Face Mask Detector using Xception

Samuele Da Mar[1]

Other group members:
Md Abdul Hamid[2], **Andreea-Daniela Badea**[3], **John E Farmer**[4]

Abstract. The pandemic caused by the coronavirus (SARS-CoV-2) has pushed the machine learning research to study more in depth the development of models for face mask detection. Me and my peers have come up with four different solutions that could hopefully bring the research forward or, at least, confirm the existing findings.

In this paper has been discussed the performance of a *Convolutional Neural Network* (CNN) model based on the Xception network, which was developed by Francois Chollet in 2017. The results are excellent as the model was able to predict correctly at a rate of 98% and 100% most of the times, that the person in the image was either wearing a medical face mask or not.

1 Introduction

Recently, we have been dealing with the contagious coronavirus (SARS-CoV-2) and we still have to wear face masks to protect us and the people physically near us. For this reason, it could be beneficial to study the implementation of face mask detectors, especially in large public venues.

Me and my peers have researched for different face mask detection models and we have tried to find which one performed better, in other words, which one had the best accuracy (a variable which will be explained in section 2).

Personally, I chose to train and test a *Convolutional Neural Network* model, shortened CNN, which I have found on the website of *Keras* [7], one of the libraries that I have used in my program. I then modify certain lines and parameters of the program following the implementation of another model on GitHub [6], so that I created something new and original.

Hence, in this paper has been described the implementation of my CNN model; the purpose of which is to determine whether a person in an image is wearing a face mask or not. Specifically, in section 2 a brief introduction to the concept of *classification* is given and some studies related to the topic of face mask detection are reported. Subsequently, in section 3 the working of my program, in which the CNN has been implemented, is outlined step by step. In section 4, a discussion regarding possible adjustments to the program is provided. Finally, in section 5 our results are compared to previous experiments in the literature and some ideas for future improvements are suggested.

[1] School of Computing and Mathematical Sciences, University of Greenwich, London SE10 9LS, UK, email: sd6049m@gre.ac.uk
[2] School of Computing and Mathematical Sciences, University of Greenwich, London SE10 9LS, UK, email: mh0009k@greenwich.ac.uk
[3] School of Computing and Mathematical Sciences, University of Greenwich, London SE10 9LS, UK, email: ab0104b@greenwich.ac.uk
[4] School of Computing and Mathematical Sciences, University of Greenwich, London SE10 9LS, UK, email: jf0425q@greenwich.ac.uk

2 Background

Fortunately, the research in the ML field has already developed some models that are able to detect whether a person is wearing a face mask or not.

These models are called classification models, since they classify the data in a given dataset by detecting the features that characterise each class (i.e. oranges and apples, two different classes of fruits, have different shapes and colours that can be recognised by the models).

In order for the classification to take place, the dataset must be labelled, which means that, in ML terms, we are using a *supervised* learning approach [2] (i.e. the oranges have label 0 and the apples have label 1). Moreover, as shown in figure 1, usually the best way to develop a predictive model is to use *supervised* learning with either *regression* or *classification*.

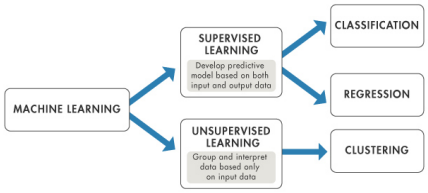

Figure 1. source: https://wiki.seg.org/.

There are several models used for classification, some of these are: *Artificial Neural Networks* (ANNs), *Support-Vector Machines* (SVMs), *k-Nearest Neighbour* (kNN) and *Adaptive Boost* (Adaboosted) [18].

One of the main types of ANNs is *Convolutional Neural Networks* (CNNs), which have been widely used in the field of computer vision tasks. After training, CNNs can recognise and classify facial images, even with slight differences, thanks to their powerful feature extraction capability. *Super-Resolution* (SR) networks, as one of the CNNs, can restore image details and can improve the classification accuracy significantly, especially when using a datasets with low-quality images, and provide a feasible solution to improve face mask-wearing condition identification performance [16].

For instance, [16] developed an original face mask identification method that combines an SR network with a classification network (SRCNet) for facial image classification. They also utilised a

deep learning method for the automatic identification of face mask-wearing conditions. They then improved the SR network structure, including activation functions and the density of skip connections, which outperformed previous state-of-the-art methods at that time (2020). Their findings indicate that the proposed SRCNet can achieve high accuracy in face mask-wearing condition identification. They found few limitations: firstly, the medical masks dataset they used is relatively small, so it cannot cover all postures or environments. In addition, the dataset does not contain video, therefore the identification result on a video stream cannot be tested.

In the experiment of [18], instead, the authors developed an integrated model of ANN and SVM combined. It is a two layers classifier. The first layer contains a number k of ANN(s) that output the classifying result based on a feature vector of one by one image. The second layer contains a SVM classifier, and its purpose is to integrate all results of the first layer. This model, called *ANN_SVM*, is easy to design and deploy for the specific classification problem. The precision is high, but the performance of processing time needs to improve, especially when applied for complex image classification such as the facial one. The training time of *ANN_SVM* is also a problem with large datasets.

Another study from [3] used a different approach. They implemented *Principal Component Analysis* (PCA), an important statistical procedure (also defined as an orthogonal linear transformation) that emphasises variation and brings out strong patterns, while reducing the size of the dataset. As [9] explained, the idea of PCA is simple: reduce the dimensionality of the dataset, while preserving as much "variability" (i.e. statistical information) as possible. In the end, [3] concluded that traditional statistical algorithm *Principal Component Analysis* (PCA) is better for normal face recognition but not for masked face recognition.

[8] carries out a similar model to the one that I used. They proposed a methodology divided into two phases. The first phase deals with over-sampling by image augmentation of the training data. The second phase is the detection of face mask using transfer learning of the InceptionV3 network model. For the evaluation they used several performance metrics: Accuracy, Precision, Sensitivity, Specificity, Intersection over Union (IoU), and Matthews Correlation Coefficient (MCC). They concluded that their model achieved accuracy and specificity of 99.92%, 99.9% during training, and 100%, 100% during testing on their dataset. This is the best performance I was able to find in the literature.

3 Experiments and results

My group chose to train and test the following models: an *Artificial Neural Network* (ANN), two different *Convolutional Neural Networks* (CNN) and a *Support-Vector Machine* (SVM). Each group member, myself included, used the same dataset [5], so that we could fairly compare our solutions. I used one of the two CNNs, which can be found in Keras' Code Examples under the Computer Vision section [7].

Having that example as a base, I imported the code on Google Colab (short for Colaboratory), an environment that runs entirely in the cloud and that allows to run your python code using Google server's computational power. It is especially suited to run ML algorithms or any python program that handles big datasets [4].

Once the environment has been set, the first step is to import the following libraries:

- TensorFlow, to develop and train the ML model [17].
- Keras, a deep learning API written in Python, running on top of TensorFlow [10].
- Matplotlib, to visualise images and plot graphs [14].
- Numpy, to enable numerical computing with Python [15].

The second step is to load into the environment the .zip file "data" from the GitHub repository. Once unzipped, the folder contains two sub-folders, "with_maks" and "without_mask", in which the images of people with or without face mask have been respectively stored.

The third step is to filter out corrupted images. Even though our dataset did not have any corrupted image, is always good practice to go through this procedure, especially if you are working with real world images.

The fourth step is to generate the dataset. In the second step we only loaded the folders with the data that we needed; now we can actually create a dataset by using the *image_dataset_from_directory* utility. In particular, we define a variable for the training set (*train_ds*) and a variable for the validation set (*val_ds*), using 0.2 for validation. Moreover, we set the *image_size* to be in a standard size, 180 by 180, and we set the *batch_size* to be 32, which is common use and therefore it lets us fairly assess the results of our experiment.

In the fifth step, as you can see in figure 2, we want to visualise the images of our generated dataset, together with their labels. The images in "with_mask" are going to have label "0" and the images in "without_mask" are going to have label "1". This way, we know that the dataset has been generated properly.

Figure 2. Training images and associated labels.

The sixth step is to augment the data. Since our dataset is not that large, we introduce new images in the training set by applying some variations and transformations to the existing images, such as random horizontal flipping or small random rotations (figure 3). This procedure enhances the effectiveness of the training and lowers the over-fitting issue, when the machine starts, in a certain way, remembering the data without really processing it.

In the seventh step we use buffered prefetching, by calling the *prefetch* utility, so we can yield data from disk without having I/O becoming blocking (*buffer_size* = 32).

The eight step is to, finally, build the actual model, which is a small version of the Xception network. The Xception network is a CNN architecture based entirely on depthwise separable convolution layers. The architecture has 36 convolutional layers forming the feature extraction base of the network [1]. At the start of our model we use a Rescaling layer to flatten the RGB channel values of the images

Figure 3. Augmented images.

from [0, 255] to [0, 1]; this way the input data is going to be much smaller, improving the training time. The next layer is a Conv2D layer (with 32 as filter size). This creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs [11]. Then a BatchNormalization layer is added. During training, the layer normalises its output using the mean and standard deviation of the current batch of inputs. Subsequently, an Activation layer is implemented, applying the activation function "relu", which is short for, *rectified linear unit*, to the output of the previous layer. After some more other implementations, a MaxPooling2D layer downsamples the input along its spatial dimensions (height and width) by taking the maximum value over an input window for each channel of the input [13]. Furthermore, a variable called *residual* stores what is returned by the *previous_block_activation*, called on a Conv2D layer. Exiting the loop, other four layers are added, a SeparableConv2D, a BatchNormalization, an Activation and a GlobalAveragePooling2D. The latter computes the global average pooling operation on the data [12]. In the end, if the number of classes is 2, like in our case, in the activation is going to be set as "sigmoid" and the units are going to be set as "1", otherwise the activation would be "softmax" and the units as the number of classes. Finally a Dropout layer, which helps overfitting, and a Dense layer, which classifies data, conclude our model.

In the eight step we are setting the epochs to 15 and we train our model. In the *compile* function, we use Adam as optimizer with a learning rate of 10^{-3}, we set the loss to "binary_crossentropy" and we set the metrics to "accuracy". The metric "accuracy" is really important as it is going to tell us if our model was able to succeed in the detection or not. This variable represents the quotient between the number of correct predictions over the total number of predictions. Calling the function *fit*, we actually initialise the training. We therefore save the output of that function (the training) in the a variable called "history". And then we call the function *save_weights* to create a file with the stored model weights.

The ninth step is to plot the training behaviour: one graph for the loss and one for the accuracy.

As you can see in figure 4, the loss for the validation grows (yellow line) until the epoch number 11 and then drops on the number 12. This probably means that the model was able to overcome the challenge that the validation brings. In figure 5, in fact, we see that after epoch number 9 the validation's accuracy (yellow line) starts increasing.

The tenth and last step is to test the model: a random image is

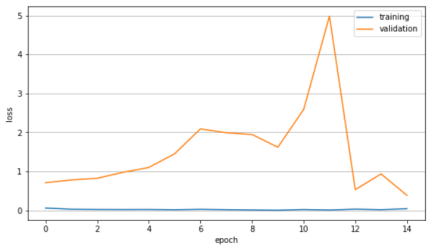

Figure 4. Loss of training and validation.

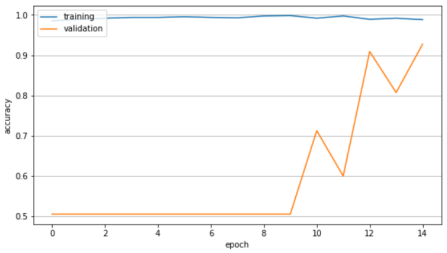

Figure 5. Accuracy of training and validation.

loaded from its directory and with the *predict* function we get the "score" variable which is either closer to 0, if the model predicted that the person in the image looks like has a face mask, or closer to 1 if the person seems to not wear a face mask.

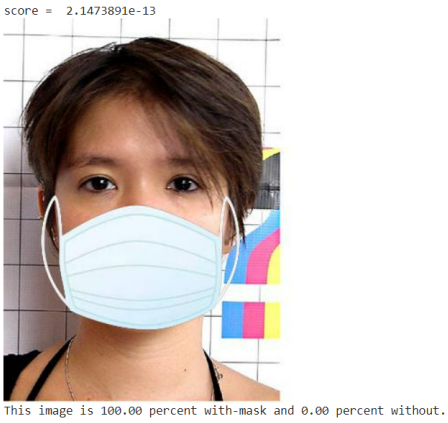

Figure 6. Prediction.

As you can see from figure 6, this image had a score of $2.14 * 10^{-13}$. This means that the model was $(100 - 2.14 * 10^{-13})\%$ sure that the person in the image was wearing a face mask.

4 Discussion

The first thought that comes to my mind is that the results of my model where too accurate to be true. Indeed, I think that the dataset was not sufficiently large and diverse. Choosing a random image from the dataset to assess the performance of the model is not enough.

The model in this study ties or overcomes the accuracy of the ones in the studies of section 2.

An interesting comparison can be done with [8], since in that experiment has been used a similar model, the InceptionV3 network.

In fact, the Xception network, on which my model is based, is an improved or, as the author of [1] wrote, an "extreme" version of the InceptionV3 network. As reported in [1]:

> *"We show that this architecture, dubbed Xception, slightly outperforms Inception V3 on the ImageNet dataset (which Inception V3 was designed for), and significantly outperforms Inception V3 on a larger image classification dataset comprising 350 million images and 17,000 classes. Since the Xception architecture has the same number of parameters as Inception V3, the performance gains are not due to increased capacity but rather to a more efficient use of model parameters. "*

Unfortunately, the ways we assessed the performance of our models are distinct and the datasets used are also different. This means that any comparison is not enough reliable.

Furthermore, looking at the performances of the models of my peers, it also seems like mine has the most accurate predictions.

5 Conclusion and future work

Considering the results in section 3 and considering what has been said in section 4, I can conclude quite confidently that the Inception networks, and in particular the Xception one, performed better than any other model from previous experiments in the machine learning literature.

As reported in section 4 and in the conclusions of [8], the work can further be improved by employing larger datsets and can also be extended to classify different kinds of masks. Moreover a facial recognition system could be combined to, not only detect that a certain person is not wearing a face mask, but also to recognise immediately who that person is.

REFERENCES

[1] Francois Chollet, 'Xception: Deep learning with depthwise separable convolutions', in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (July 2017).

[2] Julianna Delua. Supervised vs. Unsupervised Learning: What's the Difference? — IBM, 2021. note: click on title for link.

[3] Md Sabbir Ejaz, Md Rabiul Islam, Md Sifatullah, and Ananya Sarker, 'Implementation of principal component analysis on masked and non-masked face recognition', in *2019 1st international conference on advances in science, engineering and robotics technology (ICASERT)*, pp. 1–5, (2019).

[4] Google. Google Colab. note: click on title for link.

[5] prajnasb on GitHub. Mask Classifier. note: click on author and title for links.

[6] The-Assembly on GitHub. Build-A-Face-Mask-Detector-With-TensorFlow. note: click on author and title for links.

[7] fchollet on Twitter. Image classification from scratch. note: click on author and title for links.

[8] G. Jignesh Chowdary, Narinder Singh Punn, Sanjay Kumar Sonbhadra, and Sonali Agarwal, 'Face mask detection using transfer learning of inceptionv3', in *Big Data Analytics*, eds., Ladjel Bellatreche, Vikram Goyal, Hamido Fujita, Anirban Mondal, and P. Krishna Reddy, pp. 81–90, Cham, (2020). Springer International Publishing.

[9] Ian T Jolliffe and Jorge Cadima, 'Principal component analysis: a review and recent developments', *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, **374**(2065), 20150202, (2016).

[10] Keras. About Keras. note: click on title for link.

[11] Keras. Conv2D layer. note: click on title for link.

[12] Keras. GlobalAveragePooling2D layer. note: click on title for link.

[13] Keras. MaxPooling2D layer. note: click on title for link.

[14] Matplotlib. Matplotlib: Visualization with Python. note: click on title for link.

[15] Numpy. About Us. note: click on title for link.

[16] Bosheng Qin and Dongxiao Li, 'Identifying facemask-wearing condition using image super-resolution with classification network to prevent covid-19', *Sensors*, **20**(18), (2020).

[17] TensorFlow. Why TensorFlow. note: click on title for link.

[18] Le Hoang Thai, Tran Son Hai, and Nguyen Thanh Thuy, 'Image classification using support vector machine and artificial neural network', *International Journal of Information Technology and Computer Science*, **4**(5), 32–38, (2012).