to search for more patterns in the image to successfully face complex computer vision problems. This can be achieved through creating multiple output channels. Each output channel $d \in \{1, \dots, D\}$ is capable of extracting a different feature because it has its own weights and bias. The pre-activation is given by

$$a_{u,v,d} = (\mathbf{X} * \mathbf{W})_{u,v,d} = \sum_{c=1}^{C} \sum_{p=1}^{P} \sum_{q=1}^{Q} x_{u+p-1,v+q-1,c} w_{p,q,c,d} \tag{2.25}$$

and the output of the layer is

$$y_{u,v,d} = \phi(a_{u,v,d} + b_d). \tag{2.26}$$

Changes compared to the previous convolution operation are highlighted in green.

The previously introduced concept of output channels adds complexity to the convolutional layer. We want to compute the degree of complexity within a layer. We measure the degree of complexity in terms of learn-able parameters and floating point operations which have to be computed.

Firstly, to get the number of parameters we need to consider weights and biases. A 2D convolution layer consists of $C \cdot D \cdot P \cdot Q$ weights and D biases, consequently,

$$L_{\text{params}} = C \cdot D \cdot P \cdot Q + D \tag{2.27}$$

parameters in total. An important aspect to consider is that the number of parameters does not depend on the spatial dimensions of the input image. This shows that a convolutional layer can be applied to images with different resolutions without changing the architecture.

Secondly, the computation complexity is measured in multiply-add (MADD) operations. 1 MADD consists of a multiplication between two scalars followed by an addition which are two floating point operations. Consequently, a $n \times n$ vector product needs n MADDs. The exact number of $n-1$ additions does not affect the metric. The total number of MADDs for the linear part of one layer is

$$L_{\text{MADDs}} = C \cdot D \cdot P \cdot Q \cdot (M - P + 1) \cdot (N - Q + 1). \tag{2.28}$$

One aspect that has to be considered, especially when computing the semantic segmentation of an image, is that predictions are made for every pixel. Such a dense prediction task becomes very difficult, because a convolution operation reduces the input resolution $M \times N$ to $(M - P + 1) \times (N - Q + 1)$. To overcome this issue, a padding of size $P - 1$ and $Q - 1$ is appended to the spatial dimensions (width and height of an image). This padding can be created by *e.g.* reflecting the values at the image border, copying the boarder values or adding zeros.

This concludes the main computation capabilities of a single layer within a convolution neural network. Nevertheless, there are variations of convolution operations which will be discussed in the next section.

2.4.3 Different Types of Convolutions

In the previous section a standard 2D convolution was explained. However, there are also other types of convolutions which might be useful depending on the desired output of a layer and the problem. Different convolution operations can change the spatial dimensions. On the one hand a convolution with a stride larger than one and pooling reduce spatial output dimensions, on the other hand transposed convolutions are used to upscale images. Depthwise separable convolutions decrease to computation time and 3D convolutions apply the 2D concept to the third dimension. Dilated convolutions are used to increase the receptive field. The most important operations used in this work are stated in this section. We highlight the changes compared to the standard convolutions defined in Section 2.4.1 in green.

Strided Convolution A stride of size S defines the step-size of the filter moving over an image. It is introduced to reduce spatial dimensions:

$$a_{u,v} = \sum_{p=1}^{P} \sum_{q=1}^{Q} x_{S(u-1)+p, S(v-1)+q} w_{p,q}, \tag{2.29}$$

where the output resolution reduces to $U = \lfloor \frac{M-P}{S} \rfloor + 1$ times $V = \lfloor \frac{N-Q}{S} \rfloor + 1$.

Pooling Layer Another technique to reduce spatial dimensions are pooling layers. Pooling layers are often inserted between convolution layers. A max-pooling layer uses the $\max(\cdot)$ function to find the highest activation of the previous convolution layer in a predefined region:

$$y_{u,v} = \max_{p=0 \ldots P-1,\ q=0 \ldots Q-1} x_{u+p,v+q}, \tag{2.30}$$

where the size of the predefined region is $P \times Q$. Note that no bias and no activation function is applied. Furthermore, no parameters are needed. A stride can also be added to the pooling operation.

Transposed Convolution Pooling layers and strided convolutions results in lower spatial dimensions. However, to semantically segment an image, we need pixel-wise predictions. If strided convolutions or pooling layers are used, up-sampling is needed to recover the original spatial image dimensions. This can be achieved through learning the parameters of a transposed convolution. In Section 2.4.1 we saw that the convolution can be expressed using the matrix-vector product $\mathbf{W}'\mathbf{x}' = \mathbf{a}'$. As the name suggest, the transposed convolution is applied by transposing the weight matrix: $\mathbf{W}'^T\mathbf{x}' = \mathbf{a}'$. Consequently, the output $\mathbf{a}'$ has increased in dimension with respect to the input: $U = M + P - 1$ times $V = N + Q - 1$.

Depthwise Separable Convolution Another important concept are depthwise convolutions [35]. Their filters operate on each channel individually and therefore save computation time. The number of parameters decreases to $C \cdot P \cdot Q + D$, where C and D are the number of input resp. output channels. However, the constraint $C = D$ has to be fulfilled. The depthwise convolution is defined as

$$a_{u,v,c} = (\mathbf{X} * \mathbf{W})_{u,v,c} = \sum_{p=1}^{P}\sum_{q=1}^{Q} x_{u+p-1,v+q-1,c} w_{p,q,c}. \tag{2.31}$$

A depthwise separable convolution [11] consist of a depthwise convolution followed by a pointwise convolution. In the first part, the depthwise convolution creates C output channels. In the second part a convolution with filter size 1×1 combines the C channels into a single one. The depthwise separable convolution is defined as

$$a_{u,v} = (\mathbf{X} * \mathbf{W}^{(1)} * \mathbf{W}^{(2)})_{u,v} = \sum_{c=1}^{C}\left(\sum_{p=1}^{P}\sum_{q=1}^{Q} x_{u+p-1,v+q-1,c} {w^{(1)}}_{p,q,c}\right) {w^{(2)}}_{1,1,c}, \tag{2.32}$$

where $\mathbf{W}^{(1)}$ is the depthwise filter and $\mathbf{W}^{(2)}$ is the pointwise filter. The number of multiply-add computations drops to

$$C \cdot P \cdot Q \cdot (M - P + 1) \cdot (N - Q + 1) \;+\; C \cdot D \cdot (M - P + 1) \cdot (N - Q + 1) \tag{2.33}$$

compared to the standard convolution shown in Equation (2.28). Similarly, the number of parameters is reduced to

$$C \cdot P \cdot Q + C \;+\; C \cdot D + D. \tag{2.34}$$

3D Convolution All previously introduced convolution variants operate on two-dimensional inputs. In applications *e.g.* video processing an additional temporal dimension has to be considered. A 3D convolution processes input $\mathbf{X} \in \mathbb{R}^{M \times N \times O}$ with filter $\mathbf{W} \in \mathbb{R}^{P \times Q \times R}$ to produce output $\mathbf{A} \in \mathbb{R}^{U \times V \times T}$:

$$a_{u,v,t} = (\mathbf{X} * \mathbf{W})_{u,v,t} = \sum_{p=1}^{P}\sum_{q=1}^{Q}\sum_{r=1}^{R} x_{u+p-1,\; v+q-1,\; t+r-1}\; w_{p,q,r}. \tag{2.35}$$

This concept also generalizes for higher dimensional convolutions. However, it should be noted that 3D Convolutions are computational expensive.

Dilated Convolution Dilated convolutions [86] are popular at dense prediction tasks, because they enlarge the receptive field (Section 2.4.4). When used together with padding they also keep the spatial dimensions constant. The operation is defined as

$$a_{u,v} = (\mathbf{X} * \mathbf{W})_{u,v} = \sum_{p=1}^{P} \sum_{q=1}^{Q} x_{u+\rho(p-1),\ v+\rho(q-1)} w_{p,q}, \tag{2.36}$$

where ρ is the rate of dilation. Dilated convolutions are important for semantic segmentation because they enlarge the receptive field while reducing the number of parameters.

2.4.4 Multiple Convolutional Layers in a CNN

After having introduced the most relevant types of convolutions for this thesis, we want to mention how they operate within a CNN. Since the network is structured in layers, each layer can perform a different convolution operation. When stacking layers together, the number of channels at the input of each layer has to fit the number of channels at the output of the previous layer. One desirable feature of CNNs is that if every layer performs a convolution, then changing the spatial dimensions of the input does not affect the network architecture. This is very useful, because images of different resolution can be in a single data set. A network architecture which only consists of convolutional layers is called fully convolutional.

Receptive Field An important metric when deciding how many layers an architecture should at least have to tackle a certain problem is the size of the receptive field. The receptive field is the region of pixels in the input image which affect the output of a neuron at a given layer. An example is shown in Figure 2.8. After the first layer performing a $P^{(1)} \times Q^{(1)}$ convolution the size of the receptive field is $P^{(1)} \times Q^{(1)}$, because each neuron sees that many input pixels. If the second layer performs a $P^{(2)} \times Q^{(2)}$ convolution the size of the receptive field grows to $(P^{(1)} + (P^{(2)} - 1)) \times (Q^{(1)} + (Q^{(2)} - 1))$. The recursive formula to calculate the size of the receptive field at layer i is given by

$$\mathbf{z}^1 = (P^{(1)}, Q^{(1)})^T, \tag{2.37}$$

$$\mathbf{z}^i = \mathbf{z}^{i-1} + (P^{(i)} - 1, Q^{(i)} - 1)^T \cdot S^{(i-1)}, \quad \forall i > 1, \tag{2.38}$$

where $P^{(i)}$ and $Q^{(i)}$ represent the filter size with stride $S^{(i)}$ of the convolution at layer i. For example, when designing an image classification architecture, the receptive field of the output layer should be at least the size of the input image to be able to make a prediction considering the input image as a whole.

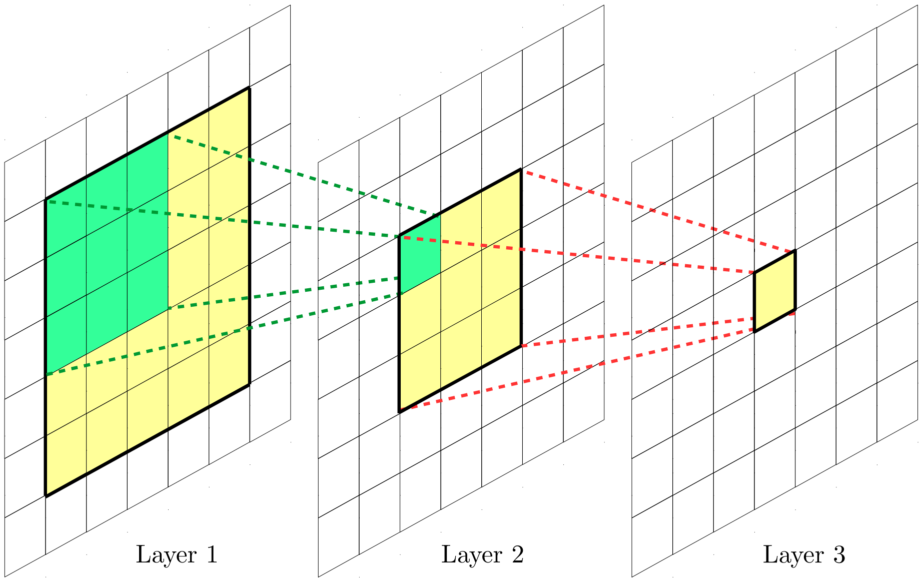


Figure 2.8: Receptive Field. This example shows a three layer architecture with convolution operations of filter size 3 × 3 at each layer. One pixel at layer 3 (yellow pixel) has a receptive field of size 7 × 7 on the input at layer 1 (yellow and green pixels).

Batch Normalization Depending on how many layers and which activation functions are used, the input for higher layers might be in an undesired range. This can cause gradient saturation during training and slow convergence. To overcome this issue batch normalization introduced by Ioffe *et al.* [37] is applied between layers. Batch normalization transforms the input batch of a layer to have the desired mean and variance:

$$y_i = \frac{x_i - \mu}{\sigma}, \tag{2.39}$$

where μ is the mean, σ^2 is the variance and $i = 1, \ldots, I$ is the size of the batch. Batch normalization can be seen as another layer of a network although it does not compute a convolution.

To summarize, this chapter focuses on how machine learning and neural networks are used to face computer vision problems. All machine learning algorithms try to find patterns and relationships in data. When semantically segmenting images, the algorithm needs to learn that *e.g.* dogs inside an image can be detected by a certain pattern. The idea of machine learning is to show training data to the algorithm in order to learn the patterns. Unfortunately, the amount of training data needed to learn complex structures is very large. During training the parameters of the algorithm are tuned. The performance is evaluated on a separate test dataset which was not previously shown to the model. Hyper-parameters are selected using a third validation dataset.

Artificial neural networks are discriminative models which provide a framework for machine learning algorithms. They are well suited for semantic segmentation because of their ability to tackle classification problems. The smallest unit, an artificial neuron, consists of weights, bias and an activation function. Weights transform the input data linearly, while the activation function transform non-linearly. Neurons are organized in layers which connect to each other. The use of multiple neurons and layers allow the network to learn complex decision boundaries between semantically different classes such as *e.g.* dogs vs cats.

Before training, all parameters of the network are initialized. Training requires evaluation of performance through target outputs and an objective function. The gap between the current output of the network and the target output is reduced through changing the parameters. This non-convex optimization problem is faced with *e.g.* the gradient descent algorithm, which involves back-propagating first order derivatives.

A method to efficiently apply the artificial neural networks to computer vision problems is through CNNs. They are characterized by many layers of neurons and few connections between them. Layers close to the input image extract low-level features *e.g.* color and gradient while layers close to the output search for complex structures such as *e.g.* shapes of objects.

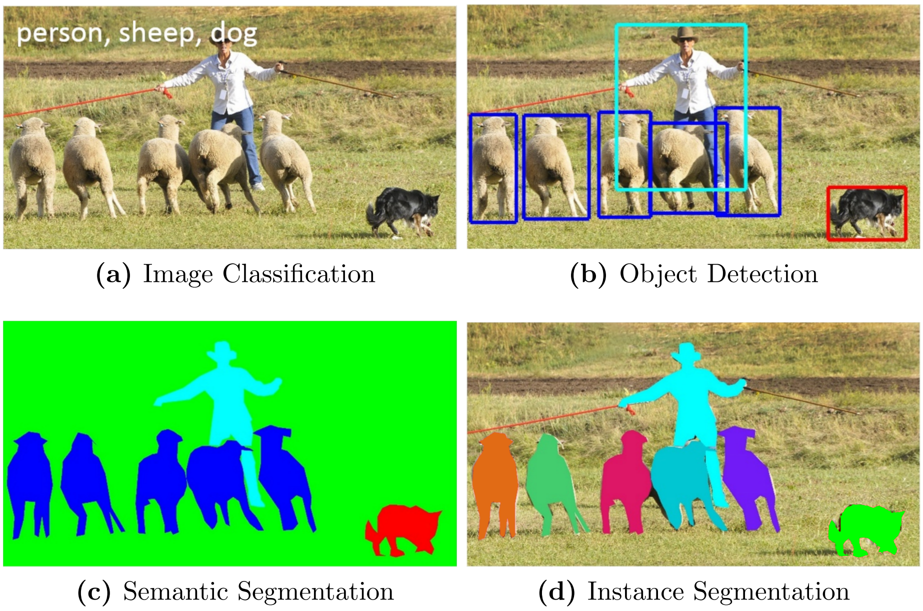
In the next Chapter 3, CNN architectures of recent work will be analyzed in detail, while focusing on the task of consistent semantic segmentation over multiple input frames.

3 Semantic Segmentation and Video Processing

Contents

In this chapter we want to examine how frame-to-frame consistent semantic segmentation can be achieved, by studying related work in this field. First, we focus on different approaches for semantic image segmentation. Second, we discuss the problems which arise when processing videos. Then, we need to look at combined methods to accomplish the task of consistent video prediction. When examining combined methods, special focus is given to recurrent neural networks. Finally, suitable street scene data sets are compared.

The main tasks of image understanding include image classification, object recognition, semantic segmentation and instance segmentation. They are compared in Figure 3.1. These tasks operate at different level of detail. Image classification algorithms assign an image as a whole to a category *e.g.* apple, bus, clown. The task of object recognition, searches for all objects inside an image and classifies all of them into a category *e.g.* an image with two sheep and a person will be recognized as three objects. Semantic segmentation algorithms assign each pixel in an image to a class as shown in Figure 1.1. Compared to the two previously mentioned problems, this task is more complex because object boundaries must be computed on pixel-level accuracy. In addition, instance segmentation distinguishes multiple objects if they correspond to the same class, which is the most detailed task. In this thesis we are especially interested in the semantic segmentation of images, although all of the problems are closely related.



(a) Image Classification **(b)** Object Detection

(c) Semantic Segmentation **(d)** Instance Segmentation

Figure 3.1: Disciplines of Image Understanding. The main disciplines of image understanding compared on the same RGB image: Image Classification (a), Object Recognition (b), Semantic Segmentation (c) and Instance Segmentation (d). The images are taken from [87]

3.1 Semantic Image Segmentation

As the name suggests, semantic segmentation combines two tasks. One of them is to segment an image into different parts, the other gives a semantic meaning to each part. In this section, we first focus on each task individually, before discussing combined work in semantic segmentation.

3.1.1 Segmentation

A very intuitive solution to the image segmentation problems is by using methods of graph partitioning. Pixels are represented by nodes and edges are drawn across neighboring pixels. The edges have weights assigned where values correspond to the intensity changes between pixels. One possibility is to maximize the normalized cut criterion [77], which measures the dissimilarity between different partitions in a graph. Other possibilities include the use of clustering algorithms *e.g.* k-means, partial differential equations and edge detection.

3.1.2 Semantics

The second task gives a semantic meaning to each of the image segments. This classification problem can be evaluated using different metrics. The prediction accuracy computes the ratio between correctly segmented pixels and total amount of image pixels:

$$\mathrm{Acc} = \frac{1}{MN} \sum_{m=1}^{M} \sum_{n=1}^{N} \delta(p'_{m,n}, s_{m,n}), \tag{3.1}$$

where $\mathbf{P}' \in \mathbb{N}^{M \times N}$ is the prediction, $\mathbf{S} \in \mathbb{N}^{M \times N}$ is the ground-truth and δ represents the Kronecker delta.

A more sophisticated metric is the mean Intersection over Union (mIoU). For each class, the Intersection over Union is given by:

$$\mathrm{IoU} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}, \tag{3.2}$$

where TP refers to true positive, FP to false positive and FN to false negative classified pixels of a specific class. The mIoU is given by computing the average over all classes. This metric ensures that each class is weighted equally, regardless of the total number of pixels for the class. Mean IoU is a good metric for street scenes, because it treats smaller semantic classes *e.g.* street signs, traffic lights, poles, bikers which are very important understanding traffic situations equal to larger classes *e.g.* sky, street, vegetation.

When combining the tasks, semantic segmentation can be computed through Conditional Random Fields (CRFs). A method which considers a fully connected CRF is introduced by Krähenbühl and Koltun [49].

3.1.3 Semantic Segmentation with CNNs

As already introduced in Chapter 2, semantic segmentation can be computed with CNNs. They outperform traditional methods on current benchmarks [13, 24]. However, they can easily be manipulated as shown in [81]. An easy to implement attack is *e.g.* introduced by Goodfellow *et al.* [26].

Another discipline which is to design architectures for CNNs. This problem can be seen as an art, because there are no mathematical principles which proof that one architecture is superior for a specific application. One approach to find a good architecture is by random testing as shown in [48]. To some extend the architecture can also be explained by the size of the receptive field as suggested in Section 2.4.4.

Most architectures are designed for the image classification problem, which means they are not capable of performing dense prediction. The compact high level feature representation can be used for dense prediction with a decoder module. The original spatial dimensions can be up-sampled by transposed convolutions and pooling layers [56].

Another possibility to keep spatial dimensions through several layers is by using padding and dilated convolutions Equation (2.36). This method is important for semantic segmentation because it enlarges the receptive field while reducing the number of parameters. Example architectures which use this technique are DeepLab [11] and

ESPNet [66]. While the DeepLab architecture has a large number of parameters and over 70 layers, the ESPNet is more efficient.

3.2 Video Processing

When considering multiple frames in a video, different computer vision problems need to be tackled to understand the time dependencies and the movement of pixels corresponding to semantic regions. The advantage of having multiple frames in a short time period (*e.g.* 30 FPS), is that objects only move slightly and correspondences between consecutive images are found easier.

Three important motion analysis disciplines include egomotion, tracking and optical flow. The task of egomotion deals with the estimation of the camera movement from one frame to another. To give one example, egomotion tries to estimate the positions of a moving automobile by considering a video sequence recorded with a camera mounted on the vehicle. Egomotion consists of rotation and translation of the camera between two images. In a 3D scenario this problem can be faced by eight point correspondences. These point correspondences can be found by traditional feature extractors (*e.g.* Harris corners [28], FAST [73]) in combination with descriptors (*e.g.* SIFT [57], BRIEF [6]) and a robust matcher.

The task of video tracking aims for following moving objects through multiple frames. In contrast to egomotion, it consists of two parts. First, the target object needs to be identified and described properly. Then, the descriptors need to me matched through consecutive frames in order to track an object (with *e.g.* a Kalman filter [42]).

The most difficult task is to compute the optical flow between two images. It describes the motion of every pixel from one image to another image, with a motion vector. This problem becomes very hard for images with many homogeneous regions. For instance, it is impossible to detect motion between two consecutive images in which all pixels have the same color. Objects need to have some texture for the flow algorithm to find correspondences and compute motion. Furthermore, pixels which are hidden behind an object in one image (occluded pixels), but then visible in another image, cannot be estimated because of missing correspondence.

3.2.1 Optical Flow

The task of optical flow tackles the problem of how pixels move from one frame to another. Under the assumption that brightness remains constant,

$$I(\mathbf{p}, t) = I(\mathbf{p} + \mathbf{\Delta p}, t + \Delta t), \tag{3.3}$$

where the function $I(\mathbf{p}, t)$ return the intensity value of pixel $\mathbf{p}$ at time t. The problem of finding the motion vector for each pixel can be tackled with various optimization methods (*e.g.* Lucas-Kanade [60], total variation methods introduced by Camille Jordan).

The advantage of having optical flow information can be beneficial, because semantic segmentation should be equal for consecutive frames, after reversing the motion transformation for each pixel.

3.2.2 Frame-to-Frame Consistency

A frame-to-frame consistent solution is important for every task of image understanding. One approach towards consistent semantic segmentation is with statistical models. Markov Random Fields [54] are used for image segmentation by modeling the joint probabilities between image features and semantic classes. They can also be extended to predict consistent semantic segmentation [8, 14, 61] by introducing temporal links [7].

Similarly, Conditional Random Field (CRF) [52] compute semantic segmentation by modeling the conditional probability between features and labels. The differences between discriminative and generative models are presented in Section 2.2.2. A 3D CRF can enforce consistency in a video setting. Kundu *et al.* [51] establish consistency with a 3D CRF as a post-processing step.

Another possibility to achieve consistency is with graph-based methods. In the 2D case, the nodes of the graphs are the pixels and the edges connect neighboring pixels in the image. When adding video data, the graph becomes three dimensional and voxels have edges into the temporal dimension. Similar pixels are grouped into superpixels [23] to reduce complexity of the graph and save computation time. Different graph partitioning algorithms (*e.g.* [77]) can be used to compute consistent video segmentation. A learned approach to find a good structure of the graph is introduced by Khoreva *et al.* [43]. Strategies to minimize energy in such a graph include hierarchical abstraction [38]. Optimization is computed on a course-to-fine level to compute dense video segmentation.

Consistency between multiple frames can also be enforced with neural networks. Consistent object detection is achieved in by combining the tasks of object detection and object tracking through time [19]. In recent work the two task are tackled with recurrent neural networks [22, 58] explained in Section 3.3. Consistent semantic segmentation takes more effort to compute than object detection, because large spatial dimension are needed throughout the neural network architecture. End-to-end training of a CNN which propagates features through time is shown by Li *et al.* [55]. They lower the computational effort by detecting key frames. High-level features of key frames are reused for subsequent frames to reduce effort. Another idea to lower complexity within the neural network is by warping images from different positions [32] or time steps. The movement of pixels between images can be computed with optical flow information Section 3.2.1. However, it has to be noted that the computation of optical flow introduces additional complexity to the problem. With the optical flow information, pixels or features of different time steps can be moved to the same spatial location.

The work of Ranjan *et al.* [71] suggests, that when stacking consecutive frames together along the channel dimension, a standard 2D convolution is capable of computing a derivative across the temporal axis. This allows us to create temporal features by means of a 2D convolution. This is applied in the work of *e.g.* Luc *et al.* [59], which combines feature at different time steps by a convolution.

3.3 Video Semantic Segmentation with Recurrent Neural Networks

The idea of combining video data with semantics is to improve the accuracy and consistency of the output semantic segmentation. One possibility to utilize video data is by using a recurrent neural network. It can forward features through time steps in a scene and recursively apply a linear transformation followed by a nonlinear activation.

3.3.1 Vanilla RNN

The most simple form of a recurrent network is given by the vanilla RNN, where the output $\mathbf{y}^{(t)}$ at time step t is computed by the current input $\mathbf{x}^{(t)}$ and the output $\mathbf{y}^{(t-1)}$ of the previous time step $t-1$:

$$\mathbf{y}^{(t)} = \phi(\mathbf{W}_x\mathbf{x}^{(t)} + \mathbf{W}_y\mathbf{y}^{(t-1)}). \tag{3.4}$$

The parameters $\mathbf{W}$ are shared through time. The output $\mathbf{y}^{(t)}$ is used as hidden state in for the next time step $t+1$. The problem of the vanilla RNN is that during back-propagation the chain rule causes many multiplications by the same factors $\mathbf{W}$. For example, to compute the gradients for time step $t-10$, the weight matrix $\mathbf{W}$ has to be multiplied 10 times. This leads to numerical problems depending on the largest eigenvalues of the matrix. In the scalar case, a factor between -1 and 1 leads to a vanishing gradient, for other values the gradient will become very large.

3.3.2 LSTM

One technique to avoid the gradient vanishing problem is with Long Short-Term Memory (LSTM) cells, introduced by Hochreiter *et al.* [34]. The LSTM has a more complex internal structure compared to the vanilla RNN. It propagates a cell $\mathbf{c}^{(t)}$ and a hidden state $\mathbf{h}^{(t)}$ through time. Within a LSTM cell, five non-linear activation functions are applied at each time step. The weight matrix $\mathbf{W}$ contains four times the number of parameters of the vanilla RNN. A diagram of the computations inside a LSTM cell is shown in Figure 3.2.

The outputs of the four different gates are computed by

$$\mathbf{i} = \sigma(\mathbf{W}_{xi}\mathbf{x}^{(t)} + \mathbf{W}_{hi}\mathbf{h}^{(t-1)}) \tag{3.5}$$

$$\mathbf{f} = \sigma(\mathbf{W}_{xf}\mathbf{x}^{(t)} + \mathbf{W}_{hf}\mathbf{h}^{(t-1)}) \tag{3.6}$$

$$\mathbf{o} = \sigma(\mathbf{W}_{xo}\mathbf{x}^{(t)} + \mathbf{W}_{ho}\mathbf{h}^{(t-1)}) \tag{3.7}$$

$$\mathbf{g} = \tanh(\mathbf{W}_{xg}\mathbf{x}^{(t)} + \mathbf{W}_{hg}\mathbf{h}^{(t-1)}) \tag{3.8}$$

and the output states are given by

$$\mathbf{c}^{(t)} = \mathbf{f} \odot \mathbf{c}^{(t-1)} + \mathbf{i} \odot \mathbf{g} \tag{3.9}$$

$$\mathbf{h}^{(t)} = \mathbf{o} \odot \tanh(\mathbf{c}^{(t)}) \tag{3.10}$$

where $\mathbf{h}^{(t)}$ is the output of the LSTM layer at time step t. If the input $\mathbf{x}$ is a vector of length X and $\mathbf{h}$ is a vector of length H, the size of the weight matrix $\mathbf{W}$ is $(4H) \times (X+H)$.

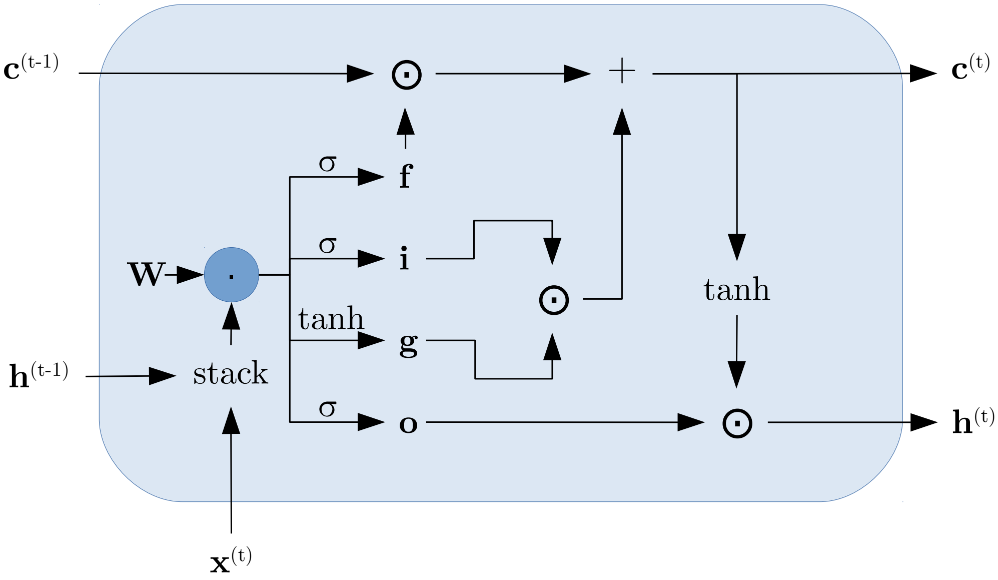


Figure 3.2: LSTM Cell. The LSTM cell uses the previous states $\mathbf{c}^{(t-1)}, \mathbf{h}^{(t-1)}$ and the current input $\mathbf{x}^{(t)}$ to compute the new states $\mathbf{c}^{(t)}, \mathbf{h}^{(t)}$ where $\mathbf{h}^{(t)}$ is the output of the LSTM at time step t. Internally, the parameters $\mathbf{W}$ compute three gate activations $\mathbf{f}, \mathbf{i}, \mathbf{o}$ and one data activation $\mathbf{g}$. The gate activations act as on/off switches to regulated the information flow based on current input and previous states. With the cell state information can travel through numerous time steps without being revealed to the hidden state. Furthermore, the cell state avoids gradient problems, because it only applies element wise multiplication with a small part of the weight matrix.

First, the current input and the previous hidden are stacked in a vector of dimension $H+X$ and multiplied with the weight matrix $\mathbf{W}$. The result is passed trough four different activation functions to produces vectors $\mathbf{f}, \mathbf{i}, \mathbf{g}$ and $\mathbf{o}$. The forget gate $\mathbf{f}$ contains values in $[0, 1]$ and determines which elements of the cell state to reset at the current time step. Similarly, the input gate determines which elements of the input needed to be added to the cell state. The input $\mathbf{g}$ is activated using a $\tanh(\cdot)$ to keep values in the range $[-1, 1]$. Before revealing the values from the cell state to the hidden state, all values are again passed trough a $\tanh(\cdot)$ function to ensure values remain in $[-1, 1]$. Finally, the output gate $\mathbf{o}$ decides how much the elements of the cell state are passed to the current hidden state.

All gates of the LSTM are activated by a sigmoid function and can be interpreted as element-wise on/off switches. A *tanh* activation functions ensure that output values remain in the range $[-1, 1]$. The hidden state reveals information to the outside, whereas the cell state acts as an internal counter. All values inside thee cell state vector can either increase/decrease by a value up to one or it can be set to zero by the forget gate at a single time step. The cell state also allows for efficient gradient flow in the backwards path.

3.3.3 LSTM and Video

There exists several possibilities how to handle the temporal information flow of a video with a LSTM layer. One idea is to create a fixed feature representation at states $\mathbf{c}^{(t)}$ and $\mathbf{h}^{(t)}$ using previous frames [79]. This feature representation can then be used by a decoder LSTM to predict the future or it could simple be used to compute semantic segmentation which is consistent with respect to the previous frames.

Another approach is to include multiple video frames in a single LSTM time step [58]. In this case the current frame t together with $n - 1$ past frames are concatenated in the input vector $\mathbf{x}$ which is passed to the LSTM cell. The output $\mathbf{h}$ again contains n frames, where only the last frame t is used for prediction. Since each frame is processed in multiple LSTM steps, a consistency loss can be applied to the other frames to keep the prediction consistent through various LSTM steps.

3.3.4 Convolutional LSTM

One problem of the traditional LSTM cell which raises when applied to computer vision problems is the size of the input vector $\mathbf{x}$. When including all channels and the spatial dimensions, the number of parameters inside a fully connected LSTM becomes very large. This can be avoided by only passing a compressed feature vector [58], instead of all feature from a CNN feature extractor. However, in case of semantic segmentation, it is very difficult to extract a compact feature representation.

Therefore, convolutional LSTMs (ConvLSTMs) [78] can be used. In contrast to the fully connected LSTM, it convolves input and weights by applying a local filter

$$\mathbf{g}' = \mathbf{W}_x * \mathbf{x}^{(t)} + \mathbf{W}_h * \mathbf{h}^{(t-1)}, \tag{3.11}$$

where $\mathbf{g}'$ contains the pre-activations of all four LSTM gates and "$*$" denotes the convolution operator. This architecture saves parameters and therefore makes an application on large feature vectors possible. It can be applied to video semantic segmentation because high-level features should remain in a local neighborhood as long as the motion is minimal such that objects stays in the same area within consecutive frames.

3.3.5 GRU

The Gated Recurrent Unit (GRU) introduced by Cho *et al.* [10] is another possible recurrent module. It consists of two gates which are activated by the sigmoid function. Additionally, a hyperbolic tangent is applied to input vector and hidden state after the linear layer. The GRU cell is defined as

$$\mathbf{z} = \sigma(\mathbf{W}_{xz}\mathbf{x}^{(t)} + \mathbf{W}_{hz}\mathbf{h}^{(t-1)}) \tag{3.12}$$

$$\mathbf{r} = \sigma(\mathbf{W}_{xr}\mathbf{x}^{(t)} + \mathbf{W}_{hr}\mathbf{h}^{(t-1)}) \tag{3.13}$$

$$\mathbf{h}^{(t)} = (1 - \mathbf{z}) \odot \mathbf{h}^{(t-1)} + \mathbf{z} \odot \tanh(\mathbf{W}_{xh}\mathbf{x}^{(t)} + \mathbf{W}_{hh}(\mathbf{r} \odot \mathbf{h}^{(t-1)})), \tag{3.14}$$

where the parameters of the weight matrix $\mathbf{W}$ are shared through time, $\mathbf{x}^{(t)}$ is the input and $\mathbf{h}^{(t)}$ is the output at time step t.

A diagram of the GRU cell is shown in Figure 3.3. Within the GRU, two gates control the information flow. The update gate $\mathbf{z}$ decides how much information of an element is reused from the previous state $\mathbf{h}^{(t-1)}$ and how much is updated from the new input $\mathbf{x}(t)$. A zero indicates that the old value is copied into the new state, whereas a one uses information from the current input. The reset gate $\mathbf{r}$ is used to forget the values of the previous hidden state by setting it to zero. If the value of the previous state is of importance, the reset gate will be zero.

The configuration of the gates varies, as multiple versions of the GRU exist. They are applied for *e.g.* future video prediction [69] and segmentation [62], but also for different domains like language processing [12]. Similar to the ConvLSTM, the ConvGRU can be used to create more consistent results while saving parameters as tested in [70].

3.3.6 LSTM vs GRU

When comparing LSTM and GRU one can observe several differences. The main difference is that the LSTM cell propagates cell and hidden state whereas the GRU only propagates the hidden state which produces the output. This gives the LSTM an additional channel to propagate information over time. It also supports efficient back-propagation.

Another difference is the number of gates. The LSTM contains three gates, whereas the GRU contains only two. The reset gate $\mathbf{r}$ of the GRU has a similar task as the forget gate $\mathbf{f}$ of the LSTM. The update gate $\mathbf{z}$ of the GRU combines the input gate $\mathbf{i}$ and the output gate $\mathbf{o}$. As a consequence, the GRU needs 25 % parameters less compared to the LSTM.

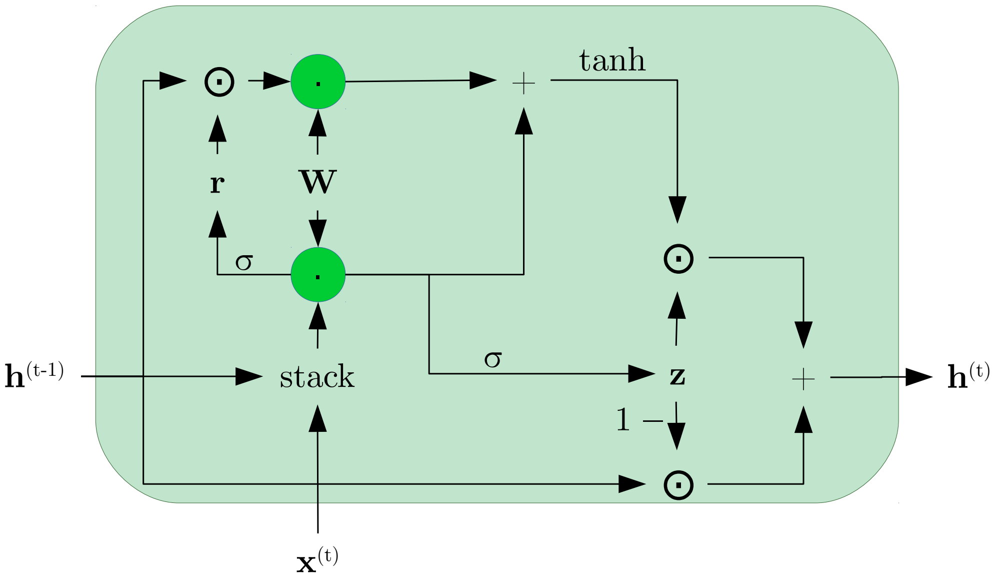


Figure 3.3: GRU Cell. The GRU cell contains a weight matrix **W** which is used to transform the input and to compute the reset gate **r** and the update gate **z** at each time step. Both gates are activated with the sigmoid function σ whereas the input is activated with the tanh function. The hidden state **h** delivers the output and also propagates the information through time.

Both LSTM and GRU outperform the vanilla RNN which is shown by Chung *et al.* [12]. When comparing the performance of LSTM and GRU, no clear winner is found. Greff *et al.* [27] conclude that variations of the LSTM cell do not significantly improve, however the GRU performs similar with reduced complexity.

The two introduced RNN architectures are the most popular ones, however, many variations exist. Jozefowicz *et al.* [41] found slightly different architectures, which all outperform vanilla LSTM and GRU on specific applications. Completely different approaches include the Clockwork RNN [47] and the Independently RNN [55].

In summary, recurrent neural networks are a good tool for modeling temporal dependencies as *e.g.* in video. They can process variable length scenes [17] as an input and create variable length outputs as well. Both LSTM and GRU avoid the vanishing gradient problem and produce good results on various applications. Depending on the application, a slight modification of the architecture can even improve the results.

3.4 Street Scene Data Sets

In this work we want to focus on the domain of street scenes, therefore, we need to select a street scene data set to apply the algorithms. Ideally, this data set should have ground-truth semantic labels to support supervised learning methods. All scenes should have

Class Name	**Group**	**Color (RGB)**	**Class Name**	**Group**	**Color (RGB)**
Road	Flat	(128, 64, 128)	Sky	Sky	(70, 130, 180)
Sidewalk	Flat	(244, 35, 232)	Person	Human	(220, 20, 60)
Building	Construction	(70, 70, 70)	Rider	Human	(255, 0, 0)
Wall	Construction	(102,102,156)	Car	Vehicle	(0, 0,142)
Fence	Construction	(190,153,153)	Truck	Vehicle	(0, 0, 70)
Pole	Object	(153,153,153)	Bus	Vehicle	(0, 60,100)
Traffic Light	Object	(250,170, 30)	Train	Vehicle	(0, 80,100)
Traffic Sign	Object	(220,220, 0)	Motorcycle	Vehicle	(0, 0,230)
Vegetation	Nature	(107,142, 35)	Bicycle	Vehicle	(119, 11, 32)
Terrain	Nature	(152,251,152)			

Table 3.1: Cityscapes Classes. The Cityscapes dataset consists of 19 semantic classes. The different classes together with their color coding and the group they belong to are shown in this table.

video data to train and validate consistency between consecutive frames.

3.4.1 Real World Data

When developing algorithms for real-world application, it is most reasonable to use natural image data. Three popular data sets are KITTI [24], Mapillary Vistas [68] and Cityscapes [13]. We want to compare them, to decide which one suits best for our needs.

Firstly, the KITTI dataset offers sequential raw data with different scene length captured at 10 FPS. The data contains urban and country scenes with a resolution of 1392×512 pixels. The ground-truth semantic segmentation is provided for 400 images. In addition, scene flow, depth and instance ground-truth exists. Accuracy results of different algorithms are evaluated on a test set are reported on a benchmark list.

Secondly, the Mapillary dataset offers 25,000 RGB images with 20,000 annotations at high resolution. The annotations exist for semantic, instance and panoptic segmentation [46]. The panoptic segmentation combines instance segmentation with semantic segmentation. Mapillary also provides a high score list for object detection and panoptic segmentation results. However, this dataset does not provide sequential images.

Thirdly, the Cityscapes dataset consists of 5,000 scenes with 30 frames each at 17 FPS with 1024×2048 resolution. One frame for each scene is annotated with semantic and instance segmentation ground-truth. Additionally, stereo data, GPS coordinates, Ego-motion data and longer video sequences are available. The highest score is currently 83.6 % mIoU and DeepLab [9] is ranked 15th with 82.1 % mIoU. The 19 Cityscapes classes used for semantic segmentation are shown in Table 3.1.

All of the three introduced datasets provide semantic segmentation ground-truth and a benchmark list. Furthermore, Cityscapes and KITTI both have raw video data, where the

Carla Class	Cityscapes Class	Carla Class	Cityscapes Class
Building	Building	Road	Road
Fence	Fence	Sidewalk	Sidewalk
Other	-	Vegetation	Vegetation
Pedestrian	Person	Car	Car
Pole	Pole	Wall	Wall
Road Line	-	Traffic Sign	-

Table 3.2: Mapping Carla to Cityscapes Classes. The mapping from Carla to Cityscapes semantic classes is shown in this table. Altogether, twelve Carla classes reduce to nine valid Cityscapes classes.

KITTI sequence length varies. Other dataset (*e.g.* CamVid [4]) do not provide sufficient video data.

3.4.2 Synthetic Street Scene Data

The advantage of synthetic data compared to real world data is that the corresponding ground-truth information (*e.g.* depth information, semantic segmentation, vehicle parameters, weather conditions) can be generated easily.

Two simulators which are build using the Unreal Engine are AirSim [76] and Carla [18]. Both can be used for realistic driving simulations. Additionally, AirSim also supports drone simulations. The access for car control and camera data collection is provided through an API which can be accessed with Python. The Python API for Carla supports semantic data generation of 12 different classes. A mapping from Carla to Cityscapes classes is shown in Table 3.2

One disadvantage of the driving simulators is that scenarios have to be created in order to generate realistic scene data. The Synthia dataset by Ros *et al.* [72] provides over 200,000 already generated synthetic images. The results indicate that the addition of synthetic data improves mIoU on real world data. Especially, training with combined real word and synthetic data works very well. It increases performance of semantic classes representing small objects. It has to be noted that number of validation classes is only 11. Benchmarks on the Cityscapes dataset consider 19 valid classes, which means that it is more difficult to achieve a high mIoU score.

Fortunately, there are a lot of street scene data sets with semantical annotations. This prevents the need of expensive manual labeling. However, no annotated video data exists, because the task takes a lot of time even if many people work together. Nevertheless, synthetic data from driving simulators can be used to create sequential images with perfectly accurate ground-truth.

To summarize, this chapter compared methods for solving the task of frame-to-frame consistent semantic segmentation. A semantically segmented image can be computed by traditional methods such as graph cut or clustering. Recently, CNNs are used more frequently for this task. Layers with dilated convolutions increase the receptive field while keeping the number of parameters low. Metrics for comparing segmentation are Total Accuracy and mIoU, whereas the latter is predestined for street scenes. CNNs can be used together with RNNs to combine the tasks of semantic segmentation and feature propagation over time. Especially, ConvLSTM are useful because of their reduced number of parameters. Currently, many different street scene datasets are used to compare the performance of computer vision algorithms. However, none of them provides ground-truth information for consecutive frames, because of the large afford of manual labeling. This problem can be tackled by synthetically created data.

4 Frame-to-Frame Consistent Semantic Segmentation

Contents

After related work has been discussed, we will focus on the methods used in this thesis. Most importantly, the focus is on semantic segmentation. From the various possibilities mentioned previously, we want to develop a method which has the potential to outperform state of the art semantic segmentation algorithms. Convolutional neural networks are selected because of recent success in the field of computer vision. Especially on the analyzed dataset benchmarks they performed very well.

Despite the selection of the architectural framework, the data for training the model and benchmarking is crucial. Most important aspects when selecting a suitable data set are size and reference scores to be able to evaluate the performance against other algorithms. Both criteria are fulfilled by the Cityscapes data set. It provides a lot of video data and is used in many recent works as a benchmark.

In addition, the utilization of temporal information is introduced as well. It has to be included efficiently to maintain low computation cost while improving consistency. For successful training, an objective function has to be selected which enforces consistent and accurate semantic segmentation over time. The objective of the methods explained in this chapter is to achieve frame-to-frame consistent semantic segmentation.

4.1 Network Architectures

Since the problem of semantic segmentation is addressed by CNNs, suitable architectures have to be defined. We focus on computational effort, because the extension to consistent multiple frame processing requires additional resources. Two architectures were chosen to proof cross network validity. The first is a simple experimental architecture named SSNet. The second is the well established ESPNet which computes semantic segmentation with low computational cost.

4.1.1 SSNet

The Semantic Segmentation Network (SSNet) is a seven layer fully convolutional architecture which computes semantic segmentation. It process three channel RGB input images and outputs 19 channels. In between the features are propagation using 64 channels. All layers use input padding to keep spatial dimensions constant. First, the low level features are created at the layers one and two by a convolution of filter size 3 × 3. Then the low level features are processed using four layers of 3 × 3 convolutions with dilate rate 2 and 4 to increase the size of the receptive field. Finally, the mapping from 64 to 19 output channels is again computed by a 3 × 3 convolution. The last layer activation function is a softmax function, which allows the network to output a probability volume. All other layers implement the ReLU activation function. A network diagram is shown in Figure 4.1. The network computation is given by

$$\mathbf{P} = \mathrm{SSNet}(\mathbf{X}), \tag{4.1}$$

where $\mathbf{X} \in \mathbb{R}^{M \times N \times 3}$ is the RGB input image of resolution $M \times N$ and $\mathbf{P} \in \mathbb{R}^{M \times N \times 19}$ is the output probability volume. $\mathbf{P}$ fulfills the properties

$$\begin{aligned} &\sum_{c=1}^{19} p_{m,n,c} = 1 \qquad \forall m = 1 \dots M, n = 1 \dots N \qquad \text{and} \\ &p_{m,n,c} \in [0,1] \qquad \forall m = 1 \dots M, n = 1 \dots N, c = 1 \dots 19. \end{aligned}$$

The semantic class prediction $\hat{\mathbf{S}} \in \mathbb{N}^{M \times N}$ is obtained by the argmax along the third dimension:

$$\hat{s}_{m,n} = \underset{c=1\dots19}{\arg\max}\, p_{m,n,c}. \tag{4.2}$$

At last layer the receptive field is 31 × 31. The total number of parameters is 197,395.

The SSNet architecture consists of standard and dilated convolutions which use the ReLU activation function. It is a simple seven layer model whose semantic segmentation output is considered as a validator for experiments.

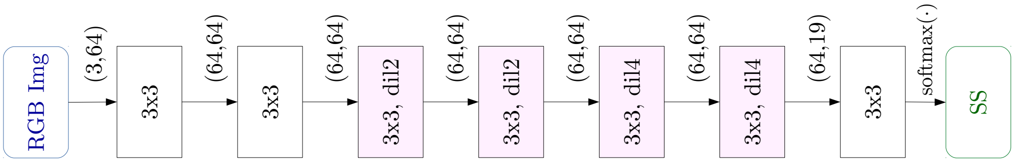


Figure 4.1: Architecuture of SSNet. The path from the RGB input image to the semantic segmentation output is detonated by arrows. Each convolutional layer is represented by a rectangular box containing the filter size. Dilated convolutions are highlighted in pink. Parenthesis next to each layer contain the number of input resp. output channels. All layers, except the last layer one, use ReLU activation functions.

4.1.2 ESPNet

The second network architecture used is the ESPNet [66]. This architecture was chosen because it provides good semantic segmentation performance on the cityscapes benchmark while keeping the model complexity low. Fast computation is especially important when considering the video setting. The network which consists of an encoder and a decoder is shown in Figure 4.2.

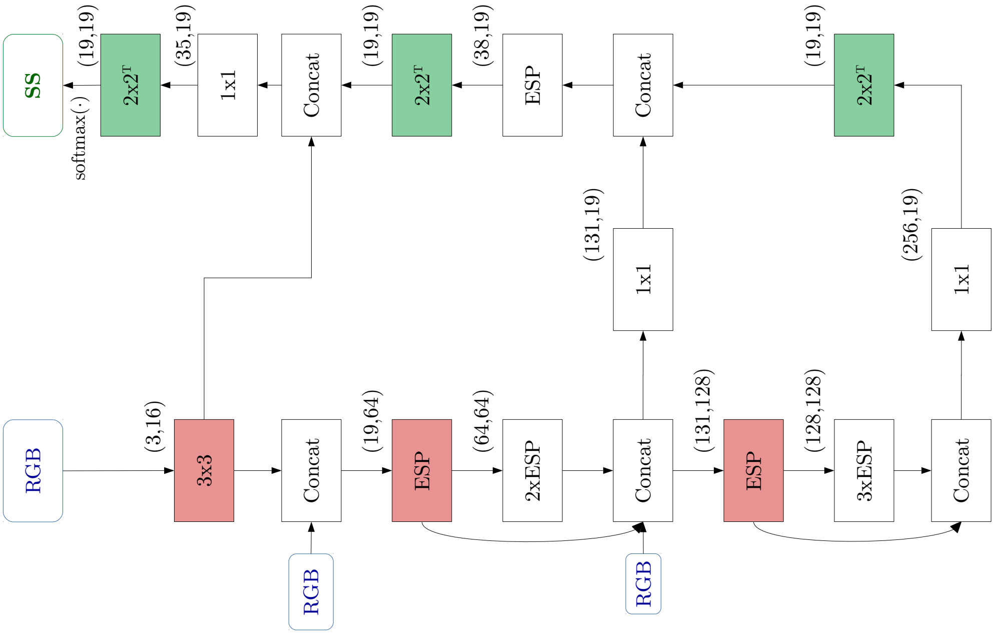


Figure 4.2: Architecuture of ESPNet. RGB images are included at three different resolutions. Each layer is represented by a rectangular box containing the module name. Red and green boxes reduce respectively up-sample spatial dimensions by af factor of two. Parenthesis next to each layer contain the number of input and output channels.

First, the encoder uses the three channel input image to create 256 channel feature encoding. Thereby, spatial dimensions are decreases by a factor of 8. At the first layer, a

3×3 convolution extracts low level features. The following layers in the decoder efficiently increase the receptive field by the ESP module. These modules consists of a pointwise convolution followed by dilated convolutions with different dilation rates. This allows a single ESP module to create an receptive field of 33×33, while keeping the number of parameters low.

Second, the decoder uses output features from the encoder at three different level. The high-level, mid-level and low-level features are up-sampled using three transposed convolutions. Finally, the 19 channel output probability volume is given by

$$\mathbf{P} = \text{ESPNet}(\mathbf{X}), \tag{4.3}$$

where $\mathbf{X} \in \mathbb{R}^{M \times N \times 3}$ is the RGB input image of resolution $M \times N$ and $\mathbf{P} \in \mathbb{R}^{M \times N \times 19}$ is the output probability volume with the same properties as in Equation (4.1). The semantic segmentation is obtained by the argmax in Equation (4.2).

The ESPNet encoder-decoder architecture with a total of 202,000 parameters which allows it to be considered a lightweight neural network semantic segmentation model. In its core, the ESP module supports efficient computation through a point-wise convolution followed by dilated convolutions with different dilation rate. All layers, except the last, use PReLU activation functions. The network outputs a 19 dimensional probability vector for each pixel in the input image.

4.2 Video Data

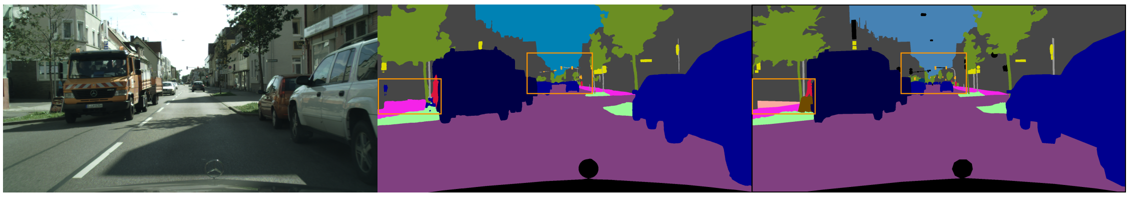
The CNNs are trained on the Cityscapes data set [13], because it provides consecutive data. It provides enough video data to allow training of the 19 semantic classes. One big advantage is that the single frame data is created by taking 20th frame of each video scene consisting of 30 frames. This allows to test and compare video trained to single frame models. Unfortunately, due to the high effort, no ground-truth labeling except for the 20th frame exists. However, this problem also exists with other datasets. All in all, 2975 training and 500 validation scenes are used.

4.2.1 Semantic Segmentation Oracle

Since the Cityscapes dataset does not provide a ground-truth semantics for video data, the labels are created using a model which performs well at single frame data. Therefore, the DeepLab Xception [11] network pretrained for Cityscapes data was used [9]. It consists of 71 layers and achieves 80.31 % mIoU semantic class score at the Cityscapes validation set. The models was pretrained using full 1024×2048 Cityscapes resolution. Consequently, the network predicts the segmentation map $\hat{\mathbf{S}} \in \mathbb{N}^{1024 \times 2048}$ for the corresponding input image $\mathbf{X} \in \mathbb{R}^{1024\times2048\times3}$,

$$\mathbf{S} = \hat{\mathbf{S}} = \text{DeepLab}(\mathbf{X}), \tag{4.4}$$

where $\mathbf{S}$ becomes the ground-truth for our experiments. One example of how the DeepLab oracle performs compared to hand-labeled groundtruth is shown in Figure 4.3. Some details of mostly distant objects are not labeled truely by the oracle. Nevertheless, differences are only minor and most objects are segmented accurately. A method of how to overcome the problems introduced by using not completely perfect groundtruth is by using synthetic data.



(a) RGB Input **(b)** DeepLab Oracle **(c)** Ground Truth

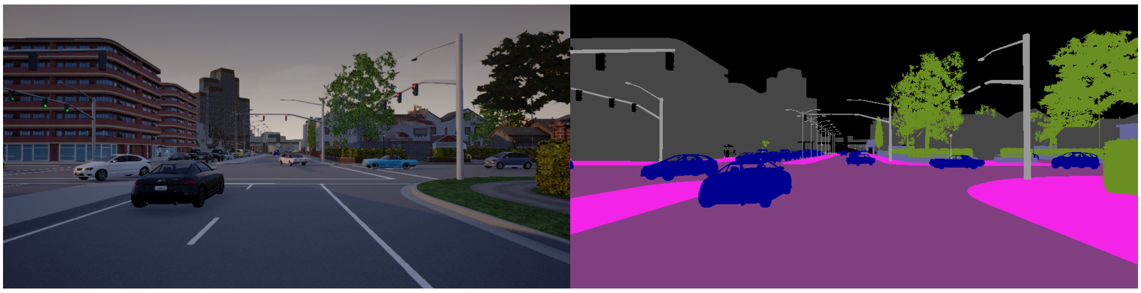
Figure 4.3: DeepLab Oracle. Comparing the semantic segmentation of the DeepLab model (b) to the fine labeled Cityscapes ground-truth (c) of the RGB input (a). Minor differences which can be observed are highlighted by orange boxes.

4.2.2 Synthetic Data

The problem with the Cityscapes data set is that it does not provide ground-truth semantic segmentation for video data. Although the previously described oracle generates ground-truth with high mIoU score, the prediction consistency over multiple frames in a scene is not given, because each frame is processed individually. Therefore, this approach is not well suited for consistency training.

To overcome this issue, the CARLA Simulator [18] is used to create perfectly accurate and consistent ground-truth. The idea is to especially enforce consistency learning with this synthetic data set. However, it should also not negatively effect semantic class scores because of the synthetic RGB images. In total 3600 scenes of 30 frames each of both input data $\bar{\mathbf{X}} \in \mathbb{R}^{512\times1024\times3}$ and semantic segmentation $\bar{\mathbf{S}} \in \mathbb{N}^{512\times1024}$ are generated. An example data pair is shown in Figure 4.4.

The method of combining data from the a slightly inaccurate prediction oracle with the perfectly accurate synthetic data generator should allow the supervised learning algorithm to produce results with high semantic segmentation accuracy while also maintaining consistency over multiple frames within a scene.



(a) Carla RGB **(b)** Ground Truth

Figure 4.4: Carla Data. Synthetic data generation using the Carla simulator: RGB image (a) and semantic segmentation (b).

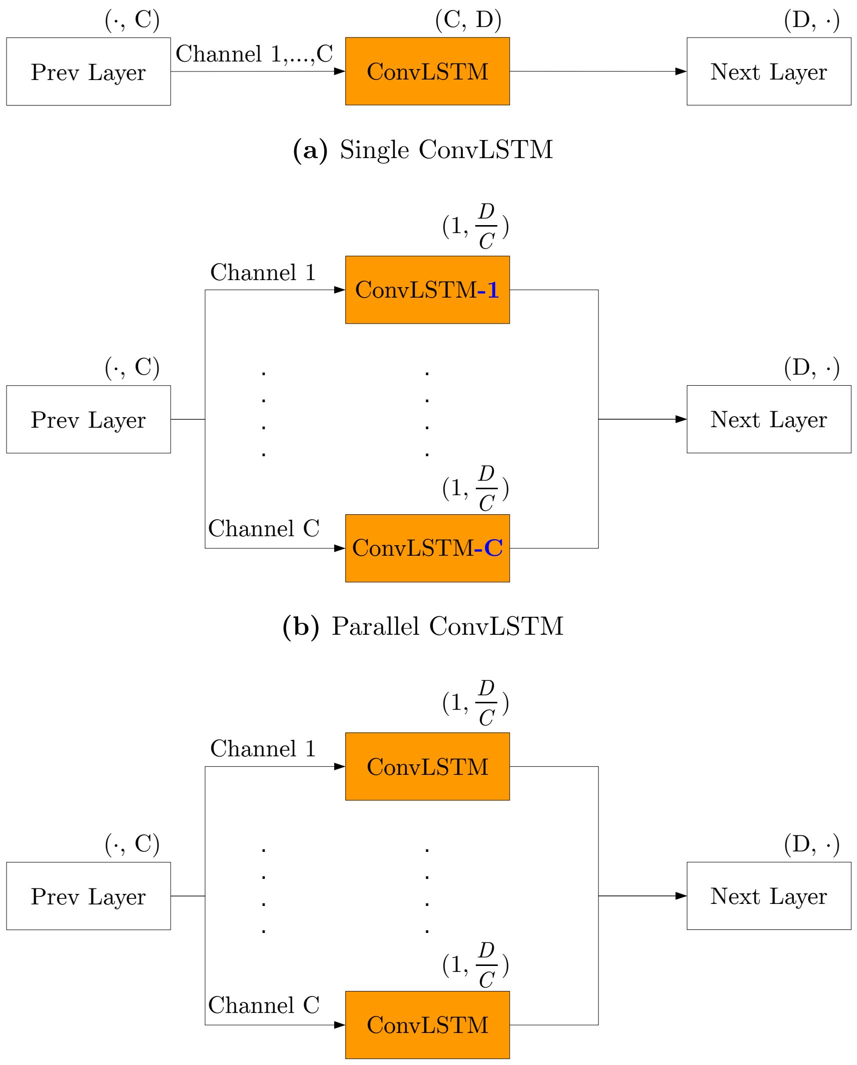
4.3 ConvLSTM

In order to use the temporal information available through the video data, a LSTM cell is added to the single frame architecture. It allows the propagation of previously extracted features to future frames. Because fully connected LSTMs need too many parameters for high resolution image data, Convolutional LSTM are used. However, it has to be ensured that the filter size is large enough to track motion boundaries.

The cell state $\mathcal{C} \in \mathbb{R}^{M \times N \times D}$ and the hidden state $\mathcal{H} \in \mathbb{R}^{M \times N \times D}$ are initialized using zero, random $[0, 1)$ or learned initialization. In the learned initialization we want the network to determine good initial values for a given data set. Then, the states are propagated through all time steps in a scene in the stateful setting. In the stateless setting, $\mathcal{C}$ and $\mathcal{H}$ are reinitialized after a given sequence length ranging from 2 to 11 time steps. The filter-size inside the cell depends on the feature level the cell is operation on ranging from 1×1 to 11×11.

Three different methods of how the ConvLSTM is integrated at a given layer are suggested. They are depicted in Figure 4.5. The Single ConvLSTM can be used in layers where the input channels cannot be distinguished semantically. In contrast, the Parallel ConvLSTM layer convolves each channel separately. Additionally, the Parallel ConvLSTM with Weight Sharing layer also shares weights across channels. This causes each channel to use the same ConvLSTM cell.

The input $\mathbf{X}$ over which the convolution is computed consists of the output from the previous layer in the architecture, stacked with the hidden state $\mathcal{H}^{(t-1)}$ from the previous time step $t-1$ of the ConvLSTM. $\mathbf{X}$ is also zero padded to ensure constant spatial dimensions throughout the layer. C and D represent the number of input respectively output channels. The variable $g = 1 \ldots 4$ selects the activation for one of the four LSTM gates.



(a) Single ConvLSTM

(b) Parallel ConvLSTM

(c) Parallel ConvLSTM with Weight Sharing

Figure 4.5: ConvLSTM Layer. Comparing three different methods to include a ConvLSTM layer (orange) into an existing architecture. The number of input and output channels for each layer are written inside parenthesis next to the box.

Single ConvLSTM Our first method uses a single standard ConvLSTM cell given by

$$a_{u,v,d,g} = (\mathbf{X} * \mathbf{W})_{u,v,d,g} = \sum_{c=1}^{C+D} \sum_{p=1}^{P} \sum_{q=1}^{Q} x_{u+p-1,v+q-1,c} w_{p,q,c,d,g}, \tag{4.5}$$

with input $\mathbf{X} \in \mathbb{R}^{M \times N \times (C+D)}$. The number of parameters for the standard method is

$$L_{\text{params}} = (C + D) \cdot 4D \cdot P \cdot Q + 4D. \tag{4.6}$$

This method is useful when the input features need to be treated independently from each other. It is the most basic method, which can be used when no meaningful separation between the channels of the layer is given.

Parallel ConvLSTM The second method uses C parallel ConvLSTM cells, where each cell treats a separate channel (*e.g.* a semantic class). In this layer, a single cell is given by

$$a_{u,v,d,g} = (\mathbf{X} * \mathbf{W})_{u,v,d,g} = \sum_{c=1}^{1+\frac{D}{C}} \sum_{p=1}^{P} \sum_{q=1}^{Q} x_{u+p-1,v+q-1,c} w_{p,q,c,d,g}, \tag{4.7}$$

with input $\mathbf{X} \in \mathbb{R}^{M \times N \times (1+\frac{D}{C})}$ for each cell. The number of output channels for each individual cell is given by $\frac{D}{C}$. The number of parameters for the whole layer is

$$L_{\text{params}} = C \cdot \left(\left(1 + \frac{D}{C} \right) \cdot 4\frac{D}{C} \cdot P \cdot Q + 4\frac{D}{C} \right). \tag{4.8}$$

Note that here each output channel is processed separately and thus it is very well suited for enforcing consistency.

Parallel ConvLSTM with Weight Sharing The third method uses C parallel ConvLSTM cells which share their weights. They can also be interpreted as a single cell where each input channel is passed through individually. The convolution computation is equal to Equation (4.7). The input for each cell is also given by $\mathbf{X} \in \mathbb{R}^{M \times N \times (1+\frac{D}{C})}$. However, every ConvLSTM has the same weight matrix $\mathbf{W}$. Therefore, the total number of parameters for the layer drops to

$$L_{\text{params}} = \frac{1}{C} \cdot ((C + D) \cdot 4D \cdot P \cdot Q + 4D). \tag{4.9}$$

This method is the computational least expensive, because it treats each channel the same over multiple time steps.

Having introduced the different types of ConvLSTM layers, it is also important to include them reasonably into the single frame architectures VSSNet and ESPNet. The single ConvLSTM layer can be used when no meaningful separation between the channels at the layer can be drawn. A parallel ConvLSTM layer can be implemented at the output layer of the network, because at this layer each ConvLSTM can operate on a separate semantic class.

4.3.1 VSSNet

The Video Semantic Segmentation Network (VSSNet) is an extension to the SSNet with a ConvLSTM layer. The main purpose for this architecture is to verify that the ConvLSTM can be used to compute frame-to-frame consistent predictions. In contrast to the SSNet the last layer is replaced by a ConvLSTM. This allows to monitor the ConvLSTM separately, because the last layer creates the 19 channel output semantic. An architecture diagram consisting of three time steps is shown in Figure 4.6.

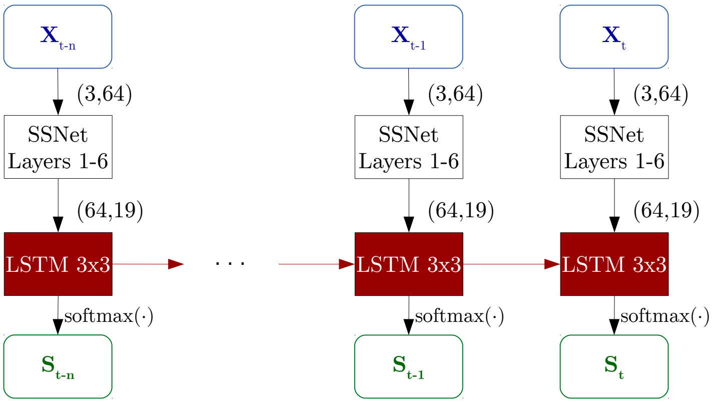


Figure 4.6: Architecuture of VSSNet. Semantic Segmentation is computed for a sequence of $n+1$ time steps (green). The ConvLSTM (red) layer propagates features through time.

When analyzing this architecture, we observe that both, the number of layers and the size of the output receptive field stays the same compared to the SSNet. However, the number of parameters increases because of the ConvLSTM. Considering a single classical ConvLSTM with filter size 3×3, the number of parameters increases by 56,848 (29 %).

4.3.2 ESPNet_L1

Similar to the SSNet, the ESPNet is also modified using ConvLSTM layers to allow for temporal information flow. Altogether, four different positions of the ConvLSTM layer are considered. Three at low (ESPNet_L1d), middle (ESPNet_L1c) and high (ESPNet_L1b) feature level at the encoder and one at the last layer of the decoder (ESPNet_L1a), similar to the VSSNet. The lowercase letter at the end of the architecture indicates the position of the ConvLSTM layer within the architecture. The ConvLSTM layer of ESPNet_L1a is directly at the output layer whereas the ConvLSTM layer of ESPNet_L1d processes feature close to the input of the network. The four different positions of the ConvLSTM are depicted in Figure 4.7.

When considering a 3×3 standard convolutional ConvLSTM cell the complexity of the architecture changes as follows. The ESPNet_L1a needs an additional 26,068 parameters at the last layer of the architecture. At the lowest feature level of the encoder, ESPNet_L1d only needs 18,496 parameters. The other ConvLSTM layers at the encoder, ESPNet_L1c and ESPNet_L1b need 102,676 respectively 188,176 additional parameters. This increases the number of parameters up to 93 % compared to the original ESPNet

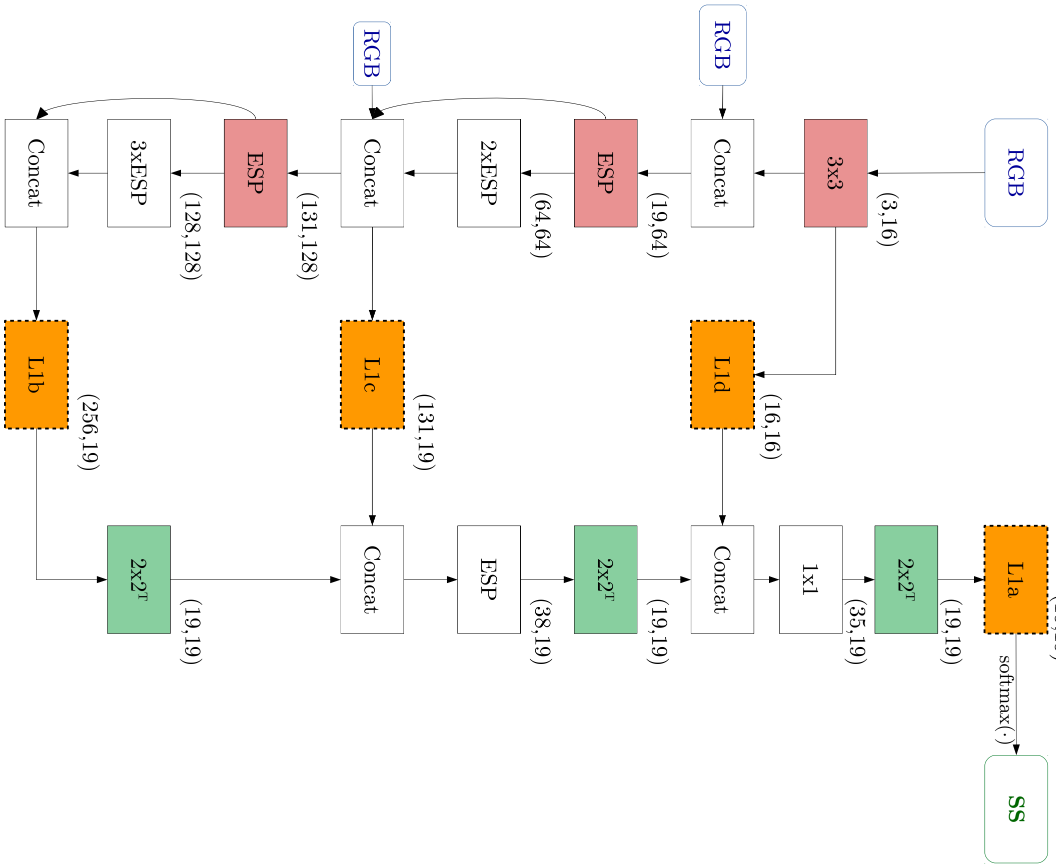


Figure 4.7: ESPNet with LSTM Layers. Four different positions for including a ConvLSTM layer (orange) into the existing ESPNet architecture. Dashed boxes indicate that only one ConvLSTM layer is present in a single architecture. L1b, L1c and L1d replace 1×1 channel reduction convolutions while L1a adds an additional layer to the architecture.

for the ESPNet_L1b extension. However, the total number of parameters is still small compared to other semantic segmentation models (*e.g.* [9]). Also, the architecture with the highest number of parameters ESPNet_L1b, has most parameters in the ConvLSTM layer which is located on a high feature layer, where spatial dimensions are low. This causes computational effort to remain low.

The introduction of the ConvLSTM layer allows for temporal information flow to share features between frames within a scene. First, the VSSNet architecture provides a simple and comprehensible model. The ESPNet_L architectures compare ConvLSTM layers at different feature levels and provide results in a state-of-the-art network.

4.4 Time Convolution

Besides the principle of using a LSTM cell to propagate temporal information, this can also be done by means of a convolution as seen in [59]. The frames of each time step are stacked together in a single volume where the input channels correspond to the time axis. A convolution then adds the filters over time. A standard convolution is given by

$$a_{u,v,d} = (\mathbf{X} * \mathbf{W})_{u,v,d} = \sum_{t=1}^{T} \sum_{c=1}^{C} \sum_{p=1}^{P} \sum_{q=1}^{Q} x_{u+p-1,v+q-1,c,t} w_{p,q,c,t,d}, \tag{4.10}$$

with input $\mathbf{X} \in \mathbb{R}^{M \times N \times C \times T}$ where T is the number of time steps.

The number of parameters increases by the factor T:

$$L_{\text{params}} = C \cdot T \cdot D \cdot P \cdot Q + D. \tag{4.11}$$

Compared to the LSTM extension, the time convolution can only consider features of the last T time steps. Also, the number of time steps is fixed and cannot changed to process shorter scenes. Convolving over time can result in filters which compute time derivatives as shown in [71]. In order to detect motion, the filter size has to be large enough to track boundaries throughout T time steps.

4.4.1 ESPNet_T

The time convolution is added at the last layer of ESPNet. It processes the 19 output layers to provide a consistent prediction. A residual connection is added to allow surpassing old features. Two different architecture are introduced. ESPNet_T computes the time convolution on T ESPNet outputs while ESPNet_T2 uses only one ESPNet output and $T-1$ outputs from the previous time convolutions.

The number of parameters increases by 16,264 for a 3 × 3 standard convolution using $T = 5$ time steps. It is valid for both ESPNet_T and ESPNet_T2. Compared to a ConvLSTM layer at ESPNet_L1a, the number of parameters is lower by 38 %.

4.5 Consistency Constraint

The aim of this thesis is to compute semantic segmentation and provide consistency through multiple frames in a video setting. As we decided to tackle these problems with neural networks, we need to enforce accuracy and consistency using a suitable objective function. Therefore, the loss function $\mathcal{L}$ consists of two terms:

$$\mathcal{L}(\mathbf{S}, \mathbf{P}) = \lambda_{\text{ce}} \mathcal{L}_{\text{ce}}(\mathbf{S}, \mathbf{P}) + \lambda_{\text{incons}} \mathcal{L}_{\text{incons}}(\mathbf{S}, \mathbf{P}), \tag{4.12}$$

where $\mathcal{L}_{\text{ce}}$ penalizes the segmentation error, $\mathcal{L}_{\text{incons}}$ penalizes inconsistencies and $\lambda_{\text{ce}}, \lambda_{\text{incos}}$ are hyper-parameters. Inside the loss function, the sum is computed over T time steps, the image dimensions $M \times N$ and the number of valid semantic classes $|\mathbb{S}|$. Two important function include the Kronecker delta function:

$$\delta(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{else} \end{cases}, \tag{4.13}$$

and the function to determine valid pixels in the groundtruth semantic segmentation $\mathbf{S}$:

$$\omega(s) = \begin{cases} 1 & \text{for } s \in \mathbb{S} \\ 0 & \text{for } s \notin \mathbb{S} \end{cases}. \tag{4.14}$$

Cross Entropy Loss $\mathcal{L}_{\text{ce}}$ is computed using the cross-entropy between target $\mathbf{S} \in \mathbb{N}^{T \times M \times N}$ and predicted probability space $\mathbf{P} \in \mathbb{R}^{T \times M \times N \times |\mathbb{S}|}$:

$$\mathcal{L}_{\text{ce}}(\mathbf{S}, \mathbf{P}) = -\frac{1}{V} \sum_{t,m,n=1}^{T,M,N} \left(\sum_{c=1}^{|\mathbb{S}|} \delta(s_{t,m,n}, c) \cdot \log(p_{t,m,n,c}) \right) \cdot \omega(s_{t,m,n}) \tag{4.15}$$

where V is the number of valid pixels in a sequence given by

$$V = \sum_{t,m,n=1}^{T,M,N} \omega(s_{t,m,n}). \tag{4.16}$$

.

Consistency Loss: All Pixels All Classes The second term which enforces consistency between consecutive frames in a scene is computed using three different methods. First, we measure the absolute difference between all consecutive frames in a sequence using

$$\mathcal{L}_{\text{incons}}(\mathbf{S}, \mathbf{P}) = \frac{1}{C} \sum_{t,m,n=1}^{T-1,M,N} \left(\sum_{c=1}^{|\mathbb{S}|} ||p_{t,m,n,c} - p_{t+1,m,n,c}||_1 \right) \cdot \omega(s_{t,m,n}) \cdot \delta(s_{t,m,n}, s_{t+1,m,n}) \tag{4.17}$$

where C is the number of valid and consistent pixels in a sequence given by

$$C = \sum_{t,m,n=1}^{T-1,M,N} \omega(s_{t,m,n}) \cdot \delta(s_{t,m,n}, s_{t+1,m,n}). \tag{4.18}$$

We can also use squared differences instead of the ℓ_1-norm, in order to penalize outliers harder.

Consistency Loss: Correctly Predicted Pixels All Classes This method only considers pixels which are already predicted correctly in one of two consecutive images. This ensures that incorrect pixels are only affected by the cross-entropy loss. Such a inconsistency loss function is defined as

$$\mathcal{L}_{\text{incons}}(\mathbf{S},\mathbf{P}) = \frac{1}{C}\sum_{t,m,n=1}^{T-1,M,N}\left(\sum_{c=1}^{|\mathbb{S}|}||p_{t,m,n,c} - p_{t+1,m,n,c}||_1\right)\cdot\omega(s_{t,m,n})\cdot\delta(s_{t,m,n},s_{t+1,m,n})\cdot \psi(\mathbf{p}_{t,m,n},s_{t,m,n},\mathbf{p}_{t+1,m,n},s_{t+1,m,n}), \tag{4.19}$$

where the number of valid and consistent pixels C is computed as in Equation (4.18). Differences compared to Equation (4.17) are highlighted in green. The function which determines if one pixel is predicted correctly is given by

$$\psi(\mathbf{p}_1,s_1,\mathbf{p}_2,s_2) = \min(\delta(\arg\max(\mathbf{p}_1),s_1) + \delta(\arg\max(\mathbf{p}_2),s_2),1) \tag{4.20}$$

Again, the ℓ_1-norm can be substituted by other error functions.

Consistency Loss: Correctly Predicted Pixels True Class Third, the inconsistency loss function Equation (4.19) can be extended by only considering the error of the true class:

$$\mathcal{L}_{\text{incons}}(\mathbf{S},\mathbf{P}) = \frac{1}{C}\sum_{t,m,n=1}^{T-1,M,N}\left(\sum_{c=1}^{|\mathbb{S}|}\delta(s_{t,m,n},c)\cdot||p_{t,m,n,c} - p_{t+1,m,n,c}||_1\right)\cdot\omega(s_{t,m,n})\cdot \delta(s_{t,m,n},s_{t+1,m,n})\cdot\psi(\mathbf{p}_{t,m,n},s_{t,m,n},\mathbf{p}_{t+1,m,n},s_{t+1,m,n}), \tag{4.21}$$

where the differences compared to Equation (4.19) are highlighted in green. In comparison, the previously introduced loss function produces approximately a $|\mathbb{S}|$-times higher error. This issue can be corrected by choosing a larger hyper-parameter λ_{incons}.

We introduced three variants of loss functions which allow training for consistency. Slightly different loss functions can be chosen by substituting the ℓ_1-norm by a different error term or by adding certainty threshold to trigger the consistency loss only when the cross-entropy has already trained for semantic.

To summarize, we presented five methods to tackle the frame-to-frame consistent semantic segmentation problem. Computationally inexpensive CNN architectures are used to compute semantic segmentation. Ground-truth video data is generated through a prediction oracle and a synthetic tool to allow supervised learning. Additionally, the CNN architectures are modified by adding ConvLSTM layers and time convolution layers to allow propagation of features through consecutive frames. To ensure that the training algorithm supports consistent semantic segmentation, the objectives are formulated in the loss function. The results of implementing the methods are presented in Chapter 5.

5 Experiments

Contents

This chapter focuses on the implementation of the methods introduced in Chapter 4. In order to produce frame-to-frame consistent semantic segmentation, the experiments are conducted incrementally. In the beginning we keep complexity low by using a small training data set and a simple model. This allows for early adjustments of the training setting and hyper-parameters. The first experiments focus on semantic segmentation using the SSNet architecture. After the experiments on single frame data deliver acceptable results, we add sequential data to train the SSNet towards multiple frame consistency. Training with the sequential data set causes changes (*e.g.* time for one iteration, sequence length for each step, order of scene processing) in the setting, which need to be evaluated. We modify the architecture SSNet to contain a ConvLSTM layer which forwards the features through time. Advantages and disadvantages of this modification need to be evaluated. After we finish the experiments on the simple SSNet architecture successfully, we test the methods suggested for the ESPNet.

Similar to the SSNet architecture, we start with single frame evaluation for the ESPNet. Results need to be inline with existing benchmarks to ensure correctness of the model. After state of the art results are achieved for single frame training, the model is extended to process video data. To process the video data, we add ConvLSTM layers to the architecture as well as a convolution over the time dimension. In addition, the inconsistency loss term from Equation (4.12) at the objective function ensures training for consistency.

To enlarge the training set we create synthetic data which provides completely precise segmented input data to the CNN. Therefore, the street scene simulator CARLA [18] is passed through various scenarios to generate a dataset similar to the size of Cityscapes. All of the experiments are compared qualitatively as well as quantitatively via suitable metrics for semantic accuracy and consistency.

5.1 Hardware and Software

We want to specify the software and hardware environment in which the experiments are performed. The hardware is build around an Intel Core i7-8700 CPU. Two graphics cards, GeForce RTX 2080 TI and GeForce RTX 2070, are used for parallel model training. To support fast parallel computation on the GPU CUDA 10 is installed together with CuDNN 7. On the implementation level, Python is used for programming tasks. Various Python extensions are used for different tasks of this work. Pretrained models *e.g.* DeepLab are evaluated with TensorFlow 1.12. We implement most CNN models in PyTorch 1.0. Within PyTorch, we use the `nn` package for model creation, the `utils.data` package for parallel data loading and the `autograd` module computes the computational path for gradient optimization methods. Training and test results are logged and visualized with the Tensorboard package. To summarize, our development environment is considered state of the art in terms of quality.

5.2 Semantic Segmentation with SSNet

We perform the first experiments on the SSNet architecture and train with the Cityscapes single frame data set. Later, we add video data together with a consistency loss function to train the single frame architecture for the video setting. The purpose of this experiments is to evaluate the performance of the SSNet architecture as well as to create a semantic segmentation baseline for future modification to the architecture *i.e.*, adding the ConvLSTM.

5.2.1 Training Setting

Our main objective is to keep the training setting as simple as possible while still producing comparable results. It should allow for fast convergence despite the additional complexity of video prediction. We use the following parameters for all experiments.

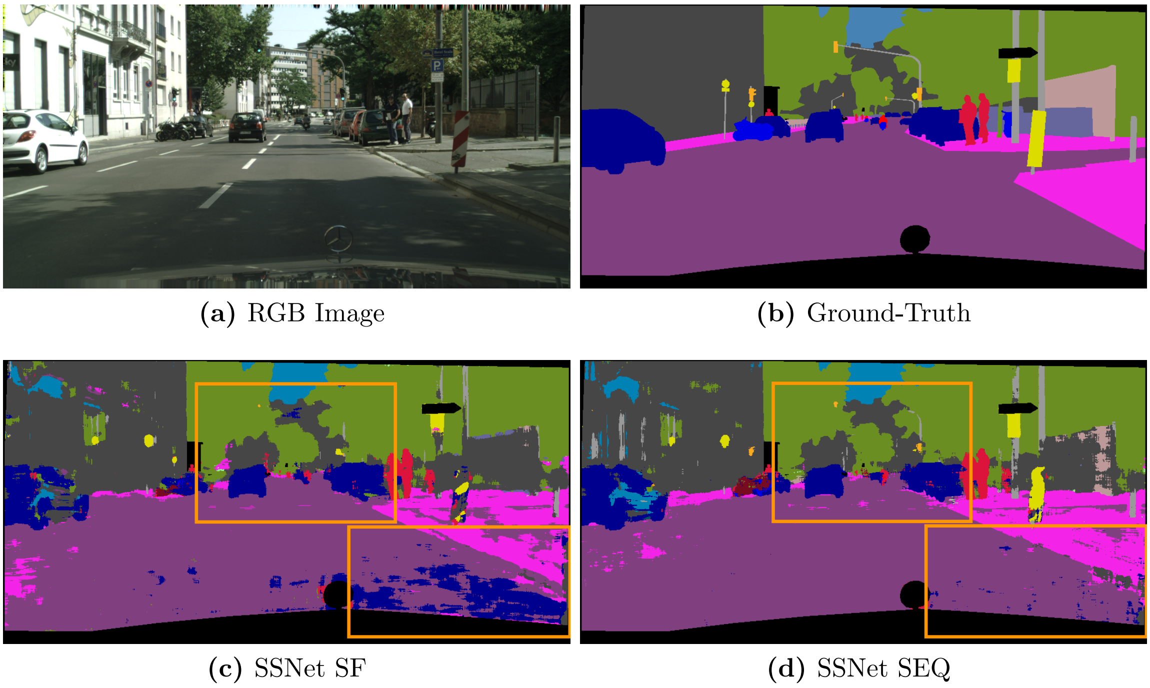
Data Preparation The training data has RGB color channels and the intensity values have 8 bit precision. Before the data is handed over to the network it is normalized resulting in a 32 bit precision floating point numbers between $[0, 1]$. No additional preprocessing steps are applied. The original data resolution of the Cityscapes data set is 1024 × 2048 (height × width), however, we down-sample images to 512×1024 and 256×512 for training using bilinear interpolation. We also down-sample the corresponding ground-truth labels using nearest neighbor interpolation to ensure that each pixel has a designated class. All experiments at image size 512 × 1024 are referred to as high resolution experiments, whereas all experiments at image size 256 × 512 are referred to as low resolution experiments. For the sake of straightforward comparison, we do not perform any data augmentation steps.

Network Initialization For every image which we hand over to the network, a zero padding is added at each layer to ensure that the spatial dimensions stay constant throughout the convolutional layers. This is necessary because of the dense prediction task semantic segmentation. Since the network uses mostly rectified activation units we initialize the weights of these layers with the method suggested by He *et al.* [31]. Biases of all layers are initialized with zero. The output of our last layer is activated using a Softmax activation function to generate the output probability volume.

Learning Parameters During the training process the cross-entropy loss presented in Equation (4.15) is computed between the prediction and the ground-truth labels. Furthermore, different inconsistency loss terms are added to train for consistency using a single frame model. After the loss is back-propagated by the `autograd` package, we use the Adam optimizer with learning rate $\eta = 10^{-4}$ to update the parameters of our model. Early stopping defines the end of the training phase.

5.2.2 Results

After training on different settings, the learned parameters are evaluated using the official Cityscapes validation set. In case of SSNet evaluation, the validation set only contains single frame 512×1024 images. Without video data, the SSNet reaches 80.3 % prediction accuracy and 27.9 % class mIoU. When using all training video frames, results improve to 87.1 % Acc and 37.5 % mIoU. This indicates that the increase in training data already improves performance. Qualitative results are shown in Figure 5.1. It can be seen that the sequence training improves overall qualitative results as well. On the one hand larger semantic areas a classified homogeneously, on the other hand, small areas in the distance are more accurate as well. Overall it can be seen that the network has difficulties identifying large objects, which is a consequence of the relatively small receptive field of 31×31 pixels. However, this network already delivers sufficient results for comparison with the VSSNet.



(a) RGB Image

(b) Ground-Truth

(c) SSNet SF

(d) SSNet SEQ

Figure 5.1: Results of SSNet. RGB input (a) and ground-truth semantics (b) from the Cityscapes validation set. We compare quantitative results of the single frame trained SSNet (c) to the sequence trained SSNet (d). Adding sequence data improves street class prediction (bottom right box) and also details in the distance are more accurate (centered box).

5.3 SSNet with Temporal Information Flow

In order to provide more consistent outputs the additional time information is propagated in the VSSNet architecture. At the last layer of VSSNet a ConvLSTM is added to forward high-level features from previous scenes to the current scene. In this multiple frame setting data has to be prepared in sequence to allow the gradient to optimize the parameters over consecutive frames. We compare different image resolutions, stateful versus stateless LSTM cells and a loss function with an inconsistency term. Adding the ConvLSTM layer also requires to adopt the training process. The parameters of the network are optimized using multiple time steps as input.

5.3.1 Validation Metrics

The results which we obtain are reported using four metrics: Mean Intersection over Union (mIoU), Accuracy (Acc), Consistent pixels (Cons) and Consistent but semantically wrong pixels (ConsW). MIoU and Acc are defined in Equation (3.2) and Equation (3.1). Cons is given by the number of pixels which keep the same semantic label in two consecutive images. Only pixels which need to have the same label according to the ground-truth are considered. We define ConsW as a subset of Cons, which consists of pixels with the wrong semantic label. Therefore, a lower ConsW score is considered better. We report all four metrics in percent.

5.3.2 Training with LSTM

We generate semantically labeled Cityscapes sequence data by the DeepLab oracle. Altogether 2,975 scenes where each scene consists of 30 frames are used for training the VSS-Net. Each scene is separated into sequences consisting of 2 to 10 frames. These smaller sequences allow LSTM parameter optimization for multiple time steps in a single training step. If we choose sequence length 5 and no overlap between consecutive sequences, we will have 6 sequences in each scene. The sequences can be seen as a subdivision of the scene. To avoid overfitting to the current scene, the sequences are processed in frame number order. This means that at the first training stage, sequence one of each scene is passed to the network. At the second training stage, sequence two of each scene is processed. At the end of each training iteration the last sequence of each scene is processed.

One problem of this approach is that during stateful LSTM training, the cell and hidden LSTM states need to be saved for each scene. However, this can not be done in 32 GB memory. Therefore, we introduce block training, where each block contains states of up to 1,000 scenes. This avoids overfitting to a single scene while also allowing states to maintain in fast CPU memory.

Res 128 × 256 and Res 512 × 1024 Training with higher resolution produces better results. After training with an input resolution of 128×256, we achieve 31.9 % mIoU and 82.5 % Acc. Switching to input size 512×1024 increases performance to 38 % mIoU and 87 % Acc. Results are compared at the validation resolution 512×1024. We upscale the segmentation outputs to this resolution with nearest neighbor interpolation.

Next, if we additionally enable the squared difference inconsistency loss term by setting $\lambda_{\text{incons}} = 10$ in the objective function, we observe more consistent outputs. At resolution 128×256 consistency increases from 97 % to 99 % and at 512×1024 consistency increases from 95.2 % to 98.2 %. Detailed results are shown in Table 5.1.

State Transistion We compare stateful and stateless training of the LSTM cell (Section 4.3). With stateless training the LSTM cell is initialized new after every sequence, whereas stateful training transfers the states trough all 30 scene frames. For this experiment, we use high resolution and no consistency loss. Results show that consistency and accuracy increase from 86.8 and 92.5 to 88.4 and 95.2 % respectively when training statefully. Therefore, the following trainings are performed with stateful LSTM cells.

Experiment	**Training**	**mIoU**	**Acc**	**Cons**	**ConsW**
Res 128 × 256	VSSNet No Cons	31.8	81.6	94.0	7.9
	VSSNet Cons	**31.9**	**82.5**	**96.0**	**7.8**
Res 512 × 1024	VSSNet No Cons	**34.2**	**88.4**	95.2	**6.4**
	VSSNet Cons	29.1	87.6	**98.2**	8.3
State Transition	VSSNet Stateless	33.7	86.8	92.5	6.5
	VSSNet Stateful	**34.2**	**88.4**	**95.2**	**6.4**

Table 5.1: VSSNet Results. The first two experiments compare a loss function with and without consistency term. We observe that training with consistency loss improves segmentation consistency on the validation set. The third experiment compares stateless with stateful LSTM training. Stateful training outperforms stateless training in every metric. All results are reported at high resolution.

Compared to the SSNet trained with sequence data, the VSSNet improves single frame accuracy and consistency. A qualitative comparison is shown in Figure 5.2. We found that extending the SSNet with a LSTM cell improves semantic segmentation results. More specifically, experiments with high input resolution give better mIoU. Furthermore, stateful training outperforms stateless training and the addition of a inconsistency error to the cross entropy loss function guides the network towards frame-to-frame consistent prediction. Having observed that the methods of this thesis can be applied successfully on the simple SSNet architecture, the next section shows results of implementing the methods on the state-of-the-art ESPNet architecture.

5.4 Semantic Segmentation with ESPNet

After validating the methods on the VSSNet, the ESPNet architecture is implemented as well. In contrast to the results reported in [66], no data augmentations *i.e.*, horizontal flipping and scaling to different resolutions are used. Furthermore, the training data is not transformed during preprocessing to have zero mean and unit variance, but only normalized to the range [0,1].

Firstly, performance on single frame data is compared. Then, sequence data is used to inspect the impact of the inaccurate DeepLab oracle. Finally, a consistency loss is added to train the single frame network for consistency without a LSTM layer. The experiments are conducted on the resolutions 256 × 512 and 512 × 1024.

First of all, the ESPNet single frame training results in 46.1 % mIoU compared to 27.9 % mIoU for the SSNet at high resolution. When validating on low resolution, the ESPNet still achieves 36.6 % mIoU. The results show that the network is designed for 512 × 1024 dimensional validation images. Because no augmentations are used, the mIoU

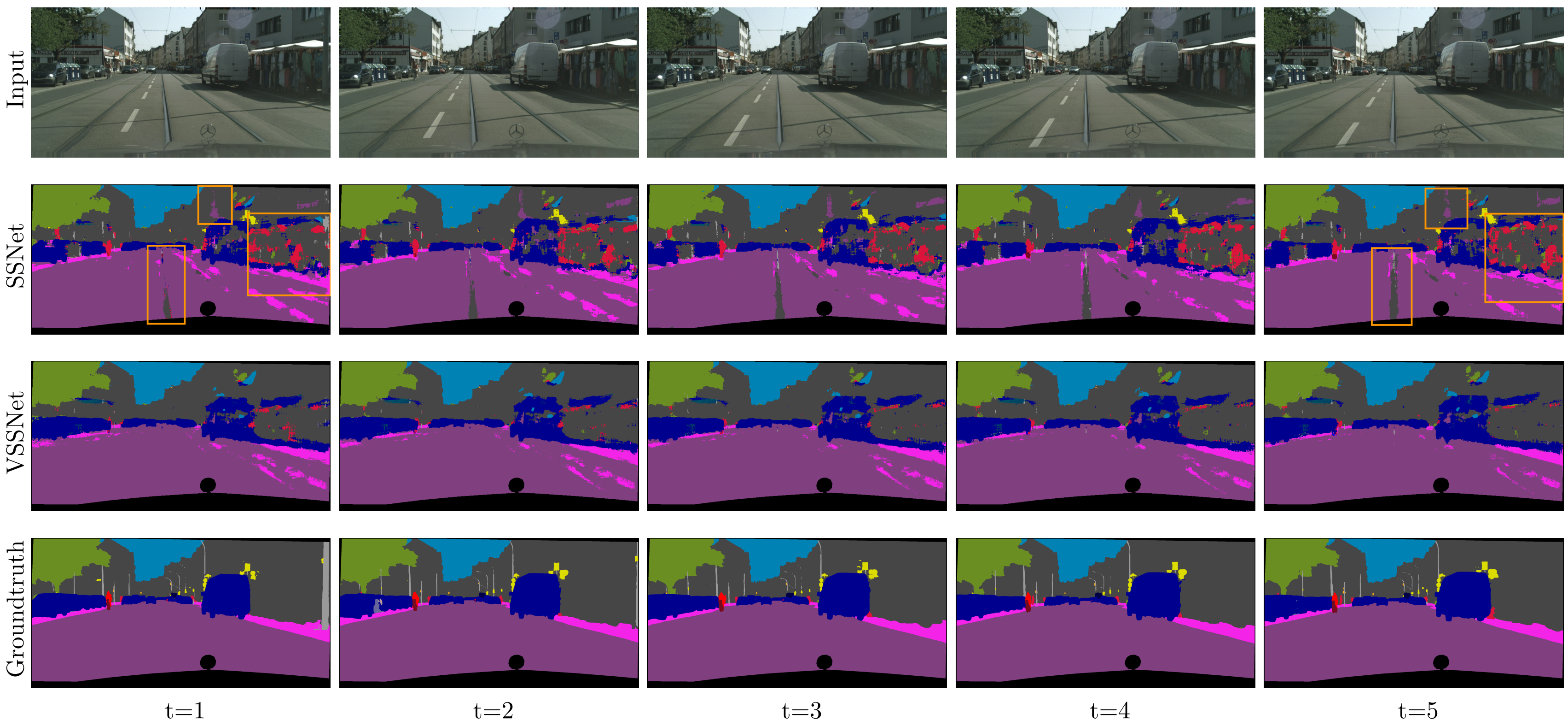


Figure 5.2: Consistency VSSNet. Comparison of SSNet (second row), VSSNet (third row) and Groundtruth (bottom) with five consecutive frames. The VSSNet gives more accurate and consistent results compared to the SSNet. This can be seen especially at the bottom (road) and at the right (building) inside the scene.

Experiment	Training	mIoU	Acc	Cons	ConsW
Res 256 × 512	ESPNet SF Data	36.6	86.2	-	-
	ESPNet SEQ Data	**47.5**	**90.5**	-	-
Res 512 × 1024	ESPNet SF Data	46.1	90.4	-	-
	ESPNet SF + Augm Data	56.3*	-	-	-
	ESPNet SEQ Data	**57.4**	**92.8**	-	-
Cons Training	ESPNet No Cons	**56.5**	**92.7**	97.6	2.6
	ESPNet Cons	50.5	91.6	**98.2**	**3.4**

Table 5.2: ESPNet Results. The first two experiments show that results are more accurate when using sequence training data, although the sequence ground-truth is obtained by the not perfectly accurate DeepLab prediction (* from [66]). Furthermore, the ESPNet performs better on high (512 × 1024) resolution, rather than on low (256 × 512) resolution. The third experiment indicates that the ESPNet can be trained to achieve higher (98.2 vs 97.6 %) frame-to-frame consistency, however this comes in together with a mIoU loss (50.5 vs 56.5 %).

of 56.3 % reported in [66] is not reached. However, this changes when adding sequence data.

5.4.1 Sequence Training

We use sequence data to train the single frame ESPNet without any further modification of the architecture. An overview of the experiments conduced is shown in Table 5.2.

Res 256 × 512 and Res 512 × 1024 Although, ground-truth oracle segmentation is not perfectly accurate, the mIoU increases to 57.4 % with 93.2 % in accuracy. It outperforms single frame results achieved with data augmentation by 1.1 %. At low resolution results improve to 47.5 % mIoU and 90.5 % Acc. It is concluded that the additional training frames improve prediction accuracy although the provided ground-truth is not completely accurate.

Cons Training To achieve consistent segmentation over all frames in every scene, the inconsistency error is enabled. This experiment was performed on high resolution using the squared difference error on correctly predicted pixels with $\lambda_{\mathbf{incons}} = 20$. Consistency increases from 97.6 % to 98.2 % while mIoU drops from 56.5 % to 50.5 %. We found that more consistency can only be reached by losing accuracy at smaller regions with the single frame architecture.

In summary, the ESPNet architecture outperforms the suggested SSNet architecture significantly, although the number of parameters is similar. When training with all 30 frames per scene, the results improve compared to the single frame training, despite the inaccurate ground-truth generated. It is possible to optimize the ESPNet for frame-to-frame consistent prediction, however details about smaller semantic regions get lost.

5.5 ESPNet with Temporal Information Flow

The methods explained in Chapter 4 are also applied to the ESPNet to allow the features to be propagated over multiple time steps in a scene. Firstly, we test ESPNet_T architecture, where a final layer is added to ESPNet. This layer performs a convolution over the last t time steps which are concatenated in the channel dimension. Secondly, we also test ESPNet_L_ architectures which we create by adding a ConvLSTM layer at different positions. Furthermore, different configurations of the ConvLSTM layer are compared. Additionally, we visualize the parameters of the LSTM cell and how the features evolve in cell and hidden state over time.

Compared to ESPNet training which we perform in a single stage, the training with temporal flow takes three stages. At first stage, the ESPNet is trained to only optimize accuracy on semantic segmentation. At second stage, the architecture is modified to include temporal information flow. Only the new parameters are trained, while the rest of the parameters stay untouched. At third stage, all parameters are fine-tuned to optimize prediction for both semantic accuracy and consistency. The benefits of this procedure are presented in Section 5.5.2.

5.5.1 ConvTime Layer

We add the ConvTime layers presented in Section 4.4 after the last layer of ESPNet. We call the resulting architectures ESPNet_T and ESPNet_T2. ESPNet_T outperforms ESPNet_T2 significantly. On 512 × 1024 resolution the ESPNet_T achieves 50.3 % mIoU while ESPNet_T2 only reaches 38 % mIoU. However, both networks achieve 98.3 % consistency. The mIoU can be improved to 52.1 % by increasing the filter size from 3 × 3 to 7 × 7. Although the ConvTime layer increases prediction consistency, the ConvLSTM layer included in the ESPNet_L1a architecture produces even better results. A comparison is given in Table 5.3.

5.5.2 ConvLSTM Layer

Various experiments are conducted on ConvLSTM settings. We first focus on different training settings, then parameters withing the ConvLSTM layer are adjusted and finally, different inconsistency loss functions are compared. The results are grouped into categories. A comparison across categories is not possible, because the training setting varies

Experiment	Training	mIoU	Acc	Cons	ConsW
ESPNet_T 5 × 5 (+45K)	ESPNet_T Stage 2	45.4	89.6	97.8	3.9
	ESPNet_T Stage 3	**49.2**	90.9	97.8	3.7
ESPNet_L1a 5 × 5 (+72K)	ESPNet_L1a Stage 2	47.1	90.6	98.3	3.5
	ESPNet_L1a Stage 3	48.8	**91.0**	**98.4**	3.3
ESPNet_L1a 3 × 3 (+26K)	ESPNet_L1a Stage 2	45.9	90.5	98.2	3.5
	ESPNet_L1a Stage 3	48.7	**91.0**	**98.4**	**3.2**

Table 5.3: ConvTime vs ConvLSTM. This table compares the ConvTime layer with the ConvLSTM layer on the ESPNet architecture. The first experiment shows results of ESPNet_T training, while the second and the third experiments show scores of the ESPNet_L1a architecture with filter sizes 5 × 5 and 3 × 3. The number in parenthesis indicates the additional parameters needed for this layer. Although the ESPNet_T achieves a slightly better mIoU score, overall consistency performance is significantly better with the ConvLSTM layer.

for different categories. All quantitative results of the ConvLSTM experiments are summarized in Table 5.4. Additionally, the qualitative improvements of using sequence data and ConvLSTM architecture with consistency loss are shown in Figure 5.4

Training Stages Firstly, training with 3 stages is compared to 2 stage training. The difference is that 3 stage training has the additional stage 2 in which only the ConvLSTM layer is optimized and all other layers are frozen. 3 stage training results in 49.6 % mIoU while 2 stage training only achieves 47.1 % mIoU. It also outperforms in the other metrics. Both training methods are compared on the ESPNet_L1a architecture with high resolution input images.

Data Size Secondly, we reduce the number of scenes in the training data set from 2,975 to 1,000. This results in a performance drop from 49.6 to 29.2 % mIoU (-20.4 p.p.) on the ESPNet_L1a. On the single frame ESPNet the performance only decreases by 15.5 p.p. (56.5 to 41.0). It is concluded that the LSTM is more prone to over-fitting. Therefore, low resolution is used to save training time instead of a smaller data set.

Sequence Length Thirdly, the sequence length determines how many consecutive time steps are given as an input in a single gradient descent step. Sequence length 5 works better than sequence length 2 in terms of mIoU (45.2 vs 44.5). When comparing consistency, the gap is even larger (97.2 vs 96.6). This suggests that a large sequence size is especially important for the LSTM to learn consistency over multiple input frames. Both experiments are trained on high resolution using a 3 × 3 ConvLSTM filter.

ConvLSTM Init Next, the initialization of cell and hidden LSTM state is also compared. The zero-initialization reaches 49.1 % mIoU and 98.3 % consistency, whereas a learned initialization reaches 49.3 % mIoU and 98.4 % consistency. Most of the other experiments we perform use learned initial states, because it gives slightly better results. Both results are produced with LSTM only training and squared difference consistency loss.

Layer Types Additionally, the best ConvLSTM layer is determined by comparing the three methods suggested in Section 4.3. The convolution over all input channels performs best in every metric with 46.5 % mIoU. The second method achieves 45.2 % mIoU which is slightly worse, however it needs less parameters. The third method reaches 42.5 % which is significantly lower than the other two methods. Method 2 can be implemented efficiently in PyTorch by using a convolution with the groups parameter. This makes it very useful especially for the ESPNet_L1a architecture.

LSTM Position 5 × 5 and LSTM Position Eq Params Another question which needs to be answered is where to position the ConvLSTM layer in the ESPNet. Four different possibilities are introduced in Section 4.3.2. At the comparison with full resolution, 2 stage training and 5 × 5 filter size, ESPNet_L1b performs best with 53.0 % mIoU. The other architectures are less accurate with 50.1, 49.6 and 47.1 % mIoU for ESPNet_L1c, ESPNet_L1d and ESPNet_L1a. Similarly, when testing on different filter sizes for each architecture to equalize the number of parameters, ESPNet_L1b produces the best results. These results suggest, that it is best to include the ConvLSTM layer at a high feature level with low spatial dimensions. At this place, the LSTM is able to achieve highest consistency and also best accuracy.

Inconsistency Loss Furthermore, the inconsistency term in the objective function can also be modified. When considering a different error function, results are quite similar. The ℓ_1-norm and the ℓ_2-norm reach about 49 % mIoU and 98.5 % consistency. Figure 5.3 compares the pixel loss for the different loss functions:

- Subfigure (1) shows all pixels which should be labeled the same in both images, but are predicted differently.

- Subfigure (2) penalizes differences on all pixels which are predicted with a certainty larger than 20 % (Threshold 20) in both images by the absolute function over all classes. The pixel error is computed by the sum over all classes.

- Subfigure (3) computes the absolute difference on inconsistent pixels which where labeled correctly in one image (Equation (4.21)).

- Subfigure (4) uses the absolute function applied on the difference of all class probabilities of pixels which should be consistent. The pixel loss is given by the sum over all classes (Equation (4.17)).
- Subfigure (5) computes the squared difference on inconsistent pixels which where labeled correctly in one image (Equation (4.21)).
- Subfigure (6) uses the squared difference of all class probabilities of pixels which should be consistent. The pixel loss is given by the sum over all classes (Equation (4.17)).

For most of our experiments we use the Squared Difference True loss function, because it only penalizes pixels which are inconsistent in the prediction. This inconsistency loss function also assigns a higher error to outliers compared to the absolute function.

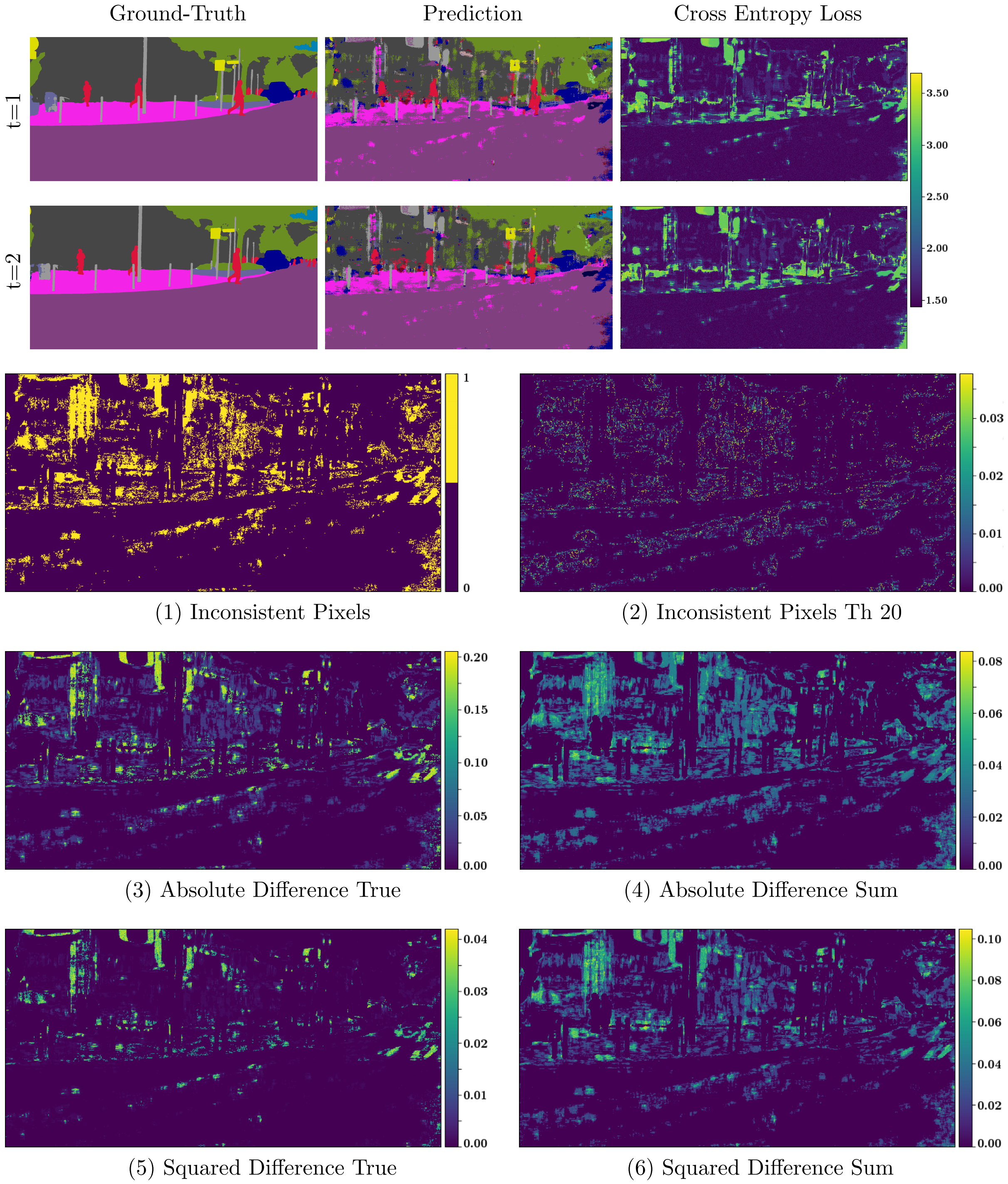
Inconsistency Lambda Finally, we modify the hyper-parameter λ_{incons} to force more or less consistent results. Setting $\lambda_{\text{incons}} = 0$ results in 49.0 % mIoU but only 98.0 % Cons. Choosing $\lambda_{\text{incons}} = \{10, 20\}$ slightly decreases mIoU but increases consistency to 98.4 %. A higher parameter λ_{incons} does improve consistency only slightly, but mIoU degrades rapidly. We observe that $\lambda_{\text{incons}} = [10, 20]$ produces best results for this domain.

Best Results We present the best results in the last line of Table 5.7. The corresponding qualitative results are shown in the last row of Figure 5.4. We achieve 57.9 % mIoU, 93.0 % Acc, 98.7 % Cons and 2.7 % ConsW computed at full Cityscapes resolution (2048 × 1024). To reach this accuracy, we train the ESPNet_L1b at high resolution (1024 × 512) without data augmentation. The training is performed in three stages. In the first stage, we train the ESPNet with video data. In the second stage we add the ConvLSTM layer and train only this layer. In the final stage we train the parameters of the whole network. The squared difference consistency loss depicted in Equation (4.21) is enabled with $\lambda_{\text{incons}} = 20$ in stages two and three. We stop the training after about two million image batches.

5.5.3 Network Visualization

We want to get an insight into the learning process to get a better overall understanding. Although this cannot be done easily on deep neural networks, there are possibilities to visualize learned filters and how states change over time. In addition, the loss function computation can be observed and multiple error functions can be compared. The most interesting visualizations on the ESPNet are shown in this section.

First, we visualize the LSTM of the ESPNet_L1a architecture. This architecture is chosen because the ConvLSTM layer operates on the last layer which creates the output probability volume. Therefore, results can be interpreted easily, since the 19 output channels can be directly mapped to the corresponding semantic classes. The propagation of cell and hidden states at an example scene is shown in Figure 5.5.



(1) Inconsistent Pixels

(2) Inconsistent Pixels Th 20

(3) Absolute Difference True

(4) Absolute Difference Sum

(5) Squared Difference True

(6) Squared Difference Sum

Figure 5.3: Inconsistency Loss Functions. In the upper part, ground-truth (left), prediction (center) and cross entropy loss (right) of two consecutive images is shown. The lower part compares different error functions which penalize the inconsistencies between predictions of the two time steps on top.

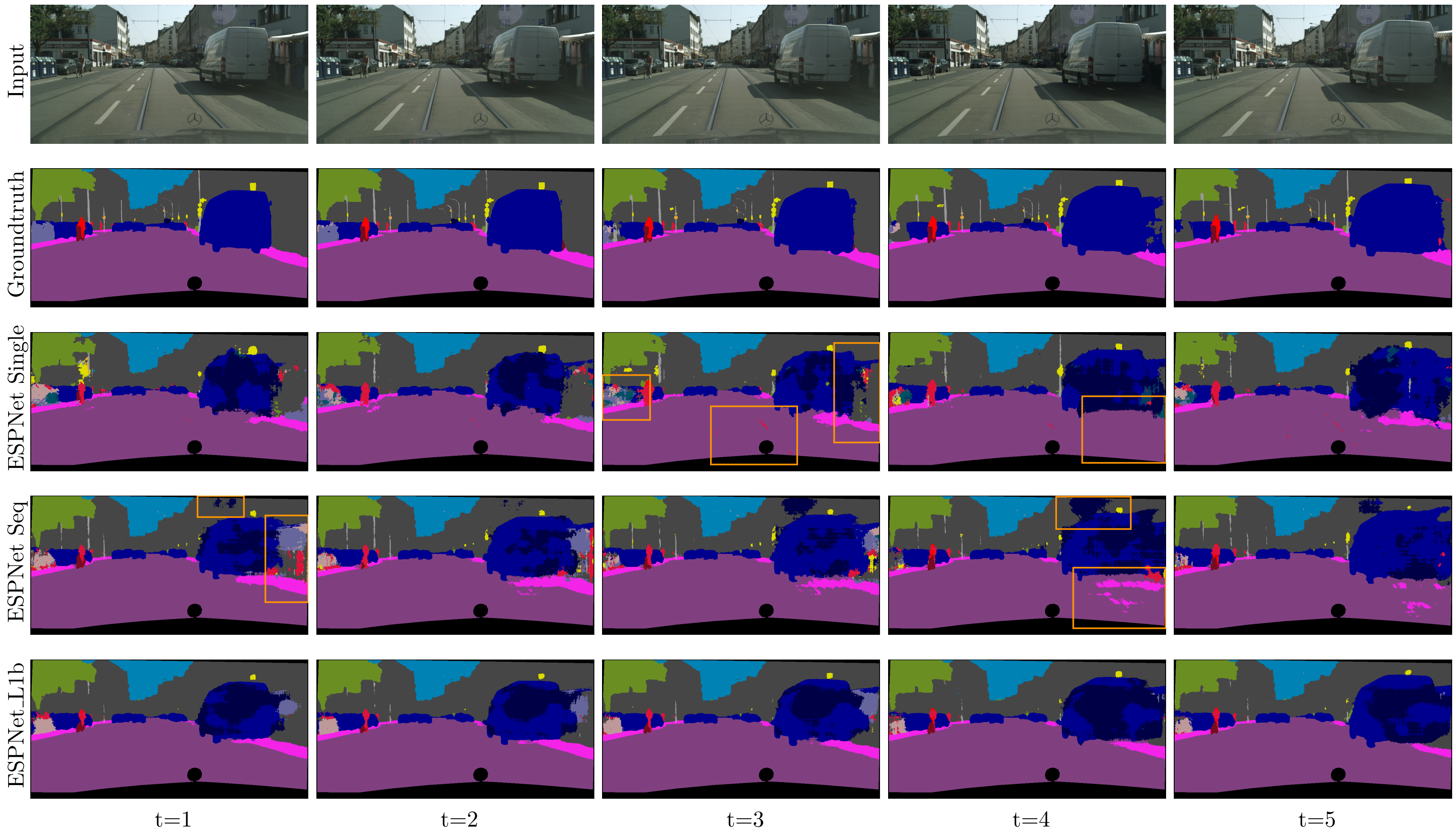


Figure 5.4: Consistency ESPNet. A comparison between input data, DeepLab ground-truth, single frame training, sequence training and LSTM training on the ESPNet (top to bottom). The horizontal axis represent the time steps. Areas with inaccurate predictions are highlighted by orange boxes. The ESPNet with single frame training shows inconsistencies at the right, left and on the road segmentation. Training with multiple frames improves overall results, but still shown inconsistencies especially in the right part of the image. The ESPNe_L1b produces the most accurate and consistent results.

	Category	**Experiment**	**mIoU**	**Acc**	**Cons**	**ConsW**
Training Setting	Training Stages	ESPNet_L1a 2 Stages	47.1	89.2	97.8	5.6
		ESPNet_L1a 3 Stages	**49.6**	**90.8**	**98.5**	**4.4**
	Data Size	ESPNet_L1a Size 1,000	29.2	79.2	94.7	13.7
		ESPNet_L1a Size 2,975	**49.6**	**90.8**	**98.5**	**4.4**
	Sequence Length	ESPNet_L1a Seq 2	44.5	87.9	96.6	6.3
		ESPNet_L1a Seq 5	**45.2**	**89**	**97.2**	**5.5**
LSTM Cell	ConvLSTM Init	ESPNet_L1a Zero	49.1	90.9	98.3	3.5
		ESPNet_L1a Learn	**49.3**	**91.0**	**98.4**	**3.4**
	Layer Types	ESPNet_L1a Type 1	**46.5**	**89.4**	**97.6**	**5.4**
		ESPNet_L1a Type 2	45.2	89.0	97.2	5.5
		ESPNet_L1a Type 3	42.5	87.7	96.9	6.5
LSTM Position in ESPNet	LSTM Position 5 × 5	ESPNet_L1a	47.1	89.2	97.8	5.6
		ESPNet_L1b	**53.0**	**91.3**	**98.5**	4.0
		ESPNet_L1c	50.1	91.1	98.1	**3.9**
		ESPNet_L1d	49.6	91.1	98.1	4.0
	LSTM Position Eq Params	ESPNet_L1a 7 × 7	50.3	91.4	98.5	3.1
		ESPNet_L1b 3 × 3	**52.0**	**91.5**	**98.7**	3.2
		ESPNet_L1c 5 × 5	49.9	91.4	98.2	3.0
		ESPNet_L1d 9 × 9	50.1	**91.5**	98.3	**2.9**
Loss Function	Inconsistency Loss	Sq Diff True	**48.8**	**90.9**	98.4	**3.5**
		Abs Diff True	48.6	**90.9**	**98.6**	**3.5**
		Abs Diff Thresh 20 %	47.9	90.7	98.4	3.6
	Inconsistency Lambda	$\lambda_{\text{incons}} = 0$	**49.0**	**90.9**	98.0	**3.4**
		$\lambda_{\text{incons}} = 10$	48.8	90.9	**98.4**	3.5
		$\lambda_{\text{incons}} = 20$	48.5	90.9	**98.4**	3.6
		$\lambda_{\text{incons}} = 100$	46.3	90.4	**98.6**	3.7
		$\lambda_{\text{incons}} = 1,000$	31.1	87.1	**98.6**	5.9
	Best Results	**ESPNet_L1b**	**57.9**	**93.0**	**98.7**	**2.7**

Table 5.4: ConvLSTM Results. Nine categories with different ESPNet_L training configuration tests are shown in this table. The first three focus on the overall training setting. Categories 4 and 5 focus on how to configure the ConvLSTM layer, while categories 6 and 7 (ConvLSTM Position) show the results of four ConvLSTM layer positions within the ESPNet. The last two categories test variations of the inconsistency loss function. The best results of each category are highlighted in bold. The last row shows the best results we are able to achieve by combining the insights from the experiments.

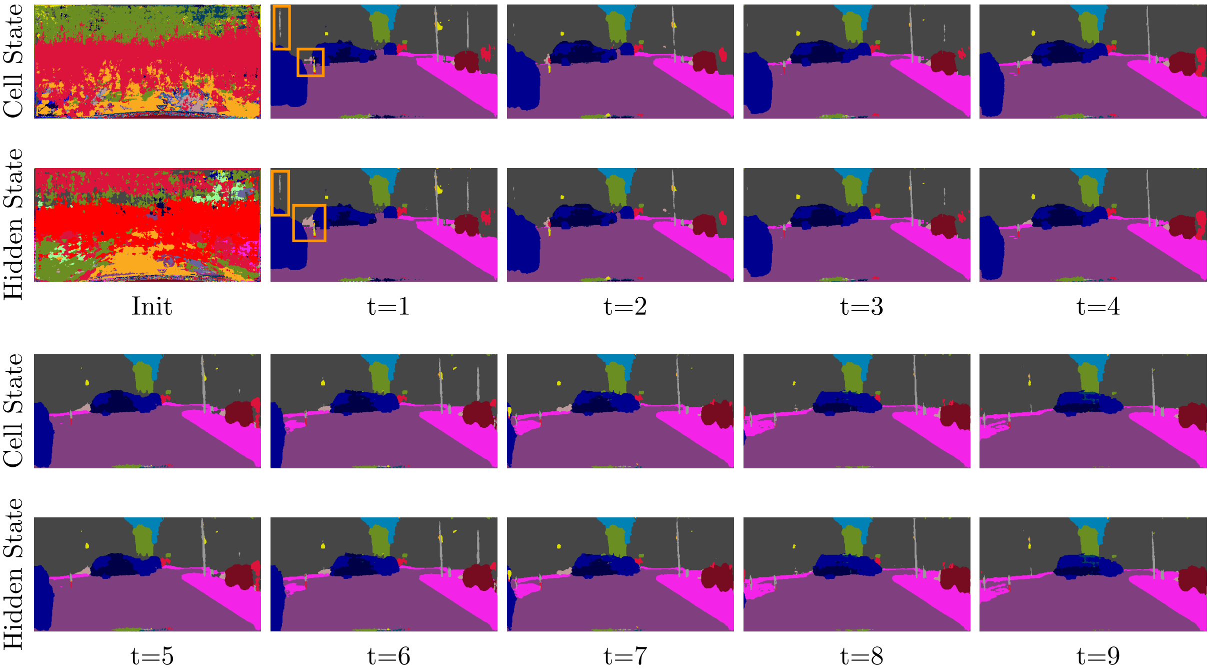


Figure 5.5: LSTM State Propagation. The first nine ConvLSTM states of the ESPNet_L1a architecture on a validation scene are shown. At the top left, the learned initial cell and hidden states are depicted. Already after the first time step in the scene, both states represent the output of that time step. The prediction of cell and hidden states behave similar throughout the scene. Few minor differences which can be seen only at the beginning of the scene are highlighted by orange boxes.

Different loss functions which penalize errors by the cross entropy function, ℓ_1-norm or ℓ_2-norm are compared in Figure 5.3. We visualize them to outline the differences and to decide on the most suitable inconsistency loss. The effect of dilation on the difference image of the groundtruth between two consecutive time steps is shown in Figure 5.6:

Subfigure (1) shows one iterations of a dilation on the ground-truth difference image using the four pixel neighborhood. (2) Inconsistent pixels after removing the dilated ground-truth pixels are marked in yellow (Consistency 90.1 %).

Subfigure (3) shows one iterations of a dilation on the ground-truth difference image using the eight pixel neighborhood. (4) Inconsistent pixels after removing the dilated ground-truth pixels are marked in yellow (Consistency 90.3 %).

Subfigure (5) shows two iterations of a dilation on the ground-truth difference image using the four pixel neighborhood. (6) Inconsistent pixels after removing the dilated ground-truth pixels are marked in yellow (Consistency 90.8 %). This dilation gives the best results, because classes with imprecise edges *e.g.* trees are not categorized as inconsistencies.

We need to compute the difference image between two consecutive images of the

Map Names	Town01	Town02	Town03	Town04	Town05
Working Spawn Points	153	83	113	115	145
Master Spawn Points	130	70	100	100	120
Vehicles in Map	150	80	300	300	300
Weather IDs	1, 2, 3, 4, 7, 8, 9, 10, 11				
Time After Spawn	[3, 10] sec				

Table 5.5: Carla Generation Settings. The settings used to generate synthetic data are summarized in this table. The first four towns are used for training whereas Town05 is used for validation. Altogether 4680 scenes are created with this settings.

groundtruth to find pixels which should not be consistent. The difference image shows the semantically changing pixels in the scene between two time steps. It is important to determine which pixels should be affected by the inconsistency loss. Pixel coordinates which alter semantics in the groundtruth are never subject to the inconsistency loss function. The idea of the dilation is to compensate possibly inaccurate groundtruth generated by DeepLab (Section 4.2.1).

5.6 Adding Synthetic Data

One disadvantage of the previously conducted supervised experiments is that they all rely on the ground-truth created by the DeepLab model. Although results of this model are fairly accurate and state-of-art on the Cityscapes high-score list, the inaccuracies occur often at edges of objects. This fact makes it more difficult for a ConvLSTM to learn motion boundaries. In order to tackle this problem, proportions of a synthetic data set created by the Carla simulator are added to the training data set. Since this data has perfectly accurate ground-truth it should allow the ConvLSTM to learn the exact motion boundaries and lead towards more consistent prediction.

The Carla server is running on the GPU while the Python client connects, defines the settings and captures data. An overview about which settings are used is shown in Table 5.5. Altogether five different maps are used, where the last one (Town05) generates test data. Within each map about 180 predefined spawn points exist, where a vehicle can start its route. Note that not all spawn points can be used, because no route is defined for some of them. Therefore, a list is generated which only contains the working spawn points. The resulting working spawn points are used to create traffic on the roads. From 80 up to 300 cars are spawned depending on the size of the map to create traffic on the otherwise empty map.

After traffic is created by spawning vehicles, the server is set to synchronous mode to pause simulation and only continue on request by the Python client. Now the master

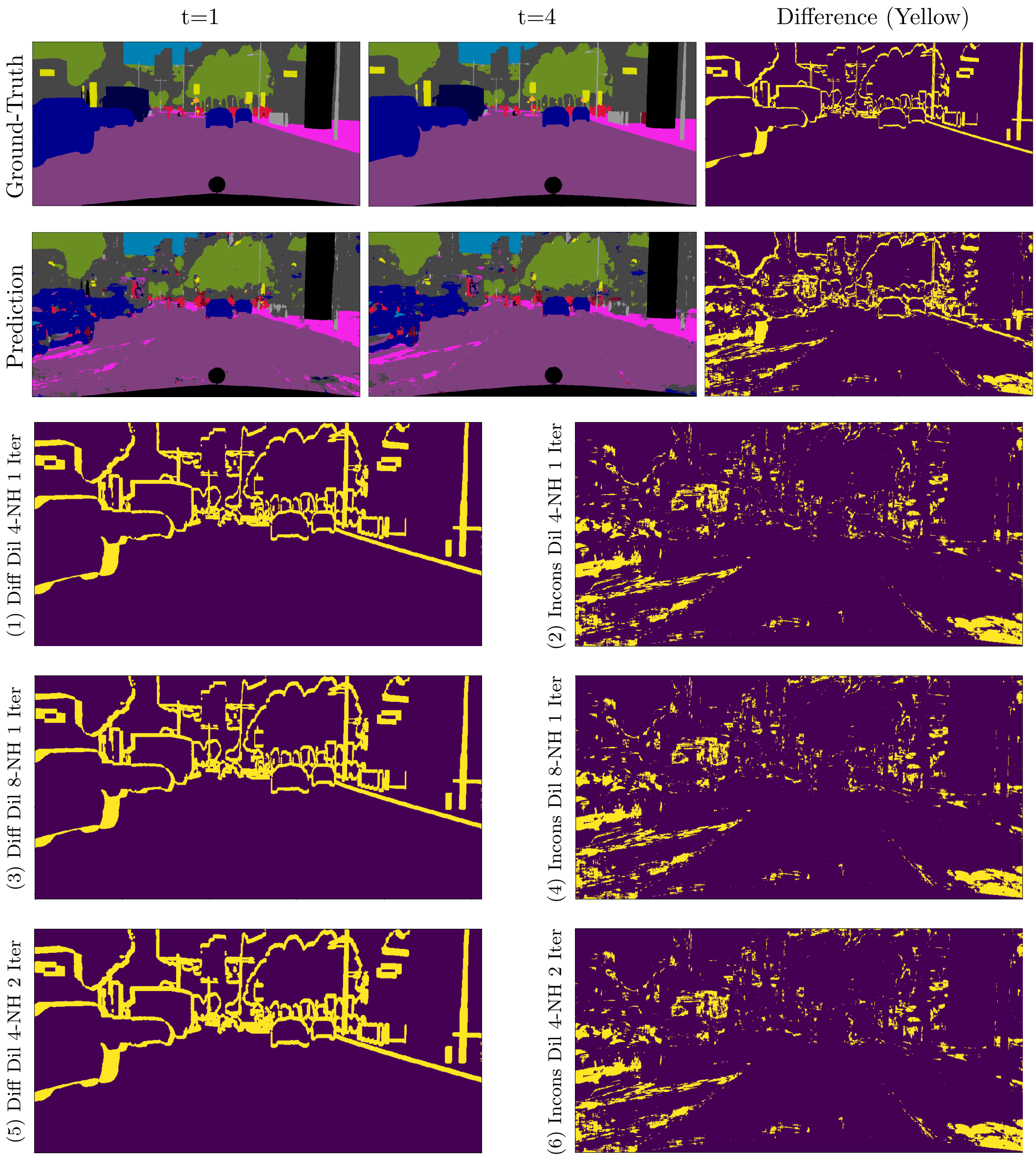


Figure 5.6: Dilation on Motion. The upper part shows semantic ground-truth and prediction of two images from the same scene, but at two different time steps. Additionally, the pixels where the semantic label has changed over time are shown (right). When subtracting the ground-truth difference from the prediction difference, the number of inconsistent pixels can be computed. In this example, 89.1 % are predicted consistent after removing the ground-truth differences throughout the two time steps. The lower part shows how the ground-truth difference image can be modified by the morphological operation dilation to overcome inaccurate edge segmentation by the prediction oracle.

Data Set	**Data Ratio**	**mIoU**	**Acc**	**Cons**	**ConsW**
Cityscapes : Carla Data	100:0	**48.5**	90.9	98.4	3.6
	90:10	**48.5**	**91.0**	**98.5**	**3.3**
	80:20	47.8	90.6	98.4	3.6
	50:50	45.1	89.9	98.2	3.9

Table 5.6: Synthetic Data Results. This table outlines the impact of gradually adding more data samples generated from the Carla simulator. The number of data samples from the Cityscapes is kept constant at 2,975 scenes. Using about 10 % synthetic data slightly improves the results. When using more than 20 % of synthetic data, the performance on the Cityscapes validation set declines significantly.

vehicle which has mounted a camera to generate data is spawned on one of the free spawn points. The master vehicle drives for a random time span from 3 to 10 seconds on the map before recording a scene. A scene consists of 30 frames recorded at 17 FPS. Both RGB data and semantic segmented data is stored to the disk. After the scene recording is finished, the master vehicle is removed from the map and re-spawns at the next free position. This process is repeated until the number of total master spawn points is reached.

The procedure of recording scenes is repeated until most spawn points of the map are covered. Once finished, the map is reset and the experiment is repeated using a different weather setting. Altogether nine weather settings are used. Finally, the procedure is applied to all maps, which allows 4,380 scenes and 140,400 RGB images to be created.

Since the performance of the model is evaluated on the Cityscapes data set, the synthetically created data is only added in some fractions to the Cityscapes training set. Otherwise the domain gap would be too large because not all Cityscapes classes can be generated by the Carla simulator. Results show that about 10 % simulation data added can have a positive impact on the validation scores. The accuracy increase slightly from 90.9 % to 91.0 % and the consistency increases from from 98.4 % to 98.5 % when adding 325 randomly selected Carla scenes. However, adding more synthetic scenes lowers the mIoU score dramatically. Detailed results are shown in Table 5.6. It is concluded that the addition of a few synthetic scenes can improve prediction scores by a small amount.

To summarize this chapter, our experiments are executed on two Nvidia consumer GPUs using CUDA and the PyTorch deep learning framework. In the simple setting, we compute semantic segmentation using the SSNet architecture. This simple architecture classifies about 80 % of all pixels correctly. After we added a ConvLSTM layer to the SSNet and used video data for training results improve further. The suggested inconsistency loss function allow the VSSNet architecture to perform frame-to-frame consistent semantic segmentation. This ensures that the suggested methods work on the small example architecture.

Training	**Architecture**	**mIoU**	**Acc**	**Cons**	**ConsW**
Single Data	SSNet	27.9	80.3	-	-
	ESPNet	**44.0**	**89.4**	-	-
Seq Data	SSNet	37.5	87.1	93.5	6.1
	ESPNet	**56.5**	**92.7**	**97.6**	**2.6**
Consistency	ESPNet consLoss	50.5	91.6	98.2	3.4
	ESPNet_L1b	**57.9**	**93.0**	**98.7**	**2.7**

Table 5.7: Path towards Consistency. The ESPNet architecture delivers much better (16.1 p.p. and 19 p.p.) results than SSNet, for both single data and sequence data. Single frame results do not contain consistency scores because they are evaluated on the single frame validation set. By using sequence data mIoU and accuracy already increase for the SSNet by 9.6 p.p and ESPNet by 12.5 p.p. Training the ESPNet with consistency loss improves consistency, but lowers mIoU. By adding a LSTM cell (ESPNet_L1b), we are able to increase accuracy and consistency.

In the next step, we test the methods on the state-of-the-art ESPNet architecture. A final quantitative comparison of these experiments is shown in Table 5.7. By training the ESPNet with video data, the officially reported performance is improved from 56.3 % to 57.4 % without any additional data augmentations. The extension of the ESPNet architecture to the ESPNet_L1b improves mIoU further to 57.9 % and also delivers high consistency over time at 98.7 %. The last line of Table 5.7 represents the best results we are able to achieve: 57.9 % mIoU, 93.0 % Acc, 98.7 % Cons and 2.7 % ConsW. The corresponding qualitative results are shown in the last row of Figure 5.4.

Visualizing intermediate network steps gives us insights into the prediction process and enhances understanding of the training process. The expansion of the training data set only improves results by less than one percent. Chapter 6 contains a final conclusion about the findings of this chapter.

6 Conclusion

In this thesis we aimed for physically plausible video segmentation by developing methods for frame-to-frame consistent semantic segmentation. One disadvantage of many state-of-the-art semantic segmentation algorithms is that they only process single images and do not use video information. We utilized this additional multiple frame information which is available in many applications to improve prediction results. In this work we mainly focused on the domain of street scenes because it is very important to have a consistent view of what is happening in traffic situations.

We used the method of convolutional neural networks in combination with long short-term memory cells to tackle the problem of frame-to-frame consistent semantic segmentation. The main task of the CNN architecture is to provide good semantic prediction whereas the LSTM should memorize the prediction of previous frames. We used the Cityscapes sequence dataset to train this architecture on street scenes. Additionally, we created synthetic data using the Carla simulator. We extended the cross-entropy loss which is usually used for classification problems with an inconsistency error term to guide the model towards consistent prediction.

Our results indicate that the use of multiple frames per scene indeed increases prediction accuracy, even when the validation set only consists of single frame images. By adjusting the architecture to allow for temporal information flow with ConvLSTM or ConvTime layers, we are able to improve prediction accuracy further. These layers increase consistency over time while maintaining segmentation details indicated by the mIoU score. Furthermore, the inconsistency error is an important tool which improves consistency quantitatively and qualitatively. It works on single frame networks as well as multiple frame architectures. The addition of synthetic data generated with the Carla simulator does only improve results slightly, when considering the Cityscapes validation set. Overall, the results show that the idea of using multiple frame information together with recurrent neural networks and an inconsistency loss function to enforce physically plausible predictions, works as indented. With this technique we are able to deliver consistent and accurate predictions with thin CNN architectures. We are able to outperform the state-of-the-art ESPNet video predictions significantly.

6.1 Positives and Negatives

It is important to point out positive and negative aspects about the methods used in this work. The positive aspects can be applied to other problems, whereas the negative ones need to be improved in the future.

Positives:

- Adding video data already improves validation results by a great amount. Although the video data is produced by DeepLab and therefore not perfectly accurate. Even when validating on hand-labeled single frame data.
- Enabling the inconsistency loss term improves consistency such that it can be observed immediately. It even works on single frame architectures.
- The ConvLSTM layer improves overall results even with a few number of parameters.
- The ConvTime layer improves results similar to the ConvLSTM layer.
- Methods proposed in this thesis can be transfered to different problems by adjusting some parameters and retraining the model. Within computer vision it can be applied to other deep learning architectures.
- The suggested methods work on the handcrafted SSNet and already deliver good results.
- Experiments work on the SSNet as well on the already optimized state-of-the-art ESPNet.

Negatives:

- Improvements through a ConvLSTM layer does also increase the number of parameters in the network. However, the total number of parameters is significantly lower that in many other architectures [11, 30].
- The ConvLSTM visualization only gives little insight into the learning process. Especially, high level feature propagation is hard to analyze.
- Adding synthetic data by the Carla simulator without preprocessing did not improve results significantly. Better outcomes might be achieved with in depth data analysis and preprocessing.

6.2 Limitations

Finally, we also point out limitations of the work in order to prevent misconceptions. The produced results are only valid on the street scenes produced by Cityscapes. Other domains might produce different results, although the methods should work similarly. Since the methods use supervised learning, training and ground-truth data are needed to predict semantics for other applications. For example, training with images taken on a sunny day does not deliver good segmentation results on test images taken on a cloudy day. Moreover, training the architecture for segmenting street scenes in New York City into 19 classes does not give any semantical understanding about 10 important classes when watching a tennis match.

When considering video data, it is important that the motion between consecutive images is small in a sense that pixels do not move large distances. Otherwise the ConvLSTM will not be able to determine the motion direction and accuracy will decrease. The ConvLSTM produces good results on street scene data with 17 FPS. Similarly, the resolution between training and test time should not differ significantly, although a change in resolution is possible because the network architecture is fully convolution. Changing the image resolution by a factor larger than two also reduces prediction accuracy.

Some of the important lessons learned when applying the methods are as follows. In order to get fast feedback on parameter changes it is important to keep to training setting as simple as possible. Image resolution and network size should be small in order to train for consistency in a video setting. Overnight convergence ensures that multiple experiments can be run. When considering the ESPNet architectures, the ESPNet-C architecture is most efficient for video experiments, because a decoder adds unnecessary complexity.

In conclusion, the proposed methods delivered consistent and correct semantic image segmentation and are able to outperform state-of-the-art algorithms. It would be interesting to apply the techniques to different image understanding algorithms on various domains. Some ideas for future experiments and improvements are explained in Future Work (Chapter 7).

7 Future Work

At the end of this thesis we would like to share some ideas about what could be done in the future in the field of frame-to-frame consistent semantic segmentation. One idea is to provide additional features (*e.g.* optical flow) as LSTM input to enforce consistency. Additional features (*e.g.* disparity information or image gradients) can be processed by the whole network architecture to improve semantic segmentation accuracy. Furthermore, RGB input data can be preprocessed and augmented to create additional information. Popular examples of augmentation are horizontal flipping and scale to different resolution. Additionally, the overall network can be further tuned by adjusting the regularization hyper-parameters. Precomputed optical flow information can be used to wrap consecutive frames. The ConvLSTM would take two to the current time step warped frames as an input. After the warping the frames should be the same except for occlusions. This method would allow to maintain consistency at frames with large motion. It would be interesting to see how the LSTM performs on very long scenes. Other research areas indicate that the LSTM is capable of remembering information over hundreds of time steps.

An improvement of data preprocessing might allow to combine the synthetic Carla data with real-world Cityscapes data more effectively. In addition to preprocessing, the cross entropy loss function can be modified to equally penalize all classes independent of their size. This would avoid inaccuracies of classes which are not present in the Carla data set. It would also weight smaller classes equally, which could improve the mIoU metric. Additionally, other real-world data sets could be used [24, 68] to enforce generalization. Moreover, methods could be tested on different video domains with other semantic classes.

Different network architectures [85, 88] can be modified to allow for temporal feature propagation. This would indicate how well the ConvLSTM works on various architectures. Within those models the function of the ConvLSTM layer can also by replaced by a GRU, 3D-Convolution or by a depth-wise separable convolution.

The concept of neural networks could be replaced by more traditional methods such as graph cut algorithm, 3D conditional random field, Markov model or Kalman filter. These methods can be used to predict semantics and ensure that the predictions remain consistent over time.

To summarize, the methods suggested for future research involve adding additional input data, testing on different domains and switching to other algorithms. Although the methods we suggested in this thesis already deliver good results, it would be interesting to investigate if the explained ideas can improve results further.

A

List of Acronyms

List of Acronyms

Acc	Accuracy
ANN	Artificial Neural Network
API	Application Programming Interface
Cons	Consistency
ConsW	Consistency Wrong
ConvLSTM	Convolutional Long Short-Term Memory
CPU	Central Processing Unit
ESP	Efficient Spatial Pyramid
FPS	Frames Per Second
GAN	Generative Adversarial Network
GMM	Gaussian Mixture Model
GPU	Graphics Processing Unit
LSTM	Long Short-Term Memory
MADD	Multiply-Adds
mIoU	mean Intersection over Union
NN	Neural Network
p.p.	percentage points
ReLU	Rectified Linear Unit
RGB	Red Green Blue
RNN	Recurrent Neural Network
Seq	Sequence
SGD	Stochastic Gradient Descent

Bibliography

[1] (2018). Road safety status. https://www.who.int/violence_injury_prevention/road_safety_status/2018/en/. (page 2)

[2] (2019). Youtube statistics. https://www.omnicoreagency.com/youtube-statistics/. (page 3)

[3] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg. (page 7)

[4] Brostow, G. J., Fauqueur, J., and Cipolla, R. (2009). Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, 30:88–97. (page 11, 40)

[5] Burges, C. J. C. (1998). A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167. (page 7)

[6] Calonder, M., Lepetit, V., Strecha, C., and Fua, P. (2010). BRIEF: Binary Robust Independent Elementary Features. In Daniilidis, K., Maragos, P., and Paragios, N., editors, *Computer Vision – ECCV 2010*, pages 778–792, Berlin, Heidelberg. Springer Berlin Heidelberg. (page 32)

[7] Chen, A. Y. C. and Corso, J. J. (2011). Temporally consistent multi-class video-object segmentation with the video graph-shifts algorithm. *2011 IEEE Workshop on Applications of Computer Vision (WACV)*, pages 614–621. (page 33)

[8] Chen, J. and Tang, C.-K. (2007). Spatio-temporal markov random field for video denoising. (page 33)

[9] Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F., and Adam, H. (2018). Encoder-decoder with atrous separable convolution for semantic image segmentation. In Ferrari, V., Hebert, M., Sminchisescu, C., and Weiss, Y., editors, *Computer Vision – ECCV 2018*, pages 833–851, Cham. Springer International Publishing. (page 39, 46, 52)

[10] Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics. (page 37)

[11] Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807. (page 24, 31, 46, 78)

[12] Chung, J., Gülcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555. (page 37, 38)

[13] Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., and Schiele, B. (2016). The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 4, 11, 31, 39, 46)

[14] Criminisi, A., Cross, G., Blake, A., and Kolmogorov, V. (2006). Bilayer segmentation of live video. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 1, pages 53–60. (page 33)

[15] Dhanachandra, N., Manglem, K., and Chanu, Y. J. (2015). Image Segmentation Using K-means Clustering Algorithm and Subtractive Clustering Algorithm. *Procedia Computer Science*, 54:764–771. (page 10)

[16] Donahue, J., Anne Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., and Darrell, T. (2015). Long-term recurrent convolutional networks for visual recognition and description. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 2)

[17] Donahue, J., Hendricks, L. A., Rohrbach, M., Venugopalan, S., Guadarrama, S., Saenko, K., and Darrell, T. (2017). Long-Term Recurrent Convolutional Networks for Visual Recognition and Description. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):677–691. (page 38)

[18] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. (2017). CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16. (page 40, 47, 58)

[19] Doulamis, A. D., Doulamis, N. D., Ntalianis, K. S., and Kollias, S. D. (2003). An efficient fully unsupervised video object segmentation scheme using an adaptive neural-network classifier architecture. *IEEE transactions on neural networks*, 14 3:616–30. (page 33)

[20] Drachman, D. A. (2005). Do we have brain to spare? *Neurology*, 64(12):2004–2005. (page 13)

[21] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159. (page 18)

[22] Fayyaz, M., Saffar, M. H., Sabokrou, M., Fathy, M., and Klette, R. (2016). Stfcn: Spatio-temporal fcn for semantic video segmentation. *CoRR*, abs/1608.05971. (page 33)

[23] Galasso, F., Keuper, M., Brox, T., and Schiele, B. (2014). Spectral graph reduction for efficient image and streaming video segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 49–56. (page 33)

[24] Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 11, 31, 39, 81)

[25] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 2672–2680, Cambridge, MA, USA. MIT Press. (page 10)

[26] Goodfellow, I. J., Shlens, J., and Szegedy, C. (2015). Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572. (page 31)

[27] Greff, K., Srivastava, R. K., Koutnik, J., Steunebrink, B. R., and Schmidhuber, J. (2017). LSTM: A Search Space Odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232. (page 38)

[28] Harris, C. and Stephens, M. (1988). A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151. (page 32)

[29] Hartigan, J. A. and Wong, M. A. (1979). Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108. (page 10)

[30] He, K., Zhang, X., Ren, S., and Sun, J. (2015a). Deep residual learning for image recognition. *CoRR*, abs/1512.03385. (page 7, 78)

[31] He, K., Zhang, X., Ren, S., and Sun, J. (2015b). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV 2015, pages 1026–1034, Washington, DC, USA. IEEE Computer Society. (page 15, 59)

[32] He, Y., Chiu, W.-C., Keuper, M., and Fritz, M. (2016). Rgbd semantic segmentation using spatio-temporal data-driven pooling. (page 33)

[33] Herculano-Houzel, S. (2009). The human brain in numbers: a linearly scaled-up primate brain. *Frontiers in Human Neuroscience*, 3:31. (page 13)

[34] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780. (page 34)

[35] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861. (page 24)

[36] Ingraham, C. (2016). How much of your life you are wasting on your commute. `https://www.washingtonpost.com/news/wonk/wp/2016/02/25/how-much-of-your-life-youre-wasting-on-your-commute/`. (page 2)

[37] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 448–456. JMLR.org. (page 26)

[38] Jain, A., Chatterjee, S., and Vidal, R. (2013). Coarse-to-fine semantic video segmentation using supervoxel trees. In *2013 IEEE International Conference on Computer Vision*, pages 1865–1872. (page 33)

[39] Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pages 2146–2153. (page 15)

[40] Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In Nédellec, C. and Rouveirol, C., editors, *Machine Learning: ECML-98*, pages 137–142, Berlin, Heidelberg. Springer Berlin Heidelberg. (page 7)

[41] Jozefowicz, R., Zaremba, W., and Sutskever, I. (2015). An empirical exploration of recurrent network architectures. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 2342–2350. JMLR.org. (page 38)

[42] Kalman, R. E. (1960). On the General Theory of Control Systems. (page 32)

[43] Khoreva, A., Galasso, F., Hein, M., and Schiele, B. (2015). Classifier based graph construction for video segmentation. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 951–960. (page 33)

[44] Kimoto, T., Asakawa, K., Yoda, M., and Takeoka, M. (1990). Stock market prediction system with modular neural networks. In *1990 IJCNN International Joint Conference on Neural Networks*, pages 1–6 vol.1. (page 9)

[45] Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *International Conference on Learning Representations.* (page 18)

[46] Kirillov, A., He, K., Girshick, R. B., Rother, C., and Dollár, P. (2018). Panoptic segmentation. *CoRR*, abs/1801.00868. (page 39)

[47] Koutnik, J., Greff, K., Gomez, F., and Schmidhuber, J. (2014). A clockwork rnn. *31st International Conference on Machine Learning, ICML 2014*, 5. (page 38)

[48] Kowsari, K., Heidarysafa, M., Brown, D. E., Meimandi, K. J., and Barnes, L. E. (2018). RMDL: random multimodel deep learning for classification. *CoRR*, abs/1805.01890. (page 31)

[49] Krähenbühl, P. and Koltun, V. (2011). Efficient inference in fully connected crfs with gaussian edge potentials. In Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 24*, pages 109–117. Curran Associates, Inc. (page 31)

[50] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). 2012 AlexNet. *Advances In Neural Information Processing Systems*, pages 1–9. (page 7, 19)

[51] Kundu, A., Vineet, V., and Koltun, V. (2016). Feature space optimization for semantic video segmentation. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3168–3175. (page 33)

[52] Lafferty, J. D., McCallum, A., and Pereira, F. C. N. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc. (page 33)

[53] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE.* (page 9, 19)

[54] Li, S. Z. (2001). *Markov random field modeling in image analysis.* Springer. (page 33)

[55] Li, Y., Shi, J., and Lin, D. (2018). Low-Latency Video Semantic Segmentation. (page 33, 38)

[56] Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* (page 19, 31)

[57] Lowe, D. G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110. (page 32)

[58] Lu, Y., Lu, C., and Tang, C. K. (2017). Online Video Object Detection Using Association LSTM. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-Octob:2363–2371. (page 33, 36)

[59] Luc, P., Neverova, N., Couprie, C., Verbeek, J., and LeCun, Y. (2017). Predicting deeper into the future of semantic segmentation. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 648–657. (page 34, 53)

[60] Lucas, B. D. and Kanade, T. (1981). An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference*

on Artificial Intelligence - Volume 2, IJCAI'81, pages 674–679, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc. (page 33)

[61] Luthon, F., Caplier, A., and Lievin, M. (1999). Spatiotemporal mrf approach to video segmentation: Application to motion detection and lip segmentation. *Signal Processing*, 76:61–80. (page 33)

[62] Lyu, Y. and Huang, X. (2018). Road Segmentation Using CNN with GRU. (page 37)

[63] Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. (page 15)

[64] Maass, W. (1997). Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659 – 1671. (page 13)

[65] Mandal, M. K. (2003). *The Human Visual System and Perception*, pages 33–56. Springer US, Boston, MA. (page 1)

[66] Mehta, S., Rastegari, M., Caspi, A., Shapiro, L. G., and Hajishirzi, H. (2018). Espnet: Efficient spatial pyramid of dilated convolutions for semantic segmentation. In *ECCV*. (page 4, 32, 45, 62, 64)

[67] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA. Omnipress. (page 14)

[68] Neuhold, G., Ollmann, T., Rota Bulò, S., and Kontschieder, P. (2017). The mapillary vistas dataset for semantic understanding of street scenes. In *International Conference on Computer Vision (ICCV)*. (page 11, 39, 81)

[69] Oliu, M., Selva, J., and Escalera, S. (2018). Folded recurrent neural networks for future video prediction. In Ferrari, V., Hebert, M., Sminchisescu, C., and Weiss, Y., editors, *Computer Vision – ECCV 2018*, pages 745–761, Cham. Springer International Publishing. (page 37)

[70] Payer, C., Štern, D., Neff, T., Bischof, H., and Urschler, M. (2018). Instance segmentation and tracking with cosine embeddings and recurrent hourglass networks. In Frangi, A. F., Schnabel, J. A., Davatzikos, C., Alberola-López, C., and Fichtinger, G., editors, *Medical Image Computing and Computer Assisted Intervention – MICCAI 2018*, pages 3–11, Cham. Springer International Publishing. (page 37)

[71] Ranjan, A. and Black, M. J. (2017). Optical flow estimation using a spatial pyramid network. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:2720–2729. (page 33, 53)

[72] Ros, G., Sellart, L., Materzynska, J., Vazquez, D., and Lopez, A. M. (2016). The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 40)

[73] Rosten, E. and Drummond, T. (2006). Machine Learning for High-Speed Corner Detection. In *Computer Vision – ECCV 2006*, pages 430–443, Berlin, Heidelberg. Springer Berlin Heidelberg. (page 32)

[74] Rudolph, G. and Voelzke, U. (2017). Three sensor types drive autonomous vehicles. `https://www.fierceelectronics.com/components/three-sensor-types-drive-autonomous-vehicles`. (page 5)

[75] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252. (page 19)

[76] Shah, S., Dey, D., Lovett, C., and Kapoor, A. (2018). Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, pages 621–635. Springer International Publishing. (page 40)

[77] Shi, J. and Malik, J. (2000). Normalized Cuts and Image Segmentation Part of the Electrical and Computer Engineering Commons Recommended Citation Normalized Cuts and Image Segmentation Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905. (page 30, 33)

[78] Shi, X., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-K., and Woo, W.-C. (2015). Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *NIPS*. (page 36)

[79] Srivastava, N., Mansimov, E., and Salakhutdinov, R. R. (2015). Unsupervised learning of video representations using lstms. In *ICML*. (page 36)

[80] Stout, D. W. (2019). Social media statistics. `https://dustinstout.com/social-media-statistics/`. (page 2, 3)

[81] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. pages 1–10. (page 31)

[82] Tompson, J. J., Jain, A., LeCun, Y., and Bregler, C. (2014). Joint training of a convolutional network and a graphical model for human pose estimation. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 1799–1807. Curran Associates, Inc. (page 7)

[83] Toshev, A. and Szegedy, C. (2014). Deeppose: Human pose estimation via deep neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 19)

[84] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144. (page 9)

[85] Xie, S., Girshick, R. B., Dollár, P., Tu, Z., and He, K. (2016). Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431. (page 81)

[86] Yu, F. and Koltun, V. (2016). Multi-scale context aggregation by dilated convolutions. *CoRR*, abs/1511.07122. (page 25)

[87] Yu, R. (2019). A brief review of object detection and semantic-segmentation. `https://richardyu114.github.io/2019/02/27/A-Brief-Review-of-Object-Detection-and-Semantic-Segmentation/`. (page 30)

[88] Zhao, H., Shi, J., Qi, X., Wang, X., and Jia, J. (2017). Pyramid scene parsing network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6230–6239. (page 81)