Face Mask Detector

Technical Report · July 2020

DOI: 10.13140/RG.2.2.32147.50725

7 authors, including:

Akhyar Ahmed
Universität Potsdam
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Saniya Adeel
Universität Potsdam
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Md Hasan Shahriar
Universität Potsdam
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Md Shohel Mojumder
Universität Potsdam
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Face Mask Detector

Prof. Dr. Christoph Lippert
Digital Health - Machine Learning
Hasso-Plattner-Institut (HPI)
christoph.lippert@hpi.de

Supervisor
Benjamin Bergner
Hasso-Plattner-Institut (HPI)
benjamin.bergner@hpi.de

Raza Ali - 805145
M.Sc. in Data Science
University of Potsdam
ali@uni-potsdam.de

Saniya Adeel - 805144
M.Sc. in Data Science
University of Potsdam
adeel@uni-potsdam.de

Akhyar Ahmed - 804864
M.Sc. in Data Science
University of Potsdam
ahmed2@uni-potsdam.de

Md Hasan Shahriar - 804693
M.Sc. in Data Science
University of Potsdam
shahriar@uni-potsdam.de

Md Shohel Mojumder - 805616
M.Sc. in Data Science
University of Potsdam
mojumder@uni-potsdam.de

Abstract

After the breakout of the worldwide pandemic COVID-19, there arises a severe need of protection mechanisms, face mask being the primary one. The basic aim of the project is to detect the presence of a face mask on human faces on live streaming video as well as on images. We have used deep learning to develop our face detector model. The architecture used for the object detection purpose is Single Shot Detector (SSD) because of its good performance accuracy and high speed. Alongside this, we have used basic concepts of transfer learning in neural networks to finally output presence or absence of a face mask in an image or a video stream. Experimental results show that our model performs well on the test data with 100% and 99% precision and recall, respectively.

1 Project Introduction

The year 2020 has shown mankind some mind-boggling series of events amongst which the COVID-19 pandemic is the most life-changing event which has startled the world since the year began. Affecting the health and lives of masses, COVID-19 has called for strict measures to be followed in order to prevent the spread of disease. From the very basic hygiene standards to the treatments in the hospitals, people are doing all they can for their own and the society's safety; face masks are one of the personal protective equipment. People wear face masks once they step out of their homes and authorities strictly ensure that people are wearing face masks while they are in groups and public places.

To monitor that people are following this basic safety principle, a strategy should be developed. A face mask detector system can be implemented to check this. Face mask detection means to identify whether a person is wearing a mask or not. The first step to recognize the presence of a mask on the face is to detect the face, which makes the strategy divided into two parts: to detect faces and to detect masks on those faces. Face detection is one of the applications of object detection and can be used in many areas like security, biometrics, law enforcement and more. There are many detector systems developed around the world and being implemented. However, all this science needs optimization; a better, more precise detector, because the world cannot afford any more increase in corona cases.

In this project, we will be developing a face mask detector that is able to distinguish between faces with masks and faces with no masks. In this report, we have proposed a detector which employs SSD for face detection and a neural network to detect presence of a face mask. The implementation of the algorithm is on images, videos and live video streams.

The rest of the report is organized as follows. In Section 2, we will go through the literature review and related work. In Section 3, the methodology of our proposed solution is discussed in detail. In Section 4, the model is evaluated, and results discussed. Section 5 and Section 6 discuss limitations and future work and finally with Section 7, the report is concluded.

2 Literature Review

Object detection is one of the trending topics in the field of image processing and computer vision. Ranging from small scale personal applications to large scale industrial applications, object detection and recognition is employed in a wide range of industries. Some examples include image retrieval, security and intelligence, OCR, medical imaging and agricultural monitoring.

In object detection, an image is read and one or more objects in that image are categorized. The location of those objects is also specified by a boundary called the bounding box. Traditionally, researchers used pattern recognition to predict faces based on prior face models. A breakthrough face detection technology then was developed named as Viola Jones detector that was an optimized technique of using Haar [1], digital image features used in object recognition. However, it failed because it did not perform well on faces in dark areas and non-frontal faces. Since then, researchers are eager to develop new algorithms based on deep learning to improve the models. Deep learning allows us to learn features with end to end manner and removing the need to use prior knowledge for forming feature extractors. There are various methods of object detection based on deep learning which are divided into two categories: one stage and two stage object detectors.

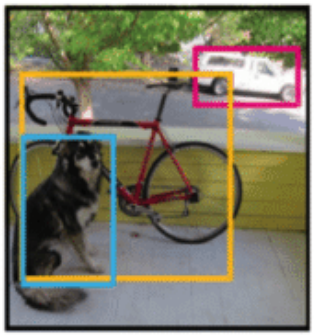

Figure 1: Bounding boxes in an image.

Two stage detectors use two neural networks to detect objects, for instance region-based convolutional neural networks (R-CNN) and faster R-CNN. The first neural network is used to generate region proposals and the second one refines these region proposals; performing a coarse-to-fine detection. This strategy results in high detection performance compromising on speed. The seminal work R-CNN is proposed by R. Girshick et al. [2]. R-CNN uses selective search to propose some candidate regions which may contain objects. After that, the proposals are fed into a CNN model to extract features, and a support vector machine (SVM) is used to recognize classes of objects. However, the second stage of R-CNN is computationally expensive since the network has to detect proposals on a one-by-one manner and uses a separate SVM for final classification. Fast R-CNN [3] solves this problem by introducing a region of interest (ROI) pooling layer to input all proposal regions at once. Faster RCNN [4] is the evolution of R-CNN and Fast R-CNN, and as the name implies its training and testing speed is greater than those of its predecessors. While R-CNN and Fast R-CNN use selective search algorithms limiting the detection speed, Faster R-CNN learns the proposed object regions itself using a region proposal network (RPN).

On the other hand, a one stage detector utilizes only a single neural network for region proposals and for detection; some primary ones being SSD (Single Shot Detection) [5] and YOLO (You Only Look Once) [6]. To achieve this, the bounding boxes should be predefined. YOLO divides the image into several cells and then matches the bounding boxes to objects for each cell. This, however, is not good for small sized objects. Thus, multi scale detection is introduced in SSD which can detect objects of varying sizes in an image. Later, in order to improve detection accuracy, Lin et. al [7] proposes Retina Network (RetinaNet) by combining an SSD and FPN (feature pyramid network) to increase detection accuracy and reduce class imbalance. One-stage detectors have higher speed but trades off the detection performance but then only are preferred over two-stage detectors.

Like object detection, face detection adopts the same architectures as one-stage and two-stage detectors, but in order to improve face detection accuracy, more face-like features are being added. However, there is occasional research focusing on face mask detection. Some already existing face mask detectors have been modeled using OpenCV, Pytorch Lightning, MobileNet, RetinaNet and Support Vector Machines. Here, we will be discussing two projects. One project used Real World Masked Face Dataset (RMFD) which contains 5,000 masked faces of 525 people and 90,000 normal faces [8]. These images are 250 x 250 in dimensions and cover all races and ethnicities and are unbalanced. This project took 100 x 100 images as input, and therefore, transformed each sample image when querying it, by resizing it to 100x100. Moreover, this project uses PyTorch then they convert images to Tensors, which is the base data type that PyTorch can work with. RMFD is imbalanced (5,000 masked faces vs 90,000 non-masked faces). Therefore, the ratio of the samples in train/validation while splitting the dataset was kept equal using the train_test_split function of sklearn. Moreover, to deal with unbalanced data, they passed this information to the loss function to avoid unproportioned step sizes of the optimizer. They did this by assigning a weight to each class, according to its representability in the dataset. They assigned more weight to classes with a small number of samples so that the network will be penalized more if it makes mistakes predicting the label of these classes. While classes with large numbers of samples, they assigned to them a smaller weight. This makes their network training agnostic to the proportion of classes. The weights for each class were chosen using the formula below:

$$Class\,Weight = 1 - \frac{Class\,Cardinality}{Cardinalities\,of\,all\,classes}$$

To load the data efficiently this project used the data loader. For instance, in this project, they used the PyTorch lighting, and to load them for training and validation they divided data into 32 batches and assigned the works of loading to the 4 number of workers, and this procedure allowed them to perform multi-process data loading. Like most of the projects, this project also used Adam optimizer. If any Model has a high rate of learning, it learns faster, but it bounces a lot to reach the global minima and may diverge from the global minima. However, a small learning rate may take considerably lower time to train, but it reaches to the global minima. If the loss of the model declines quickly for any learning rate, then that learning rate would be the best learning rate. However, it seems that this project considered the 0.00001 learning rate would be the best for their model so that it could work efficiently. To train the model they defined a model checkpointing callback where they wanted to save the best accuracy and the lowest loss. They tried to train the model for 10 epochs and after finding optimal epoch, they saved the model for 8 epochs to test on the real data. To get rid of the problem of occlusions of the face which causes trouble face detectors to detect masks in the images, they used a built-in OpenCV deep learning face detection model. For instance, the Haar-Cascade model could be used but the problem of the Haar-Cascade model is that the detection frame is a rectangle, not a square. That is why, without capturing the portion of the background, the face frame can fit the entirety of the face, which can interfere with the face mask model predictions.

In the second project [9], a dataset was created by Prajna Bhandary using a PyImageSearch reader. This dataset consists of 1,376 images belonging to all races and is balanced. There are 690 images with masks and 686 without masks. Firstly, it took normal images of faces and then created a customized computer vision Python script to add face masks to them. Thereby, it created a real-world applicable artificial dataset. This method used the facial landmarks which allow them to detect the different parts of the faces such as eyes, eyebrows, nose, mouth, jawline etc. To use the facial landmarks, it takes a picture of a person who is not wearing a mask, and, then, it detects the portion of that person's face. After knowing the location of the face in the image, it extracted the face Region of Interest (ROI). After localizing facial landmarks, a picture of a mask is placed into

the face. In this project, embedded devices are used for deployment that could reduce the cost of manufacturing. MobileNetV2 architecture is used as it is a highly efficient architecture to apply on embedded devices with limited computational capacity such as Google Coral, NVIDIA Jetson Nano. This project performed well, however, if a large portion of the face is occluded by the mask, this model could not detect whether a person is wearing a mask or not. The dataset used to train the face detector did not have images of people wearing face masks as a result, if the large portion of faces is occluded, the face detector would probably fail to detect properly. To get rid of this problem, they should gather actual images of people wearing masks rather than artificially generated images.

3 Methodology

3.1 Dataset

The dataset which we have used consists of 3835 total images out of which 1916 are of masked faces and 1919 are of unmasked faces. All the images are actual images extracted from Bing Search API, Kaggle datasets and RMFD dataset. From all the three sources, the proportion of the images is equal. The images cover diverse races i.e Asian, Caucasian etc. The proportion of masked to unmasked faces determine that the dataset is balanced.

We need to split our dataset into three parts: training dataset, test 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 hyperparameters (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. Data is split as per a split ratio which is highly dependent on the type of model we are building and the dataset itself. 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. If the model has a lot of hyperparameters that can be tuned, then we need to take a higher amount of validation dataset. Models with a smaller number of hyperparameters are easy to tune and update, and so we can take a smaller validation dataset.

In our approach, we have dedicated 80% of the dataset as the training data and the remaining 20% as the testing data, which makes the split ratio as 0.8:0.2 of train to test set. Out of the training data, we have used 20% as a validation data set. Overall, 64% of the dataset is used for training, 16% for validation and 20% for testing.

3.2 Architecture

The working of the Single Shot Detector algorithm relies on an input image with a specified bounding box against the objects. The methodology of predicting an object in an image depends upon very renowned convolution fashion. For each pixel of a given image, a set of default bounding boxes (usually 4) with different sizes and aspect ratios are evaluated. Moreover, for all the pixels, a confidence score for all possible objects are calculated with an additional label of 'No Object'. This calculation is repeated for many different feature maps.

In order to extract feature maps, we usually use the predefined trained techniques which are used for high quality classification problems. We call this part of the model a base model. For the SSD, we have VGG-16 network as our base model. At the training time, the bounding boxes evaluated are compared with the ground truth boxes and in the back propagation, the trainable parameters are altered as per requirement. We truncate the VGG-16 model just before the classification layer and add feature layers which keep on decreasing in size. At each feature space, we use a kernel to produce outcomes which depicts corresponding scores for each pixel whether there exists any object or not and the corresponding dimensions of the resulting bounding box.

VGG-16 is a very dense network having 16 layers of convolution which are useful in extracting features to classify and detect objects. The reason for the selection is because the architecture

consists of stacks of convolutions with 3x3 kernel size which thoroughly extract numerous feature information along with max-pooling and ReLU to pass the information flow in the model and adding non linearity respectively from the given image. For additional nonlinearity, it uses 1x1 convolution blocks which does not change the spatial dimension of the input. Due to the small size filters striding over the image, there are many weight parameters which end up giving an improved performance.

The block diagram [10] shows the working functionality of SSD. At the input end, we can see the VGG-16 being used as the base model. Some additional feature layers are added at the end of the base model to take care of offsets and confidence scores of different bounding boxes. At end part of the figure, we can see the layers being flattened to make predictions for different bounding boxes. At the end, non maximum suppression is used whose purpose is to remove duplicate or quite similar bounding boxes around same objects. There may be situations where the neighboring pixel also predicts a bounding box for an object with a bit less confidence which is finally rejected.

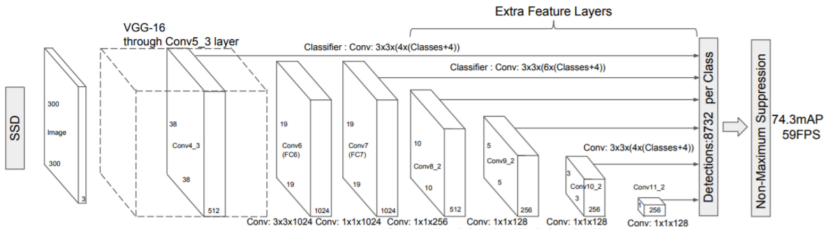

Figure 2: Working of SSD.

The problem can be solved in two parts: first detecting the presence of several faces in a given image or stream of video and then in the second part, detect the presence or absence of face mask on face. In order to detect the face, we have used the OpenCV library. The latest OpenCV includes a Deep Neural Network (DNN) module, which comes with a pre-trained face detection convolutional neural network (CNN). The new model enhances the face detection performance compared to the traditional models. Whenever a new test image is given, it is first converted into BLOBS (Binary Large Object refers to a group of connected pixels in a binary image) and then sent into the pre-trained model which outputs the number of detected faces. Every face detected comes out with a level of confidence which is then compared with a threshold value to filter out the irrelevant detections. After we have the faces, we need to evaluate the bounding box around it and send it to the second part of the model to check if the face has a mask or not.

The second part of the model is trained by us using a dataset consisting of images with mask and without mask. We have used Keras along with Tensorflow to train our model. First part of the training includes storing all labels of the images in a Numpy array and the corresponding images are also reshaped (224, 244, 3) for the base model. Image augmentation is a very useful technique because it increases our dataset with images with a whole new perspective. Before inputting, we performed the following image augmentations randomly: rotations up to 20 degrees, zooming in and out up to 15%, width or height shift up to 20%, up to 15 degrees shear angle in the counterclockwise direction, flip inputs horizontally and points outside the boundaries of the inputs are filled from the nearest available pixel of the input. For the image classification, it is now a common practice to use transfer learning which means using a model which has been pre-trained on millions of labels before and it has been tested that this method results in significant increase in accuracy. Obviously, the assumption here is that both the problems have sufficient similarity. It uses a well-structured and deep neural network that has been trained on a large amount of data set. Due to somewhat same nature of the problem, we can use the same weights which have the capability to extract features and later in the deep layers, convert those features to objects.

The base model that we have used here is MobileNetV2 with the given 'ImageNet' weights. ImageNet is an image database that has been trained on hundreds of thousands of images hence it helps a lot in Image classification. For the base model, we truncate the head and use a series of our self-defined layers. We used an average pooling layer, a flatten layer, a dense layer with output shape (None, 128), and activation ReLU, a 50% dropout layer for optimization, finally another dense layer

with output shape (None, 2), and Sigmoid activation is used. The overall process flow diagram of the algorithm is shown below.

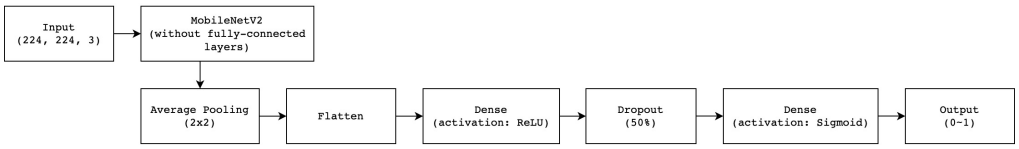

Figure 3: Process Flow diagram of the model.

3.3 Training

At the training time, for each pixel, we compare the default bounding boxes having different sizes and aspect ratios with ground truth boxes and finally use Intersection over Union (IoU) method to select the best matching box. IoU evaluates how much part of our predicted box match with the ground reality. The values range from 0 to 1 and increasing values of IoU determine the accuracies in the prediction; the best value being the highest value of IoU. The equation and pictorial description of IoU is given as follow:

$$IoU(B_1, B_2) = \frac{B_1 \cap B_2}{B_1 \cup B_2}$$

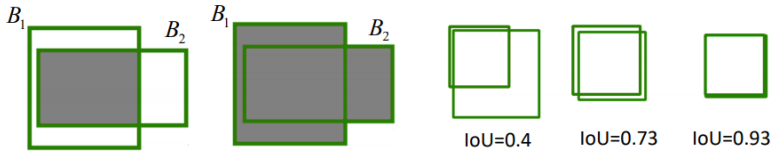

Figure 4: Pictorial representation of IoU.

3.4 Hyperparameters

A hyperparameter is a parameter or a variable we need to set before applying an algorithm into a dataset. These parameters express the "High Level" properties of the model such as its complexity or how fast it should learn. Hyperparameters are fixed before the actual training process begins. They can be divided into two categories: optimizer hyperparameters and model hyperparameters.

Optimizer parameters help us to tune or optimize our model before the actual training process starts. Some common optimizer hyperparameters are as follows. Learning rate is a hyperparameter that controls how much we are adjusting the weights of our neural network with respect to the gradient. Mini-batch size is a hyperparameter that influences the resource requirements of the training and impacts training speed and number of iterations. Epochs are the hyperparameters that determine the frequency of running the model. One epoch is when an entire dataset is passed forward and backward through the neural network only once. Model hyperparameters are parameters that are more involved in the architecture or structure of the model. They help us to define our model complexity based on the different layers like the input layer, hidden layer, and output layer of a neural network.

Initially, we trained with different values of hyperparameters by changing one and keeping the other constant and noted down the results in each case. We selected the hyperparameters that produced better performance through evaluation metrics. We have chosen the hyperparameters as follows: initial learning rate is taken as 0.0001, batch size is taken to be 32 and number of epochs as 20. In our case, the target size is also one of the hyperparameters which we kept (224, 224, 3) as it is default input shape of MobileNetV2.

3.5 Loss function

The loss of the overall detection problem can be broken into two main subsets: localization loss and confidence loss. The localization loss is just the difference between the default predicted bounding box and ground truth bounding box (g). For a given center (cx, cy), we try to alter the width and height of the box such as to decrease the loss. Respectively, the equations for the localization and confidence losses can be defined as:

$$L_{loc}(x, l, g) = \sum_{i \in Pos}^{N} \sum_{m \in \{cx, cy, w, h\}} x_{ij}^{k} \text{smooth}_{L1}(l_i^m - \hat{g}_j^m)$$

$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \qquad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h$$

$$\hat{g}_j^w = \log(\frac{g_j^w}{d_i^w}) \qquad \hat{g}_j^h = \log(\frac{g_j^h}{d_i^h})$$

and

$$L_{locf}(x, c) = -\sum_{i \in Pos}^{N} x_{ij}^p log(\hat{c}_i^p) - \sum_{i \in Neg} log(\hat{c}_i^0) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$$

The notations used in the equations are as follows:

- g: ground truth bounding box
- l: predicted box
- d: default bounding box
- x_{ij}^p: matching i^{th} predicted box with j^{th} default box with "p" category.
- cx, cy: distance from centre of box in both x and y direction
- w, h = width and height with respect to image size
- c: confidence of presence of object or not.

Confidence loss is the just measure of how high the probability of the presence of an object is, when there exists an object. Similarly, the localization loss is the measure of how much a predicted box differs from the ground truth box in dimensions. Our model will try to push down both losses by predicting the presence of object and then correctly classifying it to the right class.

4 Evaluation

4.1 Testing

We tried using three different base models for detecting 'mask' or 'no mask'. The exercise was done to find the best fit model in our scenario. The evaluation process consists of first looking at the classification report which gives us insight towards precision, recall and F1 score. The equations of these three metrics are as follows:

$$Precision = \frac{True\ Positives}{Positives + False\ Positives}$$

$$Recall = \frac{True\ Positives}{Positives + False\ Negatives}$$

$$Accuracy = \frac{True\ Positives + True\ Negatives}{Positives + Negatives}$$

Using these three metrics, we can conclude which model is performing most efficiently. The second part consists of plotting the train loss, validation loss, train accuracy and validation accuracy which also proves helpful in choosing a final model. The results of different choices are shown below.

4.1.1 Xception

The complete form of Xception is "Extreme Inception". Basically, the Xception architecture [11] is designed with depth wise separable convolution layers with residual connections. These convolutional layers are linearly stacked with each other. This architecture is easy to define and easy to modify. If we compare it with Inception V2 and V3 then Inception V2 and V3 are much more complex to define.

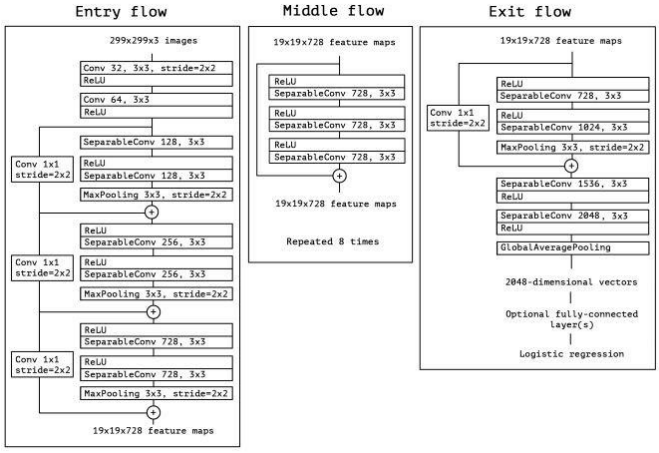

Figure 5: The Xception Architecture.

```
[INFO] evaluating network...
classification report:

              precision    recall  f1-score   support

   with_mask       0.98      0.99      0.99       384
without_mask       0.99      0.98      0.99       386

    accuracy                           0.99       770
   macro avg       0.99      0.99      0.99       770
weighted avg       0.99      0.99      0.99       770
```

Figure 6: Classification report for Xception.

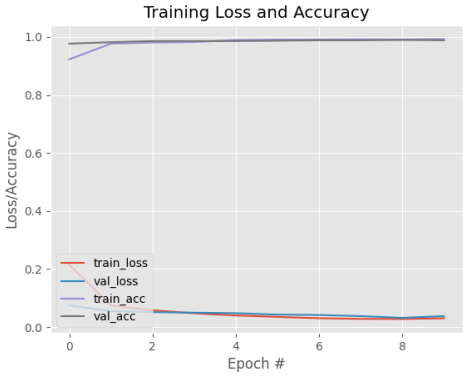

Figure 7: Loss graph for Xception.

4.1.2 MobileNetV2

MobileNetV2 [12] is an architecture of bottleneck depth-separable convolution building of basic blocks with residuals. It has two types of blocks. The first one is a residual block with stride of 1. Second one is also residual block with stride 2 and it is for downsizing.

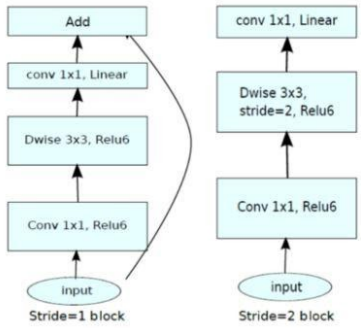

Figure 8: Convolutional Blocks of MobileNetV2.

In the discussion of the layer part, there are three layers for both blocks. First one is 1x1 convolution with ReLU6. Depth wise convolution is in the second layer and again in the third layer there is a 1x1 convolution but without any non-linearity.

Input	Operator	Output
$h \times w \times k$	1x1 conv2d , ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

Figure 9: Bottleneck residual block transforming from k to k' channels, with stride s and factor t.

classification report:

	precision	recall	f1-score	support
with_mask	0.98	1.00	0.99	384
without_mask	1.00	0.98	0.99	386
accuracy			0.99	770
macro avg	0.99	0.99	0.99	770
weighted avg	0.99	0.99	0.99	770

Figure 10: Classification report for MobileNetV2.

Figure 11: Loss graph for MobileNetV2.

4.1.3 ResNet50

It is a convolutional neural network which is 3 layers deep. The bottleneck class implements this three-layer block. From the ImageNet database we can train millions of images and then load it as a pre-trained version of a network. This network can classify images into several object categories, such as face, car, bike and many animals.

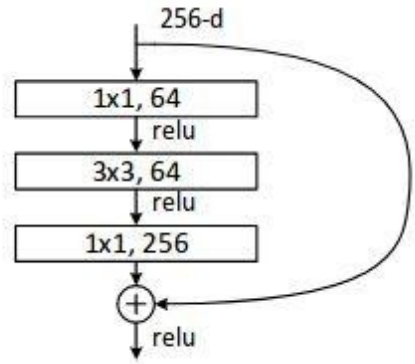

Figure 12: ResNet50 3 layer block.

classification report:

	precision	recall	f1-score	support
with_mask	0.87	0.81	0.84	384
without_mask	0.83	0.88	0.85	386
accuracy			0.85	770
macro avg	0.85	0.85	0.85	770
weighted avg	0.85	0.85	0.85	770

Figure 13: Classification report for ResNet50.

Figure 14: Loss graph for ResNet50.

After observing these results, we can conclude that the lowest performance amongst these was of ResNet50. Xception and MobileNetV2 both performed quite similar and to select the best from them was the 100% precision of unmasked faces by MobileNetV2. Since our detector can take no risk by falsely concluding that an unmasked face is a masked face, precision is the key metric to decide between these models. Since, the best performance was given by MobileNetV2, therefore, we decided to use MobileNetV2 as a base model for our face mask detector algorithm.

4.2 Inference

We implemented our model on images containing one and more faces. We also implemented it on videos and live video streams by removing and wearing masks one by one. Some screenshots of the results are shown below:

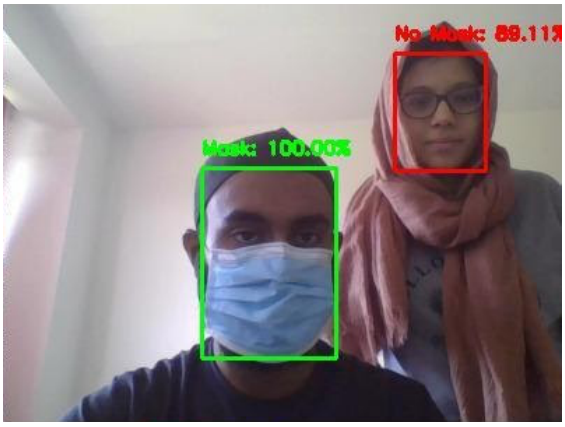

Figure 15: One person with mask and one person without mask.

Figure 16: Both persons with masks.

Figure 17: Both persons without masks.

5 Limitations

There were not many challenges faced but the two problems that were time consuming and made the tasks tedious are discussed as follows. One was the excessive data loading time in Google Colab Notebook while loading the dataset into it. Since the runtime restarting refreshes all the cells, the cell for dataset loading took most of the time while running. Secondly, the access problem in Google Colab Notebook: it did not allow the access of webcam which posed a hurdle in testing images and live video stream through Google Colab Notebook. Therefore, we had to run the code locally on the computer through which we tested the code on the live video stream.

6 Future Work

More than fifty countries around the world have recently initiated wearing face masks compulsory. People have to cover their faces in public, supermarkets, public transports, offices, and stores. Retail companies often use software to count the number of people entering their stores. They may also like to measure impressions on digital displays and promotional screens. We are planning to improve our Face Mask Detection tool and release it as an open-source project. Our software can be equated to any existing USB, IP cameras, and CCTV cameras to detect people without a mask. This detection live video feed can be implemented in web and desktop applications so that the operator can see notice messages. Software operators can also get an image in case someone is not wearing a mask. Furthermore, an alarm system can also be implemented to sound a beep when someone without a mask enters the area. This software can also be connected to the entrance gates and only people wearing face masks can come in.

7 Conclusion

To mitigate the spread of COVID-19 pandemic, measures must be taken. We have modeled a face mask detector using SSD architecture and transfer learning methods in neural networks. To train, validate and test the model, we used the dataset that consisted of 1916 masked faces images and 1919 unmasked faces images. These images were taken from various resources like Kaggle and RMFD datasets. The model was inferred on images and live video streams. To select a base model, we evaluated the metrics like accuracy, precision and recall and selected MobileNetV2 architecture with the best performance having 100% precision and 99% recall. It is also computationally efficient using MobileNetV2 which makes it easier to install the model to embedded systems. This face mask detector can be deployed in many areas like shopping malls, airports and other heavy traffic places to monitor the public and to avoid the spread of the disease by checking who is following basic rules and who is not.

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, vol. 1. IEEE, 2001, pp. I–I.

[2] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2014, pp. 580–587.

[3] R. Girshick, "Fast r-cnn," in Proceedings of the IEEE international conference on computer vision, 2015, pp. 1440–1448.

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in Advances in neural information processing systems, 2015, pp. 91–99.

[5] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in European conference on computer vision. Springer, 2016, pp. 21–37.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 779–788.

[7] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," 2017.

[8] Haddad, J., 2020. How I Built A Face Mask Detector For COVID-19 Using Pytorch Lightning. [online] Medium. Available at: https://towardsdatascience.com/how-i-built-a-face-mask-detector-for-covid-19-using-pytorch-lightning-67eb3752fd61.

[9] Rosebrock, A., 2020. COVID-19: Face Mask Detector With Opencv, Keras/Tensorflow, And Deep Learning - Pyimagesearch. [online] PyImageSearch. Available at: https://www.pyimagesearch.com/2020/05/04/covid-19-face-mask-detector-with-opencv-keras-tensorflow-and-deep-learning/ .

[10] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in European conference on computer vision. Springer, 2016, pp. 21–37.

[11] Francois Chollet "Xception: Deep Learning with Depthwise Separable Convolutions" in Proceedings of the IEEE conference on computer vision and pattern recognition(CVPR), 2017, pp. 1251-1258.

[12] M. Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in Proceedings of the IEEE conference on computer vision and pattern recognition(CVPR), 2018.

[13] Keras code inspiration taken from https://github.com/chandrikadeb7/Face-Mask-Detection