

# A First Encounter with Neural Network Learning

*SMUFN117 - UE Neural network and learning*

Julien Schulz

Department of Informatics  
Technische Universität München  
Email: julien.schulz@tum.de

Michael Cegielka

Department of Informatics  
Universität Bremen  
Email: mi\_ce@uni-bremen.de

Leonhard Zirus

Department of Informatics  
Technische Universität München  
Email: leonhard.zirus@tum.de

**Abstract**—This paper documents the entire process of creating a machine learning model, that is capable to determine, whether a person is wearing a mask or not, on basis of an image of that person. The project includes the creation of an annotation software used to edit and pre-process the used dataset as well as the implementation, training and testing of the model itself.

**Keywords**—Machine Learning, Tensorflow, Keras, Image Annotation, Deep Learning, Neural Network

## I. Introduction

Software development has grown to a point, where we have a giant base of great algorithms, that enable us to solve many problems quickly and efficiently. In todays software development it often comes down to having an idea and implementing it on the basis of these existing algorithms. Still there is a limit to what they can do and it is often reached at tasks that seem pretty simple to us humans. Algorithms are great in using big amounts of data, accumulating it to models and trends, showing it in smart graphs, generating visual representations and simplifying the analysis and interpretation. Still the final interpretation is a skill where humans are unmatched in, until now.

These days, more and more companies are starting to implement machine learning (ML) algorithms as support to their existing algorithms. These ML algorithms are the closest a program can come to the human ability to use experience and learn from it, in order to interpret new experiences on the basis of the old ones. While they have been around for a while now, it has not yet got to a reliability to be established as standard. As the topic becomes more and more important in todays world, getting to know it, can prove very valuable in the future.

This paper follows the entire process of creating a so called "artificial intelligence" that is able to take an image, grabbed from a file or a video and determine, if showing a person, which kind of mask this person

is wearing or whether he/she is wearing any at all. First the project and its goals will be layed out in detail with a short introduction of each team member and their role in this project. This is followed by a description of the used dataset and how it was created. Then there will be an outline of the process including used methodologies and encountered difficulties and finally a summarization of the obtained results and their interpretation.

## II. Project - Description

This project was conducted in context of a university course at Université Côte d'Azur in Nice, France. The main goal of the project was a first experimentation with the concept of machine learning with neural networks. It's purpose was to gain a deeper understanding of what influences the decisions of such a model and using the gained knowledge to create a simple, but fully functioning face-mask-detector.

The project was split into two parts, the data collection and preparation and the machine learning (ML) model creation and training. In the following both parts will be outlined with a introduction of the team members and their roles in both parts.

### A. Part I - Data Collection and Preparation

The data collection is already a big part of the project, as there are a lot of things that can go wrong here. The data is the main thing that influences the final model in the decision it makes. A ML model is in this context not different than humans, as it can only act on its own experience. Therefore making sure the data is chosen in conjunction with the final goals is essential. As the scale of this project is fairly small and its purpose is a first experimentation with the concept, it was sufficient to expect the the model to recognize the three members of this project.

The collected data will most likely not be in a format, that is easily readable for our model. The next step

therefore needs to manually interpret the data, label it for the model and standardize it. The level of the data preprocessing again is determined by the goal one wants to achieve. In the scale of this project, our preprocessing will go as far as already identifying the face on the image, cutting it to a good fitting square and giving it a label according to its context.

To achieve all that in a streamlined, standardized and user friendly way, the development of an image annotation software was necessary. In addition to the manual annotation there are some automation tricks that can be used to help with easy classifiable data.

### *B. Part II - Machine Learning Model*

This of course is the center of the project. Setting up and compiling a ML model, that is capable of taking an image of a face and determining whether the person shown is wearing a mask and what kind if yes.

As mentioned before, the goal was limited to being able to detect masks on the faces of the team members of this project, as we would have needed a much larger and more diverse dataset to achieve reliable results with all other faces. Still the hope is, that the model will be able to identify more faces, that have similarities to the team members. This will most likely include white, western men, while there will be difficulties with different skin colors and facial structures.

### *C. Team Members and Roles*

The team consists of three members, Michael Cegiela, Julien Schulz and Leonhard Zirus. The roles in this project were distributed equally in a way where everyone was still involved in every part of it.

In the first part of the project, Leonhard took charge of the annotation software, being responsible for the organization and collaboration. He created the framework and interfaces. Julien as our ML expert was responsible for making sure that the software was usable in the second part of the project and created a streamlined process for the creation of the dataset, augmentation and later labeling. Micheal was mainly in charge of the UI (User Interface) and UE (User Experience) of the annotation software.

TODO: Part II TODO: Who coded TODO: Who wrote the paper

## **III. Dataset**

The used dataset was created by taking pictures of our team members wearing different kinds of masks. For the ML model to use the images they need to be preprocessed and labeled to enable the learning process.

### *A. Image Creation*

The images taken will influence greatly what kind of images the model will later be able to recognize. This means, that lighting, orientation of the face, background, clothing or other kinds of patterns are very important to be aware of when creating taking the pictures.

Imagine a person choses to never wear a bonnet when walking inside, but because of corona, that is exactly when he/she always wears a mask. As it is winter, the opposite is true for being outside, he/she wears a bonnet, as it is cold, but no mask, because it is outside and not required. Now using pictures from these situations, the model might turn to identifying whether the person is wearing a bonnet instead, as the dataset reflects such an implication. This is just one of many examples, of unwilling implications that might turn up in a dataset.

In order to make sure to not have any such implications the images were tried to create, using equal distributions of different backgrounds, with no lighting differences correlating to the wearing or not wearing of a mask and different clothing styles not related to the mask. The later one was limited to wearing a bonnet or not.

The limitation of time and equipment will surely create a few problems with the dataset. As an example, the creation started in the evening, which in turn changed the lighting between changing to a different mask. As this problem became aware, the fotoshoot was moved inside to at least have similar lighting throughout all the pictures.

The improvements to the creation are countless, but would have all not been justified by the dimension of this project. To create a proper one, the diversity of the people, lighting, backgrounds, clothing, camera-lenses, etc. would all have to be greatly increased. Also the number of images created is still very small for the model to accurately recognize anyone, anywhere we put them in front of the camera. In the end the created dataset should have sufficient diversity to serve the purpose of this project.

### *B. Scaling, Labeling and Formatting*

With the base for the dataset created, it is time to pre-process the images into a format that can be used for the model. The final images should have a square format with 240 x 240 pixels saved in a folder specifying it's label. There are four labels: "no\_mask", "ffp2", "op\_mask" and "other\_mask". In this case "other\_mask" will be different kinds of cloth-masks.

To limit the manual work as much as possible, the usage of the OpenCV Haar-Cascade-Classifer, which

is a pre-trained algorithm to detect faces on an image. A short script enabled the automated loading of all images in a folder, computation of present faces, cropping of the face in the image and saving it in a new location. As the cascade classifier gives a whole lot of false positives, it is important to go through the exported files and delete anything that can not be used. To help figure out which image was already correctly processed, there is a second small script comparing the processed images to the raw images and copying all unprocessed ones to a new folder, ready to be manually annotated.

The labeling process of course can't be automated, as this would already be the finished model. The processed images therefore have to be manually copied in their respective folders.

The finally missing images, now have to be manually annotated, where finally the image annotator comes to play. Using this software it is just a matter of going through all remaining images, marking each face with a bounding box and labeling it correctly. Afterwards the Annotator offers a crop and save function, that takes care of cropping, scaling and correctly saving the images in the right labeled folders.

#### IV. Implementation and Methodologies

This project had two major implementations, the annotator and the ML model, which will be detailed in this chapter. The following will explain used methodologies and choices made in the implementation process as well as overcome difficulties.

##### A. Image-Annotator

The purpose of the image annotation software is to give an easy possibility of labeling, rescaling, cropping and saving images in order to be used as dataset by the ML model.

The software was completely written in python, an interpreted script language. Still python has object oriented features, that were partly used in this project. The program is structured in a way, to have two different classes, "Window" and "popupWindow" to take care of the UI part of the project. The rest of the functionality is structured into script-like functions. This structuring makes it easier to split the program into smaller tasks that can be distributed and worked on individually. This is a methodology oriented at the MVC (Model View Controller) software development pattern. The UI is here clearly separated from the controlling instance, that handles the backend of user interaction as well as the model, which is responsible for data storage and maintenance. As this is a simple

single instance application, of course a detailed implementation was not necessary and the model and controller are somewhat mixed and both just represented as the separate functions. Still it gives the possibility of separating the implementation of the UI, the backend and the storage system.

The UI is handled by the tkinter python library, which gives a lot of basic functionality to easily implement the frontend of the application. The UI of this project was mainly focused on the menu bar on top in addition to a right-click menu. This choice was made in order to keep the overall frame clean and focused on the most important thing, the image.

The backend storage uses a python dictionary to at all time store all edits made by the software. If need be, the entirety of the data can be saved to file using the JSON format and loaded back from it as well. To keep the storage as efficient as possible and maintain the possibility to easily change categories, they are saved separately with a second array only linking the index of the annotation to the index of a category. Listing 1 shows a reduced example of stored annotations. There is a feature enabling the user to save every processed resized image to a specified location. It was disabled as it turned out not to be needed in this project.

```

1 {
2   "categories": ["None", "mask", "no_mask", "op_mask", "ffp2", "other_mask"],
3   "images": {
4     "IMG_1342.JPG": {
5       "rectangles": [[421.0, 144.0, 853.0, 727.0]],
6       "rect_to_category": [3],
7       "dst": 0,
8       "src": "~/NNL-2021-F/img/to_annotate/IMG_1342.JPG"
9     },
10    "IMG_1418.JPG": {
11      "rectangles": [[358.0, 92.0, 939.0, 597.0]],
12      "rect_to_category": [5],
13      "dst": 0,
14      "src": "~/NNL-2021-F/img/to_annotate/IMG_1418.JPG"
15    },
16    "IMG_1597.JPG": {
17      "rectangles": [[399.0, 94.0, 1038.0, 653.0]],
18      "rect_to_category": [4],
19      "dst": 0,
20      "src": "~/NNL-2021-F/img/to_annotate/IMG_1597.JPG"
21    },
22    "IMG_1853.JPG": {
23      "rectangles": [[330.0, 137.0, 926.0, 699.0]],
24      "rect_to_category": [2],
25      "dst": 0,
26      "src": "~/NNL-2021-F/img/to_annotate/IMG_1853.JPG"
27    }
28   },
29   "destinations": ["None"]
30 }
```

Listing 1. reduced example of saved annotations

The helping features are implemented as functions, that can be called through the given menus:

##### open image

Opens a file selector to let you chose an image to annotate

##### load next image

Finds the next image after natural order (the windows file-name order) in the same folder as the currently loaded image



Fig. 1. image annotator in action

### save/import/view annotations

Lets the user chose a file location to either import or export all annotations as well as gives the possibility of viewing all currently loaded annotations in a list-view

### add/replace/show category

Lets the user add a new category, replace an existing one or show all current categories in a list

### import/export categories

Similar to annotations this gives the possibility to only share the categories by enabling exporting to json, csv or xlsx and importing from json and csv

### replace/change destination

A feature that was not used during this project enabling the user to save each image resized and as png but not cropped to a specified location, that can be different for each image

Finally the main features of the annotation lay all in event listeners for the mouse pointer. The user is able to draw rectangles in case an image was loaded. A rectangle is only allowed if it has an area greater that 40 pixels, sides longer than 5 pixels and is not overlapping with another rectangle by more than 20%.

Is a valid rectangle drawn, the user gets pointed details and is able to chose a label. Figure 1 shows the annotator in action. By double-clicking or right-clicking the rectangle, this popup can be shown again at any time and the label changed. Right-clicking also gives the choice of deleting the rectangle.

This gives the annotator the full needed functionality and enables a quick and easy labeling of all the images. Once the labeling is done, the menu item "**crop and save**" reloads each image at a time, crops all made rectangles to 240x240 square format and saves them in folders according to their labeling in an by the user specified location.

### B. Machine Learning Model

used libraries: - tensorflow - keras and keras-tuner - opencv

structured after tensorflow tutorial - load data - pre-process - ...

why this layer-structure - rescaling -> between 0 and 1 - augmentation -> small dataset artificially enlarged - convocational layer -> pattern detection - max-pooling - flattening -> for dense-layer - dense-layer - dropout -> delete low confidences

```
basic_model = Sequential([
    Rescaling(1./255),
    augmentation,
```

1  
2  
3

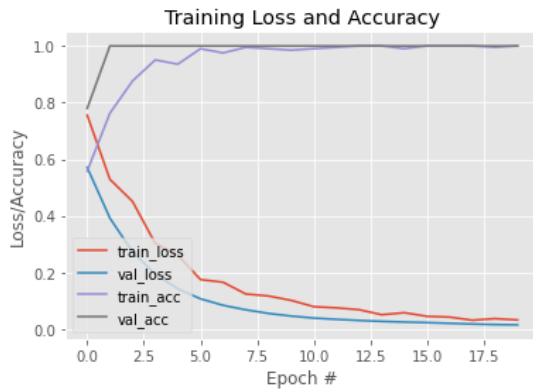


Fig. 2. MobilNetV2 classifier - training results

```

Conv2D(32, (3,3), activation='relu'),
AveragePooling2D(pool_size=(7,7)),
Flatten(name='flatten'),
Dense(128, activation='relu'),
Dropout(0.5),#drops small confidences
Dense(num_classes, activation='softmax')
1)

```

Listing 2. reduced example of saved annotations

## V. Testing and Results

Out of interest, the here used dataset using only images labeled as "no\_mask" and "ffp2", was used in another model found online. The code was written by Adrian Rosebrock and uses the already pre-trained MobilNetV2 classifier [1]. This would be the goto implementation in case one is focused on making the classifier work and not creating one from scratch. The advantage is, that this classifier has already been trained on a lot of different images, which allows it to much easier adjust weights for new purposes as long as it involves image classification. Repurposing this for a simple mask / no mask detection is very easy and very accurate. Listing 2 and Figure 2 show the performance of this model. It is very clear, that the performance is exceptional considering the training and validation data. It would remain to be seen, how the model performs on new images with very different faces, but it is clear to say, that this model outperforms the one created in this project by far.

	precision	recall	f1-score	support
ffp2	1.00	1.00	1.00	32
no_mask	1.00	1.00	1.00	27
accuracy			1.00	59
macro avg	1.00	1.00	1.00	59
weighted avg	1.00	1.00	1.00	59

Listing 3. classification report

## VI. Summary and Outlook

### References

- [1] A. Rosebrock, "Covid-19: Face mask detector with opencv, keras/tensorflow, and deep learn-

ing - pyimagesearch," 5 2020. [Online]. Available: <https://www.pyimagesearch.com/2020/05/04/covid-19-face-mask-detector-with-opencv-keras-tensorflow-and-deep-learning/>