Marius Landmann, Anna Dodik

Smart Systems

CONVOLUTIONAL NEURONAL NETWORKS

Practice Script

Table of Content

[List of figures II](#_Toc45051481)

[List of tables III](#_Toc45051482)

[List of abbreviations IV](#_Toc45051483)

[1. Lab roadmap 1](#_Toc45051484)

[2. Software set up 1](#_Toc45051485)

[3. Simple neural network creation 2](#_Toc45051486)

[3.1 GitHub set up 2](#_Toc45051487)

[3.2 Neural network environment 3](#_Toc45051488)

[3.2.1 Install the basic libraries 3](#_Toc45051489)

[3.2.2 Fork and clone 3](#_Toc45051490)

[3.2.3 Load the data set 3](#_Toc45051491)

[3.3 Train the network 4](#_Toc45051492)

[3.3.1 Pre-process data set 4](#_Toc45051493)

[3.3.2 Layer structure 4](#_Toc45051494)

[3.3.3 Training phase 4](#_Toc45051495)

[3.3.4 Evaluation 4](#_Toc45051496)

[3.4 Saving 4](#_Toc45051497)

[3.5 Save the data in your GitHub repository 5](#_Toc45051498)

[3.6 Unzip a data set 5](#_Toc45051499)

[4. Simple CNN creation 5](#_Toc45051500)

[5. Truck project 6](#_Toc45051501)

[5.1 Introduction to the truck project 7](#_Toc45051502)

[5.2 First truck connection 7](#_Toc45051503)

[6. Data set creation 8](#_Toc45051504)

[6.1 Image creation 8](#_Toc45051505)

[6.1.1 Changing the Arduino Nano program 8](#_Toc45051506)

[6.1.2 Image taking 9](#_Toc45051507)

[6.2 Image labelling 9](#_Toc45051508)

[6.2.1 Install labelimg 9](#_Toc45051509)

[6.2.2 Labelling with labelimg 10](#_Toc45051510)

[6.3 Creation of test and training data set 11](#_Toc45051511)

[6.4 Integration of the data set into a repository 11](#_Toc45051512)

[7. CNN for traffic light detection 11](#_Toc45051513)

[7.1 Explanation of the existing CNN structure 11](#_Toc45051514)

[7.2 Differences compared to previous CNNs 12](#_Toc45051515)

# List of figures

[Figure 1.1 Overview about the lab topics 4](#_Toc44847703)

[Figure 3.1 Process for the neural network creation 6](#_Toc44847704)

[Figure 4.1 Process for the CNN creation 9](#_Toc44847705)

[Figure 5.1 Overview of the truck project 10](#_Toc44847706)

[Figure 5.2 Truck at traffic light status detection 10](#_Toc44847707)

[Figure 5.3 Possible traffic light status 11](#_Toc44847708)

[Figure 6.1 Process steps for data set creation 12](#_Toc44847709)

[Figure 6.2 Arduino Nano interface 13](#_Toc44847710)

[Figure 6.3 Interface of labelimg 14](#_Toc44847711)

[Figure 6.4 Labelled picture in lableimg 15](#_Toc44847712)

# List of tables

[Table 2.1 Platforms for the project 5](#_Toc44847720)

# List of abbreviations

CNN Concolutional Neural Network

Google Colab Google Ccolaboratory

# Lab roadmap

The lab session requires you to work for roughly 60 hours. During this time you will expand your knowledge about CNNs, set up your own CNN, create your own data set, and learn how to improve and adjust an existing CNN through a project. Figure 1.1 gives an overview about the different topics that will be discussed. In the first part you will receive support in form of a documentation, code snippets, and mentoring. In the second part of the lab sessions you will have the chance to deepen your knowledge on your own and search for new solutions for a practical lab task.

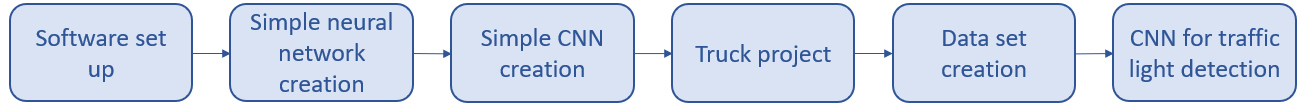


Figure 1.1 Overview about the lab topics

# Software set up

**Learning objectives of this chapter:**

* You know which programs and accounts are needed for the project
* You install and set up all important software and platforms

Before you can start to work you will need to install some software and register with some online platforms. Table 2.1 depicted below shows which software and accounts you need to set up. After all accounts are arranged you will be able to set up the first connection with the truck.

|  |  |  |  |
| --- | --- | --- | --- |
| Platform | Platform address | Account needed? | Download needed? |
| WinSCP | <https://winscp.net/eng/download.php> | No | Yes |
| VNC Viewer | <https://www.realvnc.com/de/connect/download/viewer/> | No | Yes |
| PuTTY | <https://www.putty.org/> | No | Yes |
| Arduino | [https://www.arduino.cc/en/main/software](https://www.arduino.cc/en/main/software)) | No | Yes |
| Anaconda Prompt | <https://www.anaconda.com/products/individual> | No | Yes |
| Git | <https://git-scm.com/downloads> | No | Yes |
| GitHub | <https://desktop.github.com/> | Yes | No |
| Google Colab | <https://colab.research.google.com/notebooks/intro.ipynb> | Yes | No |

Table 2.1 Platforms for the project

**Why is it necessary to install and set up some programs and accounts?**

🡪 WinSCP and the VNC viewer are both needed to transfer data between your computer and the local Raspberry Pi of the truck.

🡪 PuTTY is the console of WinSCP and is needed to send commands to the Raspberry Pi.

🡪 Arduino is needed to make adaptions to the Arduino Nano that controls the traffic lights.

🡪 Anaconda prompt will be needed for the data set creation.

🡪 Git, GitHub, and Google Colab are needed for the some exercises and the truck project.

# Simple neural network creation

**Learning objectives of this chapter:**

* Neural network skills:
  + You can set up your own neural network
  + You can explain your neural network to others
* Software skills:
  + You know how to save a trained network
  + You can set up a connection between Colab and GitHub
  + You can unzip files in the network
  + You are able to use repositories when you program

Now you can apply your knowledge. You will set up your own small neural network. The neural network should be able to distinguish between different types of clothes. Therefore, please use the fashion MNSIT data set. The fashion MNIST data set contains 70.000 different pictures of clothes. The training data set contains 60.000 pictures and the test data set 10.000.

**Task:** Please create a basic neural net that can distinguish between the 10 classes of different types of clothes. The accuracy should be at least 90%.

Figure 3.1 shows an overview of the process for the neural network creation. The following sections describe the single process steps in detail. Please follow the described steps.

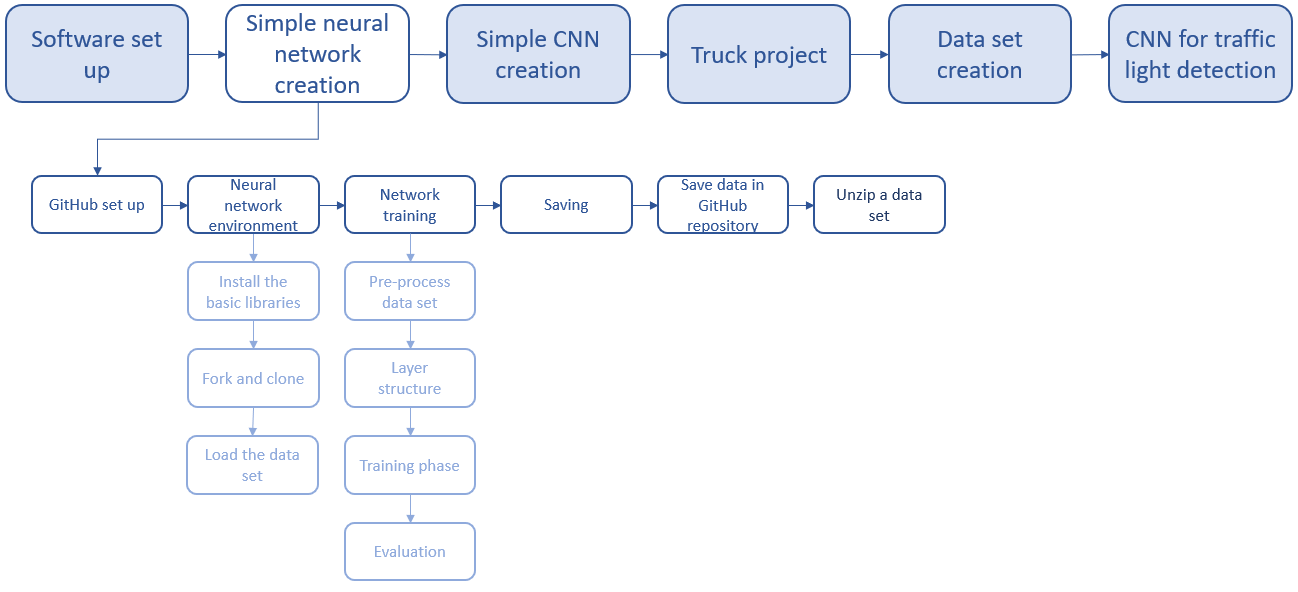


Figure 3.1 Process for the neural network creation

## 3.1 GitHub set up

Create a workspace folder on your computer with the folder structure that you would like to have in your repository. Push the designed folder structure to your GitHub repository. Then load the fashion MNIST data set into the repository.

## 3.2 Neural network environment

The neural network environment set up includes the installation of basic libraries, the cloning and forking of the repository, and the implementation of the data set. The single process steps are described in the next sections.

### 3.2.1 Install the basic libraries

Install the basic libraries as for example TensorFlow or NumPy.

Helpful additions:

* !pip install --upgrade deeplearning2020
* from deeplearning2020 import helpers
* import gzip
* !pip install h5py pyyaml
* from\_future\_importabsolute\_import, division, print\_function

### 3.2.2 Fork and clone

Fork and clone the repository in order to access it. It is possible to clone your own repository to be able to work with it and save your codes and data. You also can clone other repositories to work with their codes, data, and results.

Example:

# Forked repository

repo\_url = 'https://github.com/MariusLandmann/SmartSystems\_CNN\_TrafficLightDetection'

#Clone repository

import os

%cd /content

repo\_dir\_path = os.path.abspath(os.path.join('.', os.path.basename(repo\_url)))

!git clone {repo\_url}

%cd {repo\_dir\_path}

print('Pull it so that we have the latest code/data')

!git pull

### 3.2.3 Load the data set

Now load the data set to the environment of your neural network. Do not forget to unzip compressed data.

Example:

#mnist\_reader

def load\_mnist(path, kind='train'):

"""Load MNIST data from `path`"""

labels\_path = os.path.join(path,

'%s-labels-idx1-ubyte.gz'

% kind)

images\_path = os.path.join(path,

'%s-images-idx3-ubyte.gz'

% kind)

with gzip.open(labels\_path, 'rb') as lbpath:

labels = np.frombuffer(lbpath.read(), dtype=np.uint8,

offset=8)

with gzip.open(images\_path, 'rb') as imgpath:

images = np.frombuffer(imgpath.read(), dtype=np.uint8,

offset=16).reshape(len(labels), 784)

return images, labels

#Loading the data with Python

X\_train, y\_train = load\_mnist('dataset\_FASHION', kind='train')

X\_test, y\_test = load\_mnist('dataset\_FASHION', kind='t10k')

# X: images; y: labels

## 3.3 Train the network

The training of the CNN includes different parts. These parts are explained in the following.

### 3.3.1 Preprocess dataset

Pre-process the dataset by reshaping, shuffling, and normalizing the pixel values (depends on the data set). If necessary, convert the labels to a vector. The necessity depends on the model which will be used.

### 3.3.2 Layer structure

Create a basic layer structure for your neural network. You could use different optimizers and activation functions.

### 3.3.3 Training phase

Train your created neural network. Use 12 epochs for the training.

### 3.3.4 Evaluation

Evaluate the neural network with the test dataset. Adapt your neural network if necessary.

## 3.4 Saving

Save the trained neural network in your cloned repository and recreate/restore it afterwards.

Example saving:

model.save('./trained\_CNN/fashion\_MNIST/my\_model\_adam.h5')

Example recreate:

#Recreate whole model

new\_model=keras.models.load\_model('./trained\_CNN/fashion\_MNIST/my\_model\_adam.h5')

new\_model.summary()

## 3.5 Save the data in your GitHub repository

Download the previously saved model into your locally created repository. Afterwards push it. Now you can recreate your model in other notebooks.

Example for download:

# DOWNLOAD created files (in this case the model created previously)

from google.colab import files

files.download('./trained\_CNN/fashion\_MNIST/my\_model\_adam.h5')

# Push Anzeigen -> Inhalt -> files can display everything that download all generated

## 3.6 Unzip a data set

Download the data set from the repository (https://github.com/MariusLandmann/SmartSystems\_CNN\_TrafficLightDetection/tree/master/dataset\_FASHION/in\_csv) or search for it online and save it in your local workspace. Then load it to your repository. Unzip the compressed data in Colab.

Example:

# Unzip CSV data and assign a variable to it

import zipfile

with zipfile.ZipFile('/content/SmartSystems\_CNN\_TrafficLightDetection/dataset\_FASHION/in\_csv/train\_compressed\_fashion-mnist\_train.csv.zip', 'r') as zip\_ref:

zip\_ref.extractall('/content/SmartSystems\_CNN\_TrafficLightDetection/dataset\_FASHION/in\_csv')

train\_csv='/content/SmartSystems\_CNN\_TrafficLightDetection/dataset\_FASHION/in\_csv/fashion-mnist\_train.csv'

#!cat {train\_csv}

# Simple CNN creation

**Learning objectives of this chapter:**

* You can set up your own CNN
* You can explain your CNN to others
* You can integrate a data set in Colab

You now have some experience in setting up a new neural network. Next, you will set up your own CNN. The CNN should be able to distinguish between ten different dog breeds. Therefore, please use the ImageWoof data set.

**Task:** Please create a CNN net that can distinguish between the ten classes. The accuracy should be at least 50%.

Hint: Activate the GPU in Google Cloab (Go to Runtime > Change runtime and select a GPU hardware accelerator)

As first you should download the data set from deeplearning2020 and integrate it in Colab.

Example:

!pip install --upgrade deeplearning2020

from deeplearning2020.datasets import ImageWoof

train\_data, test\_data, classes= ImageWoof.load\_data()

You can use your old neural network as a framework. The data set and the layer structure need to be changed. The other parts of the network need to be adapted. Follow the explanations from chapters 3.2 till 3.4. and 3.6 again. Figure 4.1 shows an overview of the process for the CNN creation.

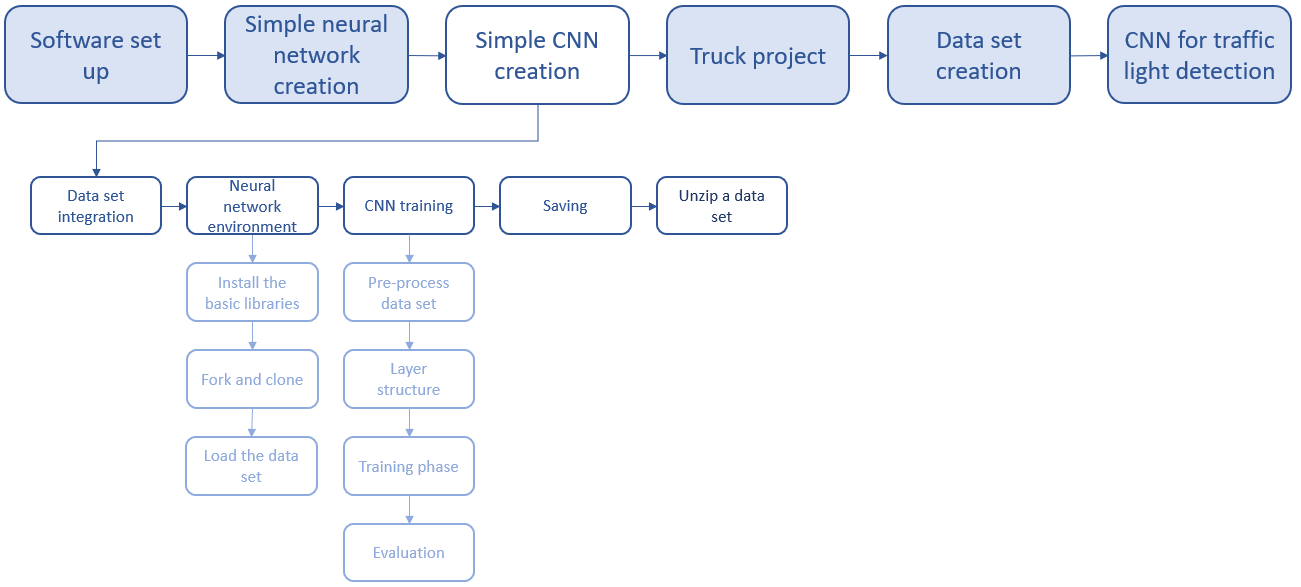


Figure 4.1 Process for the CNN creation

After the CNN was created answer the following questions:

1. What are the main differences in the layer structure between the exercises of chapters 3 and 4?
2. Why does the CNN take more time to train then the previous neuronal network?
3. What had the most influence on increasing the accuracy of the CNN?
4. Were there any surprising findings for you? If yes, which ones?
5. Which part was most problematic for you?

# Truck project

**Learning objectives of this chapter:**

* You can explain the “Smart Truck” project that will be implemented during the laboratory sessions
* You know the hardware and software design of the truck
* You can explain how to set up a connection between your computer and the truck (Raspberry Pi)

Figure 5.1 gives a short overview of topics discussed in this chapter.

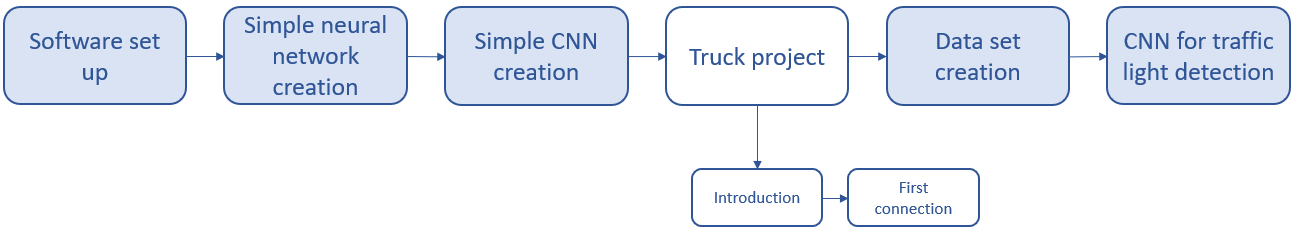


Figure 5.1 Overview of the truck project

## 5.1 Introduction to the truck project

The "Smart Truck" project continues in each semester and expands and improves through this. The overall goal is to have a truck that can drive fully autonomously. Many steps and improvements are necessary to reach this overall goal. The goal of your practice sessions is to implement a working CNN into the truck. The truck should detect traffic lights through the CNN and identify their status as illustrated in figure 5.2.

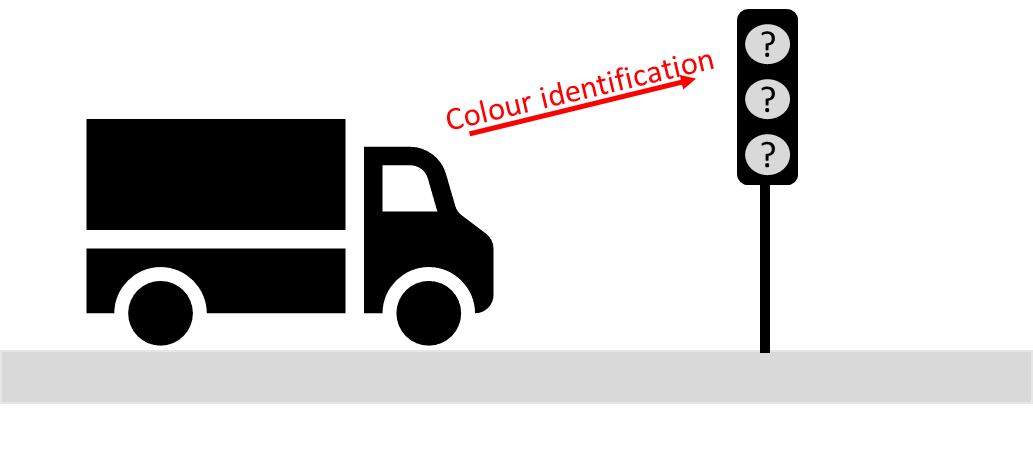


Figure 5.2 Truck at traffic light status detection

After the status was identified the truck should react according to the status of the traffic light, e.g. stop at a red traffic light and start or continue at a green one. Figure 5.3 shows the different status options of the traffic light.

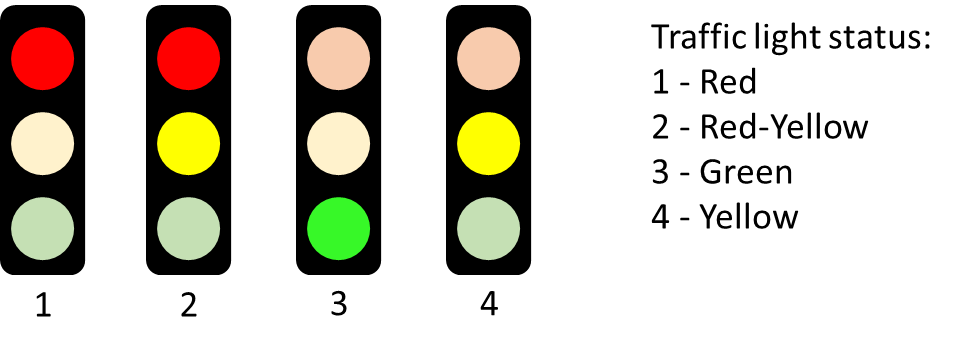


Figure 5.3 Possible traffic light status

To get a more detailed instruction to the truck itself please read chapter 1 till chapter 2.2 in the “Smart Truck Manual”.

## 5.2 First truck connection

You now have an overview about the truck and the project itself and can set up the first connection between your computer and the truck. How to do this is described in detail in the “Smart Truck Manual” in chapter 3.1.

**Task:**Please read and follow the description in chapter 3.1 and set up the connection between the Raspberry Pi and your computer

# Data set creation

**Learning objectives of this chapter:**

* You can explain and set up the single steps that are necessary to create a data set
* You can explain and justify the requirements of a proper data set

In the example before you used a prepared data set. Unfortunately, there is not always a data set existing that fits the needs of your CNN. Maybe the data sets have the wrong format or there is no data set that fits to your application at all. Therefore, you will learn in this section how to create your own data set. Figure 6.1 illustrates the single steps that are required. The following sections will describe how to create an own dataset.

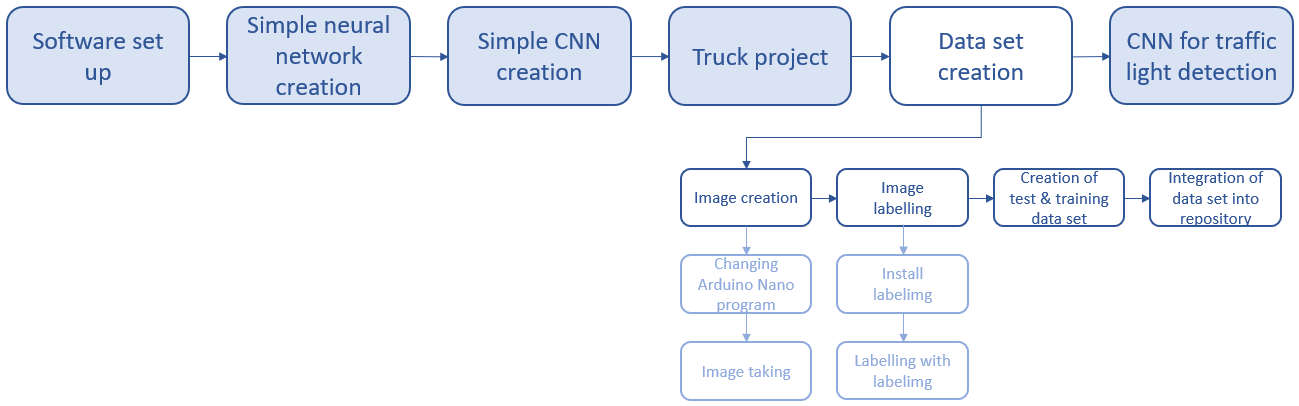


Figure 6.1 Process steps for data set creation

**Task:** Please create a data set for the traffic light detection. The data set should contain at least 300 images. The traffic light and the including webcam are shown in your lab materials.

## 6.1 Image creation

Image creation means to take images of the things that should be identified, if possible, with the device that will be used in the project. This means for you to take pictures with the webcam that is mounted on the truck. The traffic light can be adjusted for this purpose. Until now it contains a normal program that runs the traffic light with the usual phases. It makes sense to change this program for image creation purposes. The code of the Arduino Nano needs to be changed in the Arduino program interface.

### 6.1.1 Changing the Arduino Nano program

Open Arduino IDE and make some settings for smooth working. The settings need to be changed just one time. Open the tab “Werkzeuge”. Then choose the category “Board” and set *Arduino Nano*. Next, choose as category “Prozessor” and set *ATmega328P (Old Bootoloader)*. Then, choose the category “Port” and set *COM4*. Last, choose “Programmer” and set *USBap*.

Now open the code “traffic\_light\_image” which you want to load to the Arduino Nano. The code that you need to transfer will be provided.



Now push the arrow in the circle, shown in figure 6.2, to transfer the code to the Arduino Nano.



Figure 6.2 Arduino Nano interface

If you want to load the original program of the traffic light to the Arduino Nano, follow the same process.

Original program of traffic light (traffic\_light.ino):



### 6.1.2 Image taking

Now you can start to take the pictures. Therefore, open the program “take pictures” that is stored with the VNC viewer. A picture is always taken if you press space. It is recommended to take pictures of all traffic light phases for each background. Thereby you ensure that you have a balanced data set.

Please make sure that you have a high variety of pictures. This means to use different backgrounds and image sections after each traffic light phase. Sometimes the traffic light could be in the middle, sometimes left, and sometimes right. Furthermore, you should take the light conditions into account. On the one hand, you should check if the light comes from the front, side, or from the back. On the other hand, you should try to have different light conditions. For example, daylight, twilight, or night light.

## 6.2 Image labelling

The next step is image labelling. During the labelling process the things that should be identified will be marked and lettered. In your project you want to identify traffic lights and their status. Therefore, you need to label them. The sections below describe how to proceed in detail.

### 6.2.1 Install labelimg

As first, you need to set up a program with which you can label the images. There exist different programs to do this. In this case you will use the program labelimg. It is easy to operate and is therefore often used. Please follow the instructions to set up the program:

1. Install Anaconda**.** Please use the following link: <https://www.anaconda.com/products/individual>

(skip the step if you already installed the system)

1. Create a new environment in Anaconda:

* Open the Anaconda Navigator
* Select “Environments”
* Select “Create”
* Name the new environment (i.e. “test”)
* At packages select “python” 🡪 3.8
* push “create”

1. Open Anaconda Prompt
2. Change the file directory:

* To do this, write at the (base) after the “>” symbol: activate test
* (test) is now activated

1. Install labelimg by entering the following function after the “>” symbol: pip install labelimg
2. To use the program, write in the (test) environment after the “>” symbol: labelimg

After the program was successfully installed it will open automatically if you execute step 6.

### 6.2.2 Labelling with labelimg

As soon as the labelimg program is set up the image labelling process can start. The window of labelimg opens automatically and is shown in figure 6.3. The picture which should get labelled can be opened with the symbol shown at (1). If the picture is labelled it can be saved with button (2). The format with which the picture can be labelled is shown at number (3). It is possible to decide between the VOC and the YOLO format. You should save your pictures as VOC format because this format gets stored as an xml file. In this case the xml file is better than a txt file of the YOLO format, because TensorFlow can process it faster. The labelling process itself starts by pushing button (4) named “Create\nRectBox”.

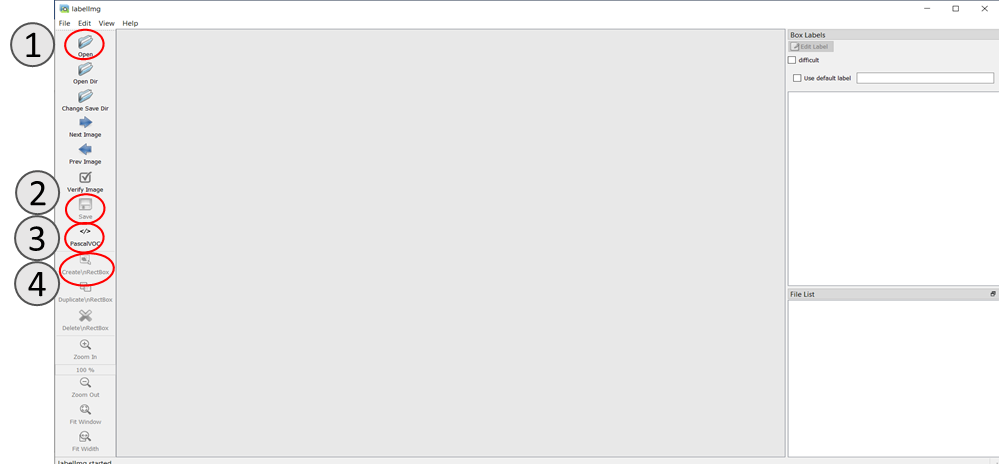


Figure 6.3 Interface of labelimg

If “Create\nRectBox” is activated, it is possible to make a rectangular around the part of the picture that should be labelled. After setting the size of the label the label needs to be set. Figure 6.4 shows how this could look like. In this case you can choose the traffic light (red traffic light, yellow traffic light, green traffic light, red- yellow traffic light) as a label.

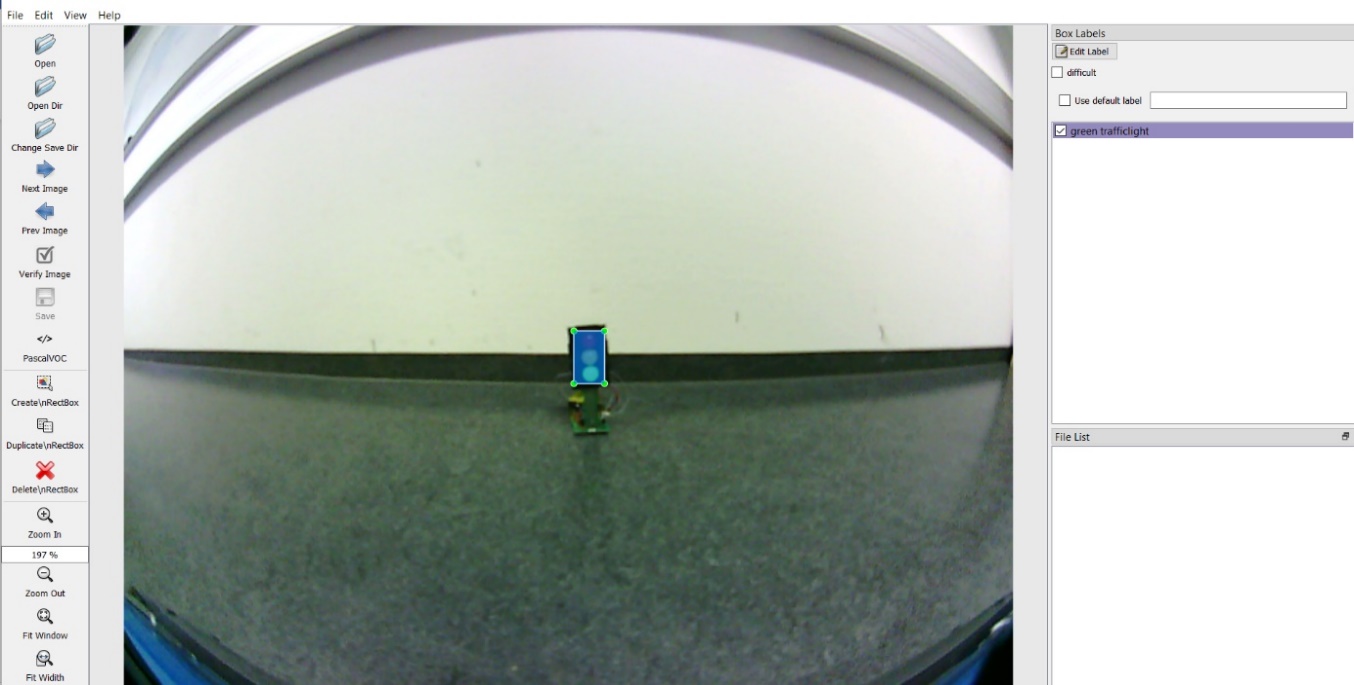
As a last step the picture needs to be saved as described in the previous section.

Figure 6.4 Labelled picture in labelimg

## 6.3 Creation of test and training data set

The creation of a training and test data set means that all data will be split into two groups. The one data group will be used just for training the CNN and the other group will be used to validate the accuracy of the CNN. Therefore, you need to create two different folders. One is the folder for training data and contains approximately 70% of the whole data set. The remaining 30% should be saved as the test data set. Consider that you always need to store the image and the related labelling file.

## 6.4 Integration of the data set into a repository

The last step of the data set creation is the integration of the test and training data set into a repository. To do so, load both folders into the project structure of your repository.

# CNN for traffic light detection

**Learning objectives of this chapter:**

* You can explain the current structure of the CNN for traffic light detection
* You understand the purpose and function of every code section and you are able to explain it to others

## 7.1 Explanation of the existing CNN structure

At the beginning I proceeded similarly as you already have learned in chapter three:

1. Setting up the GitHub repository (see chapter 3.1)
2. CNN environment:
   1. Installation of necessary and helpful libraries (see chapter 3.2.1)
   2. Forking and cloning of our repository (see chapter 3.2.2)
3. Loading and preparing our dataset
4. In the following you can see various approaches and layer structures with which I tried to train the CNN with the dataset in the TFRecord format
5. In the end the model can be saved and recreated (see chapter 3.4)

## 7.2 Differences compared to previous CNNs

1. Besides of our created dataset there are also two python codes saved in the GitHub repository which we are using for our dataset:
   1. xml\_to\_csv.py: Converts the train and test folder annotation xml files to a single csv file
   2. generate\_tfrecord.py: Takes the created csv files and generates the TFRecord files for the train and test dataset
2. By cloning the GitHub of TensorFlow Model Garden we imported additional libraries which we are using for generating the TFRecord files
3. At first, I defined the path to our dataset and then prepared the dataset by using the mentioned codes which are saved in our GitHub repository. As a result, the dataset is saved in the TFRecord format. This format is a record oriented binary format, which is often used for machine learning data. By using binary code, the system processes the input data faster than before.
4. The part in which we train the CNN is not complete yet and needs some changes or even other approaches for setting up a functional CNN.

**Task:** Create the TFRecord files for your dataset by using the existing code. Here you can see if your data set creation of chapter six was a success.

**Advanced task:** Try to change one of the existing models or create a new CNN model in which you can train your TFRecord data set. You can adapt as much as you want from the existing Traffic Light Detection code. You can also use the sources I have used to create all the neural networks of this course. You find it in the “Sources” folder in the GitHub repository.