Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Telecommunications and Media Informatics

Fábián Füleki

Image-based depth estimation with deep neural networks

Bálint Gyires-Tóth, PhD
University Supervisor

Róbert Zsolt Kabai
Industrial Supervisor

October 27, 2019

Kivonat

Az elmúlt évek egyik legnagyobb szoftvermérnöki kihívása egy olyan önvezető rendszer megalkotása, amely képes tetszőleges közlekedési szituációban önállóan helyes döntést hozni (ezt hívjuk ötös szintú autonómiának). A technológia jelen állása lehetővé teszi a részben önvezető rendszerek meglétét a már utcán közlekedő autókban (kettes szintű autonómia). Jelenleg úgy tűnik, hogy az önvezetéshez használatos szenzorok képességei és a feldolgozáshoz szükséges nagy számítási igény lehetővé tennék egy ötös szintnek megfelelő önvezető rendszer megalkotását, de a probléma bonyolultságából kifolyóan nem lehet klasszikus algoritmusokkal elvégezni ezt a feladatot.

Az utóbbi évtizedben sokat fejlődött a machine learning terület deep learning (mély tanulás) iránya, amely lehetővé teszi olyan kép és egyéb szenzoradatok feldolgozását, ami nagyban elősegítheti egy magasabb szintű önvezető rendszer létrejöttét.

Egy lehetséges megközelítés, hogy a különböző szenzoradatok feldolgozásáért összesen egy mély neurális háló a felelős, amely több, különböző részfeladatot old meg egyszerre (multitask learning). Ilyen részfeladatok az objektumdetekció, a sávok észlelése, vagy éppen a kereszteződés helyének és a közlekedési lámpa állapotának megállapítása. Egyes részfeladatok együttes tanítása hatékonyabb lehet és jobb teljesítményt eredményezhet, mintha ezeket a feladatokat külön-külön neurális hálóval oldottuk volna meg.

Dolgozatomban a távolságbecslés problémakörét és annak megvalósítását vizsgálom, mégpedig kizárólag kameraszenzorok segítségével. Célom egy olyan neurális hálózat létrehozása volt, amely képes a képen látható egyes pixelekhez távolságértékeket rendelni. Munkám részeként újszerű hibafüggvényt dolgozok ki és valósítok meg, mely esetén a távolságértékek hibájának számításakor figyelembe veszem a rendelkezésre álló szemantikus szegmentációt is. Célom ezzel, hogy egyes objektumok, mintpéldául az autók és gyalogosok, amik a kép csupán egy kis részét alkotják, nagyobb súllyal szerepeljenek a hibaszámításban, így ellensúlyozva az aránylag kis méretüket.

Abstract

In recent years, one of the biggest software engineering challenges is the goal of creating a completely autonomous system, which is capable of navigating in a complex driving environment (called the fifth level of autonomy). Today's state-of-the-art technology makes it possible to have partially self-driving cars on the roads (level two of autonomy). It seems like the capabilities of the sensor suits and the processing power of embedded systems make it possible to create a level five autonomous system, but the complexity of the problem makes it impossible to solve this task based on classical algorithms only.

In the last decade, deep learning, a subfield of machine learning has grown in popularity and in performance as well. Deep learning makes it possible to process the data coming from the sensor suits and with its help, there is a common vision that a level five autonomous system could be created in a few years.

The processing of the sensor data can be done using a deep neural network, which solves multiple tasks at the same time. These tasks are object detection, drivable area detection, intersection and traffic light state detection, etc. Some of these tasks help each other, meaning that a model creating multiple outputs can perform better than two models trained for only one task.

In this work, I aimed to explore and implement the task of distance estimation based only on camera visuals. My goal was to create a neural network which is capable of predicting distance values for each pixel on the input image. Furthermore, I created a loss calculation method based on a novel idea of taking semantic segmentation into account. This method helps to balance the size of some objects on a single image. For example, a pedestrian is relatively small on an image compared to a building, but it is more important to have a precise prediction for the pedestrian than the building.

Contents

Chapter 1

Introduction

Self-driving is one of the most inspiring yet challenging aspect of the automotive industry. A number of car manufacturing and technology companies are working on creating a software and hardware suite which is capable of navigating a vehicle on the road.

Personally, I am fascinated by the improvements that self-driving car concepts had over the recent years, therefore I have the motivation to contribute to the vision of full autonomy. However, the task of self-driving requires knowledge from a number of sub-fields of engineering. Mechanical, electrical and software engineers are working together on different kinds of solutions across the globe. As an individual, it is impossible to create every required component of a self-driving system, all the way from sensing to controlling. As a software engineer, I can contribute to the visual recognition system part, which is essential in a self-driving system.

In this work, my intention is to create a depth estimation system using convolutional deep neural networks. Depth map can be really useful for drivable area segmentation, 3D object detection, or even parking space identification. Nowadays, the easiest way for this approach is the use of lidar sensors, although these sensor are relatively expensive and also have some limitations. Therefore, my solutions are going to be based only on cameras.

A novel idea for loss calculation will be also introduced, which takes advantage of the available semantic segmentation. Some parts of the captured camera images are not taken into calculation as much as they should be. For example, pedestrians are highly involved in the traffic scenes, but they are represented by only a small portion of pixels on the image. To solve this problem, my proposed idea is to use a weight mask based on the semantic segmentation and multiply the loss with it. Important objects, such as cars and pedestrians are going to have much bigger weight on the mask. This approach turns out to be a little helpful if we look at the results in chapter 5.

Chapter 2

Background

2.1 Machine Learning Introduction

In recent years, machine learning-related algorithms have grown in popularity, thanks to the technological advancements in the IT industry and the widely available high-level libraries. Other tools and online courses are for the help of a beginner in the field of training neural networks. At first glance, some 'one click to run' methods (such as Google Seedbank) seem really controversial, but the whole training process of a neural network is far more complex than a high-level API (application programming interface) suggests. For the training process, many factors should be taken into calculation, and the cause of a poorly performing neural network can be almost any of them. As Andrej Karpathy, the head of Artificial Intelligence research at Tesla says, "Neural net training fails silently" [21]. For example, a poorly chosen learning rate decay (the slowdown of the learning speed) can cause the training process to stop at a local optima.

Machine learning is a subset of artificial intelligence related algorithms, which are mostly based on statistical data analyzing methods. Machine learning is utilized including but not limited to solve the following tasks:

- Natural language processing (See BERT [7])
- Stock price prediction
- Object identification on images (See YOLO [30])
- Network traffic analysis for cybersecurity (See [38])

In general, a machine learning algorithm has an input and an output, and we can differentiate two main types of machine learning algorithms based on the type of output:

- Classification: In this case, the network tries to predict which class a given input belongs to. Usually, cross-entropy loss is in use for multiclass classification.
- Regression: The output of the network is a vector, and in most cases, the mean absolute error function is in use as a loss.

- In some cases, the machine learning algorithm has multiple types of outputs, which have some correlation. For the input is an image of someone, and the age output is a regression and the gender output is a classification.

There are three types of machine learning techniques that are mostly utilized nowadays:

- Supervised learning: If the expected output value of a huge amount of input data is available, we can train a machine learning algorithm to produce the desired output of a given input with the method of minimizing the error between the output of the algorithm and the predefined output, also known as ground truth.
- Unsupervised learning: This type of machine learning algorithms are in use in the case of missing or partially missing ground truth. The algorithm tries to learn the internal structure of the data. The combination of the supervised and self-supervised method is called semi-supervised.
- Reinforcement learning: In this case, an agent is utilized in an environment. This environment can be dynamically changed by the agent using an available action and the environment returns a state to the agent. This procedure is called the Markov decision process. After some time, the agent receives a reward, which can be either positive or negative. The agent tries to maximize its reward by taking the best possible actions in the environment.

2.2 Deep Learning

Deep learning is a subset of machine learning, where deep artificial neural networks are in use. In this paper, deep learning will be utilized with the training method of supervised learning.

2.2.1 The concept of artificial neural networks

The human brain was the main source of inspiration in the creation of artificial neural networks. This naming can be a little misleading, as artificial neural networks (neural network or network for the rest of the paper) are not about the simulation of the brain cells.

Neural networks consist of layers and these layers consist of some neurons and a bias. A neuron has an input and an output, and the bias is a special neuron that does not have any input, it just outputs a constant number, which is 1. The connection between two neurons is called a weight, which is also a number. On the output of every neuron, there is an activation function (explained further in section 2.2.4), and the output value of a neuron can be calculated using the formula 2.1.

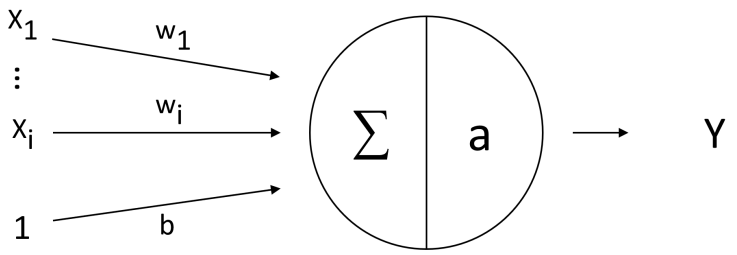


Figure 2.1: *The structure of a single neuron*

The output of a single neuron can be calculated using the following formula:

$$Y = a\left(b + \sum_i w_i \cdot X_i\right) \tag{2.1}$$

Where X_i is the output of the ith neuron in the previous layer and w_i is the weight between the ith neuron in the previous layer and the currently examined neuron. The activation function is noted as a and the bias as b.

The most straightforward way to connect the neurons of each layer is the following: every neuron in a layer is connected to the neighboring layer's every neuron. There isn't any connection between neurons in the same layer. The connection type itself is called fully connected layers, and this architecture is called the multi-layer perceptron (MLP). The first layer of the network is called the input layer and the last layer is called the output layer. All the other layers together are called the hidden layers.

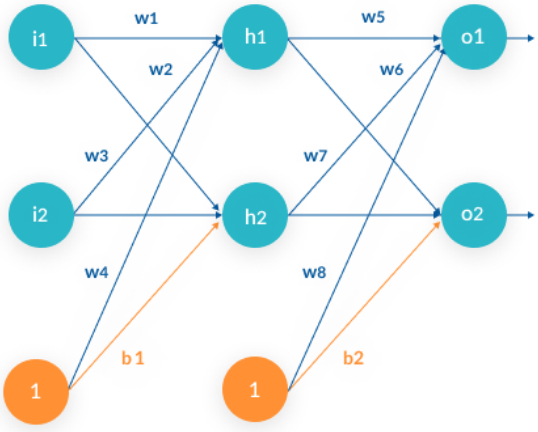


Figure 2.2: *The structure of a multi-layer perceptron (Source: [27])*

The neural network can calculate (predict) the output for a given input, this process is called forward passing or forwarding in short. The output values together are called

prediction. Forwarding consists of the following steps:

1. Fill the input layer's values with the vectorized input data.
2. For each layer calculate the output values of every neuron in the next layer based on the formula 2.1.
3. The output of the last layer contains the output values.

2.2.2 Backpropagation

In this section, I briefly introduce the backpropagation algorithm [15], which is used for the training process of neural networks. In short, backpropagation tells how much each weight (and bias) should be changed in the network to achieve better prediction results. In more mathematical phrases, backpropagation is a specialization of a technique called reverse mode differentiation or reverse accumulation, which computes the derivatives of functions that can be represented as computational graphs.

After the forwarding of a single input data is done, an error value is calculated for every neuron in the output layer. This error value is determined by the loss function, which has two inputs: the predicted value and the expected value for a neuron.

I would like to explain the calculus of backpropagation through a simple example. Let's consider a small MLP, which has one output neuron and let's also take into the calculation a single neuron from the last hidden layer. The weight between these neurons will be noted with $w(L)$. Let's note the output value of the output layer as $A(L)$, and the output of a neuron from the previous layer as $A(L-1)$.

$$A(L) = a\Big(w \cdot A(L-1)\Big) \tag{2.2}$$

Where a notes the activation function. For this sample, the loss function (noted as c) is going to be the squared error:

$$c(x, y) = (x - y)^2 \tag{2.3}$$

This function is also used in a widely applied loss function, the mean squared error (MSE). In this case, the interpreted value of cost function for this output neuron is

$$C = (A(L) - y)^2 \tag{2.4}$$

Where y is the desired output. The derivative of this cost function with respect to the output value of the last layer is:

$$\frac{\delta C}{\delta A(L)} = 2 \cdot (A(L) - y) \tag{2.5}$$

The derivative of the last output with respect to the weight (noted as $w(L)$) between the two considered neuron:

$$\frac{\delta A(L)}{\delta w} = A(L-1) \cdot a'\Big(w(L) \cdot A(L-1)\Big) \tag{2.6}$$

The final result for this weight is:

$$\frac{\delta C}{\delta w} = 2 \cdot \Big(A(L) - y\Big) \cdot A(L-1) \cdot a'\Big(w(L) \cdot A(L-1)\Big) \tag{2.7}$$

The name backpropagation makes sense if we look at this previous formula. The derivative depends on the previous layer's neuron activation ($A(L-1)$) as well, so we have to make the same steps for the second last and the last layer as well. This process should be repeated for each neighboring layer until the derivative of the cost function can be described using only the weights of the network and the values of the input. The derivative of the cost function with respect to the previous layer is

$$\frac{\delta C}{\delta A(L-1)} = \frac{\delta C}{\delta A(L)} \cdot \frac{\delta A(L)}{\delta A(L-1)} \tag{2.8}$$

This simple example can be extended for several output neurons and several hidden layer neurons. The number of output neurons will be noted as J, and the number of the previous layer's neurons is K. Let's note the jth output layer activation with $A_j(L)$ and the kth activation of the previous layer $A_k(L-1)$. Let's note the weight between the jth output neuron and the kth hidden layer's neuron as $w_{jk}(L)$.

For easier notation, let $S_j(L)$ be the weighted sum for the jth neuron in the output layer before activation:

$$S_j(L) = \sum_{k=1}^{K} w_{jk}(L) \cdot A_k(L-1) \tag{2.9}$$

And the activation will be:

$$A_j(L) = a(S_j(L)) \tag{2.10}$$

The squared error cost function is going to be:

$$C_0 = \sum_{j=1}^{J} \Big(A_j(L) - y_j\Big)^2 \tag{2.11}$$

The derivative of the cost function with respect to the last layer's jth neuron:

$$\frac{\delta C}{\delta A_j(L)} = 2 \cdot \sum_{j=1}^{J} \Big(A_j(L) - y_j\Big) \tag{2.12}$$

The derivative of the output layer's jth neuron with respect to $w_{jk}(L)$:

$$\frac{\delta A_j(L)}{\delta w_{jk}(L)} = A_k(L-1) \cdot a'\Big(S_j(L)\Big) \tag{2.13}$$

The derivative of the cost function with respect to any weight $w_{jk}(L)$ in the considered layer can be expressed as:

$$\frac{\delta C}{\delta w_{jk}(L)} = A_k(L-1) \cdot a'\Big(S_j(L)\Big) \cdot 2 \cdot \sum_{j=1}^{J} \Big(A_j(L) - y_j\Big) \tag{2.14}$$

With this proposed formula, the gradient of the cost function with respect to any weight in between two layers can be calculated. Using the formula 2.8, the derivatives can be calculated the same way in the previous layers.

2.2.3 Optimizer

A common misunderstanding of backpropagation is that it changes the weights of the neural network. Actually, this is the task of the utilized optimizer algorithm. In this section, some of the mostly used optimizers will be introduced in order to explore how the calculated gradients are utilized throughout the training process.

An epoch is a term used to note processing every input data once in the training set. The backpropagation algorithm is executed for each input data in every epoch, but the modification of the weights might not be executed immediately after each gradient calculation. The optimizer may accumulate multiple gradient values from multiple input data before changing the weights. This set of data is called mini batch and the corresponding gradients are averaged in order to create a gradient that may be able to improve the loss for some input data in the mini batch. The size of the mini batch is referred as batch size in the field of machine learning. The number of weight updates in an epoch is called an iteration. Stochastic gradient descent (SGD) is one of the simplest optimizers, which updates the weights of the network according to the following formula:

$$w = w - \eta \cdot \frac{1}{N} \cdot \sum_{i=1}^{N} \frac{\delta C_i}{\delta w} \tag{2.15}$$

Where w notes the weight values of the network, N notes the batch size, η notes the learning rate and $\frac{\delta C_i}{\delta w}$ notes the gradient for the ith input, calculated by backpropagation explained previously in section 2.2.2. For a simplified visual representation of the gradient descent, see figure 2.3.

The stochastic gradient descent algorithm has an improved variant called SGD with momentum. This improves the performance of the optimizer with introducing memory over iterations: the previous gradient values influence the current weight changes.

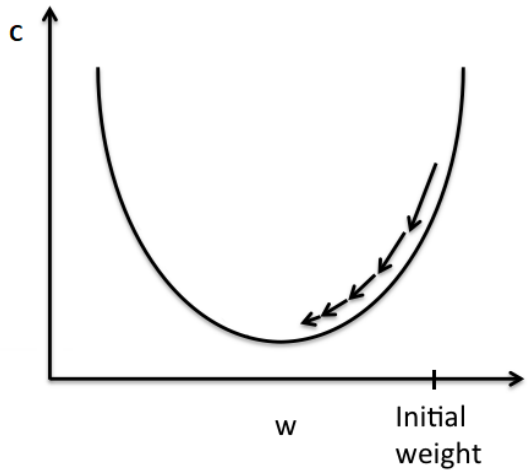
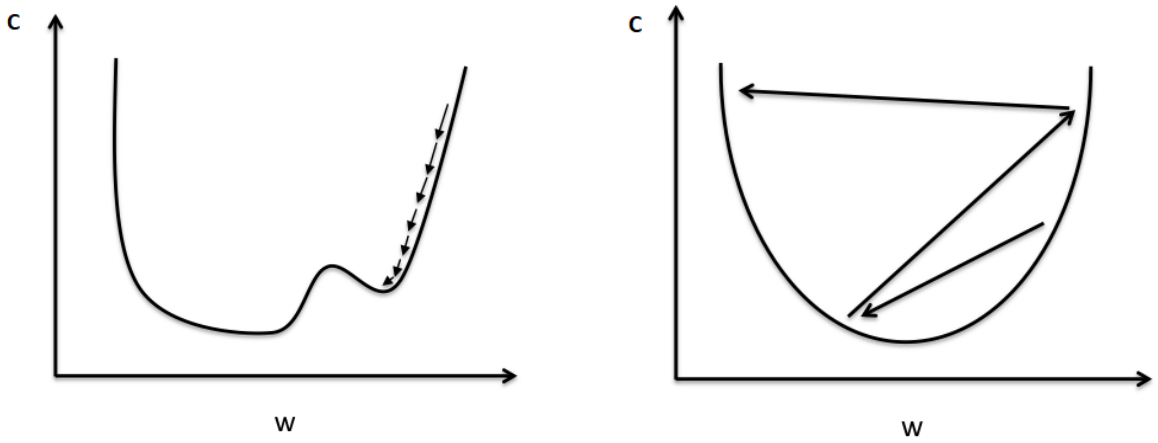


Figure 2.3: *A simplified example of gradient descent (Source: [29])*

Learning rate is a scalar hyperparameter, which changes the values of the averaged gradient. Determining the value of the learning is crucial in some cases as an incorrect value can prevent the network from learning anything. For instance, a tiny learning rate can cause the network to slow down the training process or even to stop the loss in a local optima. A giant learning rate causes overshooting of the global optima. For the corresponding examples, see figure 2.4.



(a) *Learning rate is too big, the optimizer is overshooting the optima*

(b) *Learning rate is too small: learning is stopped in a local optima*

Figure 2.4: *Picking an optimal learning rate can be crucial for the training process (Source: [29])*

Some optimizers are less dependent from the overall learning rate, as they determine a learning rate for each parameter throughout the training process. These kind of optimizers are called adaptive optimizers and the mostly used one is the adaptive moment estimation (Adam) [22]. As the amount of optimizer algorithms are rapidly growing, the between them would be a great research topic to succeed in.

2.2.4 Activation functions

The activation function is needed to introduce nonlinearities in the network. If the network had only linear activation functions, such as $a(x) = c \cdot x$, the network could not converge to any nonlinear function. Activation functions play a key part in a neural network, as the changes of the weights are going to be calculated based on the first derivative of the activation function (as explained in section 2.2.2).

Choosing an activation function for a network can be hard, if we take a look at the variety of possibilities: Leaky ReLU, ELU, LReLU, SReLU are only some subtypes of the widely utilized ReLU [28] (see figure 2.5). There are a vast number of activation functions that can be utilized in different circumstances. Most of the time, the rectified linear unit [4], sigmoid and tanh are in use. The utilization of ReLU has an advantage at deployment, as the hardware implementation is requires almost no additional computational power.

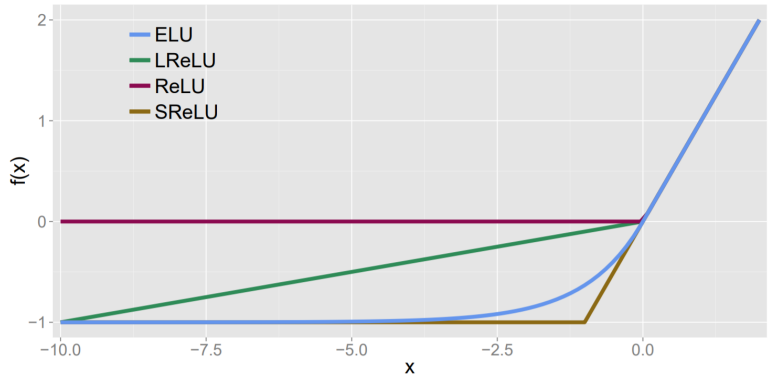


Figure 2.5: *There are a number of variety activation function based on the rectified linear unit. [5]*

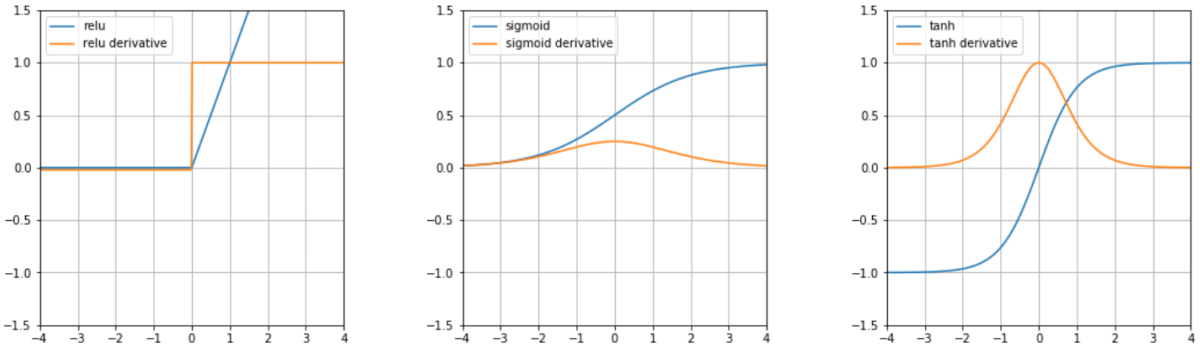
The idea of introducing nonlinearities in the network is really important because the goal is to model a nonlinear function. This can be done by utilizing ReLU as the activation function of a neural network. According to the formula, the rectified linear unit returns its input only if it is positive, and a value of zero is returned in the case of a negative input value. The derivative of this function is zero for negative x values and 1 for the positive ones. This function is also known as the Heaviside step function. According to [10], the zero value of the derivative function for negative x values does not cause any problems. For some input values, the weighted sum for a given neuron becomes negative, and both the activated value and its derivative will be zero. Because of this effect, a lot of activation values are going to be zero, which is called sparse activation.

One thing to be noted here, the derivatives of the sigmoid and tanh function is really small in the region of great positive or negative x values.

$$\lim_{x \to \pm\infty} sigmoid(x) = 0 \tag{2.16}$$

$$ReLU(x) = max(0, x) \qquad Sigmoid(x) = \frac{1}{1 - e^{-x}} \qquad tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$\frac{\delta ReLU(x)}{\delta x} = \begin{cases} 0 : x < 0 \\ 1 : x > 0 \end{cases} \qquad \frac{\delta Sigmoid(x)}{\delta x} = \frac{e^{-x}}{(1 + e^{-x})^2} \qquad \frac{\delta tanh(x)}{\delta x} = \frac{4}{(e^{-x} + e^{x})^2}$$



(a) *Rectified linear unit* **(b)** *Sigmoid* **(c)** *Tanh*

Figure 2.6: *Comparison of the activation functions*

The small gradient values in these regions can cause the well-known vanishing gradient problem, which occurs in very deep neural networks. As we propagate backward, the gradients of the activations become smaller and smaller and after a while, the network is not learning anything. In this paper, only the ReLU and the leaky ReLU are going to be utilized.

2.2.5 Convolutional neural networks

Convolutional neural networks (CNN) are nowadays the standard methods for processing images containing complex features. This type of network architecture trained with back-propagation was first utilized in [24]. These networks are able to detect low level features in the first layers and higher level features in the deeper layers. This architecture also reduces the number of neurons needed for sufficient object detection, image segmentation, etc.

In the case of images, two dimensional convolutions are in use. In short, convolutional processing in 2D is sliding a window over the input image, calculating the weighted sum and using the activation function, just like in the case of a single neuron output value calculation.

Let's consider a simple example: let our input image have a size of 5x5 and let's create the output values based on a 3x3 convolutional kernel. Because the kernel fits in 9 different

locations (3 rows and 3 columns) without hanging out at the edges, the output size is going to be 3x3. The procedure is the following: we place the window in the first possible location (colored red on figure 2.7) and calculate the weighted sum.

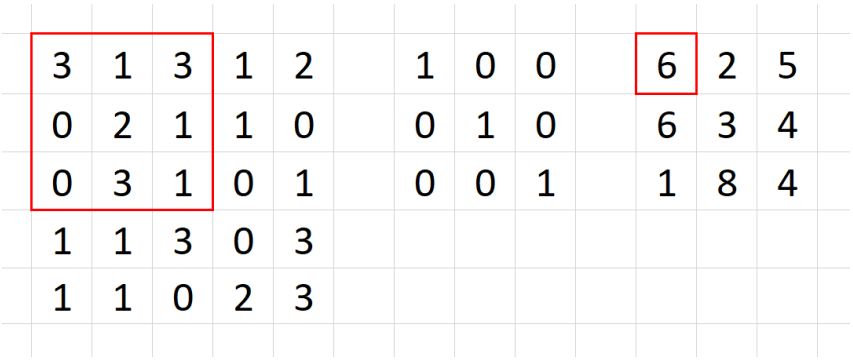


Figure 2.7: *An example of convolution over a small image*

There are some other factors which also influence how the input is processed: the stride defines the displacement of the window, and the padding defines how the edge of the input image should be filled.

After the activation, a pooling layer is in use to downsample the image and to extract some features. In the case of convolutional networks, max pooling layer is utilized, as it enables to keep only the most activated regions of the input. Maximum pooling means that we keep the biggest value from the previously calculated convolution. In order to understand why max pooling is the better choice over average pooling, examine the example on figure 2.8.

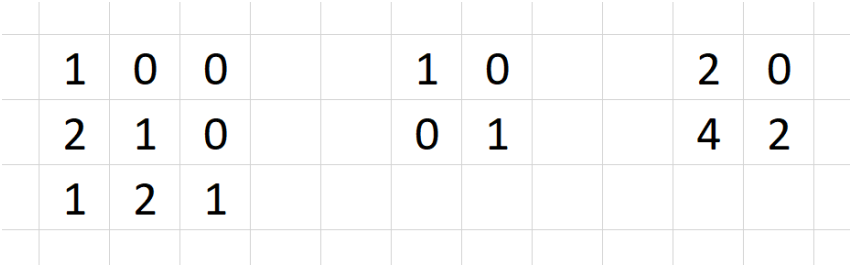


Figure 2.8: *An example of max pooling*

In this case, the filter is filled diagonally, and the input contains a diagonal line as well. On the output, the biggest activation is on the position of the diagonal line. The output of average pooling would be 2, and the output of the max pooling is 4, which we are looking for in this example.

2.3 Introduction to autonomous driving

Self-driving is one of the most challenging problems of the automotive industry at the moment. Most of the car manufacturers and other technology companies are working on

different types of advanced driver-assistance systems (ADAS). Every ADAS is aiming to lower the number of actions made by the driver, thus making the trip safer and less tedious. As the competence of the assistance systems will increase, Automated driving systems (ADS) are going to be in the focus in the next years. The difference is the driver: ADAS is focusing on helping the human driver, while ADS does not depend on one. One key technology behind these systems are neural networks.

Fully autonomous cars are going to be a vital transportation option in the near future, therefore the safety of the vehicle and its surrounding should be guaranteed. A key component achieving this goal is the visual recognition system, which supplies information about the surroundings to the intelligence and controlling systems.

The field of self-driving contains a lot of subfields, such as lane detection, traffic sign recognition, collision detection, and last but not least pedestrian movement prediction. These tasks can be solved separately with a few dedicated subsystems with or without utilizing a neural network. In this work, my goal was to create a deep neural network for solving the task of depth estimation.

Automating road transportation — even partly — has a lot of advantages. The first factor is safety: Even the simplest collision warning system in a few percentage of the cars can reduce the number of accidents on the road. Studies have shown [34] that only a small percent of autonomous cars on the road can increase safety and eliminate stop-and-go traffic jams.

According to the SAE standard [32], six levels of automation can be differentiated by who is in control of the vehicle and who is monitoring the environment:

0. No automation: Full-time monitoring and execution are required by the human driver. A vehicle is on level 0 even if it is equipped with warning or intervention systems.

1. Driver assistance: The human driver is still completely in continuous control of the vehicle, but may be assisted by different systems for different tasks. In most cases, adaptive cruise control, lane keeping system and parking assistance are implemented.

2. Partial automation: In some cases, the onboard system can take control of the vehicle, but the human driver always has to monitor the environment and must take over if needed. Keeping the hands on the steering wheel and the eyes on the road are required. Dynamic driving is completely performed by the human driver.

3. Conditional automation: The autonomous system is in full control of the vehicle and can determine in which cases human intervention is needed. The driver has to be prepared to take over, mostly for the dynamic driving tasks, such as driving in a roundabout.

4. High automation: No human driver attention is ever needed for safety. The onboard system can manage to control the vehicle in most of the driving situations. If the human driver does not take control when needed, the vehicle can safely abort the trip.

5. Full automation: The vehicle can handle all of the driving situations as good as a human driver would.

The most intuitive way of driving a car on public roads is using vision: human drivers are making decisions based mostly on their sight. The computer-based implementation of this approach is a forward-facing camera used as a sensor. The video feed is processed by the onboard computer using traditional image processing techniques or neural networks.

These sensors are described in detail in section 2.4. Adapting more than one type of sensors ensures that monitoring the environment can be achieved in all kinds of conditions. The sensor arrangement on a Tesla vehicle for example can be seen on figure 2.9.

Figure 2.9: *A modern vehicle utilizes different types of technologies at the same time (Source: [17])*

2.4 Main sensors of driving-related tasks

In this section, I introduce the human sense perception and the possible sensors of a self-driving car. The widely utilized cameras for computer vision-related tasks and the rapidly developing lidars (which stands for Light Detection and Ranging) are mostly used for distance measurement and localization problems. Both sensors have their own

applications, characteristics, advantages, and disadvantages, so I would like to provide a brief summary of them. As the goal is to create a dense and accurate depth map with a sufficient range, other tools, such as radar and ultrasonic sensors are not in use for this project.

2.4.1 Human sense perception

We, humans are able to estimate the distances of our surrounding objects. In this section, I aim to explore which aspects of human vision are mostly responsible for the 3D space detection, extend these methods for silicon-based vision systems and also introduce some methods that humans are not utilizing.

Stereo vision may seem the most intuitive way of distance estimation, and in section 2.4.2, a traditional stereo matching algorithm will be introduced, which is capable of distance estimation, if the parameters, such as focal lengths and positions of the cameras are available. In the manner of human sense, stereo vision is not that important as it seems. It may be really hard to catch a ball with one eye covered, but it is far from being impossible.

Humans are able to estimate 3D-space positions combining information from multiple effects at the same time: stereo vision, object movements, egomotion, and dynamic focus are all helping to perform well in different environments.

If the size of an object cannot be approximated, its location cannot be estimated from a single image. In this case, the movement of this object may be really useful for localization. There are some artificial neural network-based self-supervised solutions using only the relative movements of the objects [14]. If the object is not moving, the movement of the sensing person, also known as egomotion can be used as well [40] [25]. In some cases, these movement effects may not be available, as the object is far away. In this case, human vision has a great feature called dynamic focus with feedback. The feedback makes it possible to focus on certain objects correctly and estimate the corresponding distance based on the required focus level.

Humans are able to combine these types of localization techniques, and most of these methods can be implemented for computer vision based tasks as well. Stereo matching, object tracking, and egomotion calculation can be implemented easily.

2.4.2 Digital cameras

The development of digital cameras started in the 1960s. The cold war had a great impact on its advancements as a digitally transmittable image was needed in space projects. Digital cameras became really popular and widely accessible in the last decades. They are capable of producing high-quality photos with a decent framerate at a reasonable

price point. The usage of cheap, reliable, high framerate and high-resolution sensors are perfect in a lot of situations. A mobile phone's camera for example nowadays is capable of delivering up to 120 frames every second in full HD resolution, which is more than enough for most of the tasks of self-driving.

Driving is one of the tasks where humans are heavily dependent on vision, so comes the obvious idea of utilization of digital cameras. Lane detection, semantic segmentation, and pose estimation tasks can all be solved using a neural network which is based on a single camera as an input. There are some special kinds of cameras that are more efficient for some problems, so a self-driving vehicle can contain a dozen or even more cameras operating all at the same time. Some of them face different directions but overlap each other, and some of them face the exact same direction. In the latter case, the cameras have different properties, such as field of view, focus or sensor type. Some grayscale cameras have higher resolution than its color equivalent, and in some cases, a lower quality camera is utilized because of its lower price.

Artifacts and limitations of digital cameras

Every camera-based solution has some disadvantages, which mostly derive from the limitations of the used camera sensors. Digital cameras are not capable of taking perfect frames in every situation, which results in unusable photos in some cases. The most obvious limitation for the camera is the amount of light reaching the sensor. The brightness of the scene is one of the most significant measures. Other factors, such as rain, dust, and fog can occlude the camera image in a quite similar way than darkness does. On photos taken in these challenging conditions, it is much harder to identify an object or to create segmentation, etc.

Even in quite advantageous situations, other factors can also create unwanted artifacts. Artifacts are basically any of the undesired changes of an image that is caused by different effects within a camera. These side-effects can be found in almost every situation, where the cameras are in use, and most of the time, at least one of the artifacts is present. Some of the artifacts are so disturbing, that multiple consecutive captured frames cannot be used for anything.

One of the well-known artifacts is the lens flare, where a bright object shines into the lenses and leaves multiple bright rings on the image. Interestingly, the bright light source can be out of the field of view as well.

Another unwanted effect of the cameras (and sometimes even human vision) are over- and underexposure. These artifacts mostly occur when the bright scenery becomes quickly much darker (underexposure) or the dark scene becomes quickly much brighter (overexposure). In the same way as the human eye, cameras need time to adapt to the new light conditions, and in this period, some of the details can be lost around the darkest or the

brightest part of the image. Even the same frame could contain really bright and poorly lit parts, which results in lost details in these regions. For an example, see figure 2.10, where the curb is not visible on the left and the cyclist is barely noticeable on the right side of the image.

Figure 2.10: *An example of over and underexposure from the Kitti dataset [9]*

Poor light conditions can affect the captured image in other ways as well, such as noise is increased on the darker image. In normal conditions, a solution would be to increase the shutter time and lower the ISO sensitivity, but in case of driving scenes, it would result in blurred images.

In some cases, moiré can occur in the image. Moiré is a pattern, which occurs on relatively high resolution, repetitive textures. The camera's resolution is too low to capture it in all detail, so the original pattern becomes a different, wavy pattern on the image. As the pattern usually changes over time, it can be disturbing for the eye and for a computer vision system as well.

The rolling shutter effect and compression loss are not taken into account, as in practice usually, a global shutter camera provides raw images for the vision system.

Stereo cameras

The utilization of two cameras at the same time comes from the fact that humans (mostly) coordinate themselves based on stereo vision. In theory, the stereo camera setup should be sufficient for proper distance estimation at least on the same level as humans'. Stereo matching algorithms based on two cameras provide a decent baseline for the task of depth map estimation, therefore I briefly introduce the topic.

Rectification

Before stereo matching, the images are rectified, which is essentially the projection of the two images onto a common, enlarged image plane. This plane contains the two images in a way where every point in the 3D space on the left image takes the same vertical

position as on the right image. Rectification makes it easier to match the two images, because of the fact that every point has its pair on the same horizontal line. In the case of the simulated environment, the cameras are attached in a way where rectification is not necessary. For real-life sensors, both rectification and calibration is mandatory.

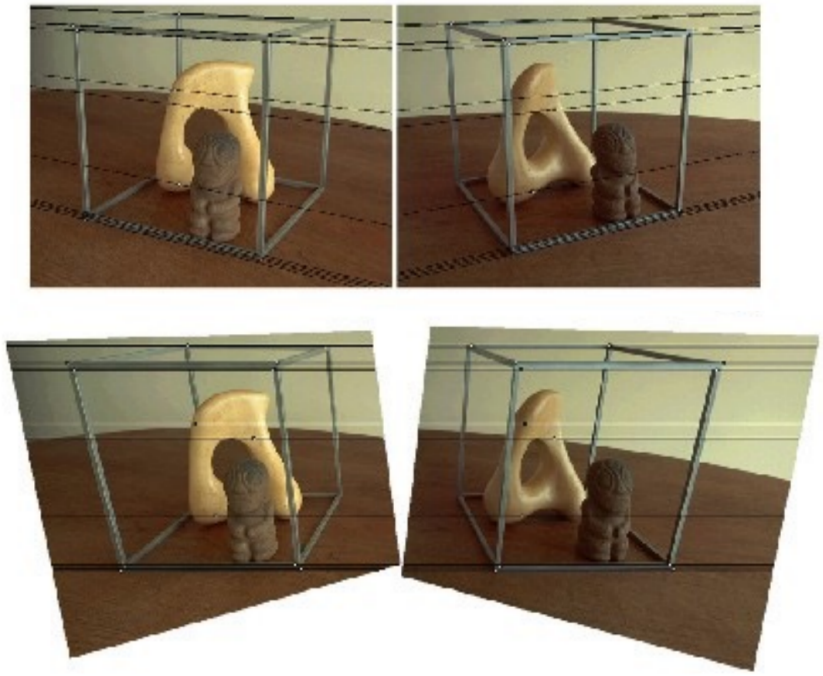
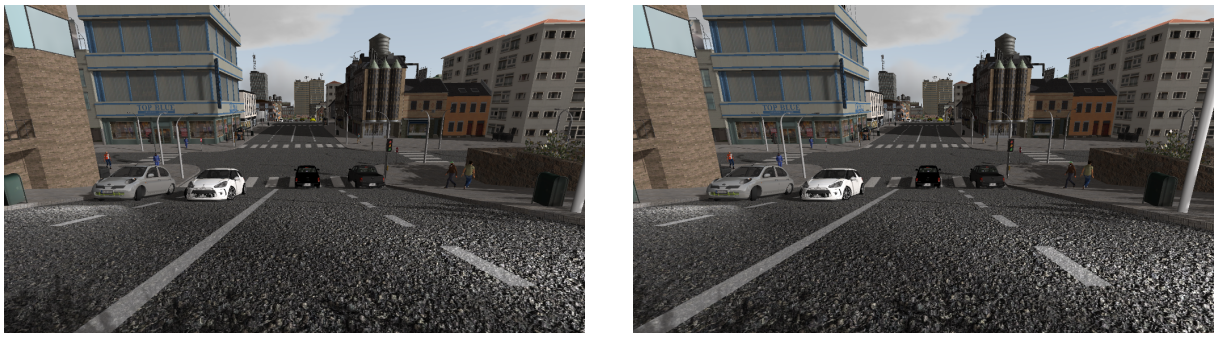


Figure 2.11: *Rectification [33]*

Stereo matching

After rectification, stereo matching algorithms are capable of calculating the disparities between the two images. The disparity map tells how much each pixel is shifted on the right image compared to the left image. In terms of self-driving, this method is implemented using two cameras, which take images at the exact same time. This setup solves the problem of moving objects in the scene, which would corrupt the results. Images taken over time are another approach for the source of the two images, but I don't consider testing its performance as the mainly utilized dataset does not provide adequately high framerate and accurate location logs.

Based on the disparity map and the properties of the cameras, the distances can be calculated for the successfully matched parts of the images. Interestingly, one of the main problems of stereo matching comes from the displacements of the cameras as well. The objects relatively close to the cameras cover other parts of the image differently on the two images. Therefore some of the pixels on the left image cannot be found on the right and vice versa. These parts cannot be matched with each other, and the stereo matching algorithm does not identify any disparities. The same effect can be noticed at the left edge



(a) *Left* **(b)** *Right*

Figure 2.12: *Example RGB stereo images from the Synthia dataset [31]*

of the image, because of the displacement. (This effect on the left side can be removed by appending blank areas to the image.)

On figure 2.13, an example of semi global matching can be seen. The coloring of the depth map is explained in section 4.1. The black areas show where SGM was not able to calculate disparities. It should be noted, that the parameters of the SGM algorithm has been optimized by hand for better visual results.

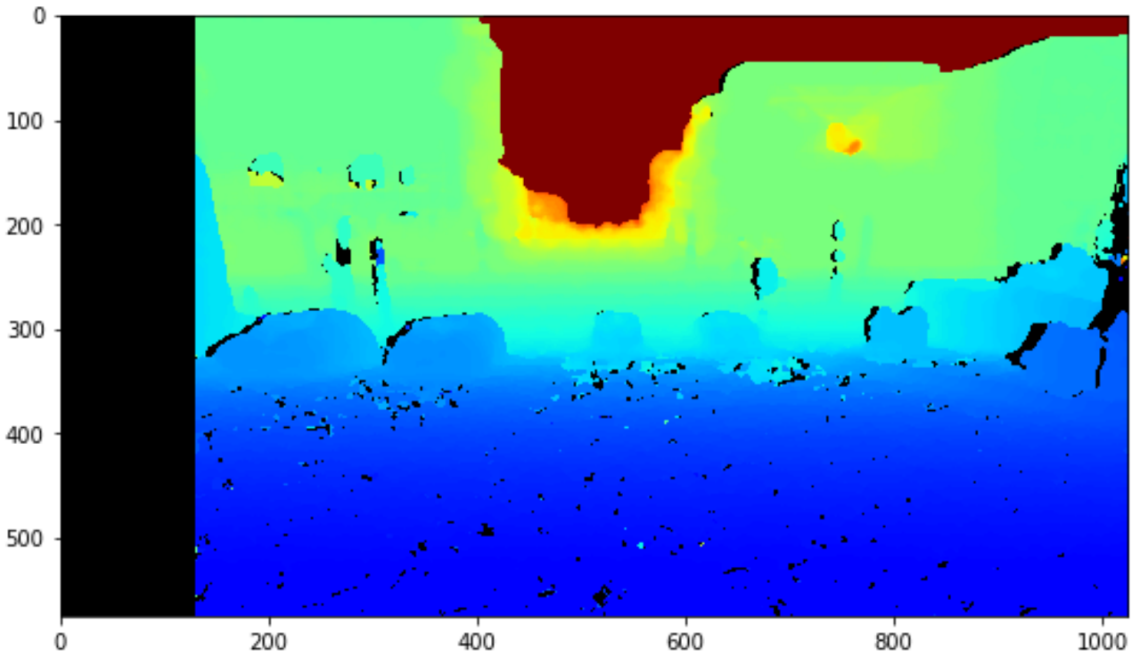


Figure 2.13: *An example of the SGM algorithm on the Synthia dataset based on the images on figure 2.12*

The biggest drawback of stereo matching is the need for a texture that can be identified. It's impossible to tell the distance of a white wall for example because the disparities of the pixels cannot be identified.

2.4.3 Light Detection and Ranging (lidar)

Lidars are the most capable distance measurement devices, but they have a lot of limitations that should be taken into calculations. Lidars are utilized in a variety of fields, including but not limited to agriculture, robotics, as well as the space industry [35]. I did not utilize any type of raw lidar measurements in my work, but as the real-world driving-related datasets are mostly collecting point clouds from lidars, it is important to have a brief overview of this marvelous technology.

Lidar is a widely used distance measurement device, which utilizes laser light to illuminate the surface and measures the reflected light with a sensor. Based on the travel time of the light and the speed of light, the traveled distances can be calculated. The provided measurements are the 3D coordinates of the measured points and every one of them has an intensity value as well. The reliability of a single measurement is relatively low, because of the way light is reflected from the surfaces. In some cases, photons get reflected, in other cases, they go through the material or get absorbed. This effect creates inconsistency over a homogeneous surface, which can be solved easily by analyzing multiple measurements to create a single data point with a sacrifice of refresh rate.

The well-known mechanical lidar consists of a rotating mirror, which reflects the outgoing laser light in the direction of the objects and also reflects the returning light into the sensor. This setup seems outdated compared to the technological advancements of digital photography. Rotating lidar in today's stage has a lot of disadvantages, such as the wearing of the mechanical parts, low resolution, and most of the time, they are very expensive. A really high-resolution rotating lidar can cost multiple times the price of the car it is attached to.

Another type of lidar called flash lidar is under development, which promises a lower entry price for a lower resolution sensor. The ultimate goal of flash lidars is to lower the price of the equipped vehicle dramatically without losing too many details about the environment. Flash lidars have some general advantages as well, such as it has no moving parts, (therefore no wearing, less possible point of failure) and no rolling shutter effect can be observed in the measurements.

Artifacts and limitations of lidar measurements

Just like cameras, lidars are far from being perfect and suffer some type of artifacts in the same way as cameras do. For example, lidars can be disturbed by the same lens flare as cameras. One of the advantages is a disadvantage as well in case of lidars: they emit the light that they use for measurement. This means that they can work well in the dark, but in the case of highly reflective and highly absorbing surfaces, no measurements can be created. The same goes for the sky, as it does not reflect any light.

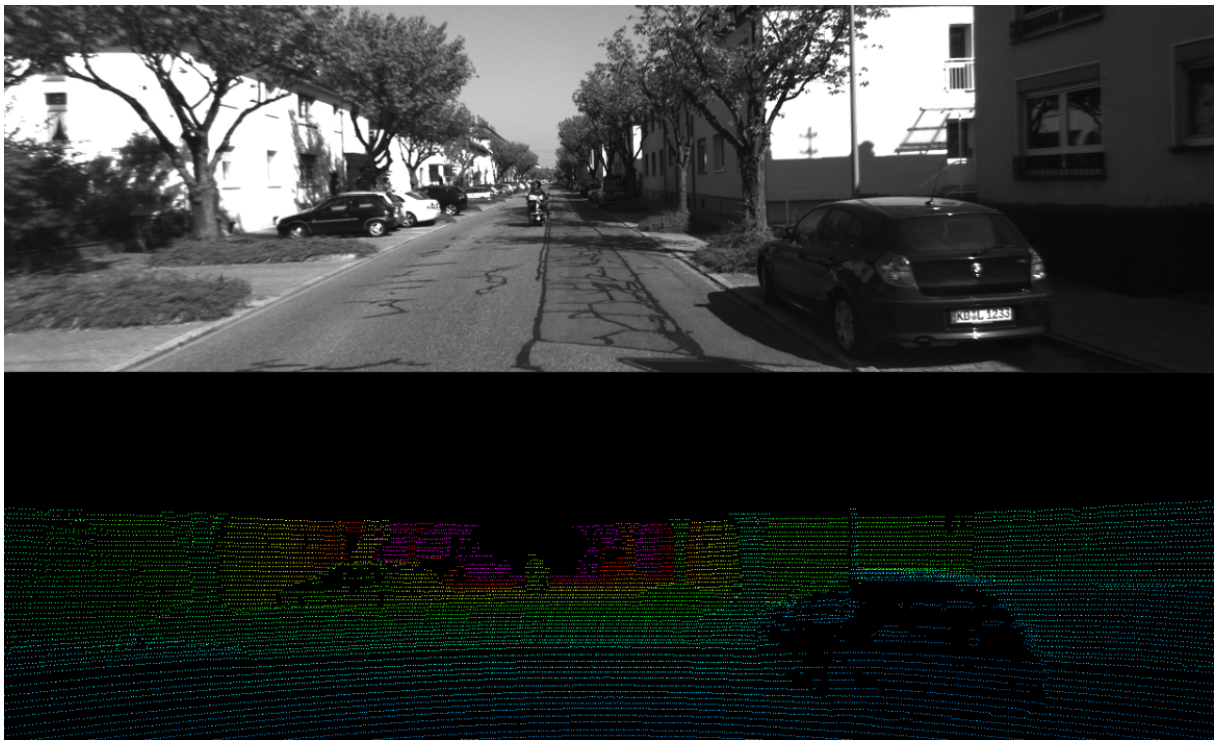


Figure 2.14: *An example lidar measurement from the Kitti dataset [9]*

Moving objects are another weak point of lidars too: the use of multiple measurements for a single data point can cause inconsistency at moving objects. In the case of mechanical lidars, the rolling shutter effect can be observed on moving objects as well.

In the case of lidars, particle occlusion, such as dust can be another challenge. Although the use of multiple measurements solves the issue in most cases, heavy rain, dust or fog eliminates a lot of useful information from the sensors, like in the case of cameras.

A significant limitation of the lidar measurements are the density of the measurements: approximately, 5-10% of the pixels contain measurement values after projection to the camera image.

2.4.4 Other sensor

In certain cases, other active measurement sensors, such as infrared cameras, ultrasonic sensors, and radars are more useful than vision-based estimation.

Radar can be used to determine the distance of an object from the vehicle. It can be used in any weather condition, at high speed, and has a long range as well. The drawback is, that radar cannot determine the shape of an object precisely. A creative way of utilizing radar is annotation for camera images [18].

Ultrasonic sensors are mostly useful for parking assistance, but they can be used for blind-spot monitoring and clearance detection at lane changing as well.

Chapter 3

Previous works

A number of different approaches can be found for the task of depth estimation. As I intended to use supervised learning, the most useful paper was the paper High Quality Monocular Depth Estimation via Transfer Learning [1]. In this work, a pretrained encoder-decoder convolutional neural network with skip connections was used, which suited my intentions as well. In this paper, the ground truth is densified from sparse distance measurements and even the authors point out that real-life datasets are not sufficient for supervised learning.

In paper [1], in section 4.4, the authors claim:

" Since our loss function is designed to not only consider point-wise differences but also optimize for edges and appearance preservation by looking at regions around each point, the learning process does not converge well for very sparse depth images. "

In the paper [8], also real-life datasets are utilized with densified ground truth values.

In [14], the movement of the camera provides information about the distances using unsupervised learning. Left an right camera images are utilized in paper [11] and [13] in an unsupervised manner.

Generating right image from left image and using a neural network with stereo input is the key idea in paper [26]. For further reading on unsupervised approaches, see [12] [23] [3] [2].

There are a huge number of other useful papers published on the topic of self-driving related tasks, which has been solved using deep learning. For such examples, see [30] and [4] for object detection, and [36] for semantic segmentation.

Although there are some papers [19] which combine both depth estimation and semantic segmentation task, I could not find any papers similar to the novel idea of weight mask for loss calculation based on the annotated semantic segmentation.

Chapter 4

Proposed work

In this chapter, important details about the proposed work will be introduced. There are some components that had to be implemented for all of the presumably working approaches. These parts are the dataset, the loss calculation and the neural network itself. Three main methods are going to be introduced and compared with each other. These are the depth estimation based on a monocular camera, stereo cameras, and weight mask for loss calculation with a monocular camera. The latter is a novel idea, which is intended to balance the size of the pedestrians and cars on a camera image compared to huge areas that are less relevant, such as the sky and buildings.

4.1 Dataset

In this work, the dataset in use is called Synthia [31], which is a set of synthetic images exported from a simulated driving environment. The reason behind the utilization of generated data is the density of the lidar measurements. As explained in section 2.4.3 real-life lidar sensors provide relatively sparse measurements, but in a simulated environment, the exportation of the z-buffer is relatively easy, so fully dense depth maps are usually provided.

Fully dense depth maps are required to train neural networks in a supervised way, and other metrics, such as the structural similarity explained in the losses section 4.2 can be calculated only on dense measurements as well.

Another advantage of the simulated dataset is the precision of semantic segmentation. For real-life datasets, the frames are annotated manually or estimated from other, neighboring frames. Because of human error, the precision of these manual annotations is not on the same level as the computer-generated semantic segmentation.

Before the RGB images and depth maps could be fed to the network, the values are normalized in between 0 and 1 in order to help the training process. In order to make the training process even faster and the video memory requirement smaller, the RGB, depth and segmentation images have been resized to 1204 by 576. This resolution is a little smaller than the HD resolution and it is really close to the resolution provided in the Kitti dataset, which has 1242 by 375. The scaling algorithm for this process was the

nearest neighbor, as in terms of the semantic segmentation images, no interpolation can be used.

For every further example, the RGB image on figure 4.1 will be used with the corresponding depth map and semantic segmentation.

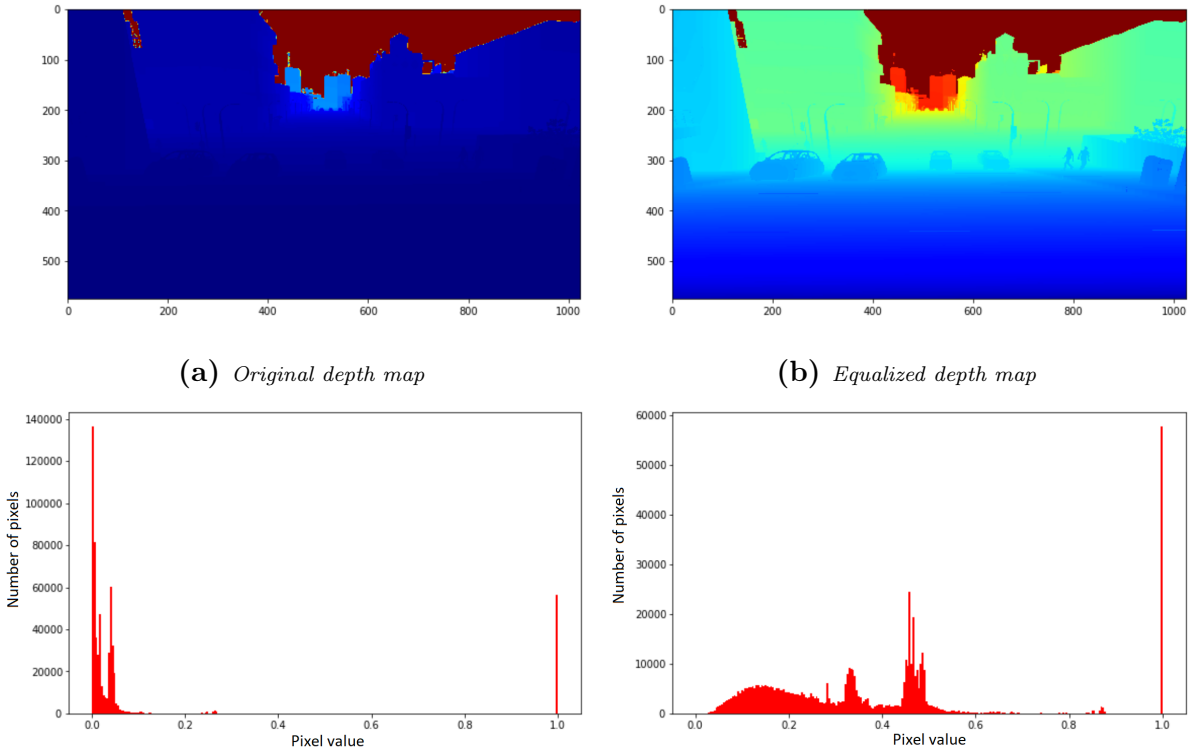
Figure 4.1: *An example of the left RGB camera from the Synthia dataset [31]*

For the visualization of the depth images, I used the jet 4.3 colormap from the matplotlib package, which provides a colorful depth map with high contrast for better visibility. The coloring follows the color order of the rainbow, the objects closer to the camera are colored blue and the objects far away are red. The sky, which basically does not have any distance value is colored claret. For even better visual results, the depth maps are equalized even more based on their histograms. On figure 4.2 the histograms of the original and equalized depth maps can be seen. On the original image, it is really hard to identify the cars on the left, for example. The original distances on the depth image were between 0 and 1000 meters, which scales to the interval of $[0, 1]$ after normalization.

Preprocessing of the semantic segmentation was needed as well. As it is basically classification, onehot encoding has been used with 23 different classes, and the predicted results have been converted back to RGB images for visualization 4.4.

4.2 Loss functions

For the previously introduced depth map estimation, the following losses are summed up throughout the training process: mean squared error (MSE) and mean absolute error (MAE) are in use to lower the differences between the real and the predicted distance



(a) *Original depth map*

(b) *Equalized depth map*

(c) *Histogram of the original depth map*

(d) *Histogram of the equalized depth map*

Figure 4.2: *Comparison of the original and equalized depth maps*

Figure 4.3: *The jet colormap from the Matplotlib package*

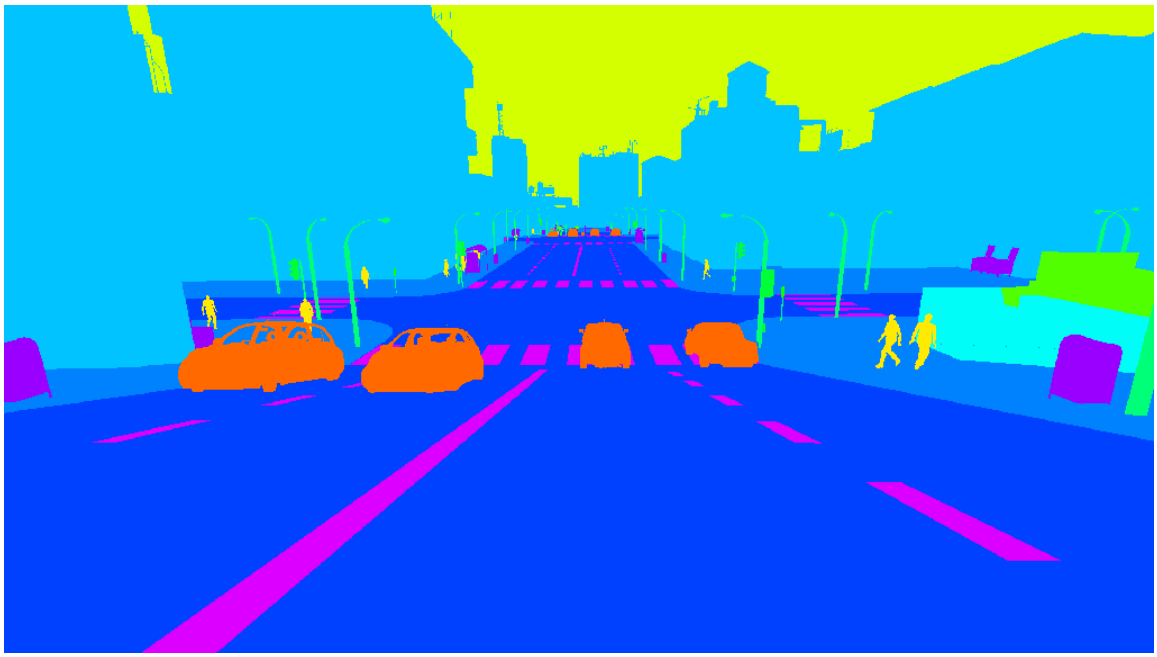


Figure 4.4: *An example semantic segmentation from the Synthia dataset [31]*

values. If the difference is relatively large (>1), MSE scales really good, as it returns the square of this difference. The problem with squaring is that it does not scale well between 0 and 1, so MAE is utilized for this reason. A loss function called Huber loss also solves this problem using MAE and MSE depending on the interval of the difference.

In some unsupervised approaches [39], the first derivative of the predicted image was also taken into the calculation in the loss function, as it can result in a smoother prediction. In a supervised manner, the first derivative of the ground truth can also be calculated, so it will not be simply minimized, but the absolute difference between the first derivative of the prediction and the ground truth will be minimized.

With the help of structural similarity (SSIM) [37], the difference of structure, contrast and luminance can also be introduced in the loss function. This helps to ensure to keep the shapes and contrasts for the different objects in the prediction image.

4.3 Model architecture

As the goal during this work was to compare a few methods of distance estimation, I tried to limit the amount of factors that influences the performance of the different solutions. The biggest factor is the model itself.

For this paper, I aimed to use only one general model architecture, which will be introduced in this section.

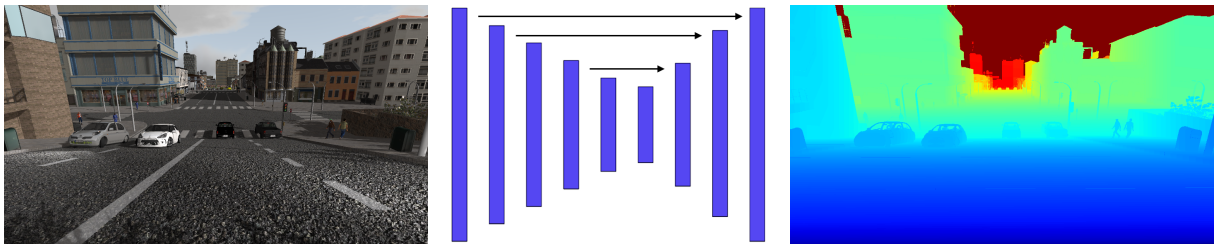
For this research project, the base of a previously proposed model is utilized from the publication of High Quality Monocular Depth Estimation via Transfer Learning [1], for every proposed solution of depth estimation.

The model is a convolutional neural network, which consists of an encoder and a decoder. The encoder part is used for feature extraction, which is a DenseNet [16] encoder, that has been pretrained on ImageNet [6]. As the output image size is equivalent to the input size, and the encoder is downscaling the image, the decoder part is used for upscaling. As the model itself contains a lot of layers and a lot of skip connections deriving from DenseNet, the detailed model architecture would be too big to include in this document. The detailed parameters of the used DenseNet-169 can be found in the paper [16], in table 1. A simplified model architecture with monocamera input and depth map output can be seen on figure 4.5.

4.4 Implemented methods

4.4.1 Single camera image as input for depth prediction

The simplest possible input for a neural network for the proposed problem is a single image from a monocamera. For the utilized dataset, both the left and right camera images and



(a) *Input RGB image* **(b)** *Neural network* **(c)** *Output depth map*

Figure 4.5: *The convolutional neural network has an encoder-decoder architecture with additional skip connections*

the corresponding ground truth values were given, which made it possible to utilize both measurements throughout this type of training. This technique can be seen as some kind of augmentation because there are only small differences in the two measurements. None of the image pairs of a single measurement was used for both the training and testing set, so the presented results are representative for this dataset.

In the case of a monocamera, there is no traditional method for calculating the distance values for each pixel. As a traditional method, the distance of a car could be estimated based on its size and solid angle, but this method has not been implemented, as it can be used only for some objects in some circumstances. As the only input is a single image, the neural network can only learn or extract some kind of features for each object, and the distance values - which is only another feature - can be predicted.

4.4.2 Stereo camera as input for depth prediction

Another intuitive solution for the input is the stereo camera setup. In this case, the number of input channels is 6, as the left and right RGB images are concatenated together. The traditional method for calculating the distances from stereo images (SGM) has been introduced in section 2.4.2.

4.4.3 Semantic segmentation for loss calculation

On the camera images, some objects are represented using only a really small amount of pixels compared to the full image resolution, but this property does not mean that an object is further away or is less relevant. A great example of this effect is pedestrians: they are highly involved in the traffic scenes and they are also really fragile, so the precise distance detection for them is a necessity. The following method provides a solution to compensate for the size of the relatively small objects. With the help of the semantic segmentation data, the loss function for some classes - pedestrians and vehicles - are taken into account with a bigger weight. For other unimportant classes, such as the sky or trees, the loss is reduced. And example of a weight mask can be seen on figure 4.6,

where the black areas are the less important parts and the loss is calculated mostly on the white areas.

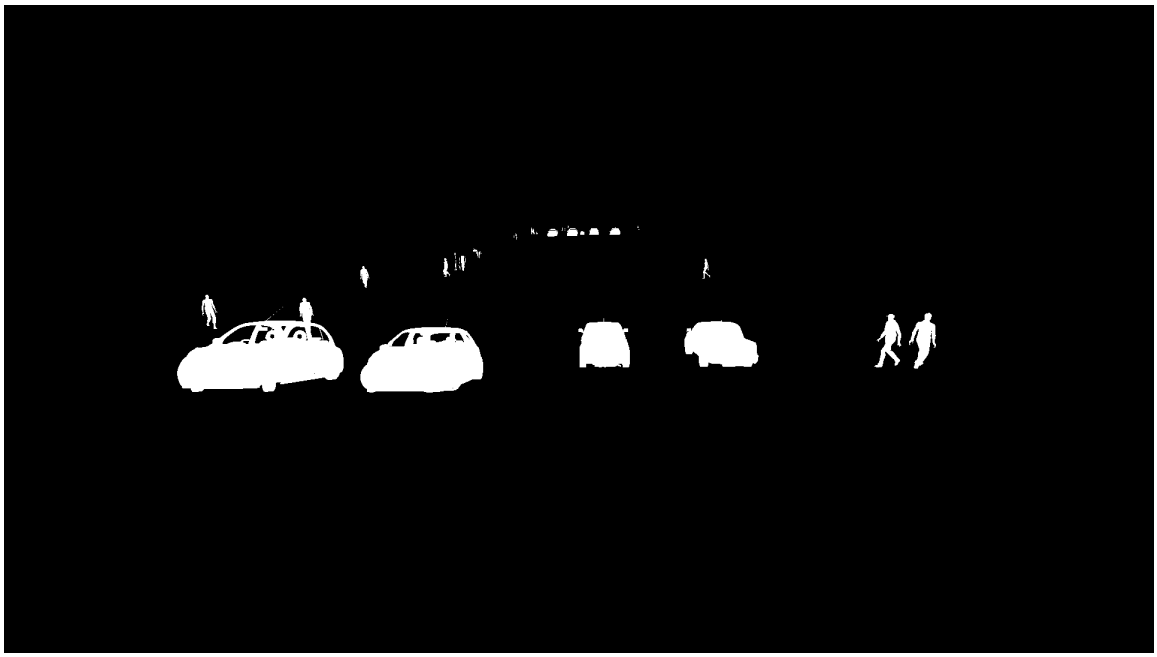


Figure 4.6: *An example weight mask for the cars and pedestrians in the scene*

Chapter 5

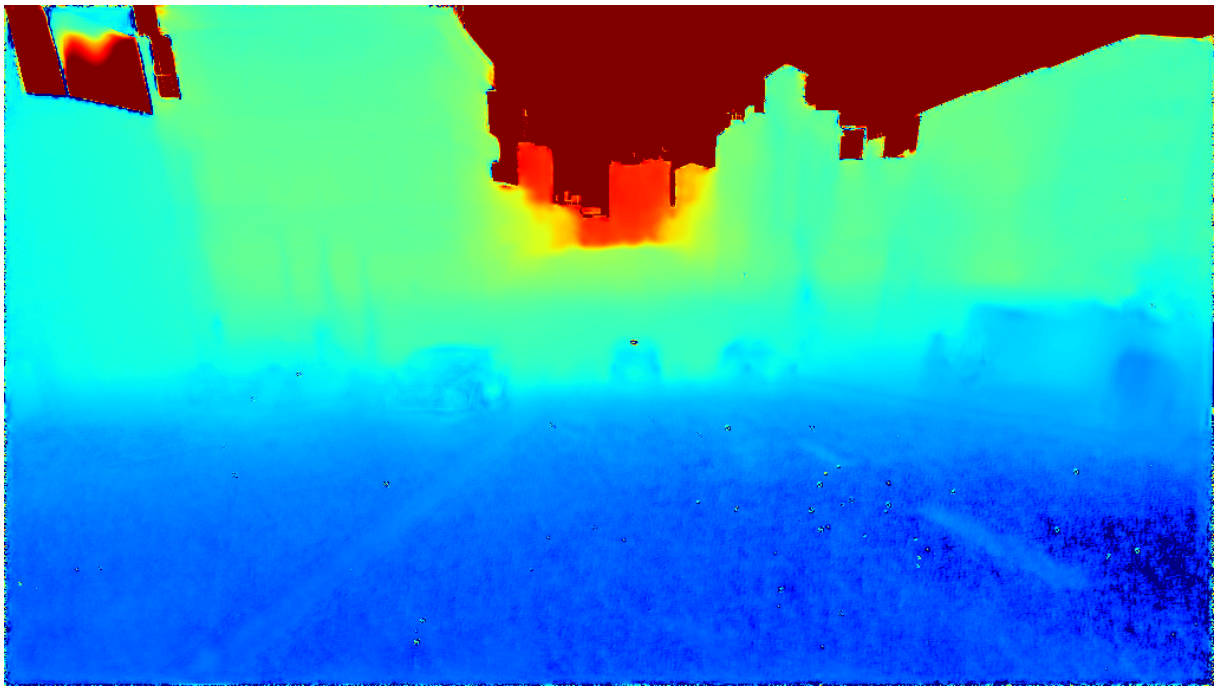
Results

5.1 Monocamera as input

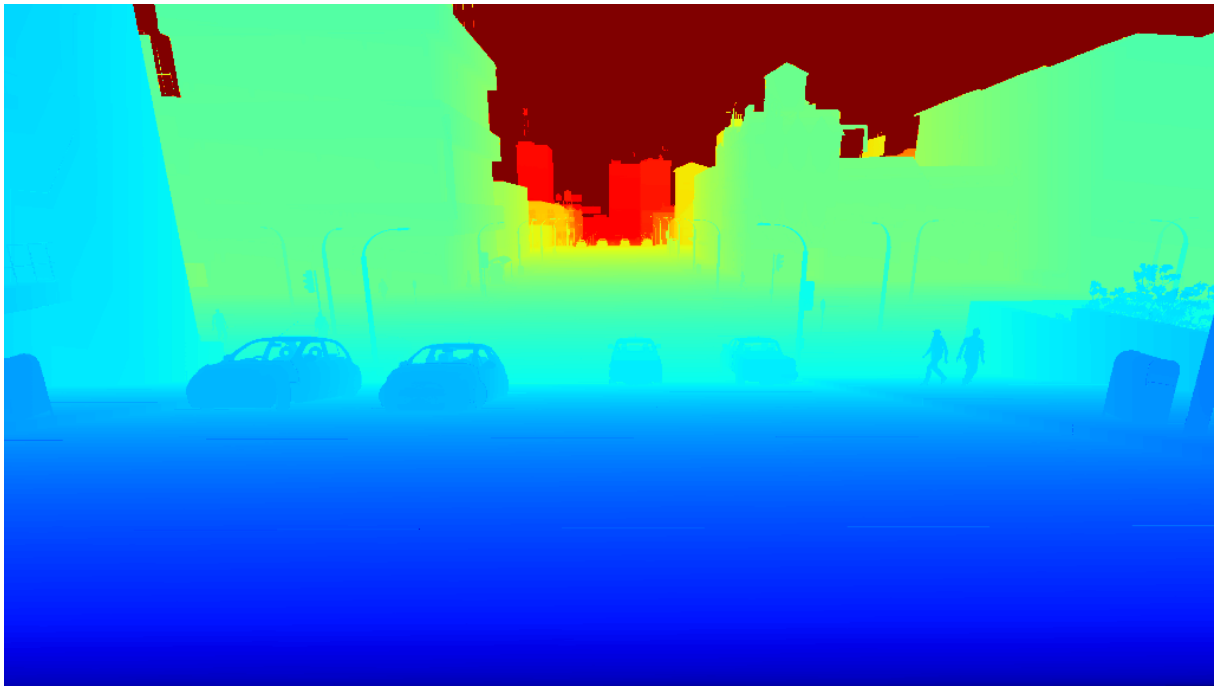
The results of the monocamera setup can be seen in figure 5.1. Visually, the results seem really good, but some details are missing from the prediction. Most importantly, the outlines of the cars on the left are barely visible, which means that the network failed to differentiate them from the background. Interestingly, the network is able to estimate the distances of the buildings far away. Notice the window on the top left corner image. On the input RGB image 4.1, the color and the position of the window would suggest it to be a part of the sky and the network has failed to identify it correctly. Some parts of the image contains noise similar to salt and pepper noise, but its amount is much smaller compared to the SGM 2.13. Also, the SGM failed to calculate correct distances for the sky on the top right region.

On figure 5.2, the error images of the prediction can be seen. These images were pretty useful determining the performance of a given model independently from calculated evaluation metrics. On the absolute error image 5.2a, the absolute difference of the ground truth and the prediction have been calculated for each pixel and the image has been equalized based on its histogram in order to improve visibility of some regions. Just like the other images, this one has been visualized with the help of the jet colormap from matplotlib. The biggest error can be easily noticed, as the previously mentioned windows are red in the top left corner. There are some important effects that should be noted: the cars, pedestrians and poles are easily identifiable on the error image. This means, that either the estimation for the background or for the object is not valid. On the prediction image, this effect can be seen as the objects are blurred into the background.

The error of the first derivative of the prediction can be seen on figure 5.2b. The same objects are easily identifiable, meaning that the edges for these objects contain high error values. The cause of this is the previously explained blurring.

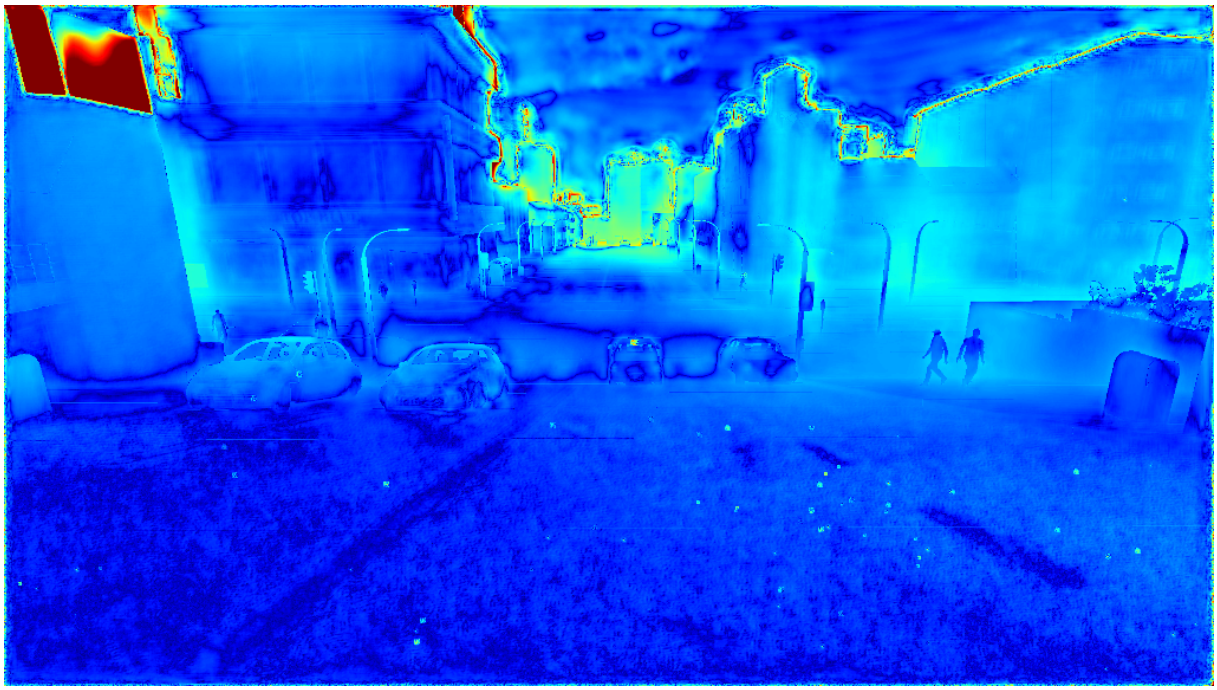


(a) *Predicted depth map*

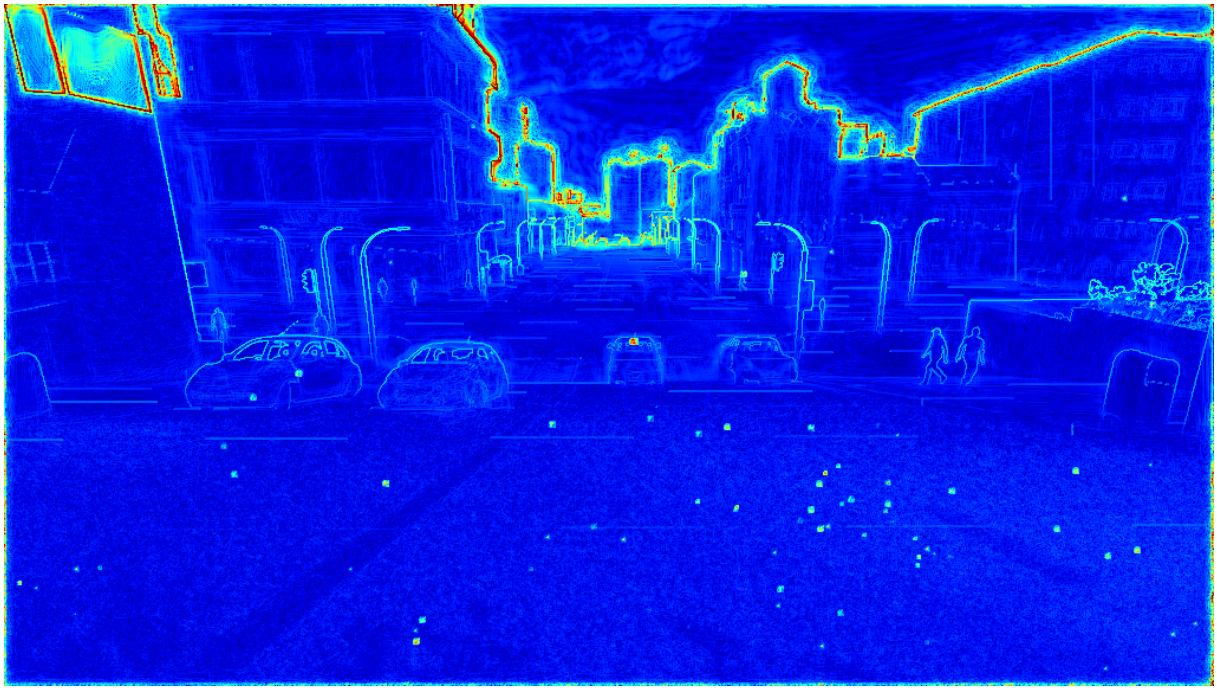


(b) *Ground truth depth map*

Figure 5.1: *Comparison of the predicted and ground truth depth maps for the monocamera input*



(a) *Absolute error image*



(b) *Error of the first derivatives*

Figure 5.2: *Equalized error images for the monocamera input*

5.2 Stereo cameras as input

The neural network with stereo input images performs a little better than the monocamera used as input. The prediction can be seen on figure 5.3. For example, the error for the window in the top left corner is much smaller, as more spacial information was available to the network. Currently, these results are not reflecting my expectations about stereo input, further exploration of this method is needed.

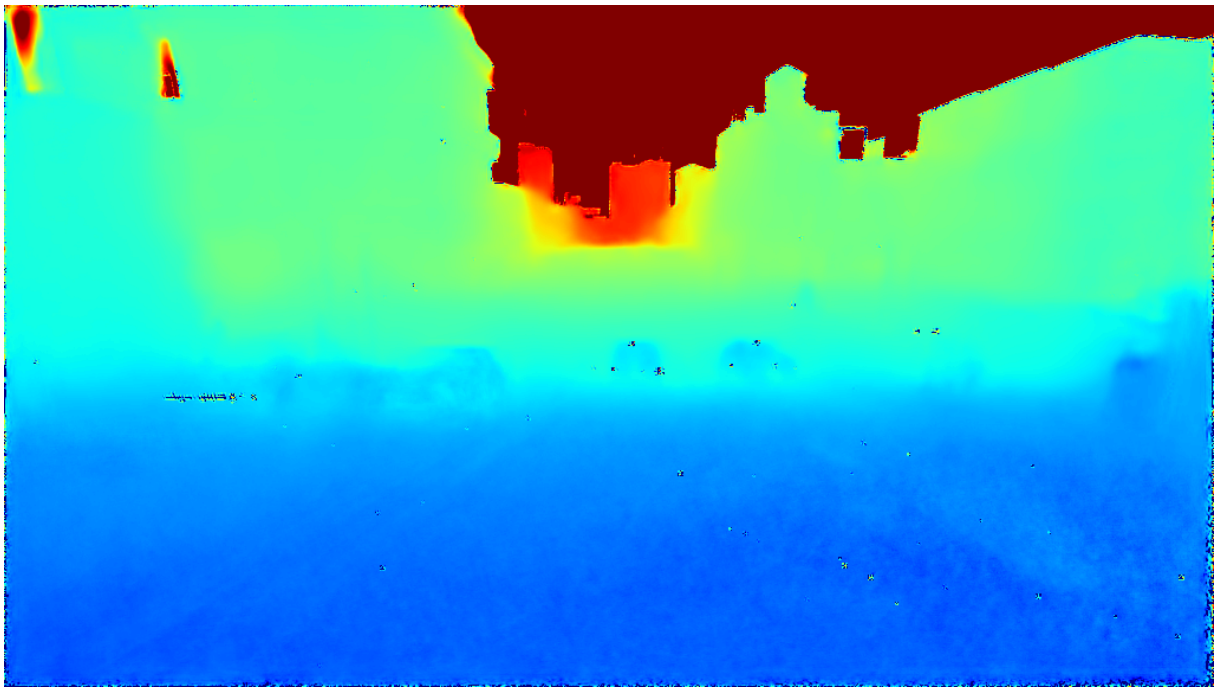


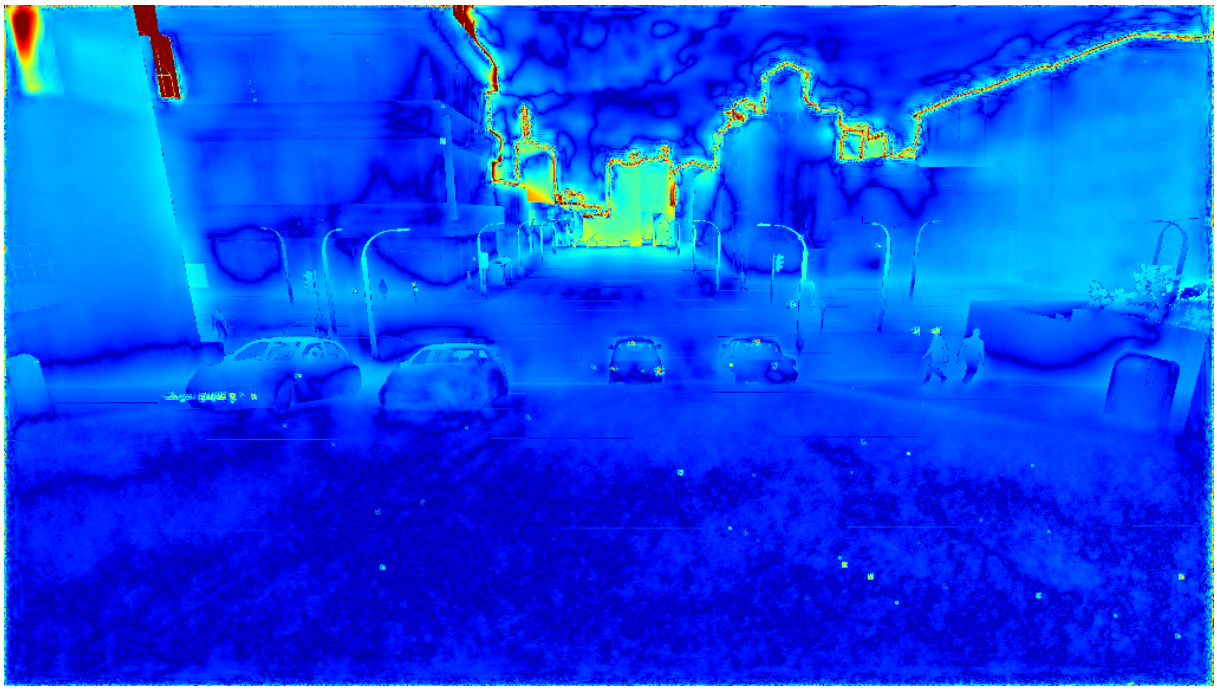
Figure 5.3: *Prediction of the network with stereo input images*

5.3 Masked loss calculation

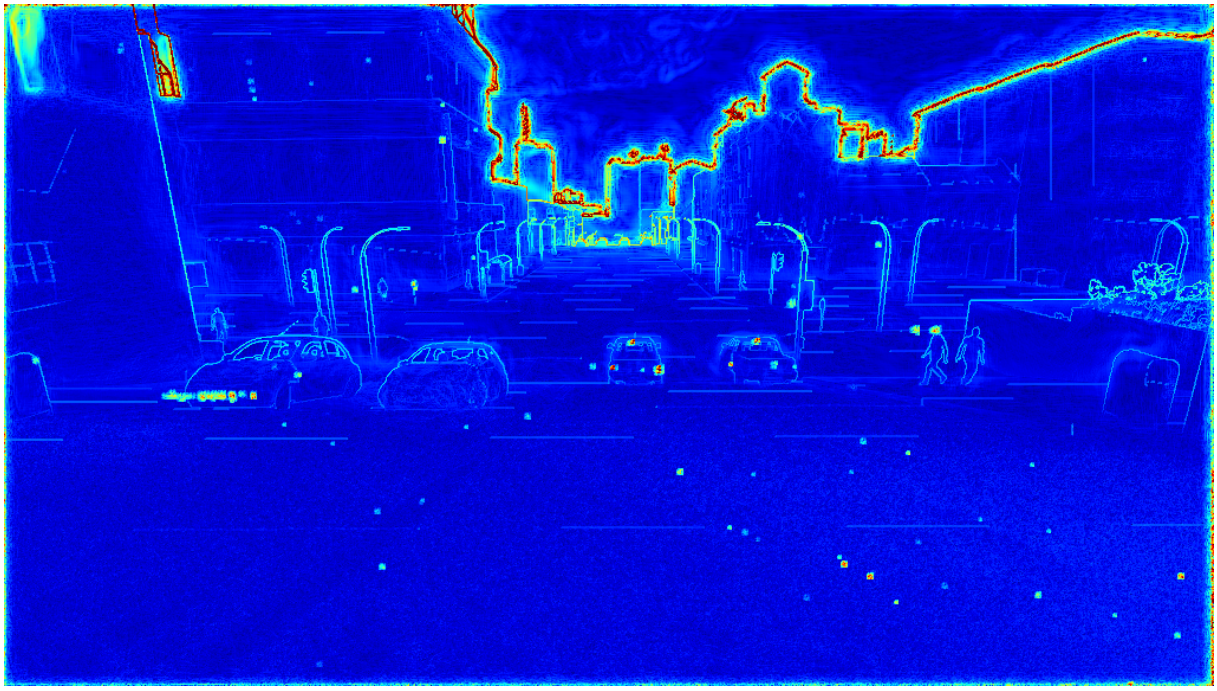
With the use of the weight mask for loss calculation, the prediction contains some areas, where the improvement is noticeable. On figure 5.5, an example prediction can be seen. For instance, the cars on the left appear to have better distance values than on figure 5.1b or 5.3. Overall, the result seems still blurry, generally speaking, the edges of the objects are hardly visible.

5.4 Comparison of the result

As all of the proposed methods use the same model, a comparison between them is reasonable. All three proposed method approximates the ground truth depth map quite well. Both in the case of the stereo input and the weighted loss, some improvement can be seen on the predictions compared to the monocamera approach. In table 5.1, the



(a) *Absolute error image*



(b) *Error of the first derivatives*

Figure 5.4: *Equalized error images for the stereo camera input*

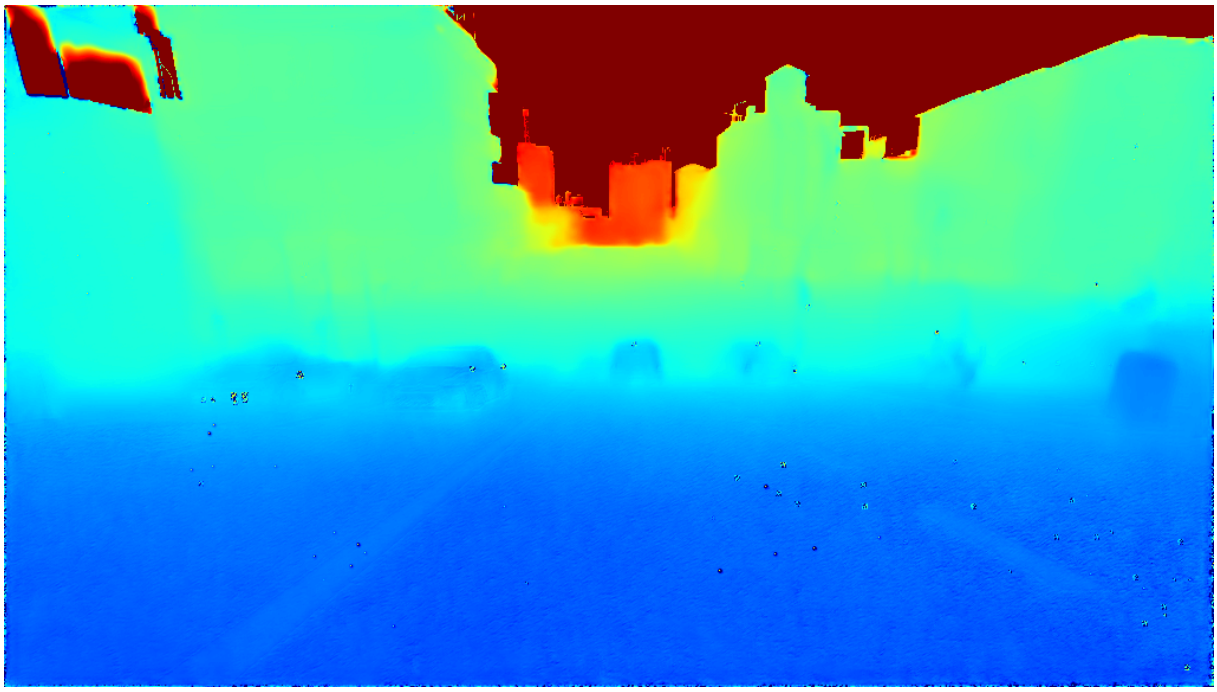


Figure 5.5: *An example prediction from the weight mask method*

comparison of the losses can be seen: as expected, the validation losses are higher than the train losses. Compared to the monocamera method, both the stereo and weight mask approaches are better. These little differences can also be seen on the previously introduced visual results. Interestingly, both losses in the weight mask method are significantly lower, but the validation losses are just a little lower than in the other tasks. This could be an indication of faster convergence to the train set.

	MAE on the train set	MAE on the validation set	SSIM loss on the train set	SSIM loss on the validation set
Monocamera setup	$4.731 \cdot 10^{-3}$	$7.853 \cdot 10^{-3}$	$3.359 \cdot 10^{-5}$	$1.333 \cdot 10^{-4}$
Stereo cameras setup	$3.299 \cdot 10^{-3}$	$6.477 \cdot 10^{-3}$	$3.748 \cdot 10^{-5}$	$1.244 \cdot 10^{-4}$
Monocamera with weighted loss	$2.3 \cdot 10^{-3}$	$7.275 \cdot 10^{-3}$	$1.697 \cdot 10^{-5}$	$1.248 \cdot 10^{-4}$

Table 5.1: *Comparison of some losses from the different methods*

Chapter 6

Implementation details

6.1 Hardware and software environment

During my work, I was able to access one of the machines owned by the Department of Telecommunications and Media Informatics. This made it possible to use a GTX Titan X GPU, which made it possible to train the neural networks within a relatively short time. Usually, overfitting was achieved within 20-28 hours. Later on, the department received a DGX Station with a really high-end hardware setup. This machine contains a 20-core CPU, 256 GB of RAM and features four Nvidia Tesla V100 GPUs with 32 GB VRAM each, which made it possible to run multiple trains with different configurations and finish them in around 15 hours. I cannot wait to test the training time improvement using all GPUs for a single train.

The software prerequisites were already satisfied, as I was working on another deep learning project on the same machine in the previous semester and my Docker setup made it possible to start the project quickly, but currently, I also miss some features and performance improvements because of the already outdated libraries. I implemented every part of the project in Python, and the mainly utilized deep learning library was the Keras API with TensorFlow backend. A future work is to update my Docker environment and use the recently released TensorFlow version 2.0 package.

The current stage of the project can be found at https://github.com/ffabi/Project_TDK

6.2 Training parameters

During the period of creation of this work, most of the training parameters were fixed in order to reduce the difference between the various results. The number of parameter in the model with stereo images as input and both depth map and semantic segmentation as output was around 29 million. The optimizer was the widely used Adam [22] with a learning rate of 0,0001. For the learning rate decay, the ReduceLROnPlateau Keras callback was used with a patience of 5 epochs and a factor of 0,7. Other Keras callbacks, such as ModelCheckpoint and EarlyStopping were pretty useful as well. During training early stopping was in use with a patience of 16 epochs. During training, after each epoch

the weights of the model has been saved if the validation loss has improved significantly. After training, the saved model has been loaded and evaluated. Tensorboard was really useful in order to keep track of the losses of the trainings. The same goes for the saving of the predicted images after each epoch, in order to visualize the current results of the network during training.

Chapter 7

Future works

With the collected experience of this project, I would be able to test some other methods that could improve the performance of the depth estimation. One of the method that I am currently curious about is multitask learning, which could help to achieve higher accuracy in several different tasks [20]. This could be implemented by using several convolutional decoders for each task. In this manner, the network would learn faster and would achieve better performance in the depth estimation than the single task version.

Chapter 8

Summary

As self-driving is going to be part of everyday life, safety has to be guaranteed in every possible driving condition. In this work, the task of depth estimation has been introduced and implemented.

Depth estimation appears to be a problem that can be solved using a monocular camera which is quite promising and appealing. My assumption is that with some improvements the proposed approaches could be utilized in the automotive industry. The novel idea of weighting the loss according to the semantic segmentation appears to be helpful. Although the results of the stereo camera setup and the weighted loss calculation do not reflect my expectations, the visual differences are noticeable. The utilization of a more robust and more varied dataset would be required in order to validate the proposed results.

Chapter 9

Acknowledgement

I would like to express my deepest appreciation to Bálint Gyires-Tóth and Róbert Zsolt Kabai for the generous and neverending support. The supervision and technical help from them was really helpful in the creation of this work. I am also grateful to Róbert Moni, who helped my advancements in the field of machine learning in the previous semester and helped me out in this project as well. This research was supported by the Department of Telecommunications and Media Informatics and Continental Automotive Hungary Kft., as they provided all the required computational hardware for this project within the Professional Intelligence for Automotive student research agreement.

List of Figures

Bibliography

[1] Ibraheem Alhashim and Peter Wonka. High quality monocular depth estimation via transfer learning. *arXiv preprint arXiv:1812.11941*, 2018.

[2] Yasin Almalioglu, Muhamad Risqi U Saputra, Pedro PB de Gusmao, Andrew Markham, and Niki Trigoni. Ganvo: Unsupervised deep monocular visual odometry and depth estimation with generative adversarial networks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 5474–5480. IEEE, 2019.

[3] Vincent Casser, Soeren Pirk, Reza Mahjourian, and Anelia Angelova. Depth prediction without the sensors: Leveraging structure for unsupervised learning from monocular videos. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 8001–8008, 2019.

[4] Xiaozhi Chen, Kaustav Kundu, Ziyu Zhang, Huimin Ma, Sanja Fidler, and Raquel Urtasun. Monocular 3d object detection for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2147–2156, 2016.

[5] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[8] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, and Dacheng Tao. Deep ordinal regression network for monocular depth estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2002–2011, 2018.

[9] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[10] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[11] Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 270–279, 2017.

[12] Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel Brostow. Digging into self-supervised monocular depth estimation. *arXiv preprint arXiv:1806.01260*, 2018.

[13] Matan Goldman, Tal Hassner, and Shai Avidan. Learn stereo, infer mono: Siamese networks for self-supervised, monocular, depth estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[14] Ariel Gordon, Hanhan Li, Rico Jonschkowski, and Anelia Angelova. Depth from videos in the wild: Unsupervised monocular depth learning from unknown cameras. *arXiv preprint arXiv:1904.04998*, 2019.

[15] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[16] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[17] Tesla Inc. Autopilot, 2016. `https://www.tesla.com/en_EU/autopilot` (Accessed: October 27, 2019).

[18] Tesla Inc. Tesla autonomy day, 2019. `https://youtu.be/Ucp0TTmvqOE?t=4151` (Accessed: October 27, 2019).

[19] Jianbo Jiao, Ying Cao, Yibing Song, and Rynson Lau. Look deeper into depth: Monocular depth estimation with semantic booster and attention-driven loss. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 53–69, 2018.

[20] Andrej Karpathy. Multi-task learning in the wilderness, 2019. `https://slideslive.com/38917690/multitask-learning-in-the-wilderness` (Accessed: October 27, 2019).

[21] Andrej Karpathy. A recipe for training neural networks, 2019. https://karpathy.github.io/2019/04/25/recipe/ (Accessed: October 27, 2019).

[22] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[23] Yevhen Kuznietsov, Jorg Stuckler, and Bastian Leibe. Semi-supervised deep learning for monocular depth map prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6647–6655, 2017.

[24] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[25] Zhengqi Li, Tali Dekel, Forrester Cole, Richard Tucker, Noah Snavely, Ce Liu, and William T Freeman. Learning the depths of moving people by watching frozen people. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4521–4530, 2019.

[26] Yue Luo, Jimmy Ren, Mude Lin, Jiahao Pang, Wenxiu Sun, Hongsheng Li, and Liang Lin. Single view stereo matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 155–163, 2018.

[27] MissingLink. Neural network bias: Bias neuron, overfitting and underfitting, 2019. https://missinglink.ai/guides/neural-network-concepts/neural-network-bias-bias-neuron-overfitting-underfitting/ (Accessed: October 27, 2019).

[28] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[29] Sebastian Raschka. Single-layer neural networks and gradient descent, 2015. https://sebastianraschka.com/Articles/2015_singlelayer_neurons.html (Accessed: October 27, 2019).

[30] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[31] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3234–3243, 2016.

[32] SAE. Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems, 2014. https://www.sae.org/standards/content/j3016_201401 (Accessed: October 27, 2019).

[33] Silvio Savarese. Stereo systems, multi-view geometry, 2015. http://cvgl.stanford.edu/teaching/cs231a_winter1415/lecture/lecture6_affine_SFM_notes.pdf (Accessed: October 27, 2019).

[34] Raphael E Stern, Shumo Cui, Maria Laura Delle Monache, Rahul Bhadani, Matt Bunting, Miles Churchill, Nathaniel Hamilton, Hannah Pohlmann, Fangyu Wu, Benedetto Piccoli, et al. Dissipation of stop-and-go waves via control of autonomous vehicles: Field experiments. *Transportation Research Part C: Emerging Technologies*, 89:205–221, 2018.

[35] Space Exploration Technologies. Spacex's dragoneye navigation sensor successfully demonstrated on space shuttle, 2009. https://www.spacex.com/press/2012/12/19/spacexs-dragoneye-navigation-sensor-successfully-demonstrated-space-shuttle (Accessed: October 27, 2019).

[36] Marvin Teichmann, Michael Weber, Marius Zoellner, Roberto Cipolla, and Raquel Urtasun. Multinet: Real-time joint semantic reasoning for autonomous driving. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1013–1020. IEEE, 2018.

[37] Zhou Wang, Alan C Bovik, Hamid R Sheikh, Eero P Simoncelli, et al. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[38] Mahmood Yousefi-Azar, Vijay Varadharajan, Len Hamey, and Uday Tupakula. Autoencoder-based feature learning for cyber security applications. In *2017 International joint conference on neural networks (IJCNN)*, pages 3854–3861. IEEE, 2017.

[39] Huangying Zhan, Ravi Garg, Chamara Saroj Weerasekera, Kejie Li, Harsh Agarwal, and Ian Reid. Unsupervised learning of monocular depth estimation and visual odometry with deep feature reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 340–349, 2018.

[40] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised learning of depth and ego-motion from video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1851–1858, 2017.