Face detection using CNN PROJ-H402 Computing Project

Franck TROUILLEZ, *Student, Ecole Polytechnique de Bruxelles,*

Abstract—Face detection is one of the main topics in AI. It is not possible to get good results using the old fashioned AI. The new candidate for this kind of problems is the deep learning. Using CNN, it is possible for an algorithm to generalize results from a restricted dataset. The goal of this project is to draw boxes around faces on images. Tensorflow on Python allows such tools. Implementing that kind of IA is thus feasible and can be used in order to detect faces on all type of images. The network proposed get pretty good results. It succeeds on detecting all the visible faces on a picture. However, this algorithm associate the presence of a face with the skin color. Thus, a post-processing is required in order to get better results. Compared to the state of the art, this algorithm is not that good but it shows the power of a simple CNN for such a difficult problem.

Index Terms—Face detection, CNN, deep learning, Tensorflow, AI, Python.

1 Introduction

This project has the goal to realise a face detector using CNN. Indeed, this kind of tools is nowadays very useful. It allows to automatically detect faces and, consequently, people. Deep learning will be a useful tool in order to implement such an algorithm. Indeed, this kind of AI trains on a certain dataset, then try to generalize the results in order to work on reality. This is exactly what is done in this project. Moreover, this project intends to show the power of deep learning. The model used is not that complex but allows to get pretty good results. It shows that these kind of AIs are very useful now and can be used in many fields, such as the face detection.

2 Methods

The way to build this project is sequential. First, an algorithm has to be chosen. Then, the way the algorithm will be implemented has to be discussed. The implementation itself and the way the dataset is used will also be explained. To finish, the way to use it on every picture will also be discussed.

2.1 Algorithm choice

The idea is to find a way to detect faces. The algorithm used should receive a RGB picture as input, and produce a mask with boxes around the face areas as output. The best idea is to use AI. However, the term AI gathers a lot of different types of algorithms. Knowing that the algorithm should work on every picture it receives, it has to generalise data. The perfect candidate for this kind of problem is deep learning. Indeed, it can train on its own using train data and then, try to generalise on pictures it has never seen.

Knowing that, the next question is about the supervision of the learning. The problem of face detection is a popular challenge. Thus, there exists some datasets with input-output pairs, which allows to use supervised learning for this project. The dataset used is *WIDER FACE, A Face Detection Benchmark, by Motaz Saad, uploaded on Kaggle* [1]. It gathers more than 10 000 pictures of people in different situations and events. It is thus the perfect candidate for an algorithm which will try to find people on every picture it gets.

2.2 Tools used

As said before, the algorithm chosen is the deep learning with supervised training. In order to implement that, Python will be used. Indeed, the library Tensorflow, which allows to easily use machine learning, is implemented in Python. It allows to build models very efficiently using C++ routines. Moreover, it includes the Keras API, which allows to train neural networks.

In order to write this code, the most interactive way to implement it is by using a notebook. Indeed, it allows to execute some parts of the code, and to add some texts in the script. Knowing that Google Colab gives the opportunity to use GPUs, the choice of using Google Colab notebooks was the obvious choice.

2.3 Implementation

First, the idea is to build a model. The input has to be a RGB picture and the output is a probability map where each pixel has a value between 0 and 1, corresponding on whether this pixel belongs to a face or not. Before getting into the model, the dataset used has been analysed to see what info are given. First of all, the pictures are all given with a target, corresponding to boxes where the faces are localised. The first big issue at this point is that the pictures of the dataset do not have the same size. Indeed, the resolution of every pictures is different, it is thus impossible to train a model in this situation.

2.3.1 Build a new dataset

The best idea to handle this kind of issue is to create a new dataset, based on the old one. The new dataset has to manage the problem faced with the original one. Thus, the pictures must be of the same size. Consequently, a defined

size had to be chosen. In the case of this project, the input pictures have a width of 256 pixels and a height of 170 pixels.

Now that the shape of the pictures is defined, the way to convert the pictures of the original dataset to the new one has to be chosen. Indeed, there exists different ways to convert them.

One idea could be to compress the pictures to the size chosen. The images could be compressed until one of its 2 first dimensions fit with the defined size chosen, and then, truncate the excess pixels. Another way to use the same idea could be to compress the picture until the original picture fit completely into the new shape. The missing pixels could be padded with 0's to fill the entire picture.

However, the simplest idea is chosen here. The original pictures are divided into sub-pictures of 256x170 pixels. For example, a picture with a shape of 1024x680 is converted to 16 pictures with a shape of 256x170.

Although it seems easy to just divide a picture into sub-pictures, the targets also need to be taken into account. Indeed, the targets for the original pictures were boxes giving a position along with a width and height. In order to convert this boxes into the corresponding boxes for the sub-pictures, the easiest way is to convert these targets given as boxes into a mask, showing where the faces are localised. Then, the mask is split in exactly the same way as the original picture. Finally, the sub-masks are converted back to boxes, to get targets corresponding to the sub pictures.

2.3.2 Model creation

Now that the dataset is well defined, with pictures of the same size, meaning that it can fit into the same model, the model itself can be created. The input layer is thus a convolutional layer with a shape corresponding to the input one.

An other important point is the shape the model outputs. Indeed, the algorithm does not need to automatically output a probability map of the same size as the input. The result we need do not need to be accurate to the pixel. Moreover, having a too high resolution for the output can greatly increase the time spent to train the model. Thus, the output of the model has a shape twice as small than the input one, corresponding to a shape of 128x85.

The hidden layers have to be chosen now, in order to generate a strong model for the face detection. First of all, knowing that a map should be returned, the localisation is very important. This information can not be lost. Thus, the model used will fully be composed of convolutional layers. Indeed, these layers allow to connect neurons between 2 layers, while keeping the localisation information. In order to get a sufficiently accurate model, 6 convolutional layers will be used, using 3x3 kernels. In order to compress the output map, a max pooling layer is also inserted. The model is described below.

- Convolutional layer using 32 filters with a 3x3 kernel, corresponding to the input layer.
- Convolutional layer using 64 filters with a 3x3 kernel.
- Convolutional layer using 128 filters with a 3x3 kernel.
- Max pooling layer using a 2x2 kernel.

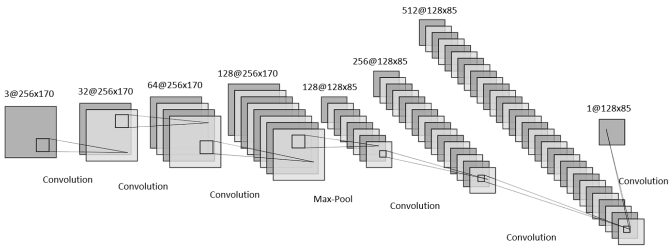

Figure 1. Convolutional neural network

- Convolutional layer using 256 filters with a 3x3 kernel.
- Convolutional layer using 512 filters with a 3x3 kernel.
- Convolutional layer using 1 filter with a 3x3 kernel, corresponding to the output layer

It can also be visualized on Figure 1.

These layers needs to be activated using functions. For all the layers, the *relu* activation function is used. Indeed, it is the simplest one, which speed up the training of this simple model. However, the output layer has a different activation function. Indeed, as said before, the aim is to get a probability map, showing whether or not the pixel concerned belongs to a face. In order to get values between 0 and 1, the *sigmoid* activation function is applied.

2.3.3 Application of the model on pictures

Having a model working on 256x170 pictures is fine, but the real goal is to draw boxes around the faces of every picture the algorithm could receive, regardless of size.

First of all, the inputs have to be extended in order to take pictures of every possible size. The input of the model itself can not be modified. Two cases have to be distinguished.

The first one is when the given input has a lower resolution than the model input one. In this case, the model can work on this picture by padding the missing pixels with 0's. When, the model outputs the probability map, the pixels corresponding to the padding ones are truncated, in order to have an output corresponding to the initial output.

The second case occurs when pictures given are bigger than the training pictures. Multiple ways exist in order to handle such issues.

One idea could be to compress the image, until it fits into the 256x170 shape. Then, the model runs on the compressed picture and returns a mask corresponding to this compressed version of the input. This idea can be very bad, since some details are surely going to be lost when compressing the image.

Another idea could be to parse the input picture with a window of a size corresponding to the model input one. This way, a bigger map can be constructed, by placing the different outputs given by the models in the corresponding area.

The solution used is a combination of both idea cited before. Before trying to parse the picture, the resolution of the picture is checked. Indeed, the model is trained to

recognise faces on pictures of 256x170 pixels. Thus, having a face bigger than this size could be very bad for the model, since it does not have any context to find that it is in the middle of a face for example. In order to avoid that, the idea is to compress the input picture if it is way too big. Indeed, the model can parse the image but doing it on too high resolution images could lead to incorrect results. Once the picture is in a range of acceptable size, the picture is parsed using a window of the model input one. In order to increase the accuracy of the algorithm, some overlap is added. This way, the majority of the pixels are analysed multiple times by the model. For the pixels passed multiple times in the model, the mean of these values is taken. Once the whole picture is parsed, the result is a probability map of the whole picture where every pixel gives the probability of being a part of a face.

Now that the algorithm can take pictures regardless of size, an other issue is that the model does not return a mask, but a probability map where every pixel has a value between 0 and 1, according to whether or not there is a face at this place. Thus, it should try to first generate a mask.

A good approximation could be to simply put a threshold on this map, selecting only the pixels with a value higher than 0.95 for example. However, the model is not perfect. Indeed, some noise will certainly be found in these masks. In order to remove this noise, the mask will be filtered using an erosion operation, which will take the local minimum. It will remove the noise, corresponding to isolated pixels on the mask. Then, a dilation is applied in order to get back the size of the face. Here, the opening operation, corresponding to an erosion followed by a dilation, is not used because the structuring element is not the same. Once the mask is well generated, boxes are generated around these face areas.

The last thing that can be managed at this state is the fact that a same face can be detected as 2 or more smaller faces. In order to handle this issue, an idea could be to parse every faces, and check if a face is too close from it, compared to a relative scale based on the biggest face detected on the picture. Consequently, it considers that the results are better when all the faces have nearly the same size. Images with faces of different sizes will be the most difficult ones. In case of such pictures, the biggest faces are considered.

3 Results

As explained before, some post-processing can be applied in order to draw boxes around faces. In order to see the improvement between before and after the last step of post-processing, which consists of merging the potential sub-faces, the results of both situations will be reported.

3.1 Before merging sub-faces

Once everything is defined, the algorithm can be tested. First of all, the model is evaluated using the test set from the dataset. In order to have a good idea about the efficiency of the model, the Matthews correlation coefficient (MCC) is used. It allows to compare algorithm used for segmentation. The evaluation of the model on the test set gives a MCC of 0.37.

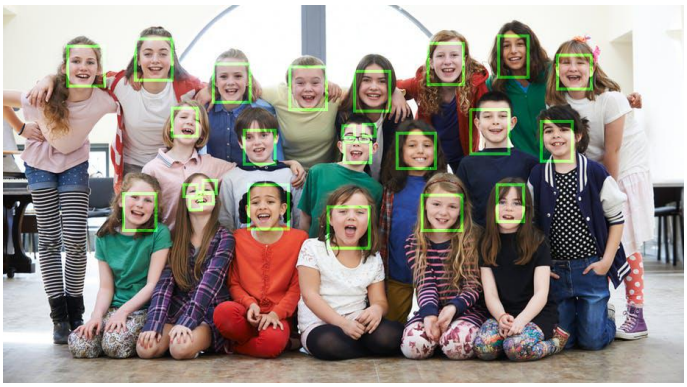

Figure 2. Example 1 before merging sub-faces [4]

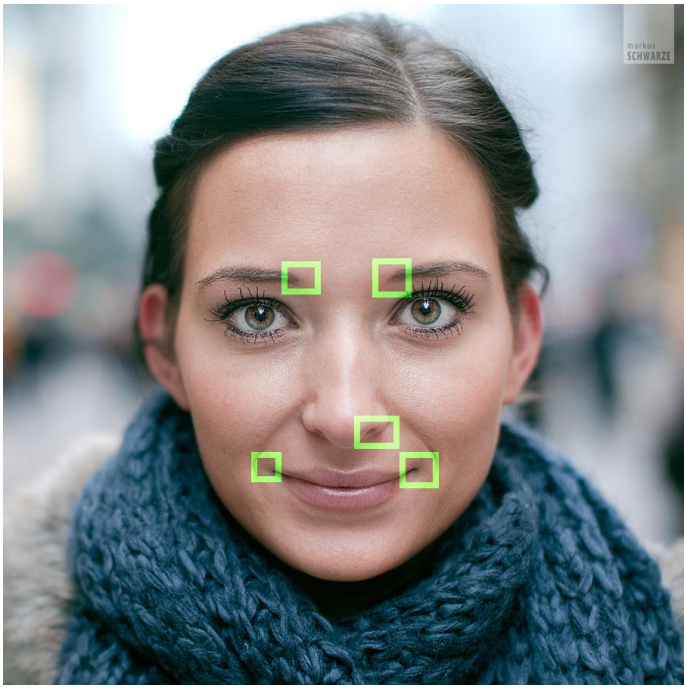

Figure 3. Example 2 before merging sub-faces [5]

In order to have a more concrete idea, the algorithm is applied on different pictures of different sizes. The results of these tests are showed on Figure 2, Figure 3 and Figure 4. .

Actually, the best way in order to evaluate the algorithm is to compare the model with state of art. Two algorithms will be taken as references. First, the frontal face detection algorithm of *OpenCV* [2] will be tested. Secondly, the *Multi-Task Cascaded Convolutional Neural Network* [3] will also be evaluated. The results of the evaluation on the test set are reported in the Table 1.

These 2 models are also applied on the same pictures

Models	**MCC mean**
This algorithm	0.37
OpenCV	0.09
MTCNN	0.26

Table 1
Comparison of different models based on MCC

Figure 4. Example 3 before merging sub-faces [6]

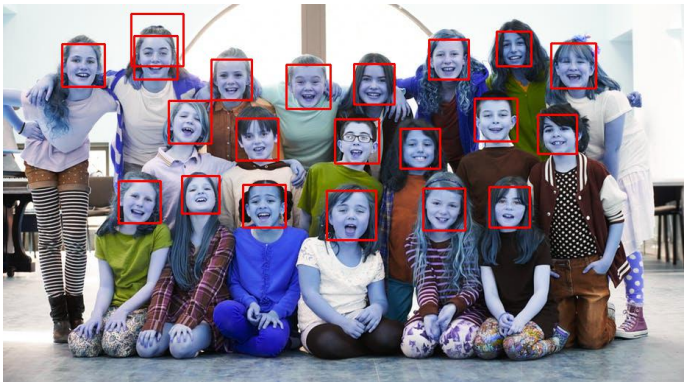

Figure 5. Example 1 with OpenCV [4]

as the algorithm described in this report. The results of these tests are also showed on Figure 5, Figure 6, Figure 7, Figure 8, Figure 9 and Figure 10.

3.2 After merging sub-faces

Now, the results were the potential sub-faces are merged can be showed. The model does not change thus, the MCC will not be used. However, the tests on the same pictures are displayed on Figure 11, Figure 12 and Figure 13.

Figure 6. Example 1 with MTCNN [4]

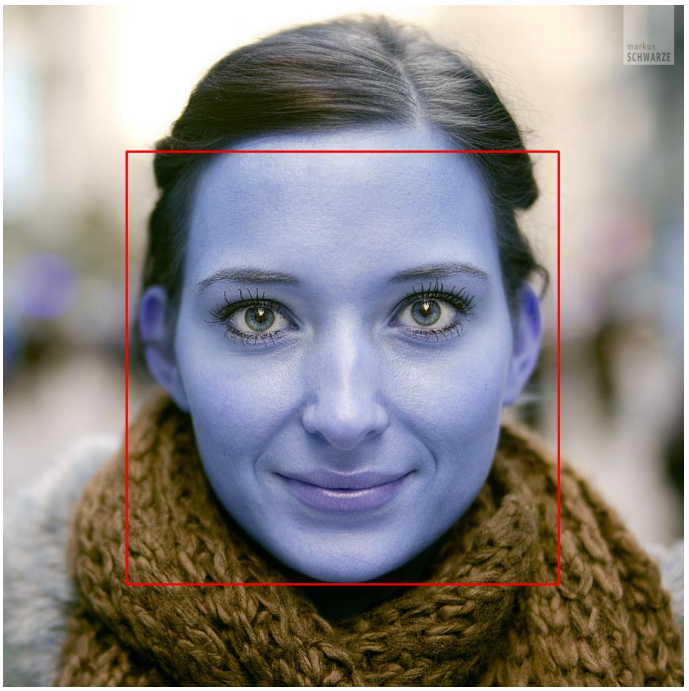

Figure 7. Example 2 with OpenCV [5]

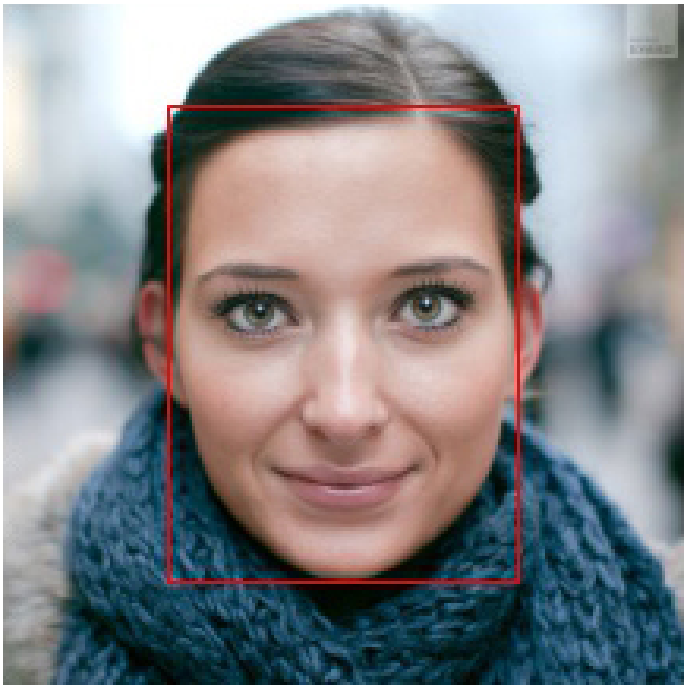

Figure 8. Example 2 with MTCNN [5]

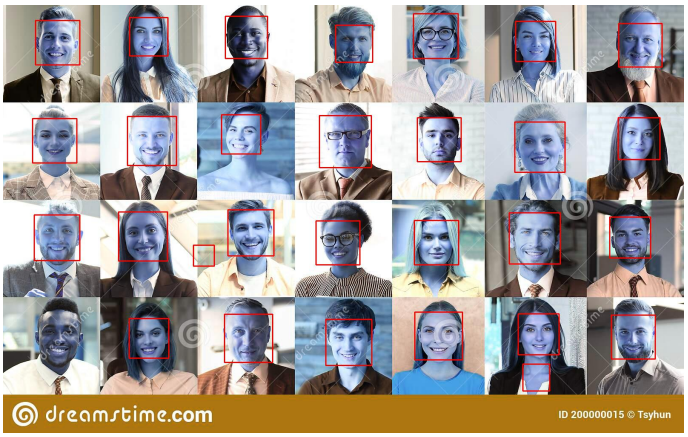

Figure 9. Example 3 with OpenCV [6]

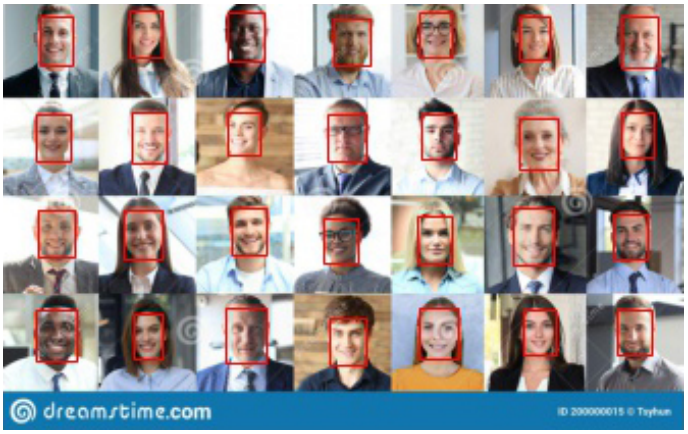

Figure 10. Example 3 with MTCNN [6]

4 Discussion

First of all, the MCC is a metric where a perfect algorithm would have a score of 1. Consequently, the Table 1 tells that the algorithm described in this project is better than the state of art. However, this conclusion can not be taken. Indeed, the different models are tested on the test set of the dataset, where pictures have a shape of 256x170. The given algorithm does segmentation, while the other ones do directly detection. This means that if a face is split among multiple pictures, only the segmentation model will work, since the other two will try to find a face on each picture separately. These results can thus not be taken into account. They are only useful to say that this algorithm do segmentation and, in these circumstances, works good.

Figure 11. Example 1 after merging sub-faces [4]

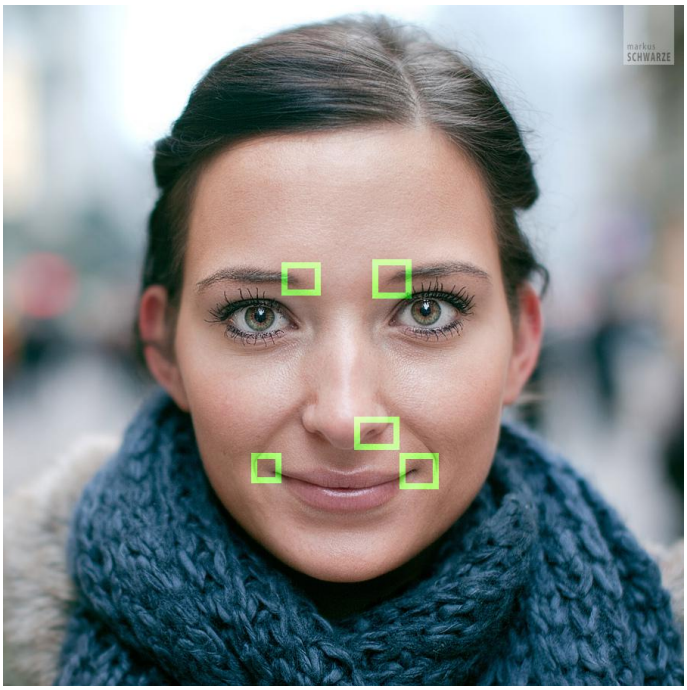

Figure 12. Example 2 after merging sub-faces [5]

Figure 13. Example 3 after merging sub-faces [6]

In order to have a better estimation of the efficiency of this algorithm, the best way is to check concrete cases, such as the different examples given before.

As expected, the results are not as good and as polyvalent as the other algorithms. However, for such a simple network, the results are quite good. Indeed, as showed before, the faces are well detected on most pictures. However, it has some problems when there is few big faces. Indeed, the model is not trained to recognise such big faces. The dataset has mostly pictures with faces which fits in 256x170 pictures. It is thus obvious that this algorithm struggles to detect such cases. Figure 12 shows exactly the problem described. One other interesting thing is that, as expected, the colour of the skin is a big feature in order to detect faces. Unfortunately,

this algorithm has difficulties to detect faces of people with darker skins, as shown on Figure 13. The boxes are very small compared to those of other faces.

As said before, besides these problems, the algorithm is quite good since it detects faces in a certain range of pictures. It gives good results and can in some cases be better than the *OpenCV* one, as it can be seen on Example 1, with Figure 5 and Figure 11, and on Example 3, with Figure 9 and Figure 13. *MTCNN* is still better than this, even if it has detected a false face in Figure 6. Indeed, it also uses CNN but is optimised for that kind of application.

One way to handle the issue with big faces could be to add a new input and output to the model. Indeed, one idea could be to have an estimation of the area of the face for example. The model would thus be able to evaluate the size of the face. Consequently, it could know the size of faces and then, try to find faces with the estimated size.

5 Conclusion

This project has got very good results for the time spent in its realisation. This also shows the power of deep learning in such applications. Using a simple CNN can already provide results. However, as expected, it is not at the level of state of art. Indeed, the *MTCNN* algorithm is more polyvalent and has better results in general. Compared to the *OpenCV*, it is not as versatile but in pictures with not too big faces, it is at the same level. This is encouraging. The fact that such a powerful tool is available gives a lot of opportunities. It can be apply in many fields, other than face detection. To conclude, at the end of this project, it can be said that this algorithm for face detection can be used with pictures where people are usually not to close from the camera. It can efficiently detect faces on that kind of pictures.

References

[1] Motaz Saad. *WIDER FACE, A Face Detection Benchmark*, Kaggle, updated 2 years ago. https://www.kaggle.com/mksaad/wider-face-a-face-detection-benchmark
[2] Open CV. https://opencv.org/
[3] Zhang, K., Zhang, Z., Li, Z., and Qiao, Y. (2016). *Joint face detection and alignment using multitask cascaded convolutional networks*. IEEE Signal Processing Letters, 23(10):1499–1503. https://pypi.org/project/mtcnn/#zhang2016
[4] Class photo example https://images.theconversation.com/files/269359/original/file-20190415-147525-1k07fcr.jpg?ixlib=rb-1.1.0&rect=19%2C142%2C944%2C523&q=45&auto=format&w=926&fit=clip
[5] Portrait example https://www.doretdespa.com/wp-content/uploads/2015/04/portrait-sigma-50mm-f14-hsm-canon-eos-5d-mk2-face-Favimcom-473053_zpsd55e8d8e.jpg
[6] Faces mosaic example https://thumbs.dreamstime.com/z/happy-group-multiethnic-business-people-men-women-different-young-old-people-group-headshots-collage-multicultural-200000015.jpg