TECHNISCHE HOCHSCHULE NÜRNBERG
GEORG SIMON OHM

IT-Projekt
Autonmously driving Remote Control Car
with DonkeyCar and Raspberry Pi 3B+

LÖHR Tim, **BOHNSTEDT** Timo, **PALPANES** Ioannis

***Abstract*—For our group IT-project at the *University of Applied Science Georg Simon OHM*, we built an autonomously driving remote control car. To accomplish this project, we used the framework *Donkey-Car*. Supervised learning is the primary component, but we provide related work and projects to compare this project with the *state-of-the-art* Reinforcement Learning. We explain how to connect the car with the microcontrollers and cables, to do a full reverse-engineering on the project. Every single component will be introduced in an appropriate complexity for non-electrical-engineers. The neural network is a *VGG16*-Network, which will be explained with all details and parts to understand the networks' architecture fully. We further provide information about how the car can be driving with the webserver. The Report gives a detailed explanation of the API and the transferred data.**

***Index Terms*—Machine Learning, IT-Project, RC-Car, RaspberryPi, Autonomously driving**

1. Project background and motivation

We decided to commit ourselves to this kind of project because we are interested in many aspects related to machine learning. We knew that our skills and knowledge were at the starting point, not deep enough. Still, after completing this project within two semesters, we gained a lot of information and understood the details of machine learning more as before. Along the many hours we spent, we had to learn new things, and we can say with some proud that we managed to make the car drive autonomously - with a lot of effort though. It is known that the car industry(especially in Germany) undergoes a shift towards electric and autonomous-driving cars currently. The old engine-driven vehicles seem to be *outdated technology*. For three eager students who are interested in computer science and everything related to it, a new opportunity arose and is still occurring. Especially in the area of autonomous-driving cars, companies invest a lot of money and time. They also search for talents which could make a difference in the future. It was especially hard, in the beginning, to split the responsibilities and figure out who takes care of which element in the project. Even the first car we bought was not compatible with the hardware we purchased (Raspberry Pi 3B+, Servodriver, e.g.). That did not stop us from learning about the equipment and the software we wanted to use more precisely. How we combine the hardware with the right software, so that the car can move on its own, was one of the most exciting and exciting aspects we wanted to explore. But in the end, an essential aspect and the reason why we wanted to do the project in the first place, was the software - especially: the neuronal networks, machine- and deep learning software. This area can not only be used for cars and hardware-related problems, but it is also used for understandable software such as movie recommendations in, e.g. the Netflix interface. The main question we asked ourselves was: *How can we teach our software to steer a car autonomously with the right velocity on a defined road with the help of images we fed into the database?* In the following, we explain to the reader exactly how we did it. We hope the reader enjoys the topic as much as we did along the hard but rewarding way.

2. Literature Survey

2.1 The Level of autonomously driving Cars

Let us start the survey by defining the autonomously driving itself first. Since this driving evolved within an extended period, different levels of autonomous driving have been invented and get officially declared. There are defined five levels of vehicle autonomy from the National Highway and Traffic Safety Administration (NHTSA) [1].

[1] https://www.truecar.com/blog/5-levels-autonomous-vehicles/

- *Level 0 - No Automation*: The car is driven by a human
- *Level 1 - Driver Assistance*: There is a single-vehicle control system like steering into the middle of the lane or braking the tires when the distance to the car in front is too close, but only one at a time.
- *Level 2 - Partial Automation*: The car performs the accelerating and decelerating by itself. The human is still supposed to look over the environment and be aware of the surroundings.
  Example: Tesla Autopilot [1]
- *Level 3 - Conditional Automation*: The autonomous car takes complete control even under special occurrences like an animal on the road. The human is still expected to intervene when it is required.
  Example: Google Waymo *self-driving car* [2]
- *Level 4 - High Automation*: The car takes over the complete control of the driving system, and the human is not required to pay any attention to the environment. The car is expected to perform all driving tasks safely and without special circumstances.
  Example: Waymo announced in 2017 that they are testing *level 4* driving [3]
- *Level 5 - Full Automation*: The car takes over the complete control of the driving system, and the human is not required to pay any attention to the environment, nor does need to be inside of the car.

2.2 Reinforcement Learning

The two significant approaches in the Machine Learning field are supervised and unsupervised learning. Supervised learning requires labelled data, and this means the data is already tagged with the *correct answer*. The model learns on the data and their labels and predicts an output based on that. Whereas the unsupervised learning describes how the model works on its own to discover the information. It deals mostly with unlabeled data and gets penalized if it produces wrong results. Unsupervised learning has some prime advantages over supervised learning:

- Finds all kind of unknown patterns: Unknown objects on the road, broken streets
- Optimizes the performance criteria with the use of experience
- Solves various types of real-world problems, such as autonomous driving

We found that a team from the North China University of Technology already approached the try to drive the donkey car autonomously [4]. The team used the *gym environment* from OpenAI [2]. This open-source project from OpenAI allows building relatively easy penalizing models to implement reinforcement Learning. OpenAI provides plenty of API's such as reset(), step() or is_game_over(). These methods are the core of the model to recognize its faults. They use the DDQN algorithm to perform reinforcement Learning. DDQN stands for Deep Reinforcement Learning with Double Q-learning [5] developed by a team of researchers at *Google DeepMind*. Explaining how this algorithm works would exceed the limits of this paper, but the final output of the team [4] was autonomously driving donkey car. The training process was fairly slow, and it didn't drive completely exact, but the experiment shows that reinforcement Learning can be implemented together with the donkey car quite easily.

[2] https://gym.openai.com/

Fig. 1. Remote Control Car

3. Hardware

Usually, the car is driven by remote control. Which sends signals to the steer of the wheels depending on which buttons the controlling user pushes. In our case, we want to control the car only with the computer itself. To fulfil that, we will need to buy some extra parts, rewire the remote control car's servomotor and connect it to the onboard Raspberry Pi 3B computer. We will also need a camera to capture the images and a servomotor controller.

3.1 RC Car

Our car is the *s-idee® 18175 9115 RC Car Buggy waterproof monster-truck 1:12*. The remote control car is the heart of our project. Even though every RC Car can be used to drive autonomously, it is a better choice to buy a real model car, because a real servomotor makes it much easier to connect the pins to. The car's ESC (electronic speed controller) must be apart from its signal receiver. The steering servo and the ESC must be connectable to the servo motor controller for the Raspberry Pi to control both fully. It is an advantage if the ESC supports fine-tuning of the steering angle to drive the car more precisely. The cost of a real model RC car is approximately 100 euros, and so was the price of ours. The car must be quite large because we need to store the Raspberry Pi, the camera and a power bank for the Pi onto the vehicle without losing to much speed of the car. To keep everything in position, we attached it with a 3D print of a plastic holding-system, which was tailored by us for our car.

3.1.a Technical Details: Our car comes with 2,4 GHz type of remote control and is 40 km/h fast. It only weights 1,88 Kg. Since the vehicle is typically steered by the remote control, in our case, we use the Raspberry Pi to drive it

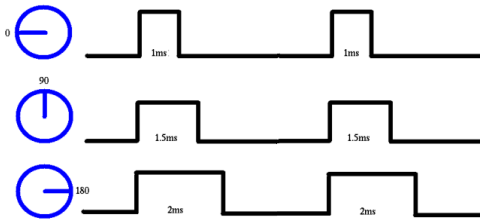

Fig. 2. PWM signal

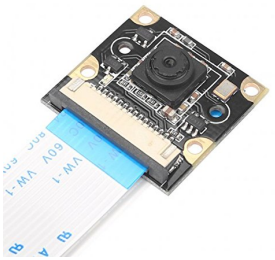

Fig. 3. Raspberry Pi Camera Module

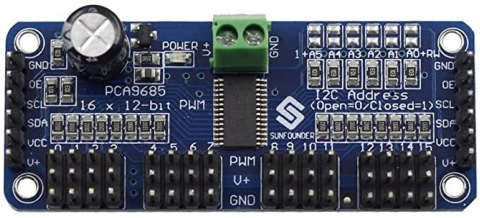

Fig. 4. SunFounder PCA9685

autonomously. The battery is a 9,6V with 800 mah, which gives us plenty of time to capture pictures of the track to train the model. The car-like case is easy to remove and to attach the Raspberry Pi and the rest onto it.

3.1.b Servomotor: The servomotor is the key to connect the Raspberry Pi with the RC car. A normal servomotor has two times three pins attached to it. One responsible for the steering and the other one for the speed. The steering is managed by a so-called PWM (Pulse Width Modulation), as shown in figure 2. Depending on the length of the signal, the steering performs a certain degree of changing its angle. We need to configure and set up the optimum values for our car in the Python *steering* file.

3.2 Raspberry Pi

Our group project aims to be affordable, so we chose the Raspberry PI 3 Model B+. There would be a better onboard computer such as the *NVIDIA DRIVE PX Pegasus* for larger projects, but the computing power exceeds our requirements by far. Even though by now the Raspberry Pi 4 is out, we still use the Raspberry Pi 3B+, because we are sure that it is compatible with our other microcontrollers and software.
We use a 16GB SanDisk to store the operating system and the machine learning model on the Raspberry Pi.

3.2.a Technical Details: The Raspberry Pi 3B+ is equipped with a lot of different extensions. It has a Wi-Fi module to communicate with the Python web server and a fairly fast processor to run the neural network and classify where to drive next. The Processor is the Broadcom BCM2837B0, Cortex-A53 with 1.4GHz and it has 1GB LPDDR2 SDRAM.
Furthermore, for the connectivity, the Pi has extended 40-pin GPIO headers. The weight is only 42 grams, which makes it easy to attach it onto the RC car. The *Donkey Car Project* [6] provides a prepared operating system for the Raspberry Pi called *donkeypi*. This OS is based on the basic *raspbian OS* from the Raspberry Pi company, but also includes the prepared Python files to start the webserver, read in the images and drive autonomously based on the images classified through the trained model.

3.2.b Cameramodule: For our purpose, we use the Raspberry Pi Camera Module 8MP v2.1. It can capture pictures with 1080p and 8-megapixel focus. It is straightforward to plug the camera onto the Raspberry Pi. The camera connects directly into the CSI (Camera Serial Interface) connector on the Raspberry Pi with a 15 pin Ribbon Cable. The CSI Bus can easily handle 20 to 30 FPS (frames per second). The weight of 3 grammes makes no difference for the RC Car.
We are not required to use a wider lens because our test track is big enough. The images captured by this camera are sequential to the input for the model to train. 1080p is relatively large and would take an eternity to prepare; this is why we need to scale the image down to 160x120 pixels. Furthermore, we use a framerate of 20, which means we capture 20 images per second.

3.3 Servodriver

We use SunFounder PCA9685. The price for this microcontroller is approximately 13 euro on the most famous marketplaces. The servo driver is needed to produce the PWM (Pulse width modulation) signals for our Raspberry Pi. It acts as joint for the RC cars' servomotor and the Raspberry Pi.

3.3.a Technical Details: he SunFounder PCA9685 consists of many different Pins [7]. The GND and the VCC pin to power it. The VCC has used the voltage of 3-5V. Furthermore are three control pins on the board, namely the:

- SCL - I2C clock pin
- SDA - I2C data pin
- OE - Output enable

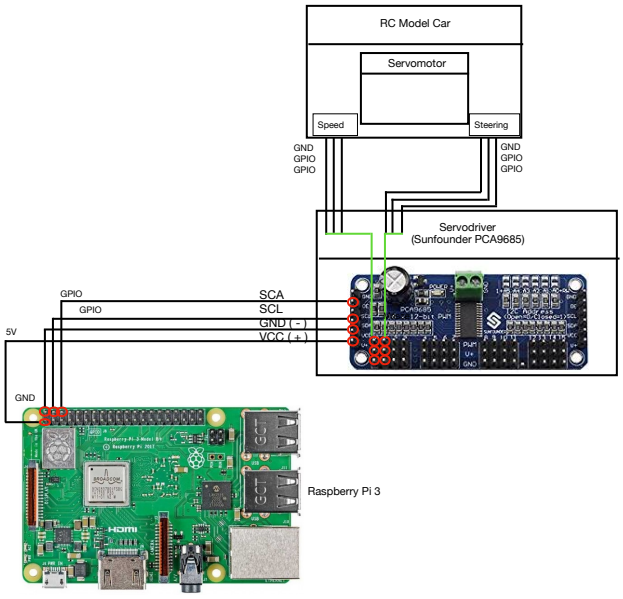

Fig. 5. Cable Management

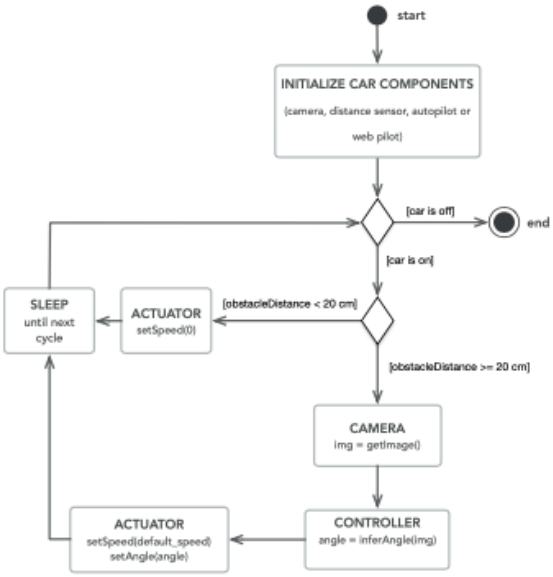

Fig. 6. UML Diagram from David Ungurean [8]

3.4 Cable Management and Battery

3.4.a Servomotor to Servodriver: A real model car always has a servomotor as a motor. This servomotor receives input from two pins, and each of the two is powered separately with V+ and the GND pin. So in total, the servomotor has six pins. Whereas the servo driver has 3x16 rows of pins available on it. The three following pins are responsible for the output:

- V+ - Power supply
- GND - Ground
- PWM

We connect the servomotor pins accordingly onto two of the rows of the servo driver. This six pins in total control the entire behaviour of the car. One of the three-pin-blocks is responsible for the steering, and the other one is responsible for the speed of the vehicle.

3.4.b Servodriver to Raspberry Pi: There are four pins that we need to connect from the Raspberry Pi onto the servo driver. Two out of those four pins are just for the power supply. We link the 5V pin from the Raspberry Pi to the VCC pins, which accordingly powers the SunFounder servo driver. To make a power supply work, we always need the ground pin (GND) as well. The other two pins communicate the data from Raspberry Pi to servo driver and vice versa. The communication pins work with the I2C bus system, which is used by the SDL (serial data) pin and the SCL (serial clock). The SDL 10Bit addressing bus sends the data, and the SCL sets up the clocking time, respectively. Both the Raspberry Pi and the servo driver have the SDL and the SCL connector pins, so we can easily plug them in.

3.4.c Battery: The Raspberry Pi 3 Model B+ has an optimum power supply of 5V and 2.1A. A power bank is responsible for the power of the vehicle. We use the power bank from the company *Anker* with *20000mAh*. It provides a 2.1A and 5V output which fully fulfils all the Raspberry Pi's power requirements.

4. Software Architecture

In this section, we give a quick overview of how the entire software is connected (Figure 7). The following UML diagram shows the whole flow of the data.

4.1 Python Files

The *Donkey Self-Driving Car* [6] Project is the basis for our group Project. So, we are constraint to use the same folder structure, which is:

- management
- parts
- templates
- test

Whereas management includes everything related to the webserver (as explained in the next subsection). The parts folder contains all the single devices related to getting the car to drive by itself. Included in the parts folder is, e.g. Pi Camera module, the Keras neural network and the image configurations to use them as input for the neural network.

4.2 Webserver

For the web server we use the Python *package* called *Tornado*. Tornado is a basic multi-purpose WebSocket which is open-source to use. The Tornado helds the bidirectional communication between the server and the browser. To receive the live transmission of the pictures of the camera module, we use the Tornado web server. As shown in figure 5, we access the webserver over the *localhost* and the *Port 37*. This Port is standardized by

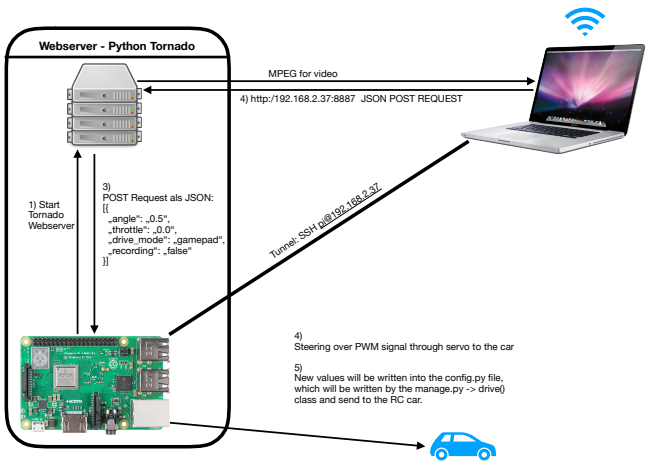

Fig. 7. Webserver Overview

IEEE [3] as the *TIME Protocol* and fits perfect to our requirements for tracking time intervals.

4.3 Unity

Based on the work of the open-source developer project from Twan Kramer, donkey car provides a simulator software. The Simulator allows us to drive with the car as with a real RCCCar. While driving, we can record labelled pictures. The C# Framework Unity is building used to implement the simulator software. More precisely, it uses the Unity 3D game engine to simulate car physics in a 3D world and generates image steering pairs to train our neural network [9]. In the Simulator, we can choose between different environments. So, we can be sure that the generated data has enough variety to get a useful model [10]. Going on a more detailed level would be too much to provide all those details in our report.

5. Machine Learning

As mentioned in our Project background and motivation section, machine learning is the heart of our project. So, before we are diving deeper into the details, we want to give a small overview of this topic. Machine Learning includes a large variety of techniques. Those techniques used for building programs which can use the experience to apply them on specific tasks. In other words, formally described as:

Definition 1: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" Mitchell (1997)

Experience can be regarded as training data [11], so that it is not necessary to programme all possible cases by hand.

[3] https://de.wikipedia.org/wiki/Liste_der_standardisierten_Ports

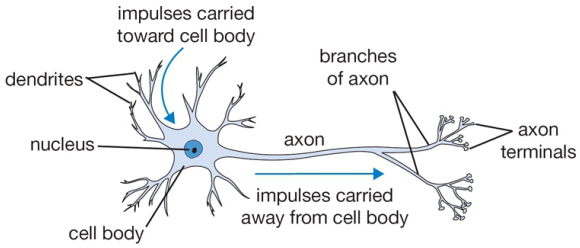

Fig. 8. A cartoon drawing of a biological neuron

To program, all possible scenarios like in earlier introduced chess computers, would not work [12]. That it is not possible to programm, because all possible scenarios are especially true for unstructured data. Unstructured data is data that isn't easy to store in a spreadsheet. For example, it could be an image, sound or text. Unstructured data is hard to interpret for computers [12][13]. In the last years, three major changes made it possible that computers are getting better in interpreting unstructured data. First, we had more data which could be used to work on different projects. Secondly, we know more efficient algorithms than before, because of the research which happened in the past 20 years. Last but not least, the increase of computional power has changed the game completely. It is essential to say that this happened mostly not because of better hardware itself, but is responsible for more a more advanced parallelization techniques running on the equipment. Besides the different data types like structured and unstructured data, plenty of tasks could be solved with machines which can learn from experience. To get a self-driving car, we must apply an image classification algorithm. The camera module fixed on the case of our car records images and sends them directly to the board computer. The computer has now to decide in which direction it should drive to stay in the track. The computers decision is based on the input images and classifies in which angle it should steer. A more detailed and technical explanation is given in the Deep Learning Subsection and the Autonomously Driving Section.

5.1 Deep Learning

Deep Learning means we have a deep neuronal network. Per Definition, we get a deep neural network if we had more than one hidden layer in a neural network [12]. Neural Networks are a concept in the machine learning discipline which try to build a program that learns like the human brain is learning. On a very fundamental level, we can say that neurons are functions [14]. If you compare the structure of both neurons, the neurons in our brain (figure 8 and the technical model (figure 9, which is used in neural networks, is similar. Besides the shape of the neurons, there are considerable differences in the way the human brain works and the way the deep neural network works. With an increase in the performance of those networks, the idea of rebuilds a human brain is almost gone. Instead, we are using modern mathematical theorems and functions

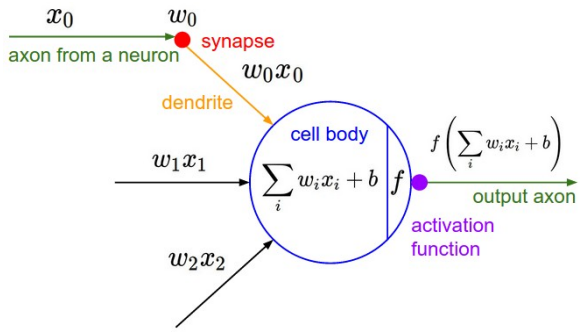

Fig. 9. A cartoon drawing of a biological neurons mathematical model

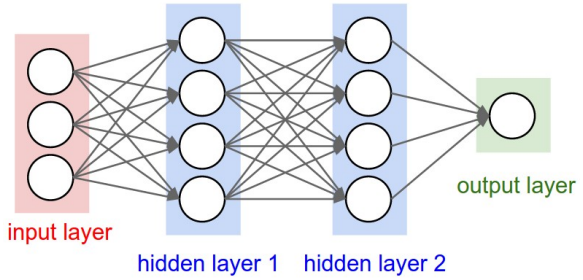

Fig. 10. A 3-layer neural network with three inputs, two hidden layers of 4 neurons each and one output layer. Notice that in both cases there are connections (synapses) between neurons across layers, but not within a layer.

that could solve machine learning tasks faster even we know that it has nothing to do with the human brain. Deep neuronal networks as you can see in figure 10 and the technical model (figure 9, can solve different types of tasks. Today we had not just the standard neural networks as in the early beginning of there first implementation. Instead, we have more advanced and complicated models. Most of them are designed to solve a specific task, and they are not able to adapt to another category of tasks. For example, Convolution Neuronal networks which are introduced in the Convolutional Neural Network Section are widely spread in the are of image classification. Recurrent neuronal networks are used for text and voice recognition. So, it would not lead to a better result, if we use a recurrent neural network which performs very good at text recognition for an image classification task, even when we had just a convolutional neuronal network that performs not so well.

5.2 Convolutional Neural Networks

To solve the image classification task, we are using a convolutional neuronal network (CNN). Convolutional Neural Networks are a particular case of feed-forward neural networks [12]. . The function of a feed-forward neural network can simply be described as $y = f(x, \emptyset)$. CNN's and feed-forward neural networks are estimating parameter values. So for $y = f(x, \emptyset)$ we estimate the parameter $\emptyset$ [12]. Through those estimated parameter values we are receiving a function back with the smallest possible difference between the predicted output values and the defined output values. A function that measures the difference between the expected - and the defined output is a so-called loss function [12]. To compare our model accuracy with other model, we are using the following evaluation criteria:

$$Acc(f, D) = \frac{1}{m} \sum_{m}^{i=1} \mathbb{I}\left[y^{(i)} = f(x^{()i)})\right] \quad (1)$$

CNN's and classical feed-forward neuronal networks differ in their basis of calculation. Feed-forward neural networks are using matrix multiplication, whereas CNN's are using convolutions. Convolutional layers, pooling layers, and fully-connected layers are specific layers for a CNN [13].

5.2.a Model Architecture: For our model, we choose pretty much the underlying implementation of the official website. It is an implementation of an architecture which is introduced by a group of researches which working for Nvidia. Their study showed that it is possible to train a model which can be used for autonomous driving. Especially the researchers showed that Donkeycar is using a small amount of labelled data — more precisely, only a few pictures from a short time of driving, where the images are marked with the direction in which the vehicle should drive[15]. As in our Literature Survey mentioned Christian Friedrich Coors implemented the model before. As he said, the model has 266.628 parameters which can be trained [16]. The architecture you can see in Figure 11. The Team around Nvidia concluded in their paper that there is more work to do for getting the driving process more stable [15]. Jelena Kocić and their team focused on this task and got a better result. They reached their goal by using other building blocks like a pooling layer. They mentioned that the training process and the driving process is more computational intense. Because of the few hardware resources of the raspberry pie, we decided to use the first model. [17]

5.2.b Convolutional Layers: The convolutional Layer performs the extracting of features from the input matrix into a feature map. For this procedure, we use matrix multiplication in the form of the dot product and a filter (feature detector, kernel) [12]. In picture 12 can be seen a calculation example of how convolution is applied to a matrix. We are iterating with the filter over the matrix, calculating the scalar product and writing the result into a feature map. An activation function like the rectified linear unit (ReLU) normalizes the feature map after we performed the convolution. Normalization guarantees a feature map which is not a linear transformation of its input value. If this is the case, we only have a linear problem which can be easily solved, but with a wrong results. In other words, the input values get just multiplied by coefficients. Note that there are plenty of reasons for activation functions in a neuronal network. We are giving a closer look at this topic at our section "Activation Functions". In Figure 13 you can see how our pictures

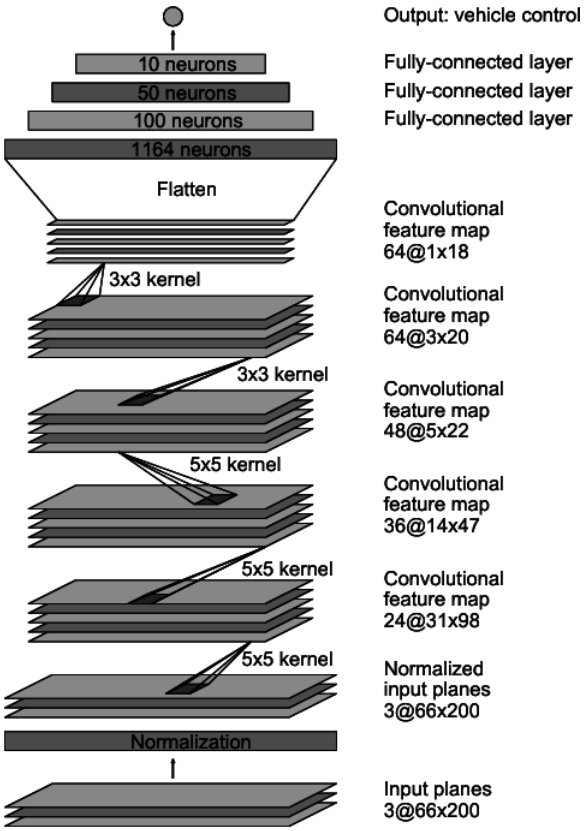

Fig. 11. Model Architecure from the Nvidia Research Team [15]

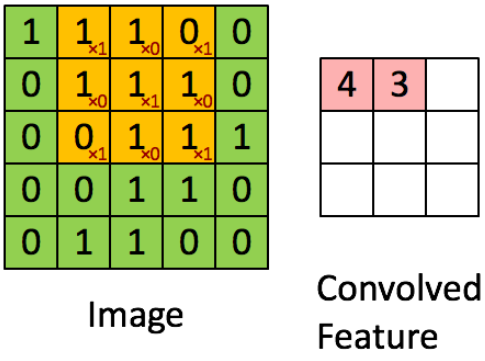

Fig. 12. Feature Map derived from http://deeplearning.stanford.edu

would change after applying the first convolutional layer. This represents the 64 first feature maps of a random picture from the dataset. We can observe that the filters already focus on certain points at the rooms environment. We further used this feature map as input for the second CNN layer. The output of the last feature map u could see in(figure 14.

5.2.c Pooling Layers: It is common to add a pooling layer in-between the convolutional layers periodically. The function exists to avoid overfitting and step by step reduce the size of the input image by reducing the number of parameters and computations of the network [18]. Anyways as mentioned earlier we decided against pooling layers as discussed in our Model Architecture Section.

Fig. 13. Feature Map from CNN Layer 1

Fig. 14. Feature Map from the last CNN Layer

5.2.d Fully-Connected Layers: The Dense layer is the last layer of our model. It is also called the *fully connected layer*. A Dense layer can be seen as a linear operation in which every input is connected to every output by weight. So, there are n inputs multiplied by n outputs weights - which can be a lot — generally followed by a non-linear activation function. This activation functions will then be computed by a matrix multiplication which is followed by a bias offset. We are using four of those Dense Layers in our model. The first two Dense Layers are used for Kernel Regularization; the others are used to get the two outputs. We will talk more about the output of our model in the Autonomously Driving Section.

5.2.e Kernel Regularization: There are many different regularizations to prevent the neural network from overfitting. Next, we are going to introduce the two most essential Kernel Regularizations. It can be integrated by penalizing the squared magnitude of all parameters directly in the objective. So, for every weight w in the network, the term $\frac{1}{2}\lambda w^2$ gets added to the objective, where λ is the regularization strength. It's usual to see $\frac{1}{2}$ in the front, because the gradient of this term with respect to w is simply λw instead of $2\lambda w$. The L2 penalizes the peaky weight vectors and prefers the diffuse weight vectors [19]. During the gradient descent weight update, the L2 regularization has the meaning, that every weight is decayed linearly (w += -*lambda* * w) towards zero. The Idea of the Drop-Out Kernel Regularisation is to drop out neurons frequently and randomly. That means we are losing Information, but our model is more adaptive to unseen data. The increase in the ability comes from the fact that the weights in dropped out neurons are temporally not updated [20]. In small experiments, the research found out that the younger drop-out approach is less computing-intensive and more robust than l2 [21]. So, we decided - as with the missing pooling layers before - that we are using drop out for Kernel Regularisation to prevent our model for overfitting.

5.2.f Batch Normalization: A technique developed by Ioffe and Szegedy [19] is called Batch Normalization properly initializes neural networks by explicitly forcing the activations throughout a network to take on a unit gaussian distribution at the beginning of the training. Normalization is a simple differentiable operation. It allows to use of higher learning rates at the beginning and is less vulnerable to a lousy initialization. In other words, neural networks that implement batch normalization layers are significantly more robust. Additionally, batch normalization can be interpreted as a preprocessing step on every layer of the network — batch normalization yields in general, a substantial improvement in the training process.

5.2.g Activation Functions: There a couple of widely used activation functions like tanh, sigmoid function, ReLU or the ELU. For our model we decided to use the *ReLU* activation function. The Rectified Linear Unit (ReLU) has become very popular in the last few years. It computes the function $f(x) = max(0, x)$. In other words, the activation is thresholded at zero. There are plenty of pro's and con's for the ReLU:

- (+) Compared to tanh/sigmoid neurons that need to compute expensive operations like the exponentials, the ReLU can be implemented by direct thresholding a matrix of activations at zero.
- (+) It was found to notably accelerate the convergence of stochastic gradient descent (SGD) compared to the tanh/sigmoid functions. That is, because of its linear, non-saturating form.
- (-) Sadly, ReLU units can be weak during training and possibly "die". For example, a large gradient streaming through a ReLU neuron could cause the weights to update in a dead-end, so that it will never activate again. If this happens, then the gradient streaming through the unit will forever be zero. The ReLU units can irreversibly die during training since they can get eliminated off the data manifold. With a good scheduling setting of the learning rate, this is less likely to happen.

5.2.h Loss function: Our compiled Keras model uses the *cross-entropy-loss* [22].

$$L_i = f_{y_i} + log \sum_j e^{f_j} \tag{2}$$

where we are using the notation f_j to mean the *j-th* element of the vector of class scores *f*. The full loss for the dataset is the mean of f_i over all training examples together with a regularization term *R(W)*. The cross-entropy loss, or also called log loss, measures the performance of our classification model with the output as probability values between zero and one. The cross-entropy loss increases as the predicted probability diverge from real value labels.

5.2.i Optimizer: There are many possible optimizers suitable for our task. The common ones are the RMSprob, Adam or the stochastic gradient descent (SGD) [23]. As before at our Section about kernel regularization we are going to introduce the essential Way to Optimize our Model. This method is more efficient than computing the gradient descent concerning the whole training set before each update is performed.

$$\frac{\partial p_i}{\partial a_j} = \begin{cases} p_i(1-p_i) & if \quad i = j \\ -p_j p_i & if \quad i \neq j \end{cases} \tag{3}$$

$$\frac{\partial L}{\partial o_i} = p_i - y\_oh_i \tag{4}$$

For the derivation of the cross-entropy loss, y_oh is the one-hot encoded representation of the class labels. But in our case, we decided to use Adam optimizer. An even more efficient optimizer. The research found out that it is compared to another optimizer a better performer in every aspect [17] [24]

5.2.j Softmax: The softmax normalizes the class probabilities to one, and it has a probabilistic interpretation.

$$f_j(z) = \frac{e^{z_i}}{\sum_k e^{z_k}} \tag{5}$$

The exponential values can very quickly explode to an infinitely large number, for example, e_{1000}. To fix this issue, it takes a one-dimensional vector of arbitrary length (in z) and puts it into a vector of values between zero and one that sums together to one. The cross-entropy loss that includes the softmax function, hence to minimize the cross-entropy-loss between the estimated class probabilities.

5.2.k Flatten: In between the convolutional layer (CNN) and the fully connected layer (Dense), there is a *Flatten layer*. The Flattening layer transforms a two-dimensional matrix of features into a one-dimensional vector that can be respectively streamed into the fully connected neural network classifier, which is our ten fully connected animal neurons..

6. Autonomously Driving

At this point, our car is equipped with a fully trained model and is ready to drive. Since our neuronal network is built with Keras, the Raspberry Pi uses the KerasPilot-Part to load the trained model and evaluate the captured pictures with that respectively. The images captured by the PiCamera are 160x120 pixels with 3 colour channels (red, green, blue). The CNN layers which the image need to pass through are shown in the figure.15.

The network produces two different outputs:

- angle_out: as PWM signal
- throttle_out: for the car speed

The car speed can be set manually to a constant value. This is especially useful when we let the car drive indoors because it cannot speed up so much as outdoors. So for our case, we only depend on the angle_out output. The

Convolutional Layer	Anzahl der Filter	Größe der Filter	Strides	Pooling	Aktivierungsfunktion
Convolution2D_1	24	5×5	2	Nein	Rectifier
Convolution2D_2	32	5×5	2	Nein	Rectifier
Convolution2D_3	64	5×5	2	Nein	Rectifier
Convolution2D_4	64	3×3	2	Nein	Rectifier
Convolution2D_5	64	3×3	1	Nein	Rectifier

Fig. 15. CNN Table

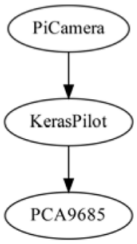

Fig. 16. Graph of Dataflow

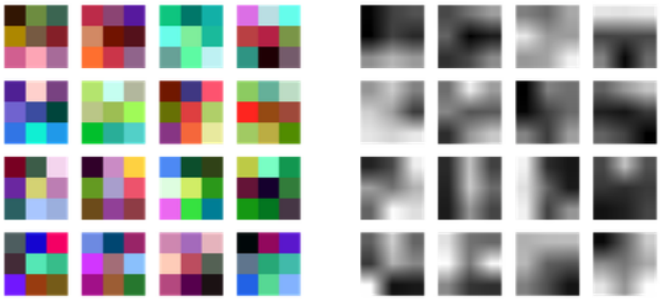

Fig. 17. Color not interpolated filter and grayscale filter with bilinear interpolation [8]

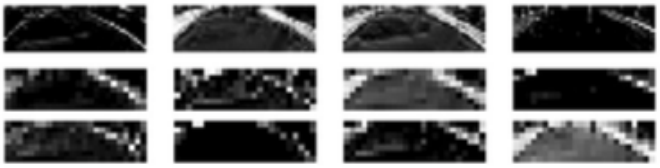

Fig. 18. Activation features after each Layer of the NN [8]

High-Level-Bibliography is Keras, which is accordingly based on Tensorflow. For our project, we use Tensorflow 1.0, not the latest release, 2.0.

As already mentioned in section PiCamera, we use 20 FPS, so we receive 20 angle_out outputs per second. These calculations processed on the Raspberry Pi. The values are directly transmitted to the PCA9685 servo driver and then to the servo motor of the car. Because of the direct transmitting, the car often shakes while driving. It receives just too many new values where it needs to steer its angle.

7. Evaluation

Even though CNN perform an excellent result in the computer vision field, it is still quite hard to decompose the decisions and the filters to understand the black box behind. Especially in image recognition, the filter decides randomly on which shapes and particles it is going to focus. By adding more and more layers to the neural network, we make it harder and harder to interpret the parameters. In the case of a fail, it is going to be very hard to evaluate the mistakes, e.g. overfitting, underfitting, too much noise in the data or the task might even be too complicated for the neural network. We can visualize the filter after, for example, the first filter of the convolutional neural network shown in figure 17. It is useful to illustrate this filter because the visualized noise could make overfitting visible in the image.

7.0.a Occlusion Maps: To evaluate the convolutional neural network with occlusion maps, we set the criterion function based on if the car drives within the lanes. Since we do not use a middle-lane, but only two paths for the border, the autonomous vehicle should identify the two border-lines correctly. The criterion function looks like the following:

$$O_{i,j} = \begin{cases} 0, & \text{if } abs(\hat{y}_{i,j} - y) > \epsilon \\ 1 & \text{otherwise} \end{cases}$$

Where y is the angle predicted from the neural network without any occlusion and $\hat{y}$ is the regressed angle when the centre of the occluding rectangle is placed at location (i, j) of the input image [8]. We do not need any more in-depth knowledge of the neural networks architecture to understand this straight-forward process because we only use the output of the network. One of the disadvantages of this technique is that we need to evaluate a lot of different pictures to detect the essential features of the image.

7.0.b Visualizing Activations: Another easy way to evaluate our model is to show each layer's activation after every convolution. We can compute with by multiplying the trained filters with the input data and apply the activation function onto it, for example, ReLU. We get a more and more decreased set of features after each layer of the neural network. We are able to extract those activations and display them (figure 18) as picture with the python package *matplotlib*.

7.0.c Occlusion Maps:

8. Conclusion

As we finished the last challenge of this hard project, a feeling of relief overcame every member. It was intellectually - and all of us agree on this - one of the most challenging tasks we encountered as students. We had to face more difficulties than we expected. In the end, we were thrilled of the successful outcome, though. We grew together as a team. The delegation of the tasked happened according to our strengths and weaknesses. Before the project, we knew each other, but we can say that after the project, we

know each other way better than before. The timespan given to us from the university of two semesters was sufficient for accomplishing the main objective. The supervised learning on our neuronal networks was successfully tested. Originally we planned to experiment regarding unsupervised learning (Reinforcement Learning), but the given timespan was not sufficient enough for this extended project, and the complexity was higher than expected. Therefore we conducted a "literature survey" where we evaluated other projects which used reinforcement learning technology and analysed whether the given technology could be implemented in our project or not. We learned the sophisticated interaction between different hardware components and the various software which run on them. Furthermore, we got a good overview of the state of the art technology of machine learning software in general. Two of the three members will try to get a job in this area. They applied for jobs or already have a situation where they have to analyse or manipulate big amounts of data.

Acknowledgment

The authors would like to thank Prof Dr Florian Gallwitz from the University of Applied Science - Georg Simon OHM in Nuremberg for an excellent supervising of our group.

References

[1] Tesla, "Motors, t. tesla autopilot," 2017. [Online]. Available: https://www.tesla.com/autopilot
[2] "W. waymo self-driving car project," *LLC*, 2017. [Online]. Available: https://waymo.com
[3] "Waymo. waymo's fully self-driving vehicles are here." *Waymo*, 2017. [Online]. Available: https://medium.com/waymo/with-waymo-in-the-drivers-seat-fully-self-driving-vehicles-can-transform-the-way-we-get-around-75e9622e829a
[4] T. D. Qi Zhang and C. Tian, "Self-driving scale car trained by deep reinforcement learning."
[5] A. G. Hado van Hasselt and D. Silver, "Deep reinforcement learning with double q-learning," 12 2015. [Online]. Available: https://arxiv.org/pdf/1509.06461.pdf
[6] autorope, "Donkey self-driving car," 2017. [Online]. Available: http://docs.donkeycar.com/
[7] [Online]. Available: https://cdn-learn.adafruit.com/downloads/pdf/16-channel-pwm-servo-driver.pdf
[8] D. Ungurean, "Deeprcar: An autonomous car model," Thesis, 5 2018.
[9] 03 2019. [Online]. Available: https://github.com/tawnkramer/sdsandbox/tree/donkey
[10] M. Gevrey, Y. Dimopoulos, and S. Lek, "Review and comparison of methods to study the contribution of variables in artificial neural networks models," *Ecological Modelling*, vol. 160, pp. 249–264, 02 2003.
[11] J. Wang, *Data mining: opportunities and challenges*. Idea Group Pub., 2003.
[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org
[13] Y. LeCun and L. Bottou, *Efficient BackProp*, 1998. [Online]. Available: https://doi.org/10.1007/3-540-49430-8_29
[14] V. D. Visin and Francesco, "A guide to convolution arithmetic for deep learning," 2018. [Online]. Available: https://arxiv.org/pdf/1603.07285.pdf
[15] M. Bojarski, D. D. Testa, and K. Zieba, "End to end learning for self-driving cars," *CoRR*, vol. abs/1604.07316, 2016. [Online]. Available: http://arxiv.org/abs/1604.07316
[16] C. F. Coors, "Approximate computing für neuronale netze in selbstfahrenden autos," Thesis, 2019.
[17] J. Kocić, N. Jovičić, and V. Drndarević, "An end-to-end deep neural network for autonomous driving designed for embedded automotive platforms," *Sensors (Basel)*, vol. 19, no. 9, 2019.
[18] H. W. Gu and Xiaodong, "Max-pooling dropout for regularization of convolutional neural networks." [Online]. Available: https://arxiv.org/pdf/1512.01400.pdf
[19] S. I. Szegedy and Christian, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015. [Online]. Available: https://arxiv.org/pdf/1502.03167.pdf
[20] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html
[21] E. Phaisangittisagul, "An analysis of the regularization between l2 and dropout in single hidden layer neural network," in *2016 7th International Conference on Intelligent Systems, Modelling and Simulation (ISMS)*, Jan 2016, pp. 174–179.
[22] Z. Z. Sabuncu and M. R., "Generalized cross entropy loss for training deep neural networks with noisy labels." [Online]. Available: https://papers.nips.cc/paper/8094-generalized-cross-entropy-loss-for-training-deep-neural-networks-with-pdf
[23] M. R. Gorti and S. Krishna, "Faster gradient descent via an adaptive learning rate," 2017. [Online]. Available: http://www.cs.toronto.edu/~mravox/p4.pdf
[24] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6980