



Neural Networks report

# Facial recognition: the mask/no mask case

*Prepared by*
**Mia Swery**
**Eva Radu**

*Supervised by*
Enrico Formenti

Master Informatique
at Université Côte d'Azur
First semester of the year 2021-2022

# Contents

# Part I
# THE FIRST PART OF THIS PROJECT

***** WARNING ******

Because of the restricted size on the moodle platform, please make sure to look at our project on our github repository : https://github.com/MiaSwery/NNL_Project
Besides, please notice that despite this file being a pdf one in our zip, this whole report has been written in LaTeX.

## A Introduction

In the first part of this project, our purpose is to build a program that enables a user to annotate images. Indeed, for every picture of the dataset, the user should be able to select boxes and associate a category for each of these boxes.

The aim of this program is to detect whether a human being in a picture is wearing a mask or not, in order to create an image classifier. It will be the second part of this project.

## B Image preprocessing

We selected a hundred pictures of people wearing a mask or not from a Kaggle database. Then we converted and saved them into a PNG format. We chose to resize the pictures that would be too big to fit into the window of the program. Their new size will be a maximum of 500x500. After being resized, the images are stored in the "resized" directory of the "img" directory. We also made sure our program is also able to deal with JPG images.

We pre-processed the images only once through our PreProcessed program.

## C The features of our interface

### i Creation and features of the boxes

If a user wants to select an area to annotate, he must first click one end of the box and release this click at another end of the box. Then a pop-up will be shown containing all the information about the box that has just been created, that is to say: the coordinates of the four ends, the height, the width, the area, and finally the category to which the box belongs.

A user can create multiple boxes as long as they are not too small, contained within another box, or containing another box.

Furthermore, whenever a box has a total intersection with the other boxes higher than 20%, this box is deleted and it will not be saved in the annotations.json file.

All the information on the different boxes is written within a json file entitled annotations.json (which is generated when all the pictures have been annotated). The category of a box is set by default to “No category”.

But a user is able to add other categories !

#### ii Import/Add of the categories

We decided to create a file named “categories.json” to save all the categories we will work with.

At first, we only have one category to annotate the pictures, which is a default category (oddly enough, it is called “No Category”). The user is then able to add new categories either from a file or manually. This file can be a json or a csv one. We considered that a json file should be written as following :

```
{
"categories": [ "c1", "c2","c3"]
}
```

And the csv files can be written in two different ways :

"c1",
"c2", or "c1", "c2", "c3"
"c3",

The categories from a file are imported thanks to the path the user will type (if the path does exist and the file is written accordingly to the rules mentioned above). These categories will be saved in our categories.json file.

#### iii Replacement of a category

A user may want to replace the name of a category with another one. It is possible thanks to our “Replace a category” button. There are two text entries. The one at the top is where the user should type the name of the category he wants to replace and the entry below it is reserved for the new name of the category. Internally, the categories.json will be modified. For consistency’s sake, all the boxes previously annotated and belonging to the replaced category will have their category’s value replaced by the new name of the category.

The user might want to replace a category with another one that already exists. This action might be seen as a replacement followed by a deletion.

Don’t worry : deleting a category is also possible !

#### iv Deletion of a category

If a user wants to delete a category, it is not a problem. This category will be safely deleted within our categories.json file and all the boxes belonging to this category will now be considered as "No Category" boxes.

All the user needs to do is to type the name of the category he wants to delete within the reserved entry and then press the "Delete a category" button.

The help button The different steps to create a box and visualize all the related information are summarized within the help window. If a user feels lost, all he needs is to click on our "Help me !" button.

#### v Next Page

Once all the boxes have been successfully created, the user can move on to the next image by clicking on the "next" button.

When a user clicks on the "next" button while the window is showing the last picture to be annotated, the annotations.json file where all the boxes and their information are stored will be generated !

## D How things were conceived internally

#### i Saving of the boxes

Since none of us ever used objects in python, we had to work with multiple functions and both global and local variables.

For instance, all the boxes are saved in a dictionary within the code. They have some "features" : [first_coords, second_coords, height, width, area, category, current_image_number, rect_id]. The "first_coords" attribute is a tuple of the first point selected by the user and "second_coords" is a tuple of the second point released by the user while drawing a box.

If a box is updated, any change will be saved within this dictionary. For a later use of all the annotations made by the user, the information about the boxes is written within our annotation.json file. It could be filled like this for instance :

```
{
"data": [
{
"image": {
        "path": "../img/resized/with_mask/image31.png",
        "boxes": [
        {
        "point top left": [
                170,
                262
        ],
        "point top right": [
                322,
                262
        ],
        "point bottom left": [
                170,
                141
        ],
        "point bottom right": [
                322,
                141
        ],
        "height": 121,
        "width": 152,
        "area": 18392,
        "category": "No Category",
        "path": "../img/annotated_images/image0.png"
        }
        ]
}
},
{
"image": {
        "path": "../img/resized/with_mask/image15.png",
        "boxes": "null"
}
},
...
]
}
```

#### ii Windows

We have a root window from which all the other windows will be created. The root window is never destroyed until the program stop running (contrary to the other windows). It is simply modified.

However, please be careful since there is a minor bug ! Our pop-up windows tend to appear behind the current window and not before it. We tried to find some way to fix that, but we didn't find any solution.

## E Our team

We tried to work together all the time so that we could keep track of the evolution of the project. Hence, it is a bit difficult to say who did what, since we did everything together.

## F Conclusion and possible improvements

To conclude, we can say that our interface seems to be working properly and to be doing everything it is supposed to do.

Of course, some improvements could be done with more time. For instance, we could add a button that helps to retrieve the image shown before the current image. We didn't achieve to do it since we didn't know how to put back all boxes on the pictures.

# Part II
# THE SECOND PART OF THIS PROJECT

## G Our Team Work

Given that some very important libraries didn't work on Mia's computer, during this part of the project, we essentially work together on Eva's computer thanks to the discord platform!

## H Introduction

In the second part of the project, we were asked to create a predictor for images containing people faces that recognizes whether these people are wearing a mask or not.

In order to fulfill this request, we must train our data. But what exactly is our data set ?

## I Our dataset

We used the first part of the project to build our data set. Indeed, all we needed to do was running the program Annotator.py we made previously, define manually the boxes that surround people face on the photographs we had and label them as with_mask/without_mask.

For each picture, we extracted the area corresponding to the inside of the box we drew.These pictures were saved and resized to form our dataset on which our predictor will train and be tested !

## J Our models to build an efficient predictor

### i General Idea

In order to realize our predictor, we needed to build a neural network. Let's recall, briefly, what is a neural network. It is a branch of computer science which was inspired by biological neural networks. A neural network learns to perform tasks by being exposed to various datasets and examples without any task-specific rules. It generates identifying characteristics from the data they have been passed.

In order to analyze our models, we thought it could be useful to add some basic definitions to our report :

- Weights : numeric values which are multiplied with inputs. In back propagation, they are modified to reduce the loss.

- Input layer : dimensions of the input vector.
- Hidden layer : the intermediary nodes which produces output.
- Output layer : the output of the neural network.

There are many models to create a neural network. One of them is the convolutional neural network model. It is a model adapted for analyzing and identifying visual data such as photographs. It is thus a model that perfectly fits our needs in this project !

Please notice that in all the following models chose to use 80% of our data set as a training set and thus 20% of our data set as a test set.

Also, you should note that we built our classifiers to recognize only one face at a time on a picture.

#### ii The classifier

First, we decided to create a predictor that will use directly the boxes associated to a picture to predict whether a human is wearing a face or not.

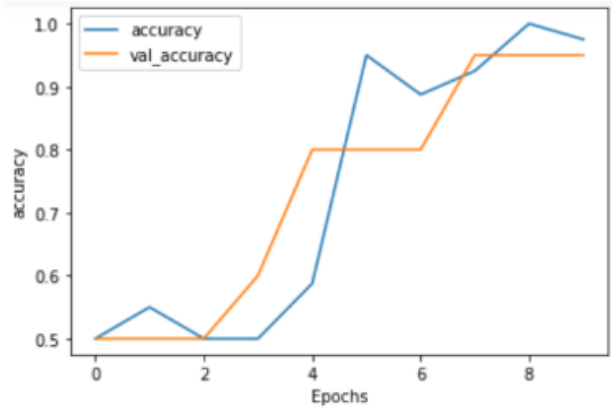
At first, we decided to use 20 epochs. As shown in the following pictures, we obtain an accuracy for the training set equals to 0.95 and an accuracy for the test set equals to 0.949999988079071. These are great results, since the closer an accuracy is to 1, the better the predictions (and thus the model) are. Besides, the matrix of confusion is also quite good.

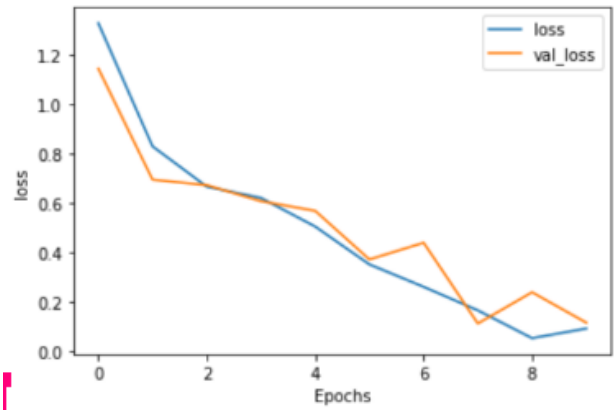
We chose to analyze the evolution of the accuracies and the loss of the model over the number of epochs (ie the time). We notice the loss decreases and the accuracy rate increase.

We can thus conclude that our predictor is working properly and seems to be really good. But can it get better ? To answer this question, we decided to change some parameters of the model and see what happens then.

First, we try to decide what is the best number of epoch to choose. So let's decide by testing and then analyzing the results we will get.

Results for the number of epochs equals to 10 :



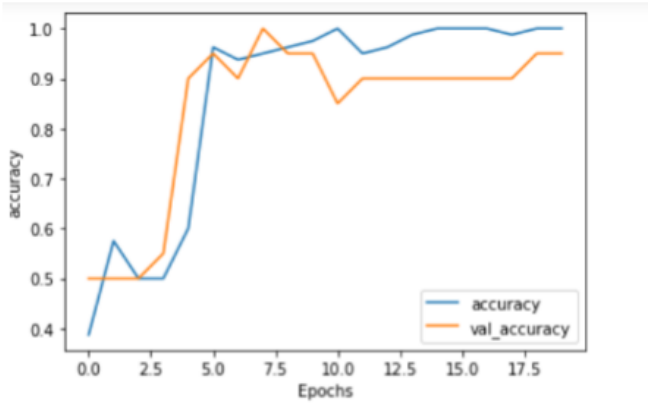




```
Accuracy :  0.95
Matrice de confusion :
[[ 9  1]
 [ 0 10]]
              precision    recall  f1-score   support

   with_mask       1.00      0.90      0.95        10
without_mask       0.91      1.00      0.95        10

    accuracy                           0.95        20
   macro avg       0.95      0.95      0.95        20
weighted avg       0.95      0.95      0.95        20
```

Results for the number of epochs equals to 20 :

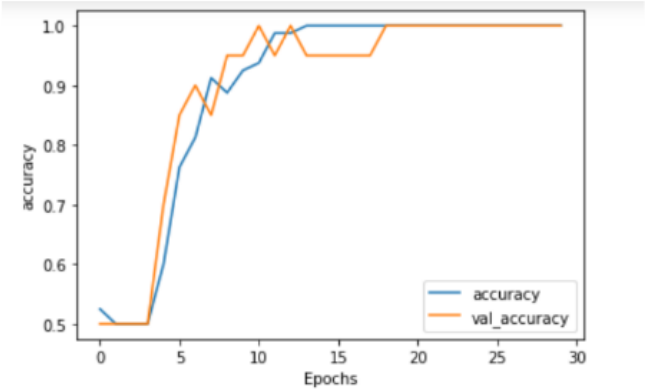


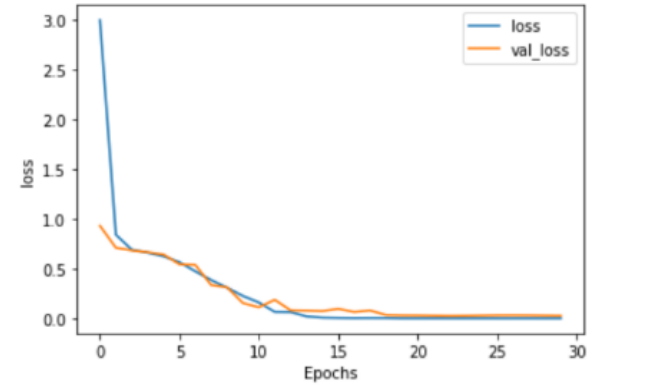

```
Accuracy :  0.95
Matrice de confusion :
[[ 9  1]
 [ 0 10]]
              precision    recall  f1-score   support

   with_mask       1.00      0.90      0.95        10
without_mask       0.91      1.00      0.95        10

    accuracy                           0.95        20
   macro avg       0.95      0.95      0.95        20
weighted avg       0.95      0.95      0.95        20
```

Results for the number of epochs equals to 30 :



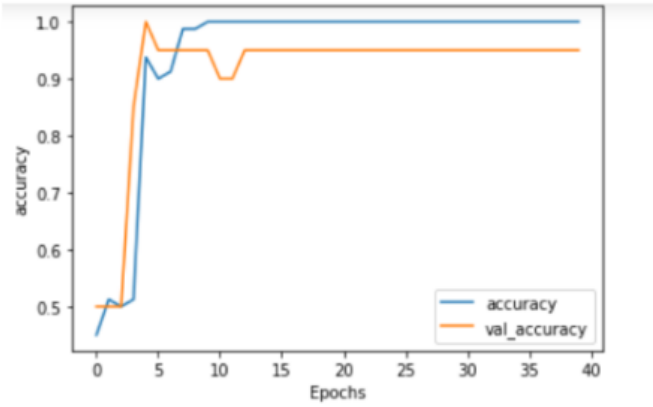



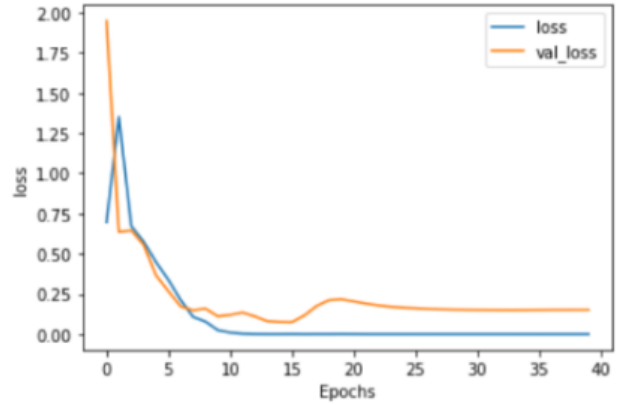

```
Accuracy :  1.0
Matrice de confusion :
[[10  0]
 [ 0 10]]
              precision    recall  f1-score   support

   with_mask       1.00      1.00      1.00        10
without_mask       1.00      1.00      1.00        10

    accuracy                           1.00        20
   macro avg       1.00      1.00      1.00        20
weighted avg       1.00      1.00      1.00        20
```

Results for the number of epochs equals to 40 :



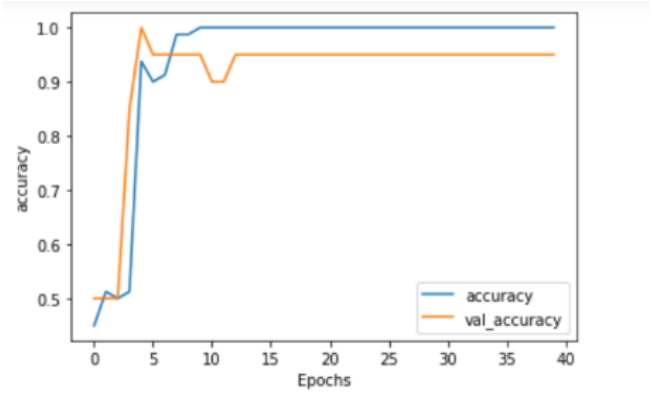



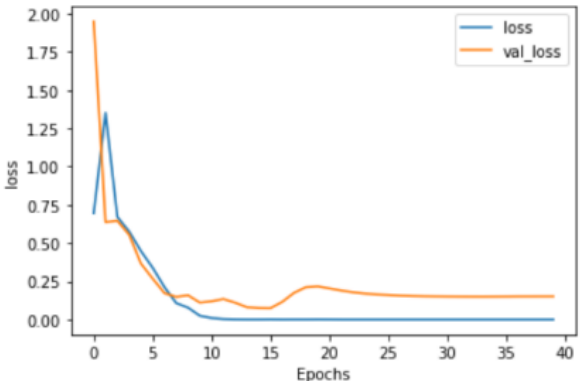

```
Accuracy :  0.95
Matrice de confusion :
[[ 9  1]
 [ 0 10]]
              precision    recall  f1-score   support

   with_mask       1.00      0.90      0.95        10
without_mask       0.91      1.00      0.95        10

    accuracy                           0.95        20
   macro avg       0.95      0.95      0.95        20
weighted avg       0.95      0.95      0.95        20
```

Results for the number of epochs equals to 50 :



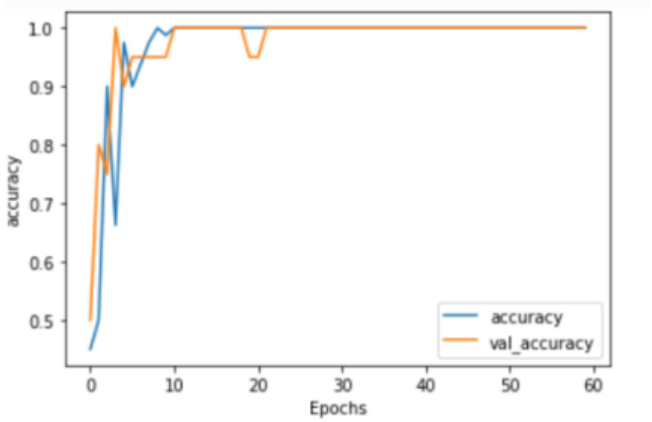



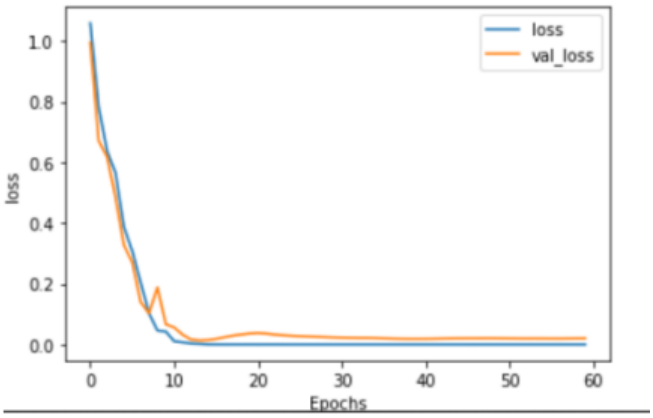

```
Accuracy :  0.95
Matrice de confusion :
[[ 9  1]
 [ 0 10]]
              precision    recall  f1-score   support

   with_mask       1.00      0.90      0.95        10
without_mask       0.91      1.00      0.95        10

    accuracy                           0.95        20
   macro avg       0.95      0.95      0.95        20
weighted avg       0.95      0.95      0.95        20
```

Results for the number of epochs equals to 60 :



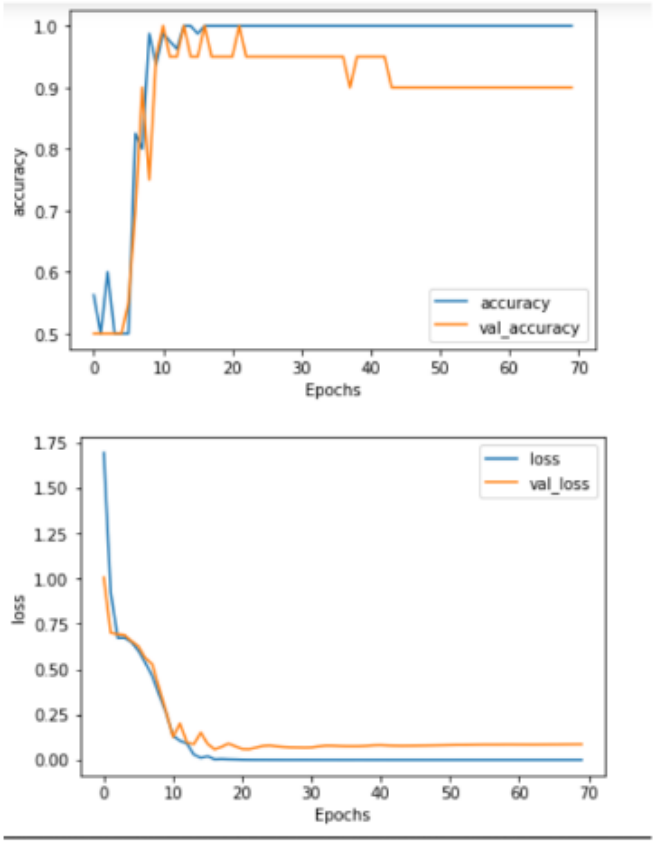




```
Accuracy :  1.0
Matrice de confusion :
[[10  0]
 [ 0 10]]
              precision    recall  f1-score   support

   with_mask       1.00      1.00      1.00        10
without_mask       1.00      1.00      1.00        10

    accuracy                           1.00        20
   macro avg       1.00      1.00      1.00        20
weighted avg       1.00      1.00      1.00        20
```

Results for the number of epochs equals to 70 :




We can observe that when the number of epochs is equal to 30 or 60, we get the best accuracy. Indeed, among all our attempts, these numbers of epochs give us an accuracy equals to 1, which means that the model always predict the correct categories. Furthermore, by comparing all the graphs, we can say that after a number of epochs equals to 30, the performances tend to be stable. These value seems to be a point from which the accuracy is very high and the loss value is very low. Thus, during the rest of the project, we considered a number of epoch equals to 30. Given that our predictor has now an accuracy equals to one, it becomes difficult to make tests about the other parameters. But since this predictor seems to be really efficient, let's try to do a more complex predictor.

#### iii Our predictor

In order to do that, we tried to create an even more interesting predictor. Instead of simply looking for the boxes related to a picture given as an entry, the model will also have to detect human faces. How do we do that ?

We actually need to use 2 neural networks. One that learns how to predict boxes and another one (on top of the first one) that learns to classify a picture. This model will still be trained on the dataset we created previously. However, it will be trained over the boxes the model defined by itself !

Note that the accuracy of this model is computed only according to the predictions of the classes, and not according to the predictions of the bounding boxes.

So first we kept the parameters of the previous model that is to say 30 epochs (and the parameter of the Drop layer is 0.5). We obtained an accuracy of 0.75 and truth to be told the predictions about the boxes are not really accurate. So we decided to test this model with other parameters and found out that 50 epochs help to gain in performance. Indeed, with 50 epochs and a the parameter of the drop layer set to 0.5, we got an accuracy of 0.85 which is quite satisfactory given the complexity of the task.

Then we tried to change the value of the Drop layer parameter for a CNN of 50 epochs.

- 0.25 : accuracy = 0.75
- 0.5 : accuracy = 0.85
- 0.75 : accuracy = 0.8

According to our results, it is best to use 50 epochs and 0.5 as a parameter of the Drop Layer to gain in performance (although 50 epochs obviously take more resources and time to run than 30).

#### iv General remarks

For our different models we decided to implement a function that determine the category of a specific image. Two different modes can be used : a mode "category" if we want to get the predicted category for an image, and a mode "probabilities" if we want to get the probabilities about the different categories for that image. Note that the file of the image should be part of the list of the paths of the test images. Moreover, for our second model, we decided to display all the images of the test images and to draw the predicted bounding box of a face. For instance, this is one example of a fairly correct prediction :


## K Conclusion and perspectives

It is now time to conclude our report. During this project, we advanced by trials and errors. Indeed, we used different parameters to understand how to build the best models and we used different type of models. Generally, our accuracies are between 0.8 and 1 which means that our models are satisfactory. The first model we implemented only focuses on classifying picture between 2 categories. Its accuracy is high but the task is easy compared to what our second model must do. Indeed, on top of classification, it must determine the area where a face is located on a picture. The accuracy is this less high than the first model but it could be worse. Besides, we thought about implementing a third model that relies on the Cascade model from the cv2 library that is quite good to detect faces on a picture. However, the results we obtained were not so good and we were no sure we had the tight to use a model as Cascade in this project so we dropped the idea. After all these attempts, we can thus conclude that both our model seem good if we look at our accuracies, although one is more complex than the other !

As it is always possible to do better, we thought about the different improvements we could make. It would be interesting to build a model that recognize multiple faces on a picture and classify each of them.