MASTER
INFORMATIQUE

UNIVERSITÉ
CÔTE D'AZUR

# Annotator and Mask recognizer

*Authors:*
Fissore Davide
Galbiati Federica
Venturelli Antoine

*Professor:*
Enrico Formenti

2021/2022

## Contents

## 1 Goal of the project

The goal of the project is to create a graphical interface allowing the user to make annotation on images by drawing a rectangle shape on the image and giving a name to it.
The cropped image can be saved in a specific folder with the *annotate-on-the-fly* button.
The graphical tool should also allow the user to load a folder of images, annotate them with the wanted tags and then make prediction on newer images.
A prediction wants to classify the new image, so if, for example, we load a set of people images and tag human face having or not mask with *Mask* and *NoMask* tag, the predictor should return the probability of the input to belong to one of these two classes.
An important remark is that the predictor has been tested only to classify faces with *Mask/NoMask* but it should also work on other situations.
It is important also to note that the image classification only works if the input is "similar" to the training sample we passed to our model, for instance, if we make a predictor on *Mask/NoMask*, it will not work on a image of a group of people, since it doesn't detect and extract human faces. We can, however, annotate this image selecting and saving human faces in a folder and then make prediction on each of the new images.

## 2 Team presentation

The project has been realized by a group of three student : Fissore Davide and Venturelli Antoine, two students at the University of Côte d'Azur of Nice and Federica Galbiati, an *Erasmus* student coming from the university Bicocca of Milan.

## 3 Task repartition

We did the most part of the project in an available room of our University and we collaborated to create together each aspect of the project. In particular we divided the work as follow:

- Antoine worked on the graphic interface, for example, he added the different options such as scrolling menu, notebook frames and tooltip.
- Davide created the classes for the back-end process such as the *tags* and *images* classes to manage tags creation an tag elimination and the skeleton of the training phase made with *tensowrflow*.
- Federica dealt with the help panel (using HTML and CSS), the parse and save of JSON files and the research of the most performing model to make predictions.

## 4 Programming language and used library

The programming language used to create the project is Python version 3.9.7 exploiting the following libraries :

- PIL to load and treat images
- tkinter to charge the graphic interface
- json to save and load json encoded files
- ttkthemes to get a larger library of theme for our interface
- tkhtmlview to display simple HTML and CSS text
- shapely to work on shapes and get coverage methods

The project has been tested on Windows OS, but it should be portable on other operating systems like Mac, Linux *etc.*

## 5 How to run the project

The project must be launched from the *src* folder : this is mandatory so that imports of local files go correctly. From this folder, you can run the *main.py* file knowing that it will take about 30 second to start since the import of *tensorflow* and some graphical libraries tend to be very slow. To speed up a bit the program opening you can add the *"-fast"* optional parameter to open the classic tk.Tk() window, but it will not really change the global charging time. An example of program launch *python3 ./main.py -fast*, note that the python3 command may not work on some OS for example in certain windows distributions you should use py.

## 6 Project structure

The project is characterized by four main folders:

- src: containing the python file sources
- img: containing a sample of images that can be used in the program to make the *Mask/NoMask* recognition (these images are taken from kaggle)
- pre: containing some useful folders such as *annotation* where we find a JSON file with all of the annotations linked to the images of *img* folder, *test_img* where there are the images we used to test our models and *text* folder containing HTML files used in the *help* panel.
- rep: containing the report of the project

# 7 Main features (code behind the scene)

The code is based on two classes that hold the main logic to create annotation on image and tag manipulation.

## 7.1 The img class

The img class takes in paramater the image path and creates the base attributes needed to store other information such as the *set* of existing tags and the *dictionary* of annotations of the images that are represented as an association between a tag name $T$ and the list of coordinates of the rectangles having $T$ as tag name.
We can create a new association in the dictionary via the *add_ tag* method which takes as parameter the name of tag and the top-left and the bottom-right coordinates of the newly-created annotation.
In this method we do some checks to verify if these coordinates are valid (see Listing 1).

Listing 1: Annotation constraints from project subject

```
surface is less than 40 pixels in total or spatial
dimensions (heigh and width) are too small, for example
less than 5 pixels. Moreover, if a box has an intersection
with other boxes for more than 20% of its surface, it
should be discarted.One should also discard a box if it
completely covers a pre-existing box or it is completely
contained in a pre-existing one
```

In this class we have other tools to handle annotation removal and tag rename.
Finally the *_str_* and *_repr_* methods (similar to the toString method in Java) are used to return the string representation of an img object used to save it in JSON format.

## 7.2 The tag class

The tag class extends the classical *set* type of Python since we do not want any repetition of a tags.
Moreover a tag object owns a list of *imgs* to know every image of our application and perform *side effect* operation if needed.
For example, when we rename tag *T1* in tag *T2*, we take every img from the *imgs* list of our current tag object and, if the image owns *T1*, *T1* will be renamed in *T2*.
Similar operations are done for tag removal where every image will pop the tag key from its *dictionary* of annotation.

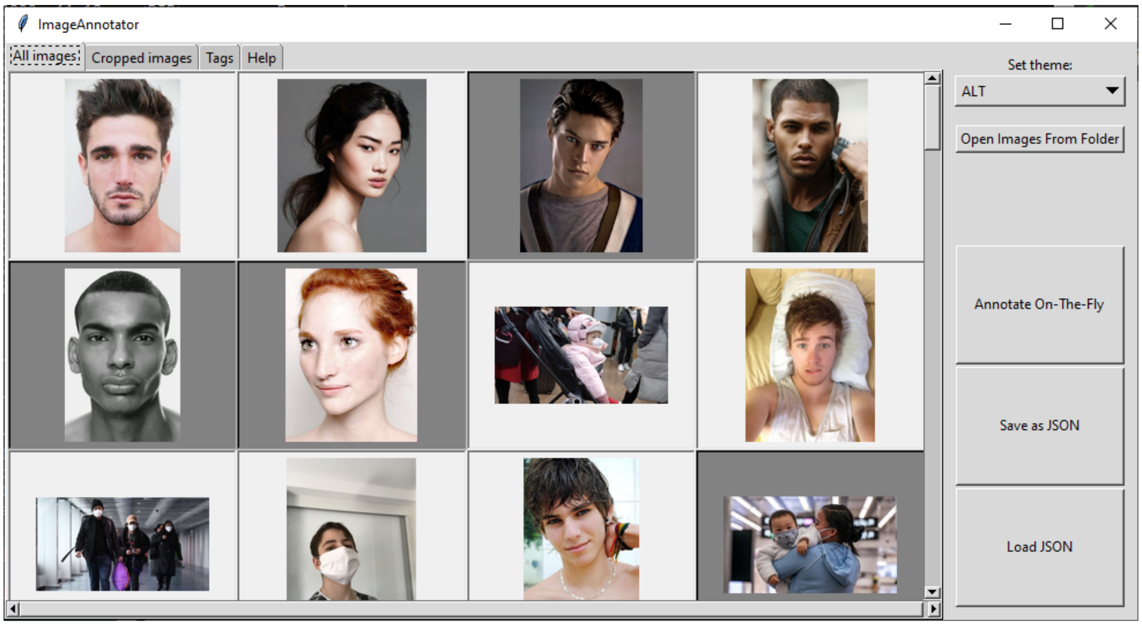


Figure 1: "All images" panel preview

## 8 Features for the graphical interface

### 8.1 The all images panel

When we run the project, we can see the first panel shown is something similar to Figure 1 where all images of the *img* folder are displayed. Here we can click on an image to open the *annotator* panel.

### 8.2 The annotator

From the annotator window, you can draw the rectangle representing a new annotation (in case of miss-click you can press the *ESCAPE* button of the keyboard to undo the annotation drawing).
Once the annotation drawn, you will see either the Figure 3a when you still have not entered a tag or Figure 3b offering you the possibility to choose an existing tag. You are also able to rename or delete an existing annotation with the right key of the mouse. This will perform a simple rename of the tag for the current image and not a whole rename of a tag *T1* in tag *T2* (this task can be accomplished in the Tag Panel).
To know the name of an annotation you can move the mouse on it and a little tool-tip will display in the bottom-left corner of the annotator window.
Finally, if you create a box that does not respect the parameter imposed (area size, sides length or covering constraints of Listing 1) an error message will be prompt to warn you about the impossibility of performing this task.

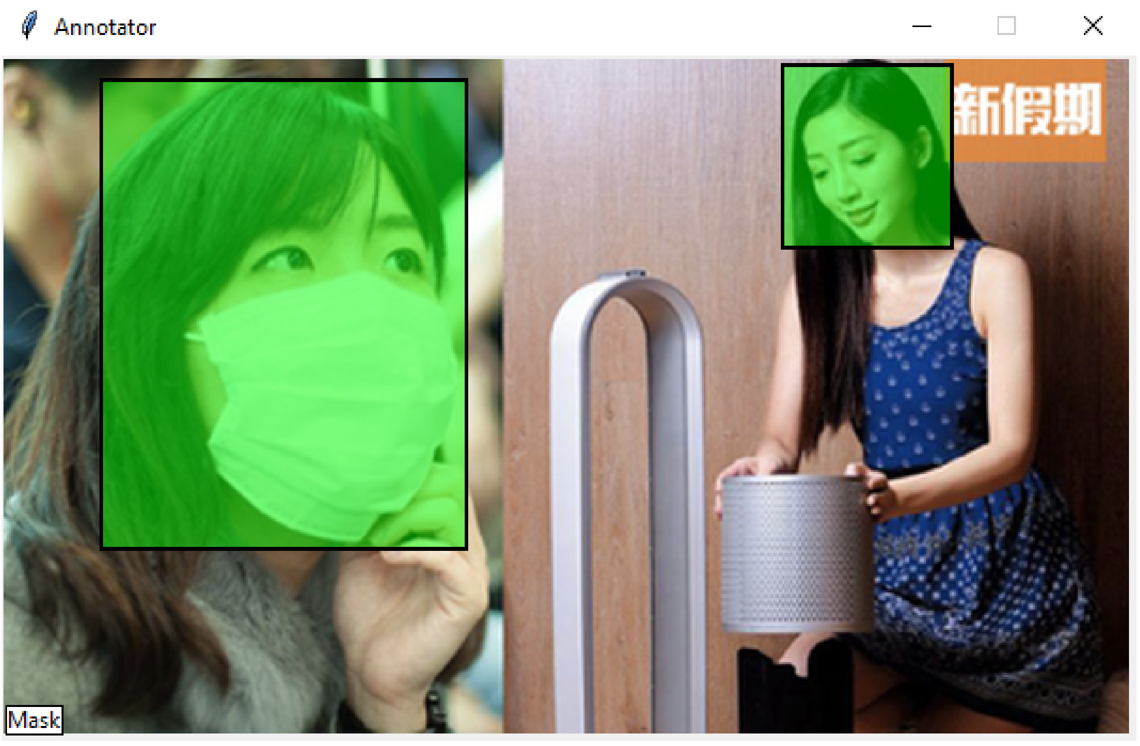
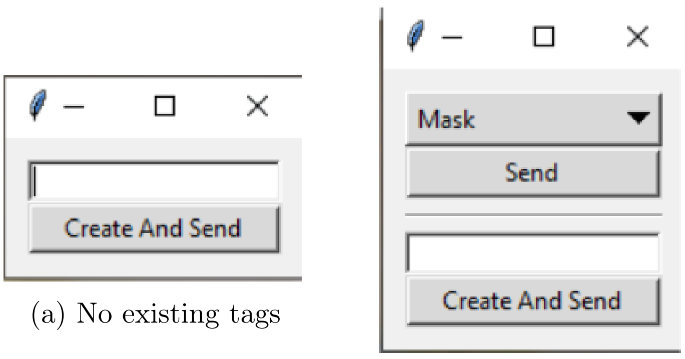


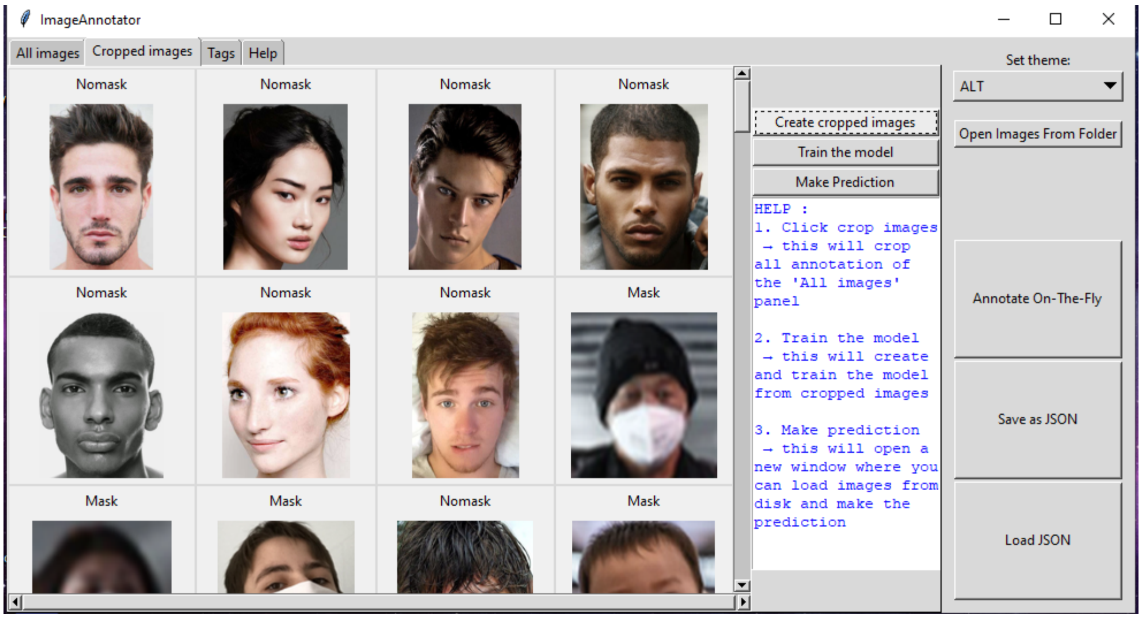
Figure 2: Annotator preview



(a) No existing tags

(b) Scroll menu with tag

Figure 3: Tag attribution

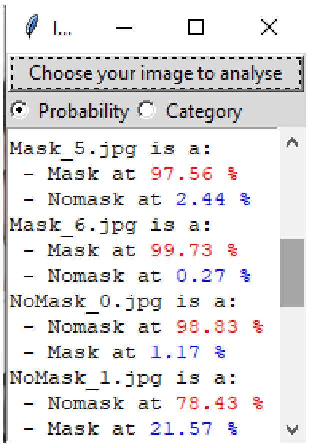


```
HELP :
1. Click crop images
 → this will crop
all annotation of
the 'All images'
panel

2. Train the model
 → this will create
and train the model
from cropped images

3. Make prediction
 → this will open a
new window where you
can load images from
disk and make the
prediction
```

Figure 4: Cropped image panel



```
Mask_5.jpg is a:
 - Mask at 97.56 %
 - Nomask at 2.44 %
Mask_6.jpg is a:
 - Mask at 99.73 %
 - Nomask at 0.27 %
NoMask_0.jpg is a:
 - Nomask at 98.83 %
 - Mask at 1.17 %
NoMask_1.jpg is a:
 - Nomask at 78.43 %
 - Mask at 21.57 %
```

Figure 5: Predictor preview

## 8.3 The cropped images panel

The cropped panel (see 4) is the panel which aims to create the model of neural network by which you will make you prediction.
In this panel, after having annotate the images you want, you click on *Create cropped images* button to crop and resize all annotations. In this panel you will see a miniature of all annotations associated with the name of the tag you gave.
Then you can *Train the model*. This is the core earth of our project, we will explain more about our model decisions in Section 9.
Once the model created, you can click *Make prediction* button and in the new opened window (see 5) you can choose the images on which you want to make a prediction (you can choose multiple files in a folder).

## 8.4 The tag panel

This panel is composed of a grid of existing tags and the *Add tag* button.
You can add a tag via the corresponding button and rename or delete an existing one by right-clicking on it (in this case the rename will impact every image's annotation).

## 8.5 The help panel

The help panel contains a brief description on how our interface works and it is composed of four clickable buttons, each of them showing its related text.

## 8.6 The right panel

The right panel of our interface contains some buttons performing different tasks.
You can modify the theme of the window via the *OptionMenu* under the *"Set theme"* label.
The other buttons:

- Open images From Folder: allows you to load the images from a chosen folder.
- Annotate On-The-Fly: allows you to load a single image and make annotations on it. You can then save these annotations in the folder you want thanks to the button *Save*.
- Load JSON (resp. Save as JSON): allows you to load (resp. save) a JSON file with the annotations of current (resp. previous) session.

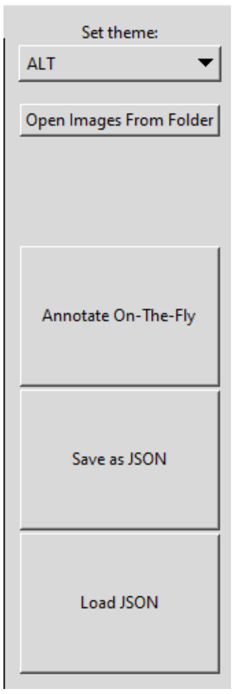


Figure 6: Right panel preview

# 9 Models

Training a neural network revolves around three main objects:

- *Layers*
- *Loss function*
- *Optimizer*

## 9.1 Layers

The core building block of neural networks is the
layer, which is a *"data-processing module that you can think of as a filter for data"*[1].
The function of a layer is to extract the representation of the data which are fed into them. Most of deep learning are characterized by a chain of simple layers.

Listing 2: Layer code example

```
model   = keras.models.Sequential([
    data_augmentation,
    layers.Rescaling(1./255),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.2),
    layers.Flatten(),
    layers.Dense()
])
```

Now let's see the different layers taken in account in our models :

- *Rescaling*: it rescale the pixels value (between 0 and 255) to the [0,1] interval.
- *Conv2D*: is the layer to convolve the image into multiple images. It takes as input tensors of shape and is defined by two key parameters: the *depth of the output feature map* and the *size of the patches extracted from the inputs*.
- *MaxPooling*: is used to extracting window from the input feature maps and outputting the max value of each channel.
- *Dropout*: it set to 0 (= *dropping out*) a number of output features of the layer during training. It is usually used to avoid overfitting. It is usually set between 0.2 and 0.5.
- *Flatten*: is used to flatten the dimension of the image obtained after convolving it.
- *Dense*: it is the hidden layer in which there are the hidden units of the layer.

### 9.2 Loss function and optimizer

The *loss function* defines the feedback signal used for learning, it is necessary to measure the performance on the training data and it is part of the compilation step of the model.
The *optimizer*, which is also located in the compilation step, is a mechanism through which the network will update itself based on the loss function.

## 10 Overfitting and underfitting

A model is underfit when, at the beginning of training, optimization and generalization are correlated. In this case, the network hasn't yet modeled all relevant patterns.
A model starts to overfit when a certain number of iterations are made, generalization stop improving and validation metrics stall and then begin to degrade.
To prevent the overfitting in neural networks usually the following methods are used:

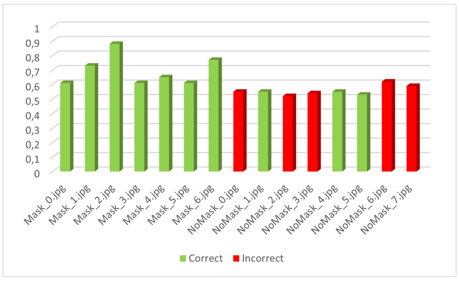
- get more training data
- reduce the capacity of the network
- add weight regularization
- add dropout.

## 11 Our models

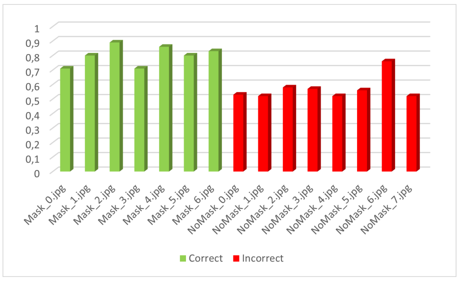
In this section we are going to observe and compare the models we have analyzed to create our neuronal network system (each model has been tested for 3 times).

### 11.1 How to read the following charts

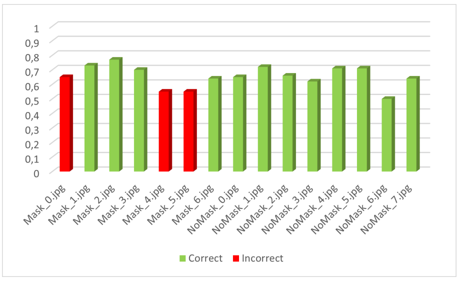
We represent some model result by a bar chart where green bars represent a correct result and red bars a false result to see how a model works.
Every model is then associated with another graph containing an average of the found results. We have information about True Positive (the *Mask* case), False positive, True negative (the *NoMask* case) and False negative. The percentage represent the mean of the statistics that we could deduce from multiple iterations on the same model, a good model is the one where the sum of values of the True column (the green zone) is as big as possible.



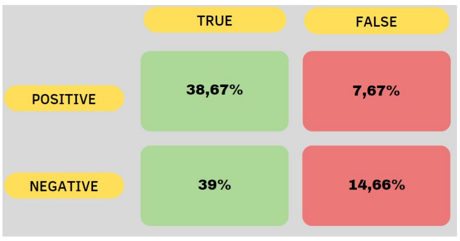
(a) Model 1 - Test 1



(b) Model 1 - Test 2



(c) Model 1 - Test 3



| | TRUE | FALSE |
|---|---|---|
| POSITIVE | 38,67% | 7,67% |
| NEGATIVE | 39% | 14,66% |

(d) Model 1 - Prediction

Figure 7: Scenario 1

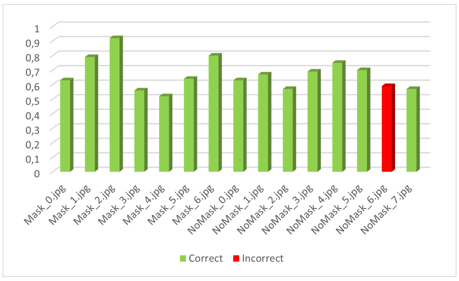
## 11.2 Model 1

Listing 3: Model 1

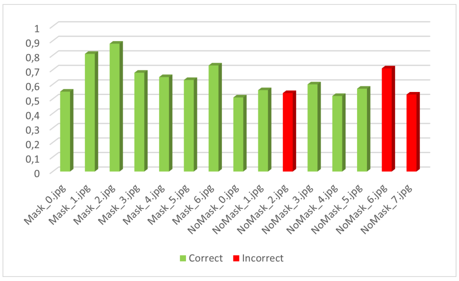
```
model = keras.models.Sequential([
    data_augmentation,
    layers.Rescaling(1./255),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(2),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.2),
    layers.BatchNormalization(),
    layers.Flatten(),
    layers.Dense(num_classes)
])
```

**Model 1 results**

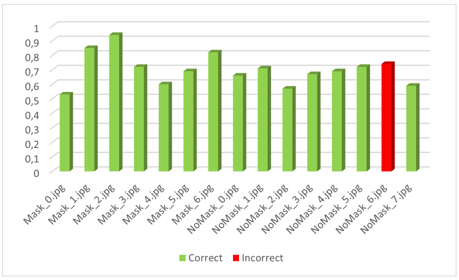
We started setting up three Conv2D layers with three different number of channels and one of the MaxPooling to 2. As we can deduce from three images 7, this model isn't a good model. It can't recognise very well in which image there is a mask and the probability percentage is low.



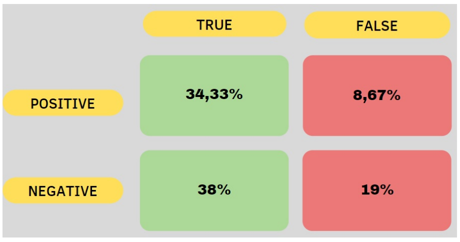
(a) Model 2 - Test 1



(b) Model 2 - Test 2



(c) Model 2 - Test 3



| | TRUE | FALSE |
|---|---|---|
| POSITIVE | 34,33% | 8,67% |
| NEGATIVE | 38% | 19% |

(d) Model 2 - Prediction

Figure 8: Scenario 2

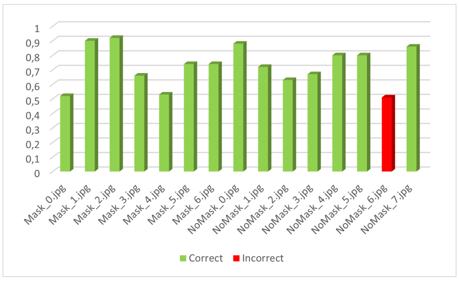
## 11.3 Model 2

Listing 4: Model 2

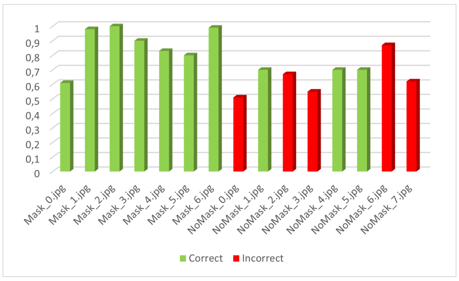
```
model = keras.models.Sequential([
    data_augmentation,
    layers.Rescaling(1./255),
    layers.Conv2D(128, 3, padding='same',activation='relu'),
    layers.MaxPooling2D(2),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.2),
    layers.BatchNormalization(),
    layers.Flatten(),
    layers.Dense(num_classes)
])
```

**Model 2 results**

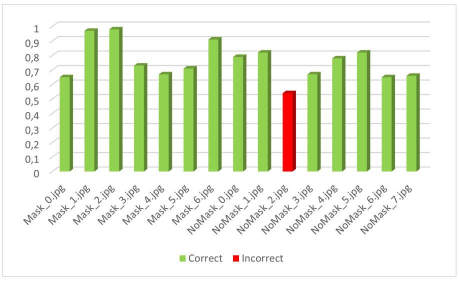
For this model we chose to set up two Conv2D layers (one of them with an higher number of channels) leaving unchanged the other layers.
The result is a better model then the previous one; for example, in Figure 8a, you can see that the recognition of the Mask/Nomask images is correct but the probability percentage is still low.



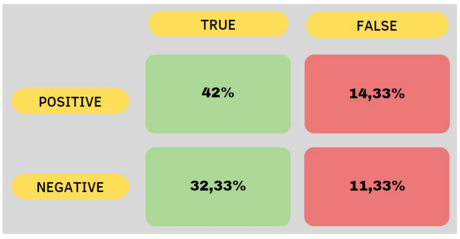
(a) Model 3 - Test 1



(b) Model 3 - Test 2



(c) Model 3 - Test 3



| | TRUE | FALSE |
|---|---|---|
| POSITIVE | 42% | 14,33% |
| NEGATIVE | 32,33% | 11,33% |

(d) Model 3 - Prediction

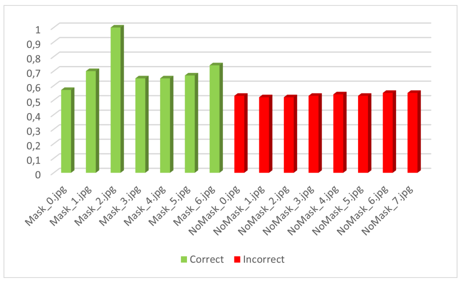
Figure 9: Scenario 3

## 11.4 Model 3

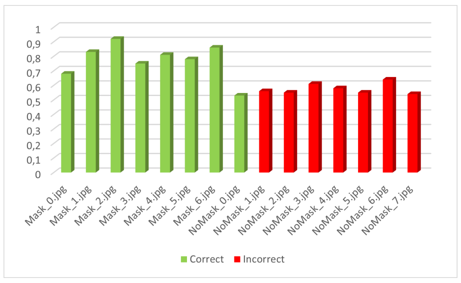
Listing 5: Model 3

```
model = keras.models.Sequential([
    data_augmentation,
    layers.Rescaling(1./255),
    layers.Conv2D(128, 3, padding='same', activation='relu')
        ,
    layers.MaxPooling2D(4),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.4),
    layers.BatchNormalization(),
    layers.Flatten(),
    layers.Dense(num_classes)
])
```

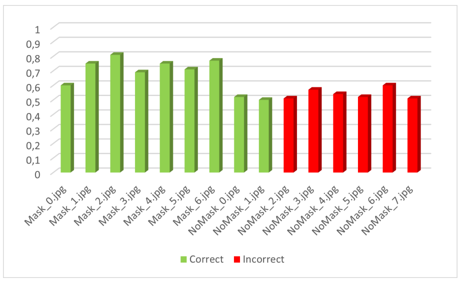
**Results**
With this model we decided to keep the same number of layers, changing some of their parameters, such as the number of channels of Conv2D layers and the parameters of the first MaxPooling and the Dropout. In this case it seems to be better then the second as regards the percentages, but it still remains a model that can be excluded.



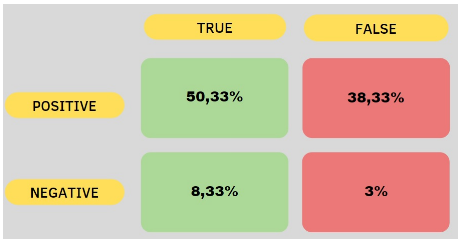
(a) Model 4 - Test 1



(b) Model 4 - Test 2



(c) Model 4 - Test 3



| | TRUE | FALSE |
|---|---|---|
| POSITIVE | 50,33% | 38,33% |
| NEGATIVE | 8,33% | 3% |

(d) Model 4 - Prediction

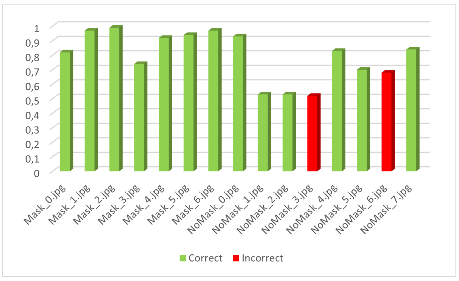
Figure 10: Scenario 4

## 11.5 Model 4

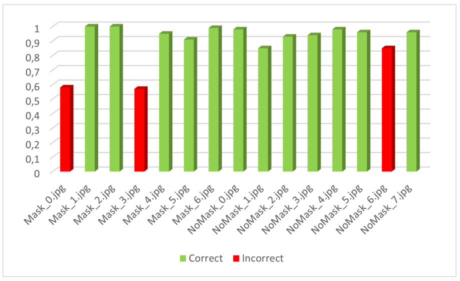
Listing 6: Model 4

```
model = keras.models.Sequential([
    data_augmentation,
    layers.Rescaling(1./255),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(2),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(2),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(128, 3, padding='same', activation='relu')
        ,
    layers.MaxPooling2D(4),
    layers.Dropout(0.2),
    layers.BatchNormalization(),
    layers.Flatten(),
    layers.Dense(num_classes)
])
```

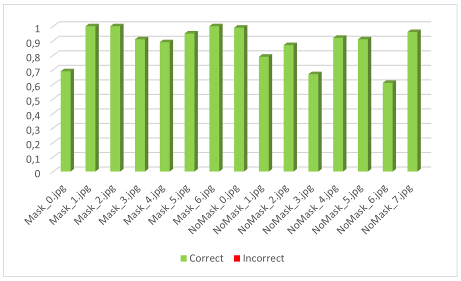
**Results**
The fourth model involves new layers, in particular three Convs2d layers each with different values, ranging to the lowest to the highest. The result is the worst model, it gathers the mask category to almost all images.



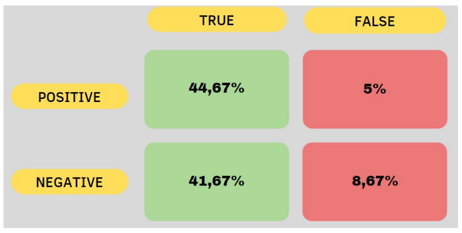
(a) Model 5 - Test 1



(b) Model 5 - Test 2



(c) Model 5 - Test 3



| | TRUE | FALSE |
|---|---|---|
| POSITIVE | 44,67% | 5% |
| NEGATIVE | 41,67% | 8,67% |

(d) Model 5 - Prediction

Figure 11: Scenario 5

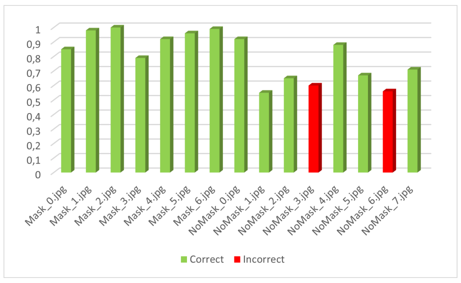
### 11.6 Model 5

Listing 7: Model 5

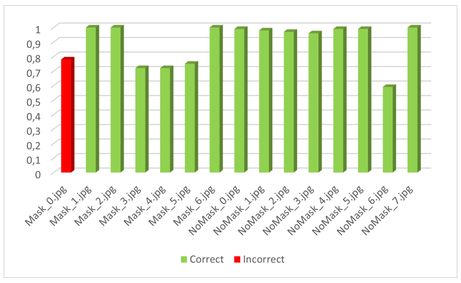
```
model  = keras.models.Sequential([
    data_augmentation,
    layers.Rescaling(1./255),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.2),
    layers.Flatten(),
    layers.Dense(num_classes)
])
```

**Results**

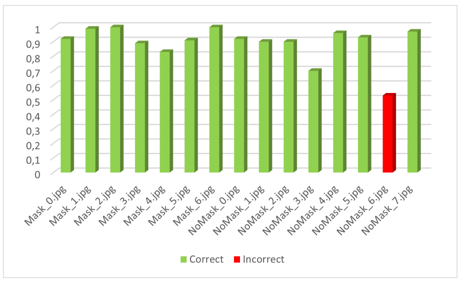
Given the results of the previous model, some layers have been removed and the number of channels controlled by the Conv2d layer has been decreased. We obtained a model that managed to get closer to a good image classification, in fact as you can see, especially in Figure 11c, the matching Mask/Nomask is correct. The probability in some cases reaches even 100%, therefore this approach is the right one.



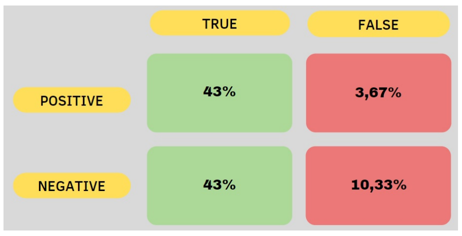
(a) Model 6 - Test 1



(b) Model 6 - Test 2



(c) Model 6 - Test 3



| | TRUE | FALSE |
|---|---|---|
| POSITIVE | 43% | 3,67% |
| NEGATIVE | 43% | 10,33% |

(d) Model 6 - Prediction

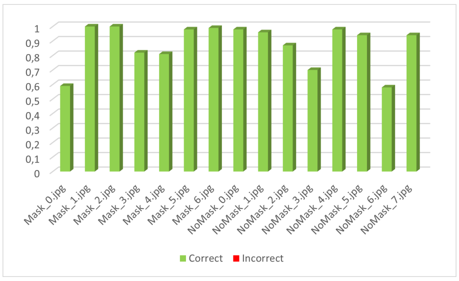
Figure 12: Scenario 6

## 11.7 Model 6

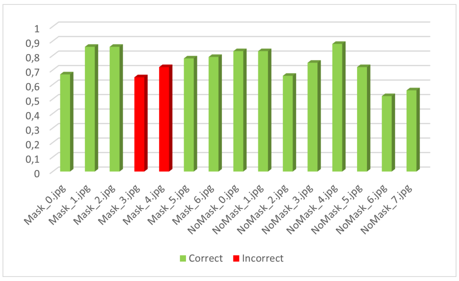
Listing 8: Model 6

```
model  = keras.models.Sequential([
    data_augmentation,
    layers.Rescaling(1./255),
    layers.Conv2D(16, 4, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(16, 4, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 4, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.2),
    layers.Flatten(),
    layers.Dense(num_classes)
])
```

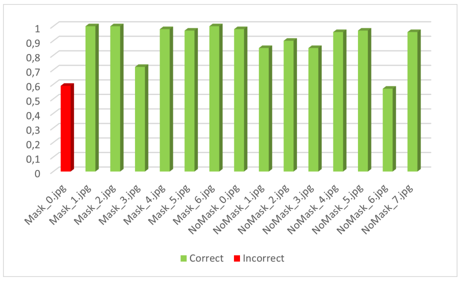
**Results**
Adding a new Conv2D layer the results is pretty the same, but the probability is slightly lower, but in some cases it still reaches 100%.



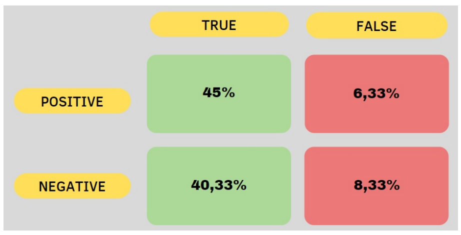
(a) Model 7 - Test 1



(b) Model 7 - Test 2



(c) Model 7 - Test 3



| | TRUE | FALSE |
|---|---|---|
| POSITIVE | 45% | 6,33% |
| NEGATIVE | 40,33% | 8,33% |

(d) Model 7 - Prediction

Figure 13: Scenario 7

## 11.8 Model 7

Listing 9: Model 7

```
model  = keras.models.Sequential([
    data_augmentation,
    layers.Rescaling(1./255),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.2),
    layers.Flatten(),
    layers.Dense(num_classes)
])
```

**Results**
Proceeding in the direction of previous models, we decided to add another Conv2D layer and MaxPooling2D layer to verify if the results would improve further. The results obtained are close to those of the sixth model, but the percentages of probability are slightly lower.
Analyzed all the seventh models, it is possible to choose those that have obtained the best results and these are the fifth, sixth and seventh models.

In particular, the choice fell on these models because they obtained percentages close to or equal to 100% and, in some tests, it correctly classified all the images dividing them correctly between *Mask/Nomask*.
So, for future model training/testing, you could test several Conv2d, but keeping a lower value of channels controlled by the first Conv2D layer argument (in these cases 16).

## 12 Conclusion

Finally, we have a version of our application in which we can easily manipulate and crop the images that interest us and where our intelligence is clearly able to distinguish between a person wearing a mask, and one not wearing it.
To carry out this project, we have therefore, in the first part of the project, created a functional interface to load and annotate images. We learned to use the tkinter library to place labels and menu correctly in panels and in the end we have a nice software to use. The the shapely library has been very useful and fast to easily check the area and converge different shapes such as rectangular boxes.
In the second part of the project, we realized a learning model, trained it through multiple experiments allowing us to define an image predictor which, thanks to the bounding box of the face, will classify the image in the categories *Mask/NoMask*.
To compile the learning model, we use the the *keras* framework and we tested a lot of different model (seven of them are cited in this report). We draw from these different tests very satisfactory results, by carrying out multiple tests for each model, we conclude that the best of our models are model 5 and 7, with 44.7% of True Positive (resp. 45.0) and 41.7% of True Negative (resp. 40.3). On the contrary, some of the worst models we found are model 4 (a lot of false predictions) and model 2 (a very slow model).
The majority of the basic features have been implemented and work well, but it could be improved further by trying to find even more advanced models. Moreover, it is interesting to think that we could even consider other applications than just mask detection.
This work is finally a project that allowed us to realize how interesting it is to use the notion of neural network to identify, here in this case, a nowadays problem, the wearing of the mask and we have finally realized how complicated can be the creation of an artificial intelligence that understands what a human can detect immediately and instinctively.

## 13 Bibliography

## References

[1] F. Chollet. *Deep Learning with Python.* 2018.