Université Côte d'azur

Master Informatique
Neural network and learning
Rapport

---

# ImageAnnotator and Face Mask recognition

---

***Students :***
Mokhtar Samy
Mohamed Belhassen
Omar Alami

***Instructor :***
Enrico Formenti

January 27, 2022

# Contents

# 1 Description of the project and its goals

## 1.1 Description of the project

The first part of this project is about creating a simple application that allows users to upload an image and manually select the bounding box of each category in that image, and save the annotations containing the category and the coordinates of each bounding box into a **JSON** file.

The second part is to create a model that can classify an image with a face with or without mask.

## 1.2 Goals

The goal of the first part is to prepare the data-set for training the model created in the second part, this data-set is a set of photos with people wearing face masks and others without masks, with a JSON file containing the annotation as mentioned in the previous section. Those annotations will be used to generate smaller images for the training phase.

# 2 Data

The input data is an image that we upload to our application. The output data are the list of categories of each image and the list of coordinates of each category in an image.

## 2.1 ImageAnnotator

We have build the ImageAnnotator with documentation, respecting the given restrictions on the itersection given in the first part of the project, we have also integrated the bonus part, deleting a category, modifying a category.

As shown in the figure above, we can select as many boxes as we want and save the annotations to use it later.

## 2.2 How we produced the data

**kaggle:** We used images we found from kaggle. We separated the images on 2 folders containing 2 classes.

We produced the data for the training by selecting the bounding box with faces of people only, in order to get rid of the additional unnecessary details in the images, those smaller images were the input for our model training set.

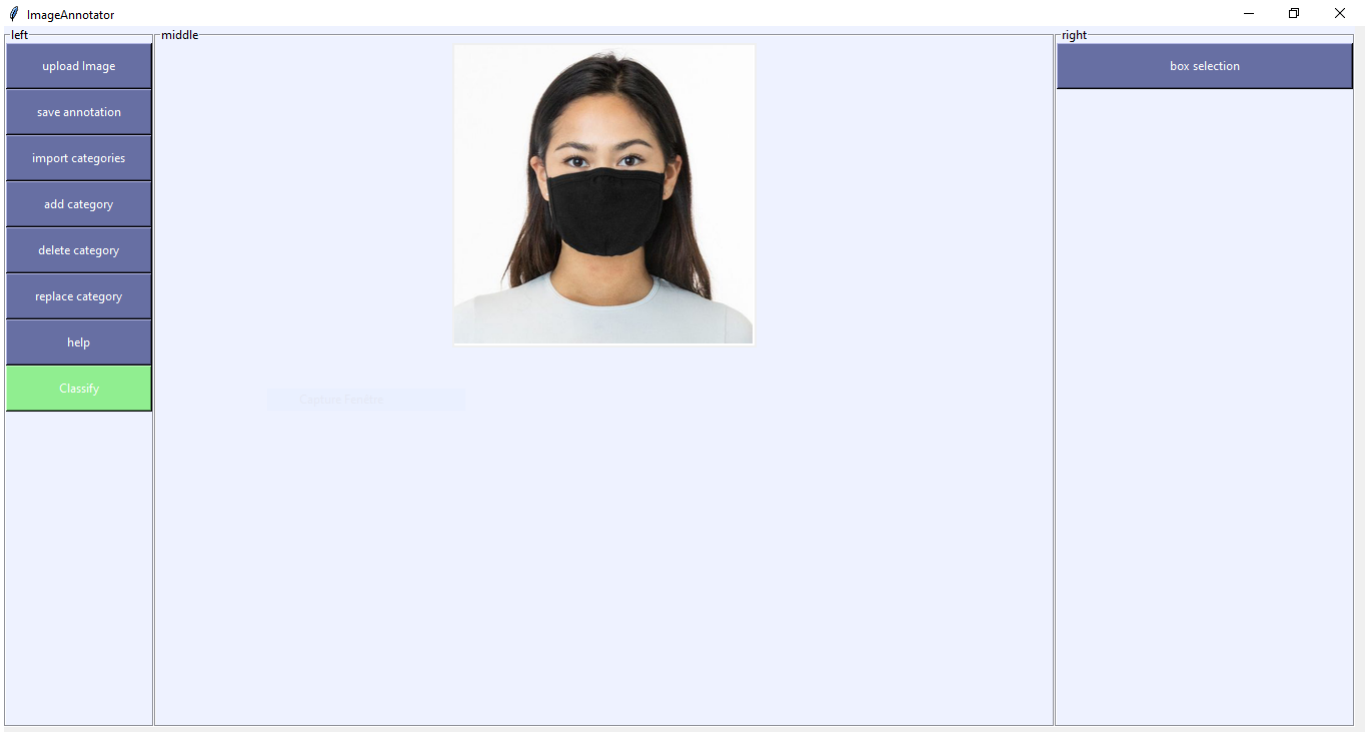


Figure 1: The image annotator interface

# 3 Used methodologies and technologies

## 3.1 The technologies used

We have used Python version 3.90 to code our program. For the Image Annotator , we used several libraries such as:

- Tkinter to create the GUI.
- Shapely to check if there's any intersection between two rectangles of the bounding boxes, Pillow to load the images.
- Json to save and import the data into and from Json files.
- Opencv to select a bounding box or a rectangular region of interest (ROI).

  For the second part we used:
- Tensorflow's Keras sub-library to create and train our model.
- Scikit-learn to split our data into train and test sets, as well as the label encoding of the images.
- IPython to display the images with their predicted class for testing our model's performance.
- Matplotlib to display the graphs illustrating the performance of the model.
- We have used Jupyter notebook to organize our code by separating our code with the right description of each section and with a track of the execution result.

## 3.2 The methodology adopted

**Part1:** To meet the requirements of the first part of the project, we used all the technologies suggested in the subject. Tkinter was the main tool to build the graphical user interface for the Image Annotator, we searched for documentation and tutorials to learn how to build a GUI using Tkinter.

We managed to separate the logic part from the GUI part, **Part2:** To generate smaller images for the training phase, we created a script that fetch JSON file with the annotation and iterate through it to get each annotation and generate an image based on its coordinates. Then save each image to the corresponding class folder.

We used Sequential from Keras to build the CNN model and tested it with different parameters to compare its performance.

To briefly explain how CNN models work, there are 2 main phases, the first is the features extraction:

1. We first input an image, then the convolution process will apply different filters( filters map) on it to extract the different features of that image, each filter detect one feature, and all the filters in a model are of the same size.

2. The second step is to apply the maxpooling, this allows the CNN to be able to detect an object in whatever position, this is done by manipulating the image by flipping it, zooming ...etc.

3. Step one and two can be done more than once to improve the performance of a model.

4. The third step is flattening, once we get to this step we should already have a pooled map of features, flattening those features who are in a matrix shape into a column shape, this vector of values will be given to the ANN to process it.

5. After flattening, regularization can be applied to improve the accuracy of the model, so we add Dense and Dropout.

To overcome the difficulties we kept looking for more tutorials and documentations about CNN and Tensorflow library that simplify everything about machine learning.

At the beginning there were a lot of errors and unclear concepts about the parameters, for example, using sigmoid of softmax for the activation of the last layer, which loss function should we use...etc.

# 4 Build and optimize our model

So in this section we explain how we have done different experiments to find the best parameters for our model.

we will start by asking the right question :

- How can you get better performance from our deep learning model?
- How can we improve accuracy?
- What should we do if our neural network performs poorly?

so in this section we will try to answer this questions by :

- Improve Performance With Data.
- Improve Performance With Algorithm Parameters.

## 4.1 Model architecture

we will use as architecture for our model a lookLike simplified version of the **LeNet architucture by Yann LeCun**.

Firstly because it's A great idea to start with a good architucture which it is become the standard for this kind of problem .Secondly because it is one of the simplest architectures.


Figure 2: basic looklike lenet architecture1

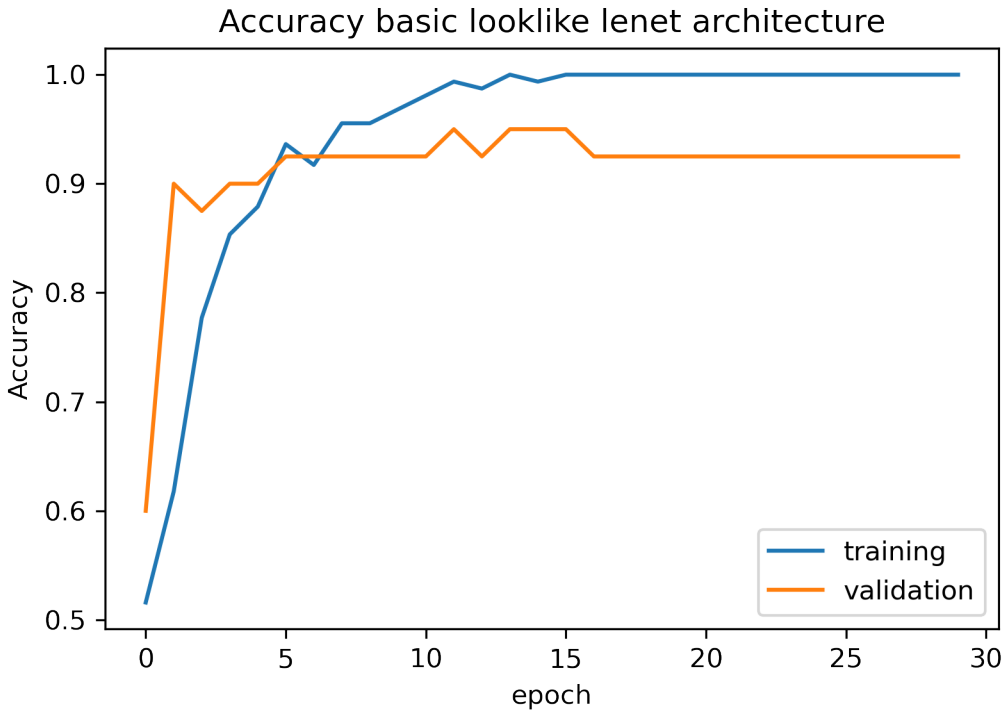


Figure 3: basic looklike lenet architecture2

After we run our model with the basic parameter of our initial version, we got a very high accuracy on our training data-set and we tested our model in our test data-set we got only 92% but with a loss equal to 0.25 which is a high value .

so this is an example of underfitting because in training set it tends to do so good (0.01) but on test set it is having only 0.25 lost so most likely there is an underfitting happening

## 4.2 Improve Performance with Data

so to skip the underfitting we will try to reuse the old model with two changes :
- supplying data augmentations to our model as our first layer to produce new samples
- adding a dropout layer to drop randomly in each pass à 20% of neurons to have better generalization

### 4.2.1 Data augmentation

We'll explore how data augmentation can reduce overfitting and increase the ability of our model to generalize via two experiments.

To accomplish this goal we will "replace" the training data with randomly transformed, augmented data. **Data augmentations** on images would include transformations like :

- Flipping the image either horizontally or vertically
- Rotation the image

- Zooming in and out on the image
- Cropping the image
- Varying the color on the image

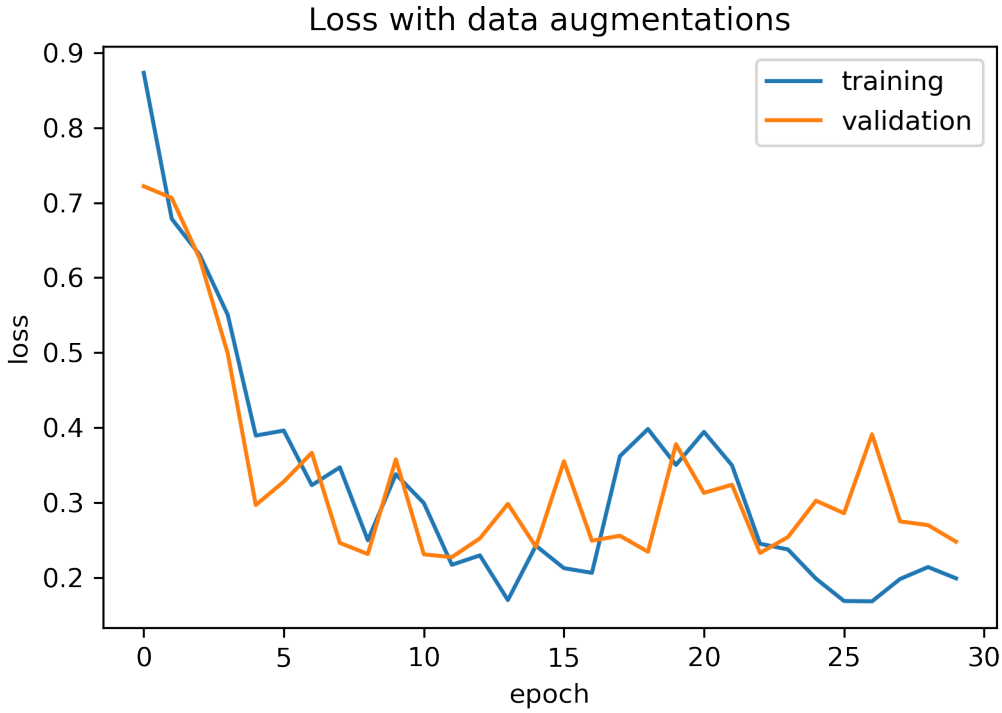


Figure 4: with data augmentations

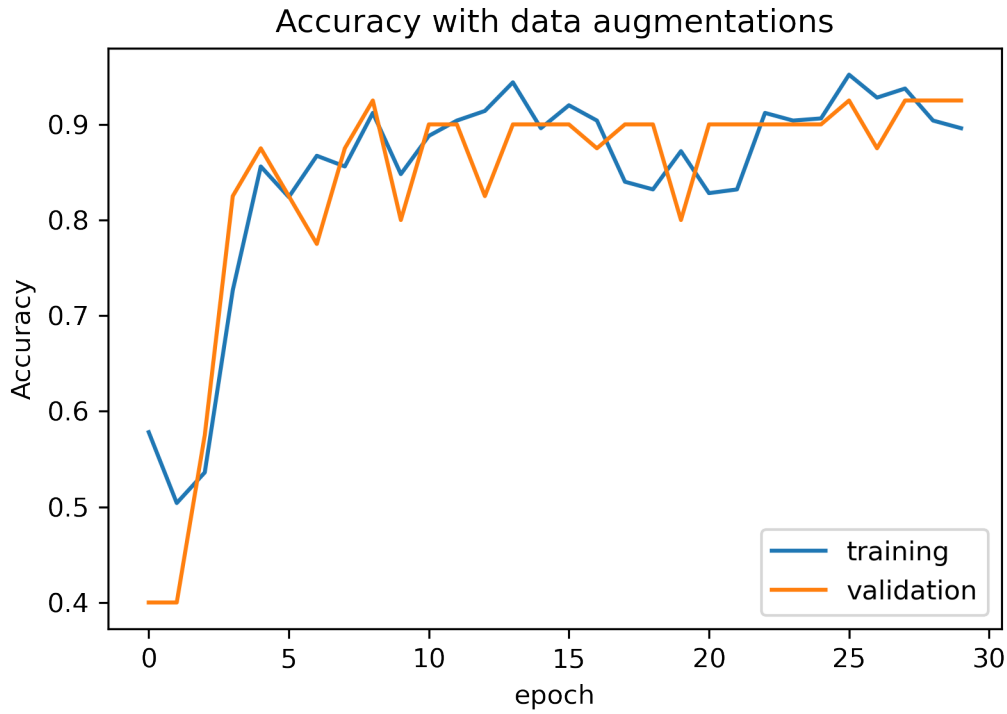


Figure 5: with data augmentations

After we run the model, we got a new val-loss that is accurate with the loss of training data-set.
but we got a accuracy value that is slightly worse than the previous version of our model ( due to the variations in data caused by the random transforms).

## 4.3 Improve Performance With algorithm paramter

### 4.3.1 Regularizations

Dropout is also a great approach to curb overfitting of the training data. So we have tested different values of the Dropout

We can see that for this problem and for the chosen network configuration that using dropout in the hidden layers did not lift performance.

It is possible that additional training epochs are required or that further tuning is required to the learning rate.

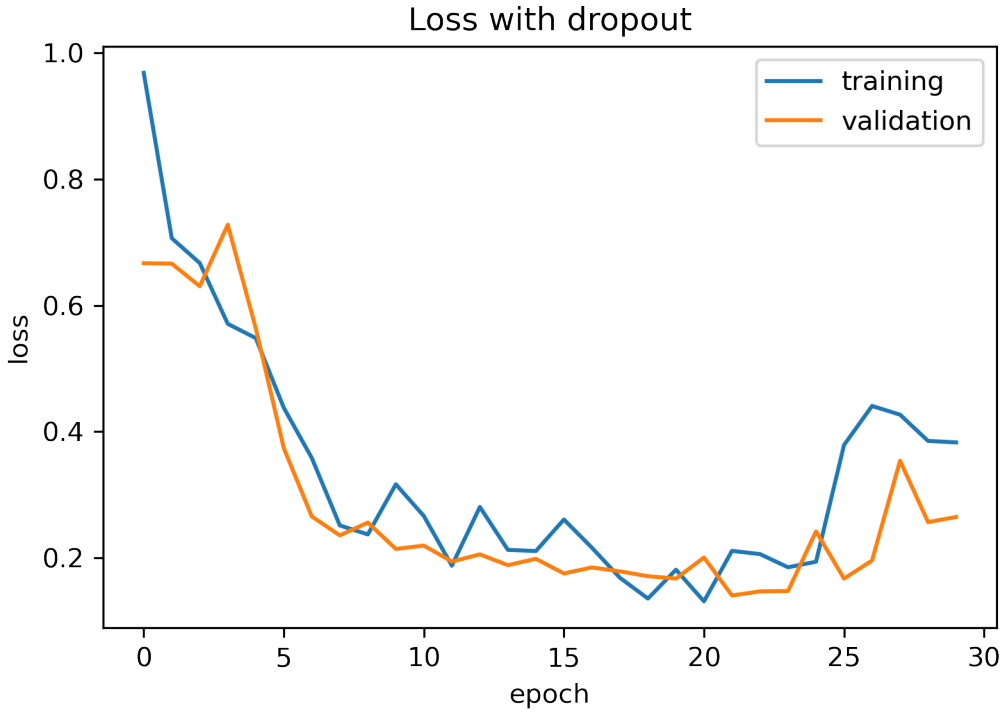


Figure 6: with dropout

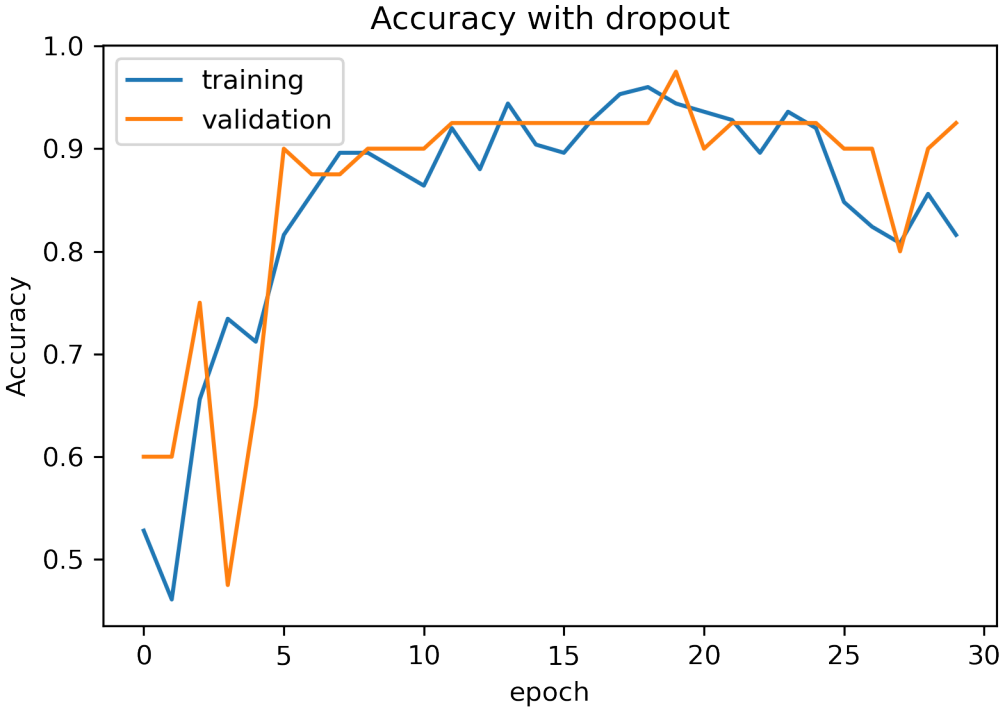


Figure 7: with dropout

### 4.3.2 Number of Epochs

In this section we will experiment large number (500) of epochs and see their effect in the global learning curve .

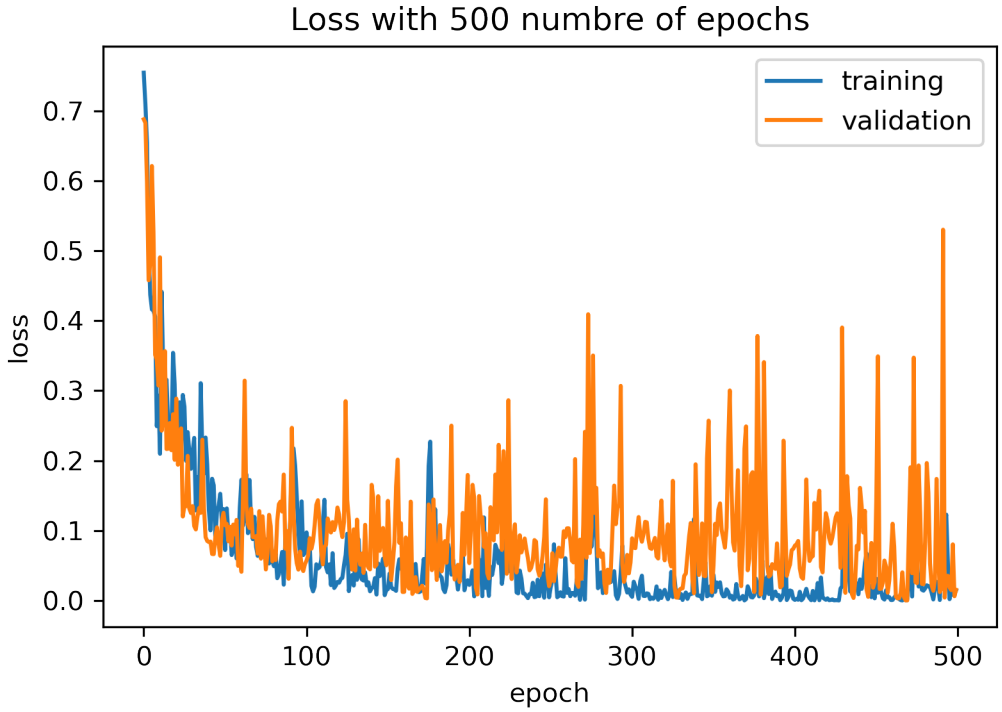


Figure 8: with 500 number of epochs

The figure above shows the effect of the number of epochs on the loss rate, starting from 100 epochs the lost rate in the training set was almost constant, however, it varies a lot with the validation set ranging from , approximately 0.02 to 0.5.

These plots can help to diagnose whether the model has over learned, under learned, or is suitably fit to the training data-set.
in our case we see that start to decrease until it reach 100 - 150 and after it stared to increase again so we can see after a 150 epochs our model well start over learning .

### 4.3.3 Size of Batch

We have tried with different sizes of batch:

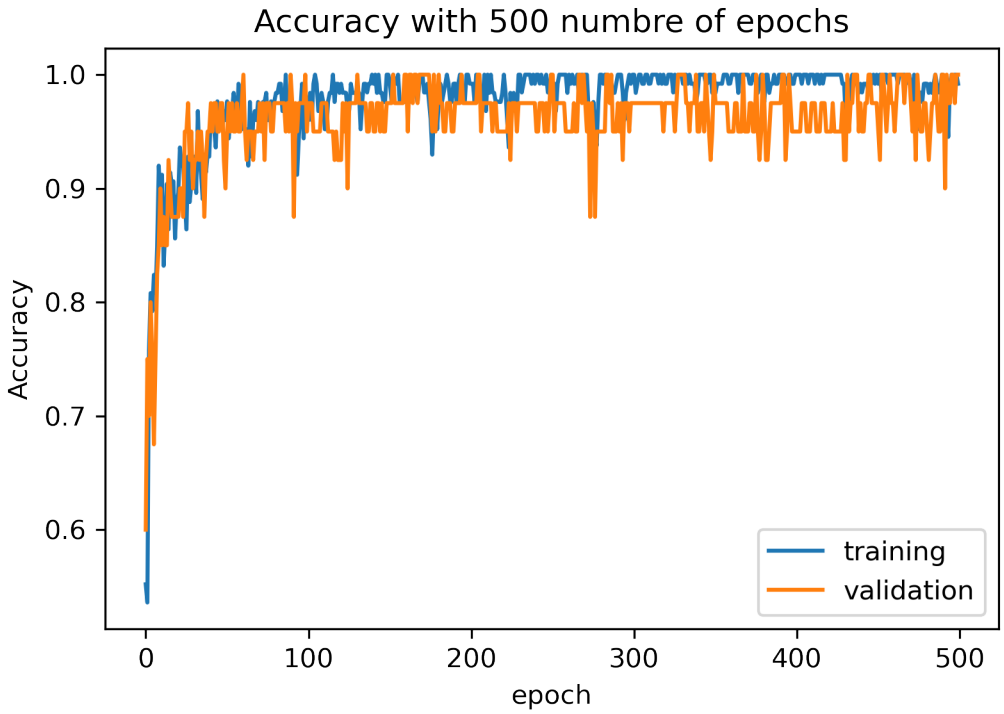


Figure 9: with 500 number of epochs

| Size of Batch | validation accuracy |
|---|---|
| Size of Batch: 1 | 0.949 |
| Size of Batch: 8 | 0.9499 |
| Size of Batch: 16 | 0.9750 |
| Size of Batch: 32 | 0.9250 |
| Size of Batch: 64 | 0.9750 |
| Size of Batch: 128 | 0.899 |

So finally we decided to use 16 as the batch size as it gave the best result in terms of accuracy.

- A batch size equal to the training data size, depending on the memory (batch learning).
- A batch size of one (online learning).
- Tried a grid search of different mini-batch sizes (8, 16, 32, . . . ).
- Try training for a few epochs and for a lot of epochs.

### 4.3.4 Number of neurons

| Number of Neurons | validation accuracy |
|---|---|
| Number of Neurons: 32 | 0.925 |
| Number of Neurons: 64 | 0.925 |
| Number of Neurons: 128 | 0.949 |
| Number of Neurons: 500 | 0.925 |

As the figure shows, we decided to use 128 as the number of neurons as it gave the best result in terms of accuracy.

# 5 Integrating the model into the Image Annotator

We have integrated the model with the Image Annotator, so after saving the model we load it and use it to predict the current uploaded image in the Image Annotator, and display the result.

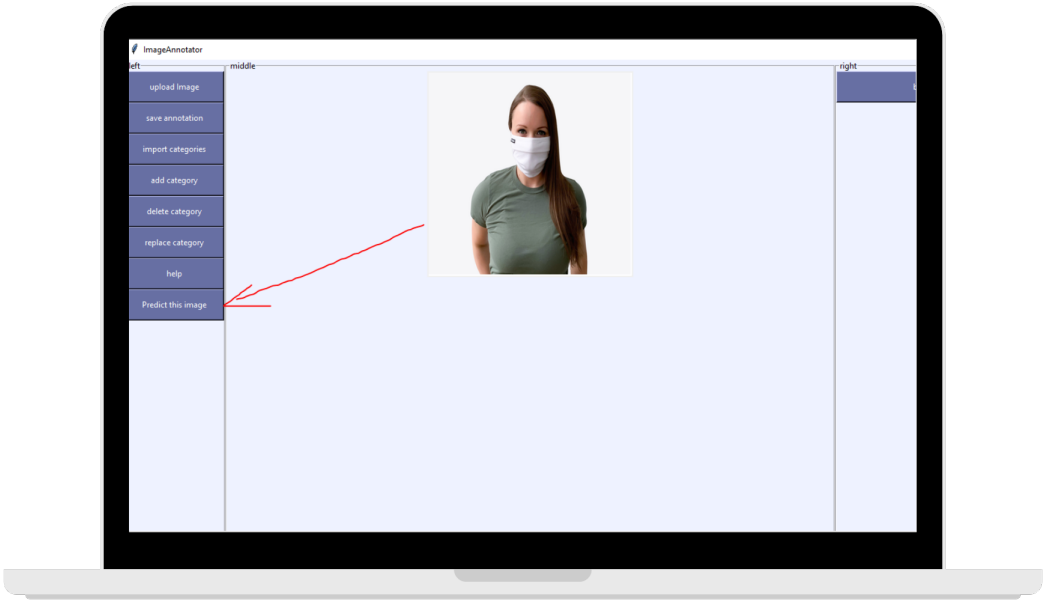


Figure 10: button to detect image

# 6 Real-time face mask detection

we have implement a real time mask detector .that use the webcam to capture every frame . the real time mask detector use our saved pretrained model to classify the frame. and show a label in it


Figure 11: real time detector

# 7 Object detection

We spent too much time trying to make the object detection functional with our CNN model, however, it was far from accurate, despite it does recognize different object in a photo, the predictions most of the time is wrong.

We first tried to adopt the concept of sliding window to detect objects in a photo, that sliding window is a smaller box scanning the whole image, we can consider that window as a cropped part of the original image, each cropped part will be saved in a list of smaller images, at the end, that list of images will be given to our CNN classifier to predict the classes of each one of those images.

Then we set a minimum confidence ratio, for example 0.8, so from those list of smaller images, we take only the images classified with a probability higher than 0.8, then add

bounding boxes for each of those images after filtering our list, those will be the objects we have detected on an image.

However, when we used this concept, we found a problem with probabilities as the model predict a class for each smaller image with a very high confidence ratio, so it usually detects, for example, a mirror, as a no-mask class, this is probably due to lack of training as it doesn't know anything other than mask and no mask, it doesn't recognize a mirror as it is not a class in our data-set, however, if the model have classified it as a no-mask with a probability of 0.6 for example, it wouldn't be an issue, but that was not the case and we couldn't solve this problem after too many experiments.

We tried using OpenCv to let it detect the bounding boxes for us then take each box and pass it to the classifier, it worked much better than our sliding window, and another much simpler way is to use the pre-trained models for object detection available in Tensorflow library, but we didn't implement it in our project as it doesn't meet the requirements of the project.

# 8 Conclusion and perspectives

To conclude, the overall performance of the model is satisfying and very accurate in its predictions, and changing the values in the parameters can dramatically change the results.

Increasing the number of photos into the training is the most important step.

## 8.1 Reason of the delay to submit our project

Please note that we have already finished the classification part on 12/01, we respected the deadline you gave us and we have pushed a commit on that day, please check the commit on that day on our Github repository to verify the date and the content, The link to our Github repository commit, 12/01/2022 .
However, after I sent you a message asking about the object detection, we wanted to complete the project and add the object detection to our model, that took very long time for us, with several attempts, we succeed to make it work.
In addition to that, BELHASSEN was infected by covid and had was very sick and had to rest, the positive result of the covid test have been given to the scolarité, we will send you a copy of it, and his exams were postponed.

## 8.2 Possible improvements

Our CNN model performed very well with only 2 classes, when we tried to add different types of masks it was not accurate, so this is one of the possible improvements that could be added to our model.

The second one is the object detection, we spent too much time trying to make it work, but we couldn't make its performance even acceptable to be added to our project.

We also could improve the performance of our model by extending our data-set so the training will be better, hence, better predictions.

# 9 Members of the group and their roles

## 9.1 Members and roles

**SAMY Mokhtar:** Creating the ImageAnnotator, building the CNN model, preparing the dataset and generating the smaller images from annotations, testing the object detection.
**BELHASSEN Mohamed:** Creating the ImageAnnotator and improving it, building the CNN model, building real-time mask detection model, doing experiments on the CNN model and comparing different parameters.
**ALAMI Omar:** Participate to the creation of the imageAnnotator and the CNN model ,preparing the data-set, and doing some experiments on it, documentation.