EIASR 21Z - Face Mask Detection

Michał Smutkiewicz, 283538

Tomasz Miazga, 293071

Contents

Task description

The objective of this project is to, in general, **recognize if people, identified on given pictures can be recognized as people with face masks**, a feature that can be useful in fighting a pandemic. Our main idea is to use convolutional neural networks to deal with the recognition. To get a network working like this, we will try to teach it to classify people with and without masks. As a result, based on given input images, system output should provide a classification of each input image.

Choice of initial model

We will be taking advantage of *Transfer Learning*. For the initial model, we chose **ResNet50**. It is a very popular model when it comes to image processing mainly due to its great accuracy. ResNet50 was appreciated and confirmed at the ILSVRC 2015 classification competition, achieving an error of only 3.57%. The name corresponds to the term *Residual Networks* and number 50 references the number of layers the model consists of.

In the field of deep convolutional networks, a typical solution for increasing the accuracy of a network is introducing a greater number of stacked layers. Each layer can be trained for different variations of tasks to make the overall result better. Yet with the increasing number of layers a problem of saturation and degradation of the accuracy might occur. To prevent the unwanted downgrades, ResNet architecture introduced a key concept: **connections skipping via addition**. This concept is illustrated on the figure below:

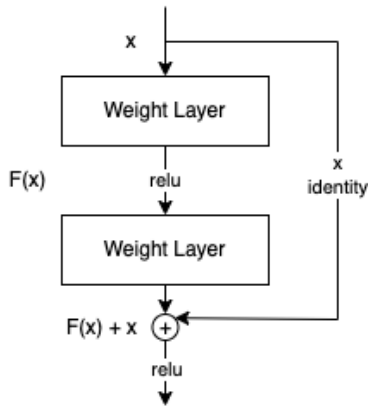

Fig. 1: Residual blocks of ResNet architecture

The connection skipping vastly enhances the possibilities of the network in two ways. At first, it eliminates the issue of vanishing gradients by introducing an alternative shortcut for gradient to go through. Secondly, it allows the layers to learn an identity function very simply. The identity function is also used to preserve the gradient alone.

The effect of identity function and gradient preservations is that for the greater number of layers no degradation of accuracy occurs. The residual networks allow users to train much deeper networks with greater number of stacked layers resulting in better results.

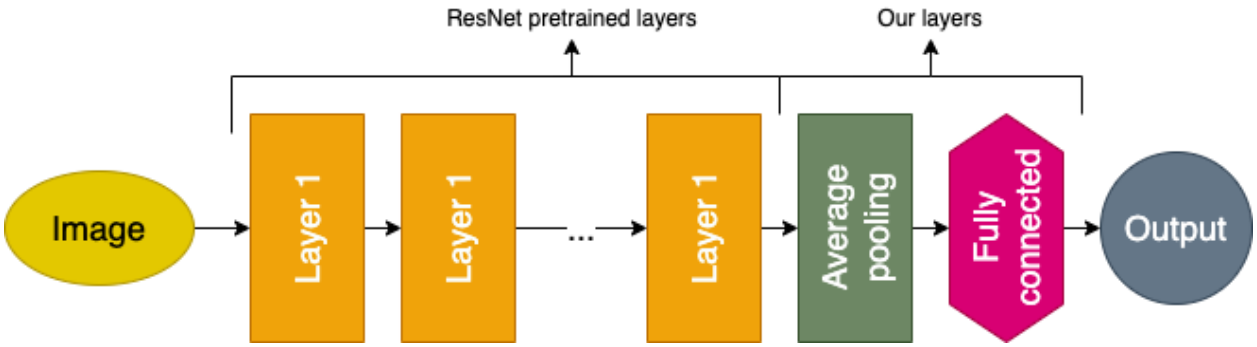

Fig. 2: Our ResNet solution overview

An overview of important layers in the architecture:

- **Average pooling**: required to downsample the detection of features in a feature map. It maps square patches (usually 2x2) into the average of values inside, dividing the size of the feature map by 2 in every dimension. This layer answers the problem of feature location in the input image sensitivity (local transition invariance).
- **Fully connected**: so called *dense layer*, is a part classification layer of a network. It is used for supporting the final classification, returning the probability of belonging to a specified class (in our case - whether a person has a mask on, or not).

Model training

1. Testing images selection

   Classifier will be taught in two classes: “has mask” and “no mask”, so we need to gather a dataset containing information about two classes of images. We also need a set for verification of our classifier accuracy. We have chosen *kaggle* platform as our source of example datasets:

   - **Face mask faces**: https://www.kaggle.com/andrewmvd/face-mask-detection
   - **Validation set**: https://www.kaggle.com/aditya276/face-mask-dataset-yolo-format

2. Training algorithm

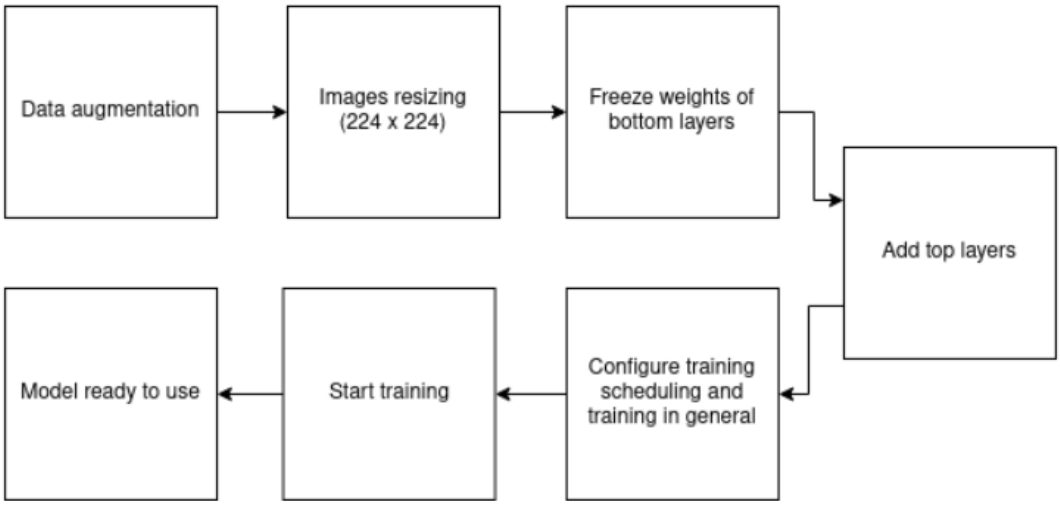

Fig. 3: Training algorithm

- **Data augmentation**: for a more diverse dataset and for better results, we will try *data augmentation*. Augmentation can include image preprocessing operations like: rotation, vertical/horizontal flip and histogram equalization.
- **Images resizing**: For compatibility with our initial model, we will have to resize images to size of 224x224.
- **Freeze weights**: freeze weights of pretrained bottom layers.
- **Add top layers**: we will use additional global average pooling and dense layers for training of the network.

- **Configure training**: choose optimal number of epochs (to automatically stop training when there will be no improvement visible), choose learning rate for optimizer.
- **Start training**: compile and save the classifier.

As we will be using a pretrained model, we will load the base model containing pretrained weights with classification layers stacked on top. At this point we will have numerous ways of setting up our model. We might choose different ways of approaching transfer learning (feature extraction, fine tuning). At first we aim to train the whole model including the ResNet50 already calibrated weights. The training process (which depends on random initialization) will be run for several epochs with an early stopping mechanism, to avoid overtraining the network.

Algorithm description

We divided the algorithm into two main phases: detection and classification. For detection, we will use the OpenCV library and for face classification, we will use pre pre-trained convolutional neural network - ResNet50.

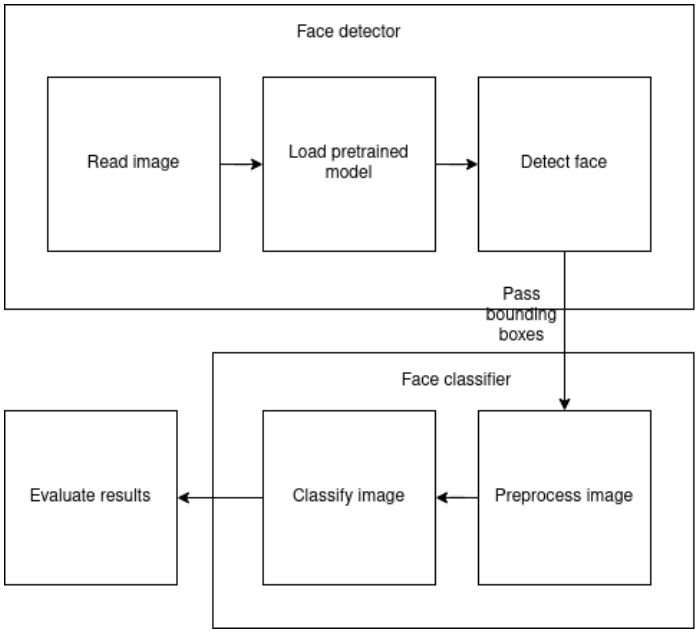

Fig. 4: General algorithm diagram

1. Face detection

   Our first general step is to detect all possible faces on a given image. We will be using a pre-trained cascade classifier in OpenCV.

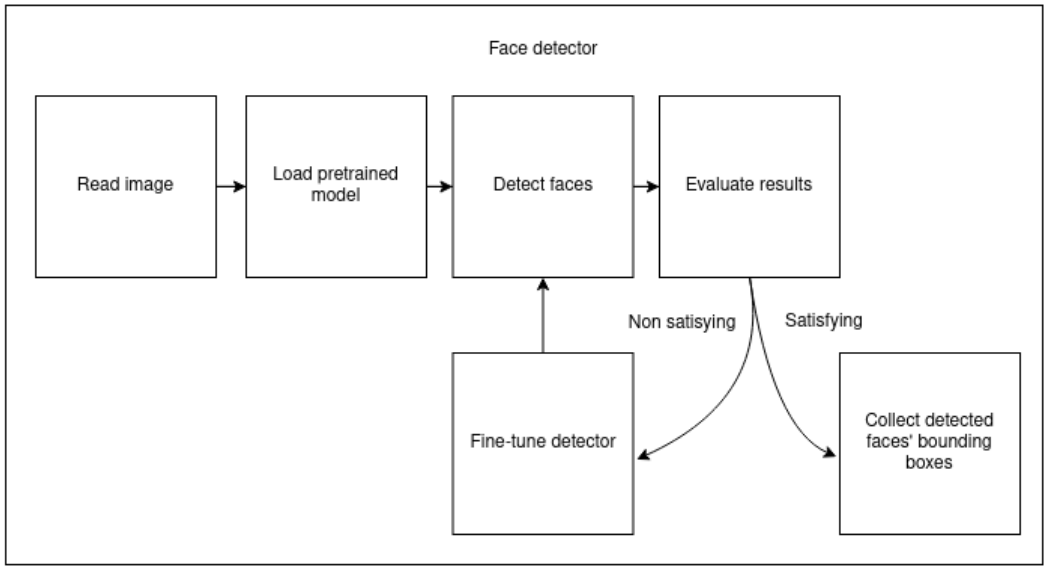

Fig. 5: Face detection algorithm diagram

- **Reading image**: randomly pick an load an image from our validation image set
- **Loading pretrained model**: we will use the *Open Frontal Face Detection* Model from OpenCV Github project resources.
- **Detecting faces**: we will use the Haar Cascade Classifier, which is already trained with several hundred positive sample views of a particular object and arbitrary *negative* images of the same size. After the classifier is trained it can be applied to a region of an image and detect the object by moving a search window across the image and checking every location for the classifier.
- **Evaluating results**: depending on how accurate the detection will be on images from our validation images set, we can fine-tune the cascade classifier by modifying *scale factor* or *minimum neighbors* parameters.
- **Collection of detected faces' bounding boxes**: cascade classifier should provide us boundaries as x and y coordinates of corners of rectangle over each detected face.

2. Face classification

Fig. 6: Face classification diagram

We will leverage our trained model which has an easy API to run the prediction process. We only have to resize the input image, which is a cropped part of the original image (part on which the detector previously found face).

Results evaluation

In order to properly analyze the results of our work, we will prepare three types of plots:

- The first type of plot will present **training and validation accuracy** depending on the number of epochs.
- Second type will present **training and validation loss** also depending on the number of epochs.
- **Confusion matrices** which will contain crucial information of our examination and will be very interesting to compare with previous plots.

Choice of tools

- Python,
- OpenCV for detection,
- Tensorflow/Keras for classification,
- Google Collab for model training in cloud (or optionally on our own GPUs)