# Face-Mask Detector using CNNs

**Ihtisham Ahmad**
MS in Data Science
University of Potsdam
*ahmad@uni-potsdam.de*

**Raheleh Majidi Tehran**
MS in Cognitive Systems
University of Potsdam
*majiditehran@uni-potsdam.de*

**Muhammad Zeeshan Mazhar**
MS in Data Science
University of Potsdam
*mazhar@uni-potsdam.de*

## Abstract

The ongoing global pandemic of Covid-19 and the slow progress of vaccination in most countries make the continuation of our newly acquired preventive behaviors inevitable, at least to some degree. Among all the precautionary steps, studies show the association between face masks use and significant reduction of the airborne transmission of the virus and hence, the spread of infection. These findings have found object detection field a new mission: Face mask detectors. In this project, we developed a model to determine whether a person uses a mask or not, both on images and live stream videos. Our model uses a deep convolutional neural network as its architecture's backbone. We used RMFD (Real-World Masked Face Dataset) to train and evaluate a CNN-based model and employed OpenCV and PyTorch as our implementation tools. Our final experiment showed 98.9 accuracy in the test phase which is a good result.

## 1 Introduction

In December 2019, Life as it used to be, has been ended for the majority of people worldwide. The Covid-19 pandemic took over our lives, forcing us to add some new routines to our hygiene habits. Among these newly acquired procedures, wearing a mask in public places is somewhat the only measurable one. Some studies are showing a negative correlation between the rate of wearing masks in the area and the number of infected people by the virus. These findings are critical for better prediction of virus spread patterns. These patterns give the authorities insight into how to adjust preventative measures and medical facilities

To obtain this kind of information and also stop people from removing their masks in high-risk places, we need face mask detectors. These systems are based on object detection models and usually consist of two different steps: Detecting the face in an image or real-time video and determine whether there is a mask on the detected face or not.

In this project, we propose a model based on CNN (Convolutional Neural Network) for face mask detection task. Our model consists of three convolutional layers followed by two fully connected layers. We trained our model on RMFD dataset which is an image dataset with images from 525 different people, 5000 masked and 90000 normal faces.

## 2 Background

Object detection is a combination of two separate tasks: object localization and image classification. Object localization means locating pertinent objects in an image and pinpointing them by drawing a box around those objects. Image classification, in general, is to assign a class label to an image. Object detection integrates these two separate tasks by indicating one or multiple objects of interest in the image by bounding boxes and assigning class labels to those objects. This way, an object detection algorithm makes it possible to decide whether a particular type or class of objects is present in the image or not.

The core of our project is face detection which is a popular sub-branch of the object detection field. Early face detection approaches used classical methods, extracting manually tailored features from the images and fed them into classifiers to detect the potential face regions. One of the most favored classical object detection frameworks was developed by Viola & Jones in 2001[1]. The idea behind this model was a combination of Haar-like features and the cascade classifier concepts. Another popular classic method was presented by Dalal & Triggs in [2], using Histogram of Oriented Gradients (HOG). Both models were breakthroughs in the field at the time, but they also suffered the same problems regarding their sensitivity to pose, illumination, and image quality.

Computer vision took a quantum leap in 2012, when AlexNet[3], an AI system from the University of Toronto, won the IMageNet computer vision contest with 85 percent accuracy. The runner-up contestant scored 74 percent on the test. AlexNet's core strength resided in using Convolutional Neural Networks (CNN), trained on a large dataset (ImageNet), and exploiting vast computer resources. CNNs are a specialized version of artificial neural networks and were in the object detection game for a long time. They were first introduced by Yann LeCun in the 1980s and soon were employed in banking and postal systems to read postal codes and checks' digits. However, CNNs need a lot of data and processing resources to work with large or high-resolution images, which none was available at the time. As a result, CNNs' remained on the sidelines of AI, and their application was limited to basic recognition tasks until recently.

CNNs architecture is inspired by the human visual system. It leverages the spatial relationship among data, which is essential for dealing with vision-related problems and was lost from simple feed-forward networks (e.p., MLPs). Each CNN consists of two major blocks. The first layer in this block is responsible for feature extraction. CNNs utilize filters for this goal. Filters are small squares of data, each representing one of the features we aim to find in the inputs. Unlike previous approaches, these filters (features) are not predefined. Instead, they will be learned as the network's weights during the training phase. Hence they can be different based on the type of images in the training data. A filter is slid over the input image and build a feature map. This process is repeatable for different filters, and they are also manipulated in terms of norms and size. The final feature map will be feed to the second block. The second block resembles the last section of any other neural network. It is responsible for image classification and consists of fully connected layers and a final activation function.

Revised version of CNNs adapted to affected the approaches in the computer vision field dramatically, and object detection was no exception. One of the first DL models proposed for object detection was Region-based CNNs (R-CNN) [3]. Girchick et al. suggested extracting a group of regions from the image using selective search. The goal is to reduce the number of regions. The next step is to feed these regions into a deep CNN and extract features from them. As the final step, the model's SVM classifier receives these features as input and assigns them a label from one of the known classes for the model. An extension of R-CNN was introduced in [4]. Fast R-CNN aimed to improve the speed issue of its successor and increasing detection accuracy. The proposed solution was to replace the former pipeline with a single model responsible for region suggestion and classification. Although the new model is much faster than the original one, the speed issue remained one of the shortcomings of this family of models, especially in real time applications. The root of this problem is the need to propose a set of regions per each input image.

Another popular DL model designed for object detection is Single Shot Multi-Box Detector (SSD) [5]. Unlike the R-CNN family, SSD uses a single deep neural network and replaces the region proposal network step by applying convolutional filters to extracted feature maps. For each feature map location, the model makes predictions by utilizing four default boundary boxes. We can increase the accuracy of the SSD model by adding to the number of boundary boxes, albeit at the cost of speed. SSD model is a practical choice for real-time object detection.

## 3 Methodology

### 3.1 Dataset

To train our deep learning model to classify that a person is wearing a mask or not, we need a dataset which has both masked and unmasked images. We are using an open source data set Real World Masked Face Dataset (RMFD).

This dataset is further divided into two categories.

1. First one has real people images. Total 525 people faces, 5000 masked images and 90000 unmasked images.
2. Second one is simulated masked and unmasked images.

For our model training we will test both datasets.

We have to split our dataset into three parts, training dataset, testing dataset and validation dataset. The purpose of splitting data is to avoid overfitting which is paying attention to minor details/noise which is not necessary and only optimizes the training dataset accuracy. We need a model that performs well on a dataset that it has never seen (test data), which is called generalization. The training set is the actual subset of the dataset that we use to train the model. The model observes and learns from this data and then optimizes its parameters. The validation dataset is used to select hyper-parameters (learning rate, regularization parameters). When the model is performing well enough on our validation dataset, we can stop learning using a training dataset. The test set is the remaining subset of data used to provide an unbiased evaluation of a final model fit on the training dataset. If our dataset and model are such that a lot of training is required, then we use a larger chunk of the data just for training which is our case too. If the model has a lot of hyper-parameters that can be tuned, then we need to take a higher amount of validation dataset. Models with a smaller number of hyper-parameters are easy to tune and update, and so we can take a smaller validation dataset.

In our approach, we have dedicated 70% of the dataset as the training data and the remaining 30% as the testing data, which makes the split ratio as 0.7:0.3 of train to test set.

## 3.2 Implementation tools and libraries

### 3.2.1 IDE

The main skeleton of the project has several parts that are needed to ensure that the project functions the way it should. For hardware environment, Google Colab GPU was used in the research. So, the first step was to mount the dataset from Google Drive, where it was uploaded.

### 3.2.2 Libraries

Now, the core of the project involved usage of PyTorch Lightning, which is a famous library built on top of PyTorch, and Pandas, another key library in the world of Python. The dataset has separate masked and unmasked image datasets. So, the project needs separate masking and unmasking pictures to be added.

After the former step was done, the next key part of this research involved implementation of CNN and its training. Scikit-learn, PyTorch, PyTorch-Lightning, and Pandas was used for doing the CNN training and implementation stage. For face detection in video or image we used open cv2.

## 3.3 Architecture

### 3.3.1 Convolutional neural network structure

There are mainly three specific kinds of layers in ConvNets: Convolution, ReLU, and Max Pooling layer. We explain each of them briefly.

A convolution layer contains kernels or filters that are smaller than the dimensions of the input image. We apply the kernel, which is also called a mask, on each pixel (dimensional value) of the image and get an output, which is sent to the next step. In total, three convolution layers and one linear layer were initialized and used in the project.

ReLU is an acronym for Rectified Linear Unit, which is a very important linear activation function whose job is pretty simple. It takes an input, which in our case is a 2-D image and outputs a 2-D matrix with all positives. The main job of the Rectified Linear Unit is to convert all negative values in the 2-D to zero, and it leaves all other positive values as they are and doesn’t change them by any means.

ReLUs are also helpful in avoiding some problems. In CNNs, ReLU layer helps in avoiding the ‘vanishing gradient’ problem, which is very common in CNNs and MLPs. It is always applied after the convolutional layer.

### 3.3.2 Training process

First of all, the given data is divided into training and validation datasets, and 0.3 split size was chosen in our case. Validation datasets are important for checking the accuracy or performance of the model, and training dataset is the actual data used in the model.

As Almost every other training process, our process needed epochs, so a maximum value of 10 epochs was used for efficient but effective processing. It is extremely optimal to choose the ideal epoch value in each project as epoch means how many times a model will go through the data in the modeling process. A less epoch value can lead to underfitting while a value that is too large leads to overfitting. Only an optimal value can lead to an optimal curve.

Now, testing the project with ready-made images is one thing, but validating the effectivity is something else. So, the idea of using a live video feed to test the model came out, and the SK-Video library was used for it. The code detects whether a person is wearing a mask or not from the live feed at the exact same time.

The trainer in the project was chosen as GPU as CPU is generally very slow in cases of large datasets. So, a GPU trainer is always recommended, which is why we also went ahead with the latter in our case.

In a feature map, the main task of max pooling is to calculate the max value in each mask or area of a 2-D image. We can define it to be a 2x2 or 3x3 or any other specific size depending on the image. In the project, a 2x2 kernel was used for max pooling. It helps to eradicate issues such as overfitting in the model, and decreases the computational cost by quite a bit, because when only maximum values are chosen among each patch of a 2-D image, the total number of parameters are reduced.

## 4 Evaluation

Once a machine learning model has been built, the most crucial task is to evaluate how good its performance is regarding the unseen data. To delineate our model's predictions, we applied the accuracy, and loss formula to our results and plotted the outcomes. We will discuss our findings in the following.

You can choose to log your model's loss data after each iteration or after each epoch. We chose the latter approach and calculated the loss function across every item during every epoch.

We plotted both training and validation loss curves on the same graph to make comparison easier (Fig. 1). Training loss experience a sharp drop at the beginning and continues to decrease more steadily during the following epochs to the point that it became approximately flat. Also, by comparing the validation loss graph with the training loss curve, we can observe that they both gradually decrease to the point of stability after the first swift dive. There is a tiny difference between the two curves, known as the generalization gap. Hence, we can conclude that our model reached a good fit during the chosen number of epochs. Further training of the model would likely lead to an overfit.

Same as the last step, the curves for training and validation accuracies are plotted in the same graph. The gap between the training and validation accuracy indicates the amount of overfitting. We can observe that the training accuracy plot had a smoother increase through the epochs compare to the other curve. Despite this fact, both graphs have reached a good accuracy in the final epochs, which is another sign of a good fit of our model.

Same as the last step, the curves for training and validation accuracies are plotted in the same graph (Fig. 2). The gap between the training and validation accuracy indicates the amount of overfitting. We can observe that the training accuracy plot had a smoother increase through the epochs compare to the other curve. Despite this fact, both graphs have reached a good accuracy in the final epochs, which is another sign of a good fit of our model.

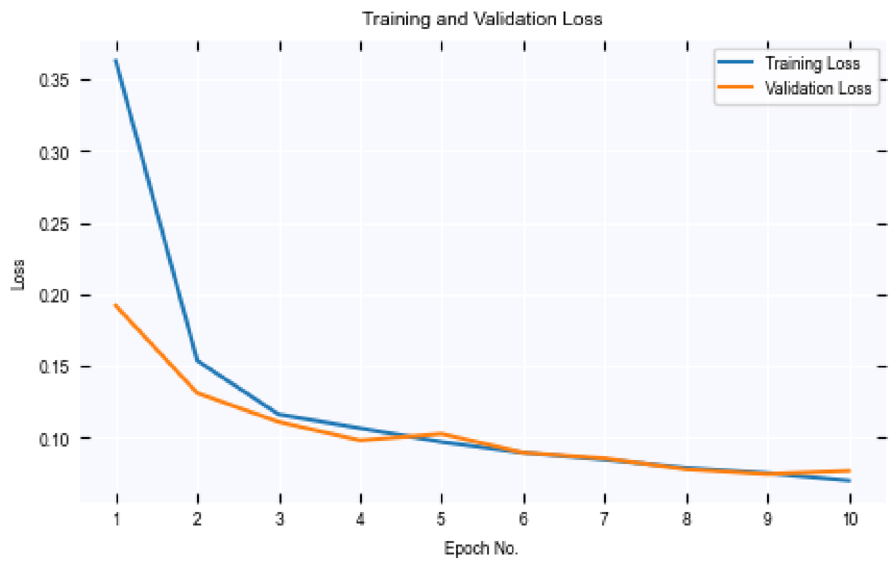

*Figure 1: Loss graph for training and validating steps*

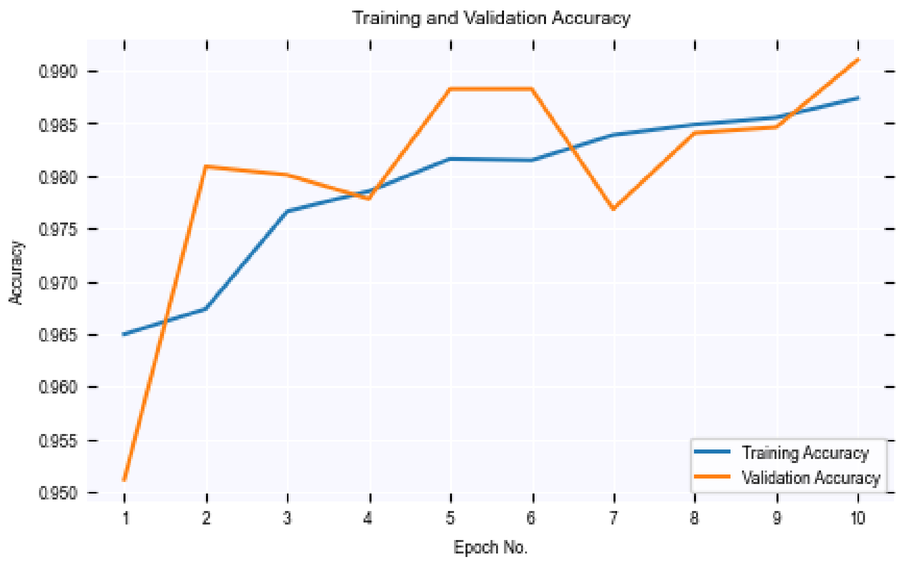

*Figure 2: Accuracy graph for training and validating steps*

## 5 Limitations

CNN’s generally take more time, which comes as a disadvantage. So, our research did comprise of CNN’s and it did elongate the time taken for the process to finish. With a bigger dataset, the model took more time than it would with a smaller one. We also tried rotated images, and the results weren’t too promising.

## 6 Future works

With larger datasets, it could take ages for the current model to train. So, one important future work would be to formulate ways by doing research that would enhance the current model, which in turn will increase the speed of the model because the more efficient a model is, the better.

## 7 Conclusion

In this report, a CNN-based mechanism was proposed for face mask detection in people using PyTorch Lightning. The first step involved inserting the RMFD dataset snaps into a data frame w.r.t images with mask and without the mask. Then, CNN implementation was done. The maximum number of epochs was set to 10. After training, the next step involved taking image frames from a live video feed and detecting if the person in the frame had a mask on his face or not. We used two evaluation metrics in this project, loss and accuracy. Applying these metrics to our results helped us configure whether an overfit or underfit happened during the training phase or not. We got 98.9% accuracy for our validation step, which is very promising. This model has a simple architecture; hence, its performance is comparable to more complicated ones. It can be used in many populated places, which manual controls are laborious and help authorities prevent hygiene codes' violations.

## References

[1] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, Dec. 2001, vol. 1, p. I–I. doi: 10.1109/CVPR.2001.990517.

[2] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, Jun. 2005, vol. 1, pp. 886–893 vol. 1. doi: 10.1109/CVPR.2005.177.

[3] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *arXiv:1311.2524 [cs]*, Oct. 2014, Accessed: Jul. 14, 2021. [Online]. Available: http://arxiv.org/abs/1311.2524

[4] R. Girshick, "Fast R-CNN," *arXiv:1504.08083 [cs]*, Sep. 2015, Accessed: Jul. 14, 2021. [Online]. Available: http://arxiv.org/abs/1504.08083

[5] W. Liu *et al.*, "SSD: Single Shot MultiBox Detector," in *Computer Vision – ECCV 2016*, Cham, 2016, pp. 21–37. doi: 10.1007/978-3-319-46448-0_2.