Group B
Team 2

Piotr Bouta
Michał Januszewski
Kamil Olszewski
Albert Rasiński

Detecting faces in images using NN and OpenCV.

1. Motivation & Objectives

Face recognition and detection is a very popular topic nowadays. It is implemented in a vast array of electronic devices and softwares. The usage of neural networks in the process of recognition speeds up the process as well as improves its precision. Writing face detection code without using neural networks would be very difficult and complicated, and it would be hard to get satisfactory results. Preparing a neural network with an appropriate library is much easier and can produce very good results. Our motivation for taking this topic of the project was that it has many applications and could be useful in our future projects, also it would be a good learning experience due to many sources and scientific articles on the Internet.

The main objective of our project is to implement a Convolutional Neural Network that is capable of face recognition. Initial goal of our team is to learn the basics of neural networks and make an accurate algorithm for face detection.

2. Algorithm

We decided to work in Python (to be able to work together at the same time the program has been made in Google Colab) and that we will use the Keras library in our project. Keras is a deep learning API, that helps programmers to write easier code. Keras is running on top of the Tensorflow, which is an open-source machine learning platform

At the beginning we started with searching for a dataset. As a result, we managed to make a testing dataset consisting of 36823 images, a validation dataset consisting of 11670 images and testing dataset consisting of 9253 images. All the datasets have close to 50% quantity of face image. Samples of images are presented below:

Face images:

No face images:

The overall program consist of below steps:

1. Loading images (testing and validation datasets) from Google Drive.
2. Adjusting parameters of images and preparing augmentation parameters
3. Preparation of the structure of the convolutional neural network model
4. Defining the loss function, optimizer and the metrics
5. Training and evaluation of the model
6. Testing model
7. Visualization of the parameters of the CNN model
8. Predicting the given image

The images from the testing database were augmented before every epoch, the main reason for this action was not the small size of the dataset but the fact that every image of face was similar in terms of rotation of faces as well as most faces were similar in size (photo was taken from similar distance). All the photos are grouped in batches of 64 and resized to 128x128px.

In the process of augmentation many arguments were used such as: resizing images, rotation of images, changing width or height of image, shearing the images by angle, changing the zoom of images and horizontal flip.

Example processed image with augmentation process:

Our model has the following structure:

Model: "sequential"

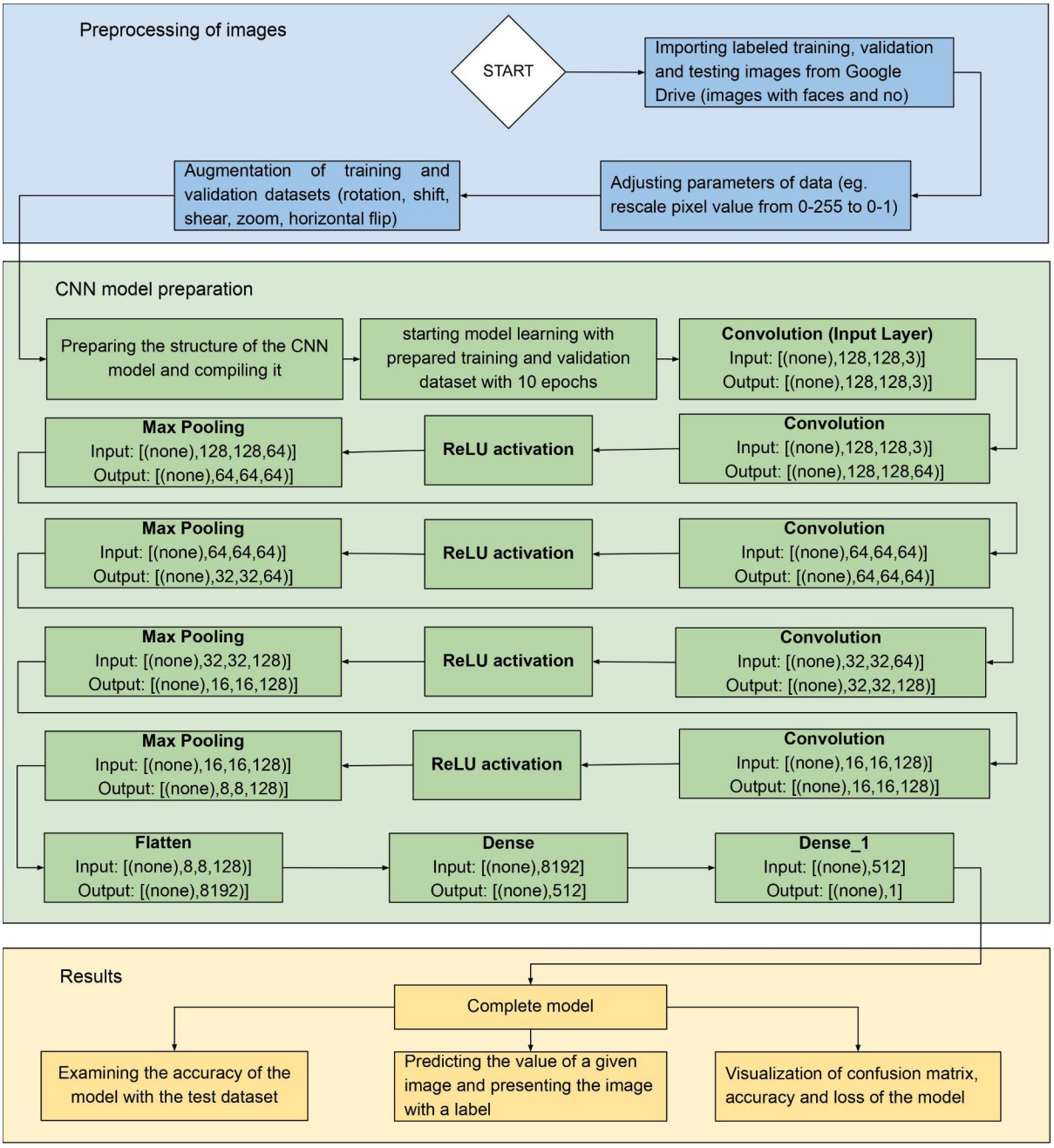
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 64)	1792
max_pooling2d (MaxPooling2D)	(None, 64, 64, 64)	0
conv2d_1 (Conv2D)	(None, 64, 64, 64)	36928
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 64)	0
conv2d_2 (Conv2D)	(None, 32, 32, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 128)	0
conv2d_3 (Conv2D)	(None, 16, 16, 128)	147584
max_pooling2d_3 (MaxPooling2D)	(None, 8, 8, 128)	0
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 512)	4194816
dense_1 (Dense)	(None, 1)	513

Total params: 4,455,489

Trainable params: 4,455,489

Non-trainable params: 0

Our program has the following structure:



Conv2D is a 2D Convolution Layer that is randomly initialized to some distribution kernels/filters that are moved across the image. It takes as the arguments: number of filters (in our case it's 64 or 128), size of filters (in our case it's 3x3) and activation function (in our case it's relu - rectified linear activation function).

MaxPooling2D down-sample an input image, reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions. (in our case it's (2,2) that means the image after this operation gets two times smaller)

Flatten is used to flatten the input. Changes two dimensional image into one dimensional vector.

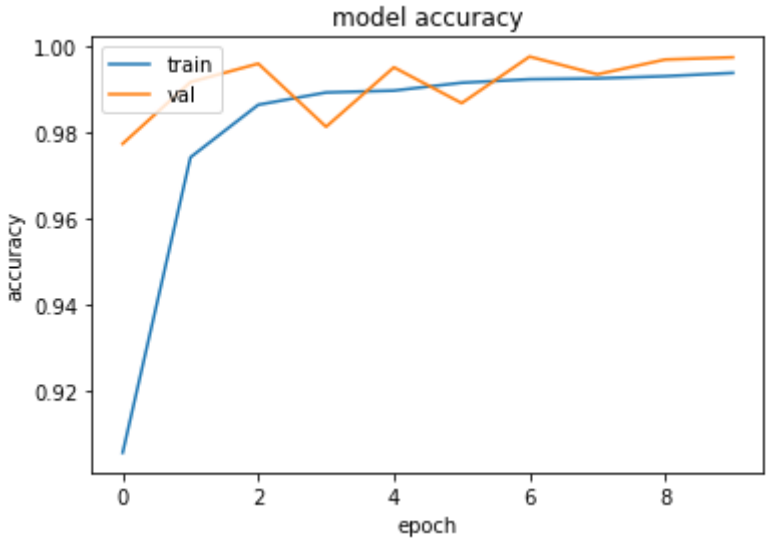
Dense layer feeds all outputs from the previous layer to all its neurons, each neuron providing one output to the next layer. (In our case we have 512 neurons and the activation

function is relu in second to the last layer and 1 output neuron of the whole model in the last layer with sigmoid activation function)

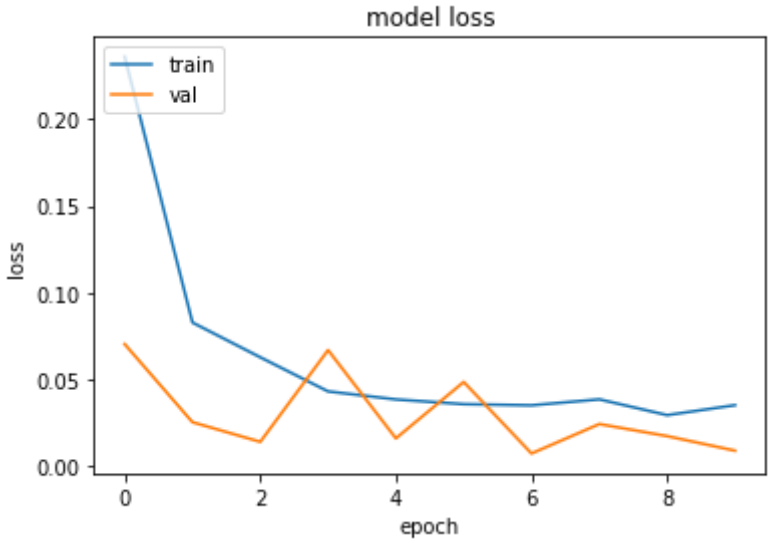
Our model was compiled with binary_crossentropy loss function, RMSprop(lr=0.001) optimizer and binary_accuracy metrics. Loss function is necessary during model compilation in order to measure the error of the model. There are plenty of various ways the error can be calculated, such as mean_squared_error or mean_absolute_error. The loss function that was chosen for the model is binary_crossentropy since the model is based on polar values of labels: 1 and 0. They represent the states of either presence of face in the image or lack of it.
Just like with loss functions there are plenty of optimizers and each optimizer has its own inside parameters. Optimizers are responsible for adjusting weights based on the difference between the prediction and the loss function. The most commonly used optimizers are SGD, RMSprop Adam, Adamax. The optimizer that was chosen for the model compilation process is RMSprop. The value of the parameter called learning_rate of the optimizer was experimentally changed but in the end it was left at the default value equal to 0.001.
The last argument that is used in the process of compiling the model is metrics. The preferred outcome of the compilation is high total accuracy. Therefore, binary_accuracy was chosen as the metric for the compilation.

The teaching of the neural network takes place in stages determined by the number of epochs (the number that learning algorithm will work through the entire training dataset). We tried different numbers of epochs during modelling our process. Eventually after a few tries we decided to choose epoch=10 as with this parameter we were able to obtain very good accuracy in a decently short time. When we tried adding more epochs the accuracy was not better but computation was taking longer.

Model accuracy (the number of classifications a model correctly predicts divided by the total number of predictions made):



Model loss (the sum of errors made for each example in training or validation sets):



The accuracy of our model in every epoch is presented below:

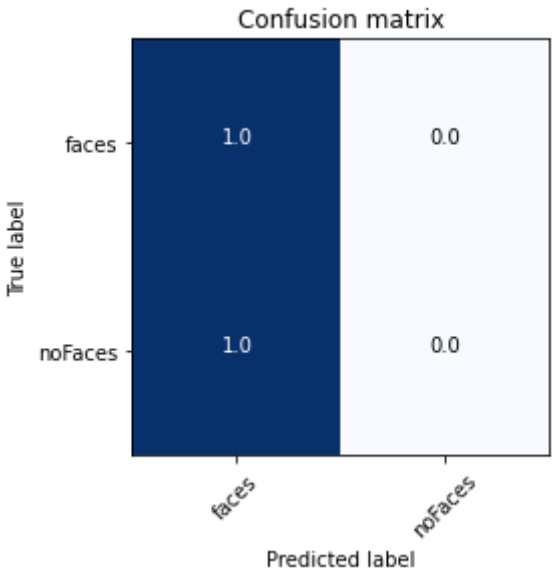
Epoch	time [s]	loss	accuracy	val_los	val_accuracy
1	13330	0,4013	0,8203	0,0702	0,9773
2	315	0,0926	0,9698	0,0250	0,9916
3	315	0,0498	0,9861	0,0137	0,9959
4	311	0,0414	0,9894	0,0667	0,9812
5	311	0,0405	0,9891	0,0157	0,9950
6	311	0,0322	0,9918	0,0483	0,9867
7	317	0,0319	0,9924	0,0069	0,9975
8	317	0,0426	0,9915	0,0240	0,9934
9	314	0,0303	0,9930	0,0170	0,9968
10	313	0,0449	0,9929	0,0086	0,9973

As the end result, our program achieved very good accuracy of above 99% on training and validation datasets. Loss value is also low for both datasets. We consider this outcome to be satisfactory and it proves our program is working properly.

Next, the model was evaluated with the test dataset. The results we have achieved are as shown below:

- accuracy - 99,91%
- loss - 0,0088

Those results are enormously satisfying. The model we have managed to prepare is incredibly accurate.
Confusion matrix of our model represents itself as:

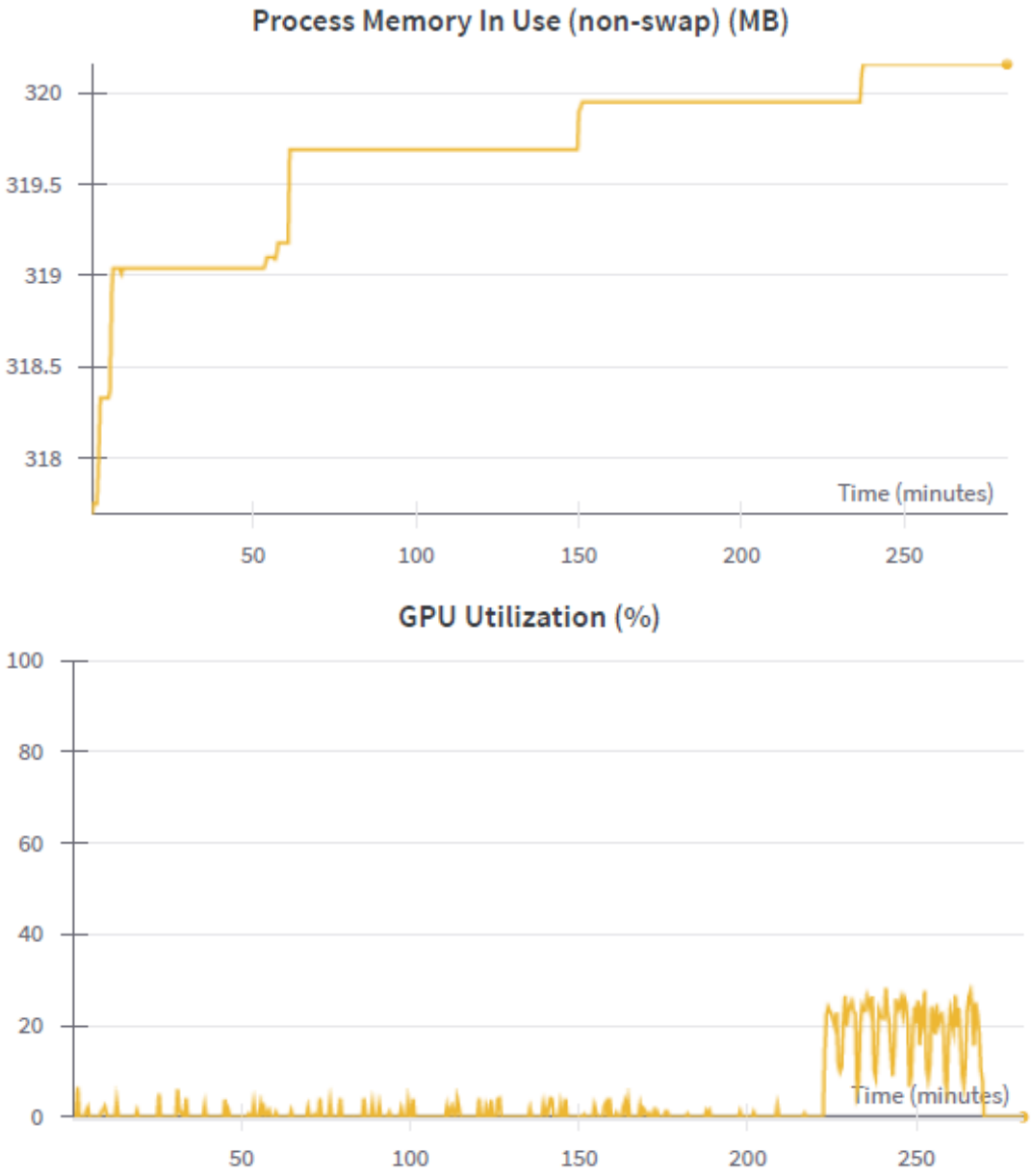


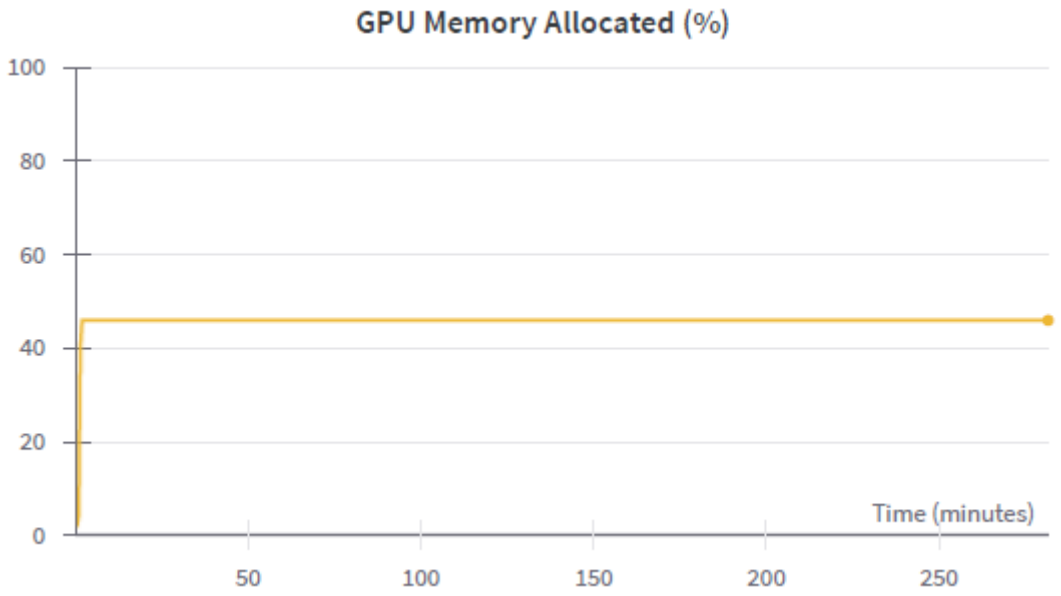
Final stage is preparing an interface for the user so he could give a photo to the model and predict the results. Output information is represented as an image with a label in the upper left corner. Here are few example results:

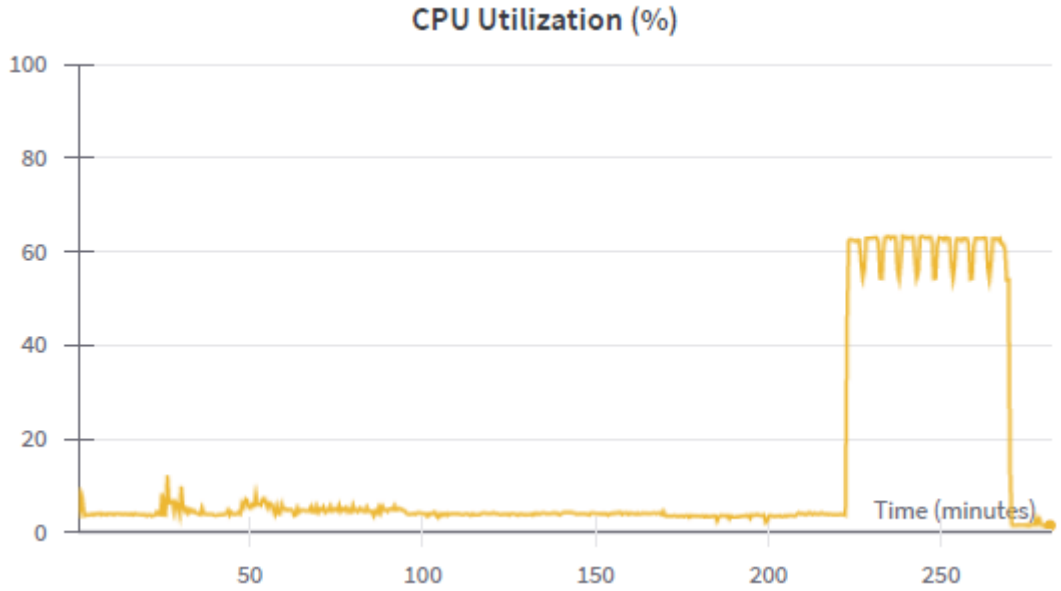
3. Computational Complexity
   The time needed to fit the model in each epoch is shown in the 2nd column in the table in the previous section. The first epoch needs the most time to compute all the data and it takes up to 3,5 hours. Remaining epochs take up to slightly more than five minutes.
   Overall, training the model takes about 4,5 hours on GPU from Google Colab service.

   All the computer load analysis was gained through wandb.

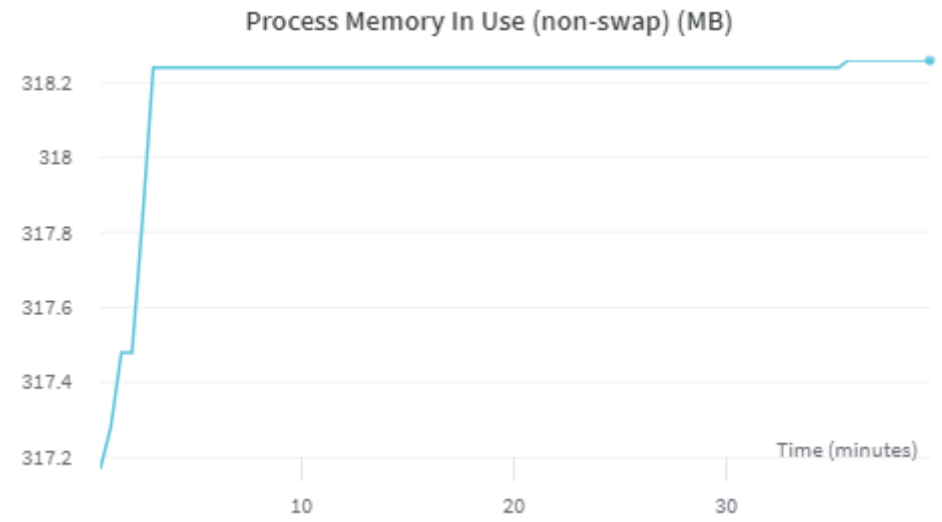
   During training and validation the computer load is as shown below:

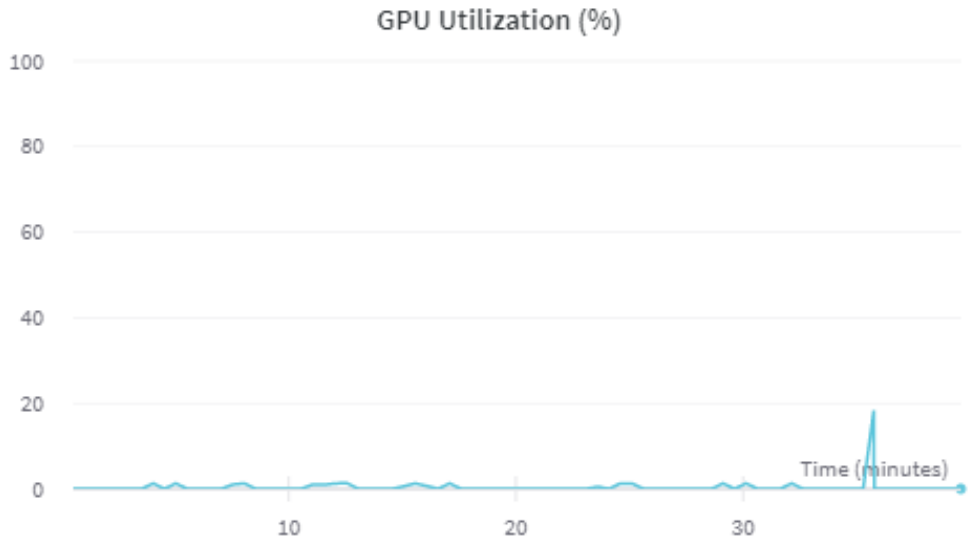


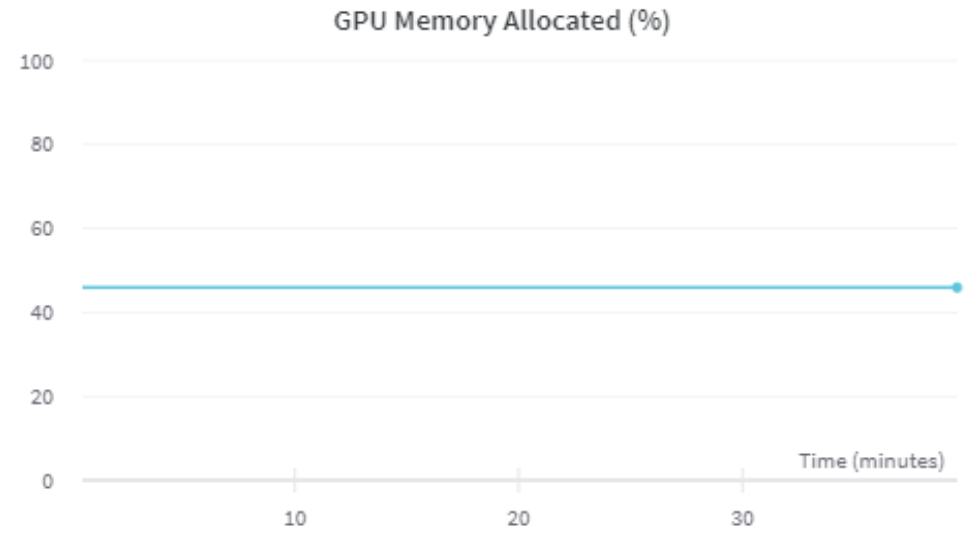

GPU Memory Allocated (%)
100
80
60
40
20
0
Time (minutes)
50
100
150
200
250

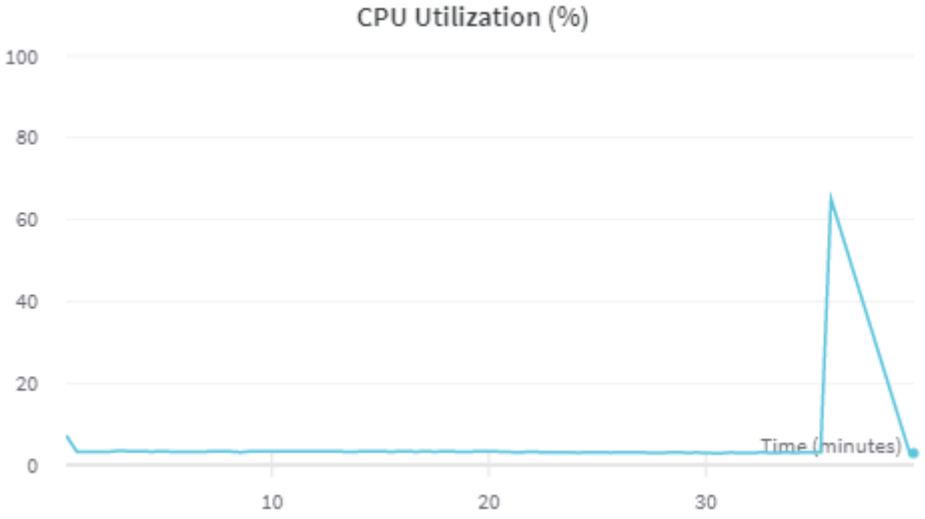

CPU Utilization (%)
100
80
60
40
20
0
Time (minutes)
50
100
150
200
250

Model evaluation with the test dataset took 35 mins. The computer during this process is loaded as shown below:









Times needed to predict a single image (batch size = 1) on an already trained model are shown in the table below.

	1	2	3	4	5	6	7	8	9	10	11	12
Time [ms]	52,8	45	48,7	53,5	44,6	48,7	46	46,8	50,2	45,4	49,2	42,4

Predicting a single image takes an average of 47.8 ms.

4. Advantage & disadvantage

   The programmed Convolutional Neural Network has extremely high validation accuracy at the level of **99,73%**. Also the value of its loss function is very low at **0,0083**.

   Examination of the accuracy of the model based on the testing dataset proved the satisfactory results: the final accuracy at the level of **99,91%**
   and its loss function at the level of **0,0088.**

   The values of the Confusion Matrix are approximated to the decimal value so it shows sensitivity and specificity as 100% accurate as a result of this approximation.

   That means CNN has an extremely high success rate. Our model seems to be very complex, universal and resistant to different types, poses and expressions of the faces. Even if part of the face is covered, our program can detect properly the presence of the face. The main reason for this is the huge amount of data we have managed to gather.

   Another advantage of this solution is online availability of the datasets for reviewing, expanding and editing from Google Drive.

   In total there's almost 60 thousands of images available for CNN, from which 37 thousands is reserved as training data, 11 thousands for validation data and 9 thousands for testing data. Additionally, training and validation images were subject to augmentation. This made the model more flexible in terms of face rotation, point of view, etc. Huge data size allows the CNN to be universal and definitely adds to the properties of the CNN.

   The biggest drawback of this solution is the training time of the NN. On average it takes 16154 seconds for the CNN to be compiled. That’s almost 4,5 hours of uninterrupted computations of the program in Collab Keras. Increasing the batch_size slightly helped in lessening the time of compilation.

   Another significant issue came to be the online environment itself - Google Collab. While it makes working on the same project convenient in a group and there’s no need to use your own gpu on computational tasks, the environment and connection to the hosted GPU is unreliable. Very often what happens is that the connection to the hosted GPU drops and the session crashes during compilation, which requires recompilation of the entire script from the beginning. After that servers might be unavailable for a couple of hours. Another issue is the limit of the available resources at once on these GPUs. If the limit is reached the compilation is interrupted, which again needs restarting.

   This makes applying small changes and testing extremely inefficient and risky in such a demanding implementation. Despite knowing how long the computation should take we cannot control how reliable Google servers and their GPUs are. Once it even took a couple of days to have an uninterrupted, successful run of the program.

   Therefore, if the compilation is very demanding it is better to use a local *runtime* using for example Jupyter Notebook instead of a hosted runtime in Google Collab even though the Google GPUs have more computing power.

5. Coding

   The code of the algorithm can be seen on the following page:
   https://colab.research.google.com/drive/1vpmrB3ZI9F408uHKv3lRjOKGb7n7pLvy?usp=sharing