Face Mask Detection Model for Covid-19 Pandemic Using Convolutional Neural Network & Deep Learning Techniques

Jawed Ahmed[1] Ilmaan Zia[1]

[1]School of Engineering Science &amp; Technology, Jamia Hamdard, New Delhi-110062, India.

Abstract. In the present scenario, due to covid-19, there are no efficient face mask detection applications which are now in high demand for transportation means densely populated areas, residential districts, and other enterprises to ensure safety. This model could potentially be used to help ensure your safety and the safety of others. In this paper, a Convolutional Neural Network model using Deep Learning & Machine Learning tools will be presented. The proposed model was build and trained in Google colab since it offers free GPU, and training the model on Colab was much faster. Some necessary libraries are imported into the model such as OpenCV, Keras, and Tensorflow. The Dataset was directly imported from Kaggle on the Colab environment of masked and non-masked people. Image augmentation was performed to increase the size of the data set for zooming, rotational, and horizontal features of the images. CNN was implemented to classify images into their respective categories as CNN is meant to be best for image classification purposes. Two different graphs are plotted for training and validation loss and for training and validation accuracy to depict how the model performs during the training phase. This model is also used to predict a new set of data that can be uploaded from a local machine. This model can also be used to detect face mask on video streams as well.

Keywords.COVID-19, Face Mask Detection, Deep learning, Convolutional Neural Network, Machine Learning

1. Introduction

The Covid-19 pandemic has set a new rule to wear a face mask all over the World so as to ensure self and others safety, as wearing a mask is one of the effective methods to reduce the spread of the virus, earlier people used to wear a face mask to protect themselves from pollution, but this virus led people to wear a mask in non-polluted areas as well. It is scientifically proven that wearing a face mask reduces Covid-19 transmission. The virus spreads through close contact with infected persons.

Since the COVID-19 virus has emerged there is a massive development of data related to the virus-infected persons and so there is a massive rise in the worldwide scientific and medical research, and talking of the IT sector there is seen an extraordinary rise as there massive data created every day. Data science techniques as Artificial Intelligence (AI), Machine Learning, and And Deep Learning have helped to fight COVID-19 in many ways. These technologies allow medical researchers to evaluate and analyze the vast amount of data for the spread of COVID-19 to provide estimation by an early warning for such potential pandemics. And so the responsible body can take measures and detect future predictions.

Many Governments have enforced the law for wearing a mask as mandatory in public as well as in private places to fight the increasing challenges and risks that can be caused by this virus. But it is very often that seen people are not following the law strictly and avoiding wearing masks that are resulting in the rise of COVID-19 cases. And for places with a huge population, it has become extremely difficult for the government to monitor the new rules and laws introduced as in many cases it is seen the government officials get exposed to the virus and get affected online monitoring has also become difficult due to large groups and large gathering.

In this paper, a face mask detection model has been introduced to build using Convolutional Neural Network by Deep Learning and Machine Learning techniques. The proposed model can be used for surveillance via cameras to fight with COVID-19 by detecting and classifying between people not wearing masks and people wearing masks, which would reduce human interaction and human effort to a great extent. Deep Learning Technique is used for feature extraction by Deep Convolutional Neural Network implemented with some libraries as Keras, Tensorflow, and OpenCV. The model is trained and then Validated to achieve the highest accuracy, and two different Graphs have been plotted for Training and Validation Loss and for the Training and Validation Accuracy. A new set of data have also been used to make predictions and train This trained model is also used to make predictions on a new set of data that can be uploaded by the user. This model is also used to detect face mask on video streams as well.

2. Related Works

In General, most of the publications have laid their main focus on image classification of the human face, construction, and identity recognition but as wearing a face mask is almost new for the current year hence not many models-efficient models are created to detect face mask for COVID-19. In [1] Guosheng Hu has shown the use of a Convolutional Neural Network to classify an image and explained how it can be used for face recognition. Weihong Wang in [2] have characterized Convolutional layers and Pooling layers in a Convolutional Neural Network, then outputted to a fully connected layer.

Medical researchers and scientists have proven that wearing a face mask can reduce the risk of COVID-19 and helps in reducing its transmission to a great extent [3]. And so the government of many countries has laid down new policies so as to reduce the spreading of the COVID-19 virus and have made it mandatory for people to wear a face mask in public places. And officials are making sure that everyone is following the rule strictly.

In [4]Mohamed Loey used YOLO-v2 with ResNet-50 to create a deep learning model for medical face mask detection, the YOLO-v2 with ResNet-50 provided high average precision. And to improve the object detection process, he used mean IoU to estimate the best number of anchor boxes. It positively achieved a result that was concluded that Adam optimizer achieved the highest average precision percentage of 81%.The authors proposed a model and dataset to find the normal and masked face. They introduced a large dataset Masked Faces, which has 35, 806 masked faces. The proposed model based on a convolutional neural network called LLE-CNNs, which consists of three modules. The works showed that LLE-CNNs using MAFA achieved the average precision equal to 76.1%.

In the following research the focused aim was to recognize and detect the people who are not wearing a face mask and provide it to the concerned authority so as to help in reducing the transmission and spreading of COVID-19. A face mask-wearing identification method has been developed, that is able to classify between different mask-wearing conditions. The categories are wearing a face mask and not wearing a face mask.

3. Dataset Characteristics

This research conducted its experiment on Datasets that were taken from Kaggle5 uploaded by Prithwiraj Mitra. This dataset contains about 1006 equally distributed images of 2 distinct types, face masked and non-face masked people. The author [5] further distributed the data set into 3 subdirectories of Test, Train, and Validation set [figure1].

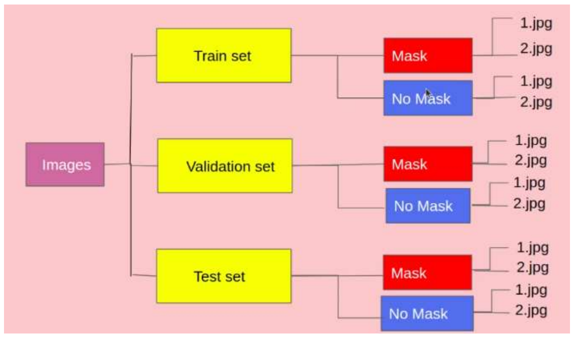


Figure 1 Dataset Distribution

The Test data set have been divided into two subdirectories of the mask and non masked data set which contains 50 images in each set [figure2].

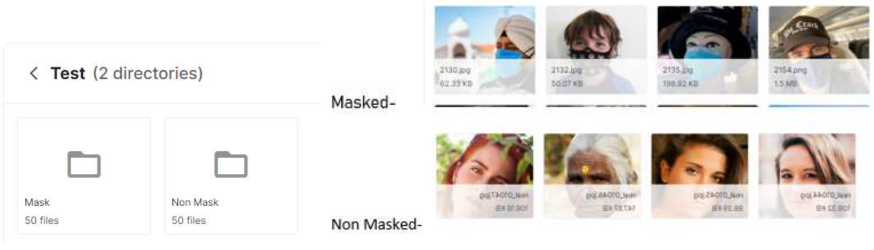


Figure 2 Test Dataset

The Train data set have been divided into two subdirectories of the mask and non masked data set which contains 300 images in each set [figure3].

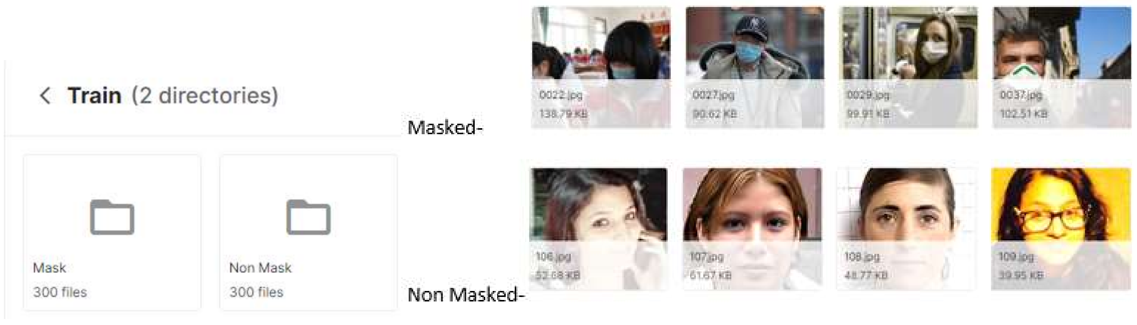


Figure 3 Train Dataset

And lastly in the Train data set have been divided into two subdirectories of the mask and non masked data set which contains 153images in each set [figure4].

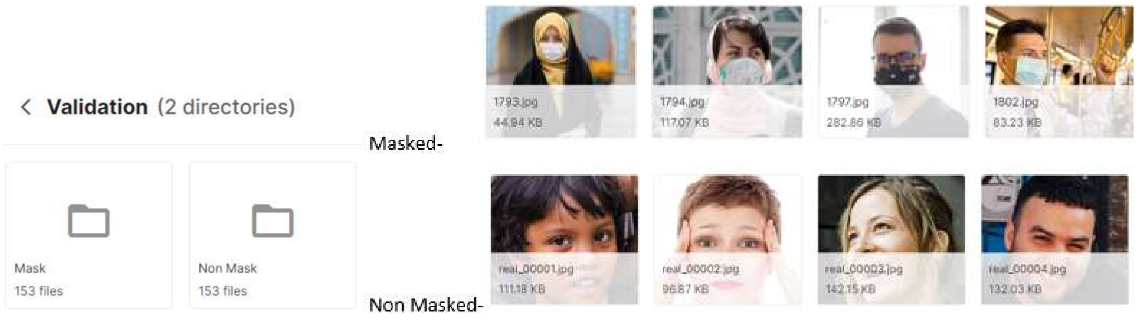


Figure 4 Validation Dataset

The Dataset was downloaded and used on the google Collab directory in the runtime session. And four directories of the main directory Train directory, Test directory, and Validation directory were created distinctly and joined using the path in Google Collab.

4. Proposed Model

The introduced model is built on Convolutional Neural Network by Deep Learning tools and technique and the data set used is divided into three categories of Training, Validation, and Testing. This data set would be used during the testing and training purposes of the Neural Network.

4.1.1. Training Set

The training set is the set of data that is used to train the model. So during each epoch, the model will be trained over and over again on the same data on our training set, and it will continue to learn about the features of this data so that the trained model can be deployed and have it accurately predict on a new set of data that it has never seen before. It will be making all these predictions based on what it learned during the training process on the training data [figure 5].

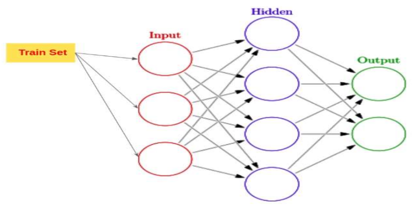


Figure 5 Training Phase

4.1.2. Validation Set

The Validation set is the set of data separate from the training data that is used to validate the model during training. As I mentioned earlier, that with each epoch during training, the model will be trained on the data, on the training set, well, it will also be simultaneously validating all the data in the validation set. So during the training process, the model will be classifying the output for each input in the training set, and also during training, the model will be classifying each input from the validation set as well.

The validation data in the validation set is separate from the data in the training set. So when it's by validating on this data, this data does not consist of samples that a model is already familiar with from training one.

So the main reason why we need a validation data set is to ensure that our model is not over fitting to the data and the training set.

So during training, if it is also validating the model and our validation set and see that the result is giving for the validation set are just as good as the results it's giving for the training data, then it can conclude that our model is not over fitting [figure6].

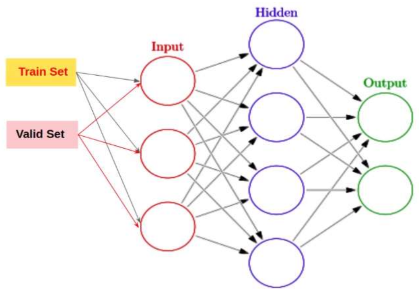


Figure 6 Validation Phase

4.1.3. Test Set

The Test set is the set of data that is used to test the model after the model has already been trained. The test set is different from both the train set and the validation set and after the model is trained and validated using the training and validation set, then the model will be used to predict the output of the data in the test set.

So the main difference between the test set and the remaining two sets is that the test should not be labeled the training and the validation set has to be labeled so that we can see the matrices given during training, like the accuracy and loss from each epochs.

So when the model is predicting on the unlabeled data in test set this would be the same type of process that would be used if we were to deploy our model into the real field [figure7].

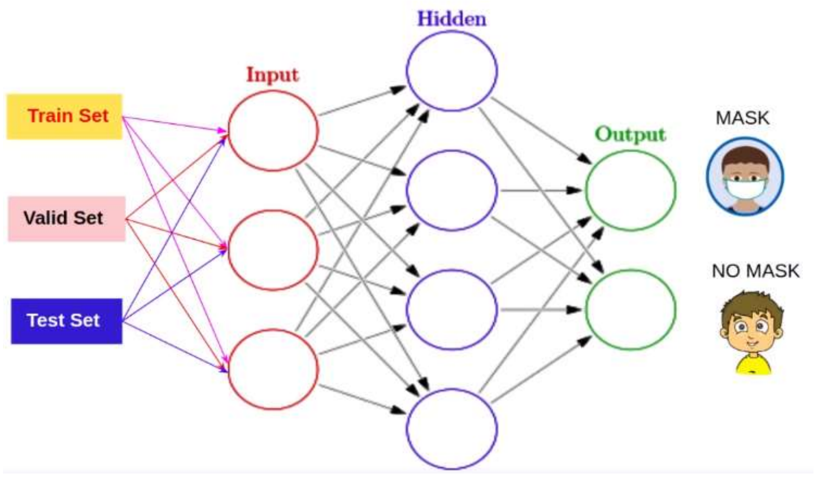


Figure 7 Test Phase

4.2. Image Visualization using Matplotlib

Images from the Train data set are visualized. A grid of 16 images is created, where 8 images are visualized are created from the mask directory and the other 8 images are visualized from the no mask directory of the Train data set, taking the number of columns and number of rows as 4[figure8].

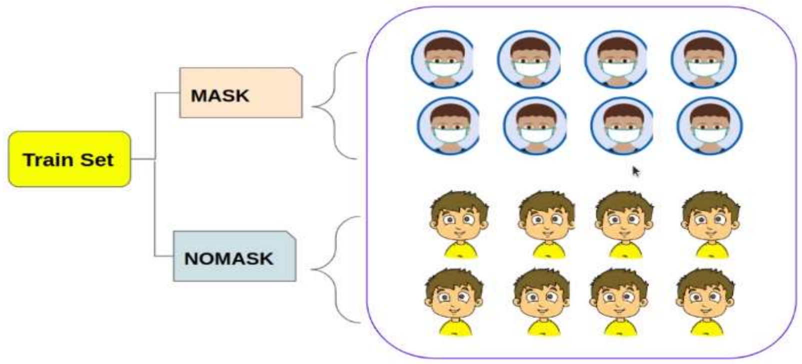
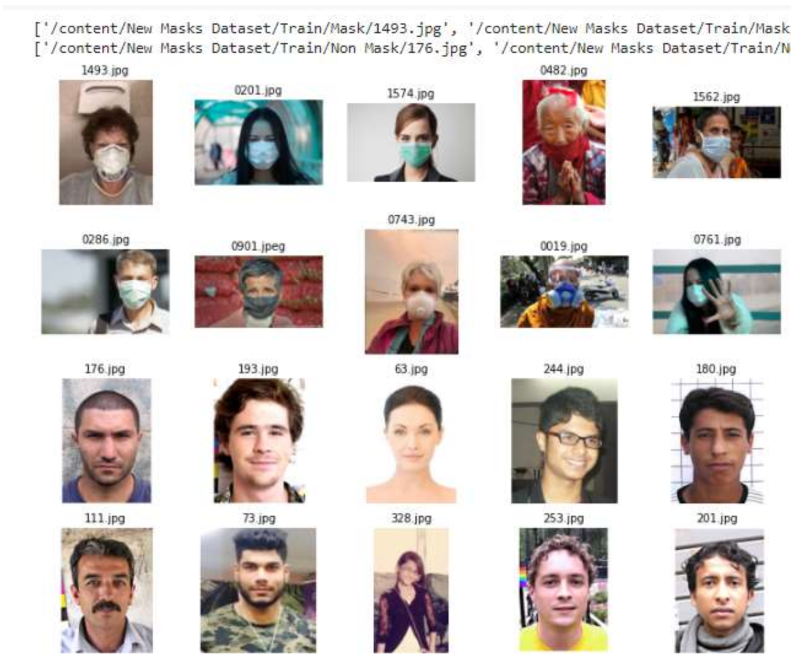


Figure 8 Dataset Distribution

From the Matplotlib image module is imported, the image module in the Matplotlib package provides functionalities required for loading, rescaling, and displaying the image.



```
['/content/New Masks Dataset/Train/Mask/1493.jpg', '/content/New Masks Dataset/Train/Mask
['/content/New Masks Dataset/Train/Non Mask/176.jpg', '/content/New Masks Dataset/Train/N
```

Figure 9 Dataset Preview

4.3.1. Image Data Augmentation

Data Augmentation is the process when new data is created based on the modifications of existing data. A new augmented data is created by the reasonable modification of the data in the training set.

Image Augmentation is a simple but very powerful tool to help you avoid over fitting in your data. The concept is very simple, though if you have limited data, then the chances of you having data to match potential future predictions are also limited. And logically, the less data you have the less chance you have of getting accurate predictions for data that your model hasn't seen yet.

To put it simply, if you are training a model to spot cats and your model has never seen what cats look like when lying down it might not recognize that in the future.

For example, we can augment image data by flipping the images either horizontally or vertically. We can rotate the image, we can zoom the images in or out [figure10], and all these are the common augmented techniques. From a single image, we can generate around three to four images with various modifications.

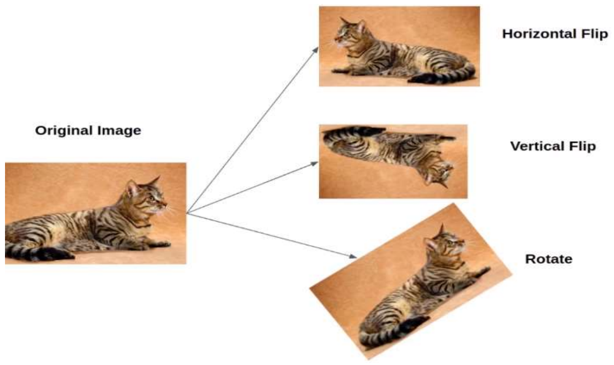


Figure 10 Data Augmentation

Image Data Augmentation is needed, as first thing is that more data can be added to the training set. Suppose there is a relatively small amount of samples or data to include in our training set, and it's difficult to get more, then we could create new data from the existing dataset using data augmentation to create more samples. And additionally, data augmentation can be used to reduce over fitting. If the model is over fitting, one technique to reduce over fitting is that more data can be added and more such data can be created using data augmentation techniques.

The more data we have, to train our model on the more it will be able to learn from the training set. Also, with more data, it will be adding more diversity to the training set as well.

The general idea of augmentation allows us to add more data to our training set that's similar to the data that we already have, but it's just reasonably modified to some extent so it's not the exact same.

4.3.2. Over fitting

Over fitting occurs when the model becomes really good at being able to classify or predict the data that was included in the training set. But it is not so good at classifying data that it wasn't trained on. It can be said that the model is over fitting based on the matrices that are given for the training and validation data during the training process.

During the training, we get both the training, accuracy, and loss as well as the validation accuracy and loss. If the validation matrices are way worse than the training matrices, then that's the indication that the model is over fitting. It can also be concluded that the model is over fitting if during training the model matrices are good.

Over fitting, simply means that the model is not able to generalize well, meaning the model has learned the features of the training set extremely well. But if we give the model any data that slightly deviates from the data used during training, it's unable to generalize and predict the output. Over fitting is a very common issue while developing models.

We can identify with the model is over fitting or not by the graphs of loss and accuracy. In the following graph [figure11], there is a training accuracy of around 97 percent, but validation accuracy is only about 60 percent. While in the loss graph, the training loss decreases, but the validation loss increases. This is a clear indication that the model is over fitting.

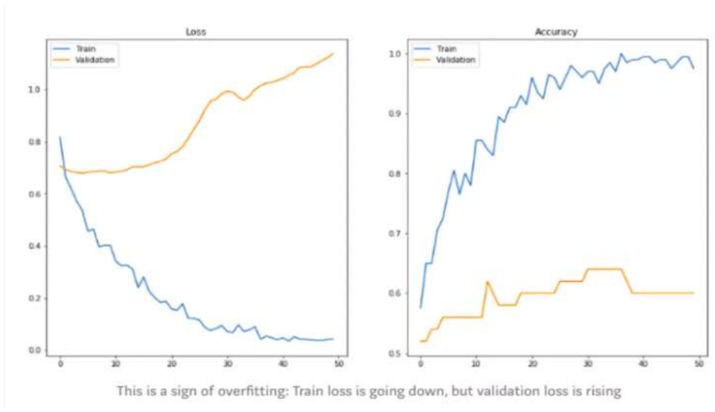


Figure 11Overfitting model graph

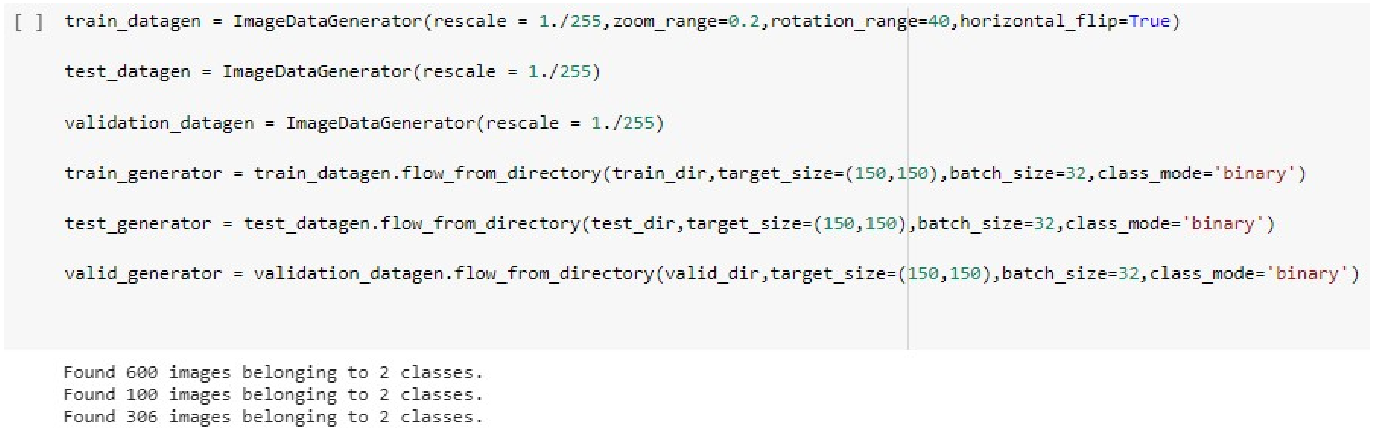
This is a clear sign that the model is over fitting. It's learning the train data really well but fails to generalize the knowledge to the test data.

4.3.3. Keras Deep Learning Library – Image Data Generator

Keras comes bundled with many helpful utility functions and classes to accomplish all kinds of common tasks in any machine learning pipelines and one commonly used class is the Image Data Generator.

It generates batches of image data with real time data augmentation. Data augmentation is supported in Keras Deep Learning Library via the Image Data Generator class.

The entire data set is looped over in each epoch and the images in the dataset are transformed as per the options and value selected, these transformations are performed in memory, hence no additional storage is required. Keras image data library is used to make the data ready for the model.



```
[ ]  train_datagen = ImageDataGenerator(rescale = 1./255,zoom_range=0.2,rotation_range=40,horizontal_flip=True)

     test_datagen = ImageDataGenerator(rescale = 1./255)

     validation_datagen = ImageDataGenerator(rescale = 1./255)

     train_generator = train_datagen.flow_from_directory(train_dir,target_size=(150,150),batch_size=32,class_mode='binary')

     test_generator = test_datagen.flow_from_directory(test_dir,target_size=(150,150),batch_size=32,class_mode='binary')

     valid_generator = validation_datagen.flow_from_directory(valid_dir,target_size=(150,150),batch_size=32,class_mode='binary')
```

```
Found 600 images belonging to 2 classes.
Found 100 images belonging to 2 classes.
Found 306 images belonging to 2 classes.
```

Figure 12 Image Data Generarot

4.4.1. Convolutional Neural Network

Convolutional Neural Network, also known as the CNN is the artificial neural network that has so far been most popularly used for analyzing images. Generally, we can think of CNN as an artificial neural network that has some type of specialization for being able to pick out or detect patterns and make sense of them. This pattern detection is what makes CNN so useful for image analysis.

CNN has hidden layers called convolutional layers [figure 13], and these layers are precisely what makes a CNN. They also have other non-convolutional layers as well. But the basis of CNN is the convolutional layers.

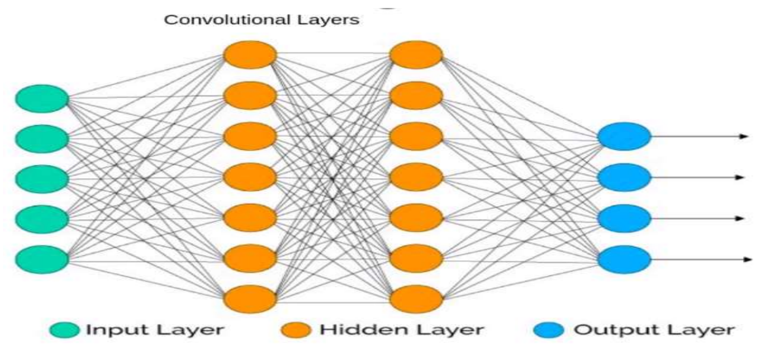


Figure 13 CNN Architecure

The function of the convolutional layers is, that just like any other layer, a convolutional receives an input then transforms the input in some way, and outputs the transform input to the next layer. With the convolution layer, this transformation is a convolution operation.

With each convolutional layer, we need to specify the number of filters the layers should have. These filters are actually what detects the patterns. When this convolution layer receives the input, the filter will slide over each filter matrix pixels from the input image itself under its lead over every matrix pixels blocks of pictures from the entire image [6]. This sliding is called convolving. A filter is going to convolve across each block of pixels from the input. Convolution is a dot product of a kernel or a filter and a patch of the image of the same size. Different values of the filter matrix will produce different feature maps for the same input.

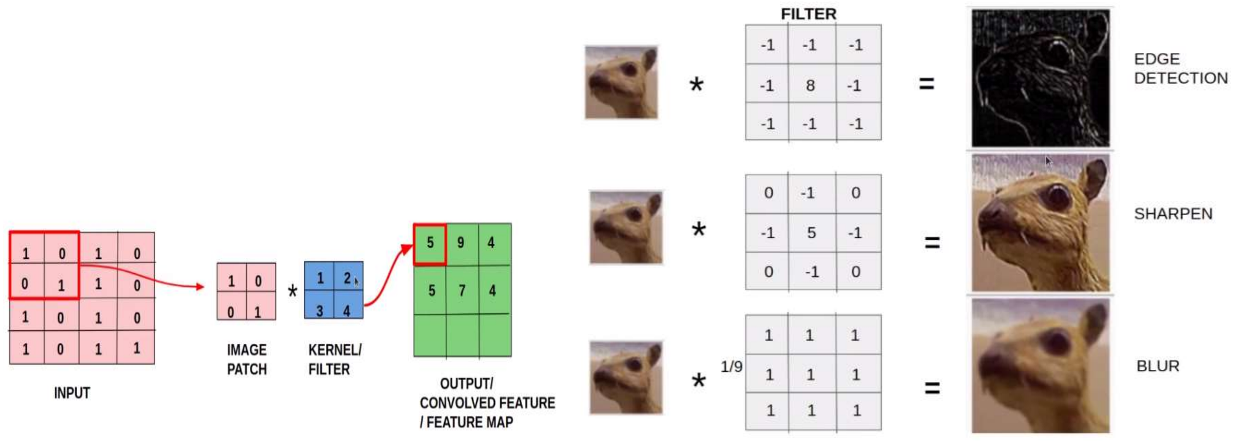


Figure 14 Image Filters

CNN learns the values of these filters on its own during the training process, although we still need to specify parameters such as the number of filters, filter size, the architecture of the network, etc. the more filters we have, the more image features get extracted and the better the network becomes at recognizing patterns in unseen images.

Our network includes a filter that just detects images or corners, but the deeper our network would go, the more sophisticated this filter becomes. On later layers rather than edges and simple shapes, out filters, would be able to detect specific objects like eyes ears. Feather's hairs. And again, if it moves further in deep layers the filters are able to take more sophisticated objects, like a full human face or a full dog or a full cat.

4.4.2. Model Architecture

A Convolutional Neural Network, perform some process beforehand we feed data to the densely fully connected artificial neural network. The entire network takes an image as input and generates the target label as output.

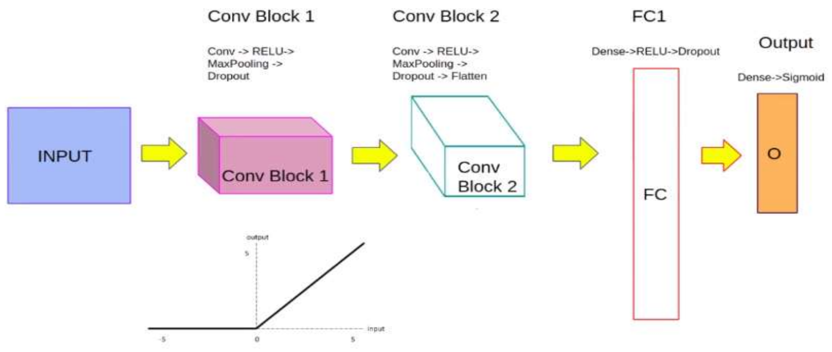


Figure 15 Model Architecture

A Convolutional Neural Network, perform some process beforehand we feed data to the densely fully connected artificial neural network. The entire network takes an image as input and generates the target label as output.

At first, we perform feature extraction, we apply convolution to simply extract features out of the input image. The input, is fit to the Conv2D block, after which we apply a RELU (Rectified Linear Unit) nonlinearity, followed by the MaxPooling layer, and lastly with the Dropout regularization.

The convolution is repeated, but this time there are more filters present, which is followed by, the activation function RELU (Rectified Linear Unit) and then it is followed by the MaxPooling layer and lastly, the Dropout layer. And then we flatten them up using the Flatten layer into a one-dimensional array so that it can feed to the fully connected layer. Any value, less than zero will be set to zero, and any positive value will be parsed along.

We are applying an activation function to improve the scarcity of the feature maps that actually dramatically improve and enhance the performance of the network. And after a RELU activation, we are down sampling the dimension of the image with a MaxPooling layer. After each convolution block, the height and weight are reduced by around a factor of two roughly, while the number of channels roughly doubles. After this convolution block2, the output is flattened and passed through one fully connected layer. And lastly, we use a dense layer with a sigmoid activation function.

4.4.3. Layered Architecture

The first layer is added as a convolution layer as a Conv2D layer. Specifying the feature extraction filter as 32 and the size of the filter as three by three. The activation function is set as RELU. The input shape is set as 150 by 150 by 3. The MaxPooling2D layer is added and the pool size is set to 2 by 2. The Dropout layer is added at 50% dropout to avoid overfeeding the data [figure16].

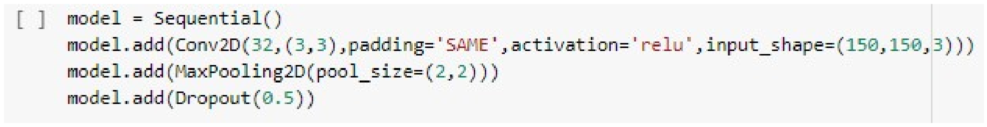
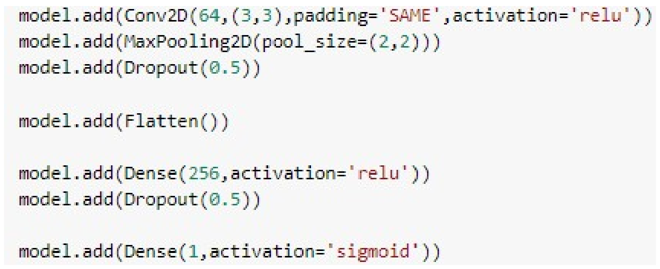

```
[ ]  model = Sequential()
     model.add(Conv2D(32,(3,3),padding='SAME',activation='relu',input_shape=(150,150,3)))
     model.add(MaxPooling2D(pool_size=(2,2)))
     model.add(Dropout(0.5))
```

Figure 16 Layer 1

The second layer is added followed by the first layer with the same parameters but changing the feature extraction filter to 64 and then added the flatten layer to transform it into a 1D image. A Dense layer is added with 256 nodes with RELU activation function and 50% dropout. The final output layer added with a dense layer with the number of node as 1 as there is a binary classification with a sigmoid activation function. [figure17]



```
model.add(Conv2D(64,(3,3),padding='SAME',activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(256,activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(1,activation='sigmoid'))
```

Figure 17 Layer 2

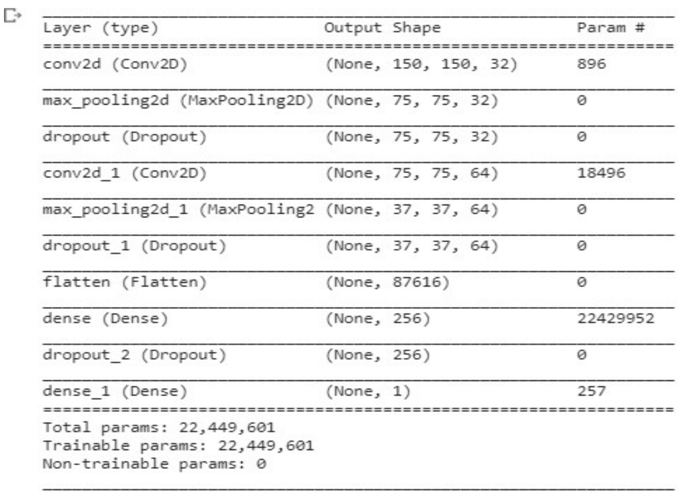
5. Experimental Results

The following model trials have been conducted on Google Colab and a computer equipped with an Intel(R) Core(TM) Duo processor (2.93 GHz), 6 GB of RAM. Visual Studio Code was used in this research for the development and implementation of the following model. The experiments trails include the following specifications and setup:

- 1 Dataset with real face mask and non-face masked for(training, testing and validation) taken from Kaggle
- Google Colab Environment
- Visual Studio Code with all necessary libraries installed on local system
- Tensorflow -2.3.0
- Keras -1.3
- OpenCv -1.2

5.1. Model Summary

Summary of the model. [figure18]



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 150, 150, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 75, 75, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 75, 75, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 75, 75, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 37, 37, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 37, 37, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 87616)             0
_________________________________________________________________
dense (Dense)                (None, 256)               22429952
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 257
=================================================================
Total params: 22,449,601
Trainable params: 22,449,601
Non-trainable params: 0
_________________________________________________________________
```

Figure 18 Summary

5.2. Training & Evaluating Performance of Model

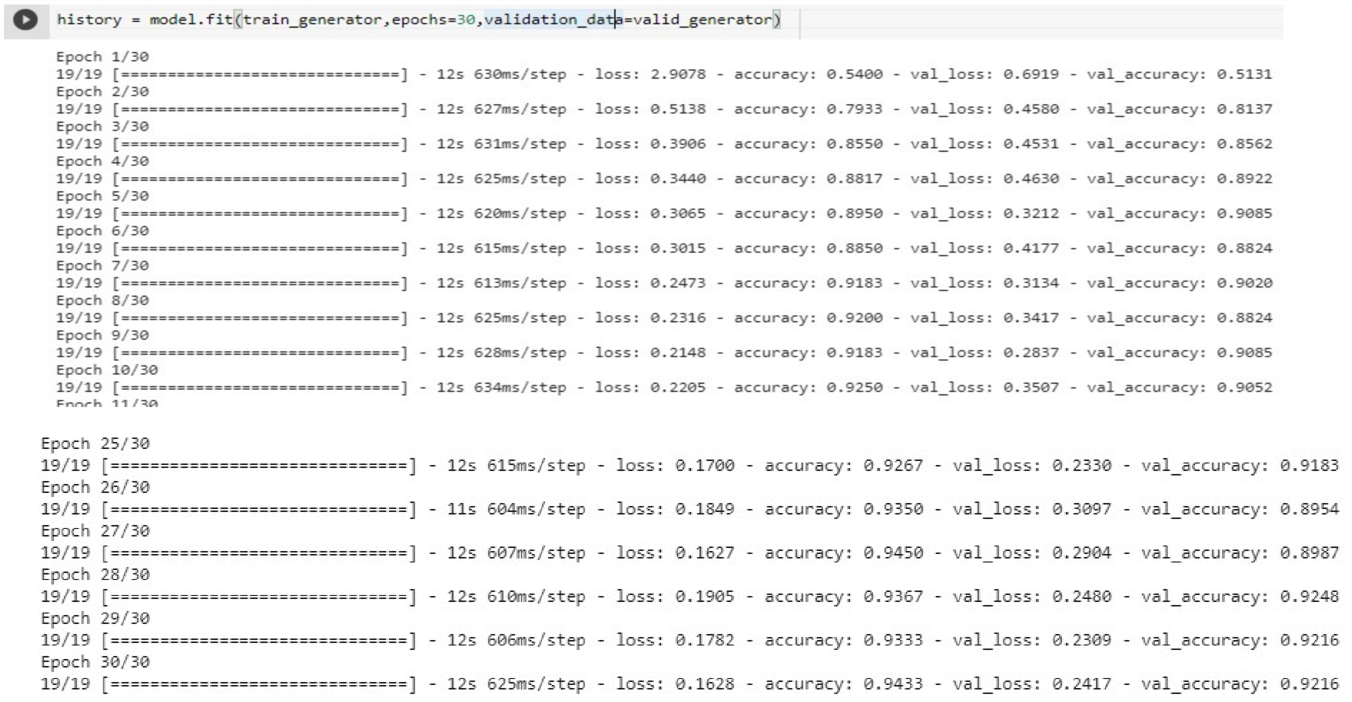
Adam optimizer with a learning rate of 0.001 with a loss of binary cross entropy type (as there are only two classes) and accuracy metrics.

These metrics will be calculated for each epoch during training, along with the evaluation of lost function on the training data.

With the following code, the compiler method will keep track of the performance metrics in addition to the binary cross entropy loss.

5.2.1. Training Model

Training is performed on all images available for 30 epochs and validation is performed on all the validation images that we have.



```
history = model.fit(train_generator,epochs=30,validation_data=valid_generator)

Epoch 1/30
19/19 [==============================] - 12s 630ms/step - loss: 2.9078 - accuracy: 0.5400 - val_loss: 0.6919 - val_accuracy: 0.5131
Epoch 2/30
19/19 [==============================] - 12s 627ms/step - loss: 0.5138 - accuracy: 0.7933 - val_loss: 0.4580 - val_accuracy: 0.8137
Epoch 3/30
19/19 [==============================] - 12s 631ms/step - loss: 0.3906 - accuracy: 0.8550 - val_loss: 0.4531 - val_accuracy: 0.8562
Epoch 4/30
19/19 [==============================] - 12s 625ms/step - loss: 0.3440 - accuracy: 0.8817 - val_loss: 0.4630 - val_accuracy: 0.8922
Epoch 5/30
19/19 [==============================] - 12s 620ms/step - loss: 0.3065 - accuracy: 0.8950 - val_loss: 0.3212 - val_accuracy: 0.9085
Epoch 6/30
19/19 [==============================] - 12s 615ms/step - loss: 0.3015 - accuracy: 0.8850 - val_loss: 0.4177 - val_accuracy: 0.8824
Epoch 7/30
19/19 [==============================] - 12s 613ms/step - loss: 0.2473 - accuracy: 0.9183 - val_loss: 0.3134 - val_accuracy: 0.9020
Epoch 8/30
19/19 [==============================] - 12s 625ms/step - loss: 0.2316 - accuracy: 0.9200 - val_loss: 0.3417 - val_accuracy: 0.8824
Epoch 9/30
19/19 [==============================] - 12s 628ms/step - loss: 0.2148 - accuracy: 0.9183 - val_loss: 0.2837 - val_accuracy: 0.9085
Epoch 10/30
19/19 [==============================] - 12s 634ms/step - loss: 0.2205 - accuracy: 0.9250 - val_loss: 0.3507 - val_accuracy: 0.9052
Epoch 11/30

Epoch 25/30
19/19 [==============================] - 12s 615ms/step - loss: 0.1700 - accuracy: 0.9267 - val_loss: 0.2330 - val_accuracy: 0.9183
Epoch 26/30
19/19 [==============================] - 11s 604ms/step - loss: 0.1849 - accuracy: 0.9350 - val_loss: 0.3097 - val_accuracy: 0.8954
Epoch 27/30
19/19 [==============================] - 12s 607ms/step - loss: 0.1627 - accuracy: 0.9450 - val_loss: 0.2904 - val_accuracy: 0.8987
Epoch 28/30
19/19 [==============================] - 12s 610ms/step - loss: 0.1905 - accuracy: 0.9367 - val_loss: 0.2480 - val_accuracy: 0.9248
Epoch 29/30
19/19 [==============================] - 12s 606ms/step - loss: 0.1782 - accuracy: 0.9333 - val_loss: 0.2309 - val_accuracy: 0.9216
Epoch 30/30
19/19 [==============================] - 12s 625ms/step - loss: 0.1628 - accuracy: 0.9433 - val_loss: 0.2417 - val_accuracy: 0.9216
```

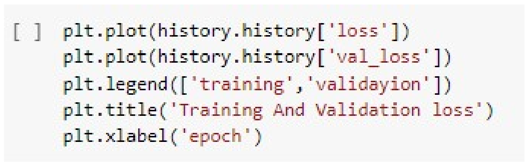
Figure 19 Training epochs

Training is performed on all images available for 30 epochs and validation is performed on all the validation images that we have.

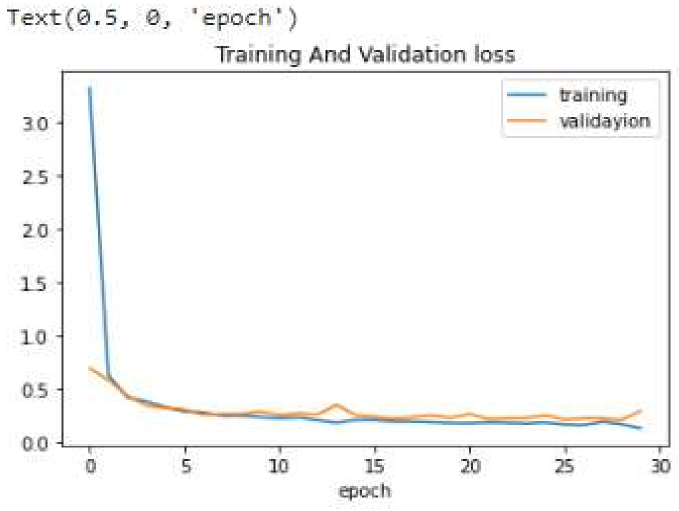
The training is completed with a final training loss of 0.1628 and training accuracy of 94.33% with validation accuracy of 92.16%.

5.2.2. Training and validation Loss

The training and validation loss went down which is good for the model. With every epoch, the loss is going down and the data is fitted well, keeping both the training and validation loss at a minimum. Both the loss for training and validation are going in the same direction.



```
[ ]  plt.plot(history.history['loss'])
     plt.plot(history.history['val_loss'])
     plt.legend(['training','validayion'])
     plt.title('Training And Validation loss')
     plt.xlabel('epoch')
```



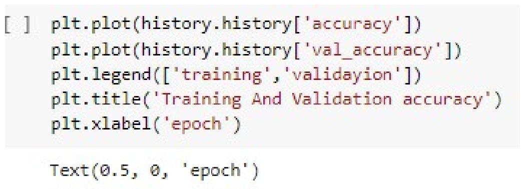
```
Text(0.5, 0, 'epoch')
```

Figure 20 Training & Validation Loss

5.2.2. Training and validation Accuracy

The training and validation loss went down which is good for the model. With every epoch, the loss is going down and the data is fitted well, keeping both the training and validation loss at a minimum.

The training accuracy is around 97 percent and the accuracy for validation accuracy has reached around 96 percent. Over the number of epochs, the accuracy has increased.



```
[ ] plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.legend(['training','validayion'])
    plt.title('Training And Validation accuracy')
    plt.xlabel('epoch')

    Text(0.5, 0, 'epoch')
```

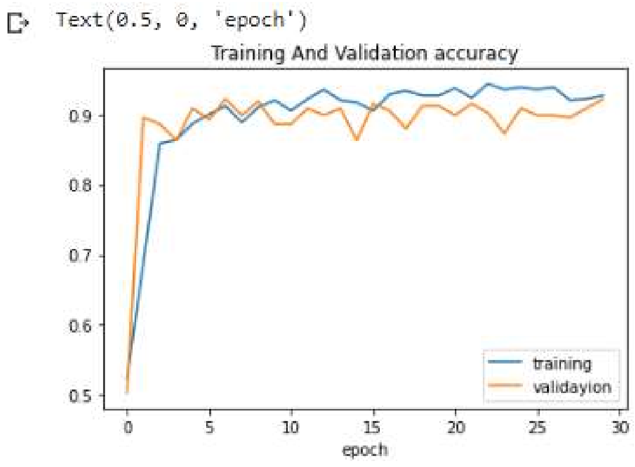
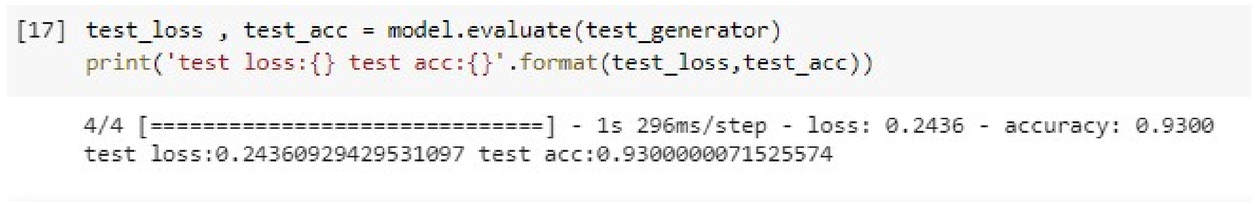


Figure 21 Training And Validation Accuracy

5.2.3. Test Accuracy and Loss

The testing data is feeded to the trained model, testing data is the data that a model has never seen during the training process. Around 93% accuracy was achieved on the testing data.



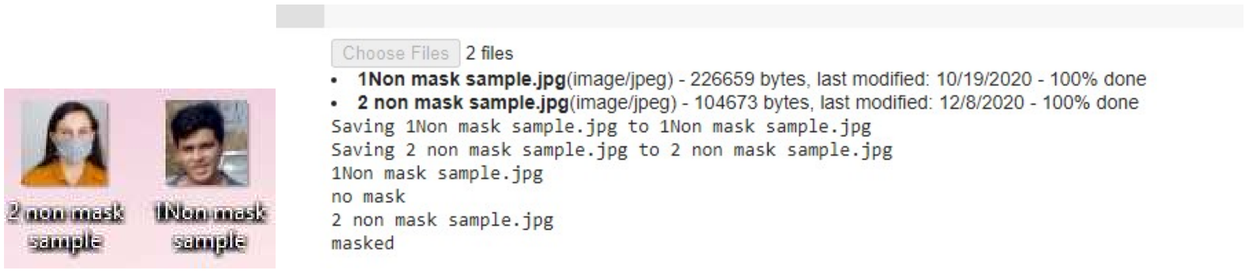
```
[17] test_loss , test_acc = model.evaluate(test_generator)
     print('test loss:{} test acc:{}'.format(test_loss,test_acc))

     4/4 [==============================] - 1s 296ms/step - loss: 0.2436 - accuracy: 0.9300
     test loss:0.24360929429531097 test acc:0.9300000071525574
```

Figure 22 Testing Accuracy

5.2.4. Working application

Image files from the local system were uploaded to the Colab notebook, and the prediction was used for the following uploaded images.

Here the CNN model has predicted the no mask sample file as no mask and the mask sample as masked. The model has predicted the right class labels, hence the model is working correctly.



- **1Non mask sample.jpg**(image/jpeg) - 226659 bytes, last modified: 10/19/2020 - 100% done
- **2 non mask sample.jpg**(image/jpeg) - 104673 bytes, last modified: 12/8/2020 - 100% done

```
Saving 1Non mask sample.jpg to 1Non mask sample.jpg
Saving 2 non mask sample.jpg to 2 non mask sample.jpg
1Non mask sample.jpg
no mask
2 non mask sample.jpg
masked
```

Figure 23 Working Application

The prediction performed on the video stream taking the video from the local system.

The CNN model has predicted from the video stream and labelled the no mask face as "No mask" and the masked face as "Mask" and the video is broken down into a stream of an array and stored in the model to further train the model and improve accuracy.

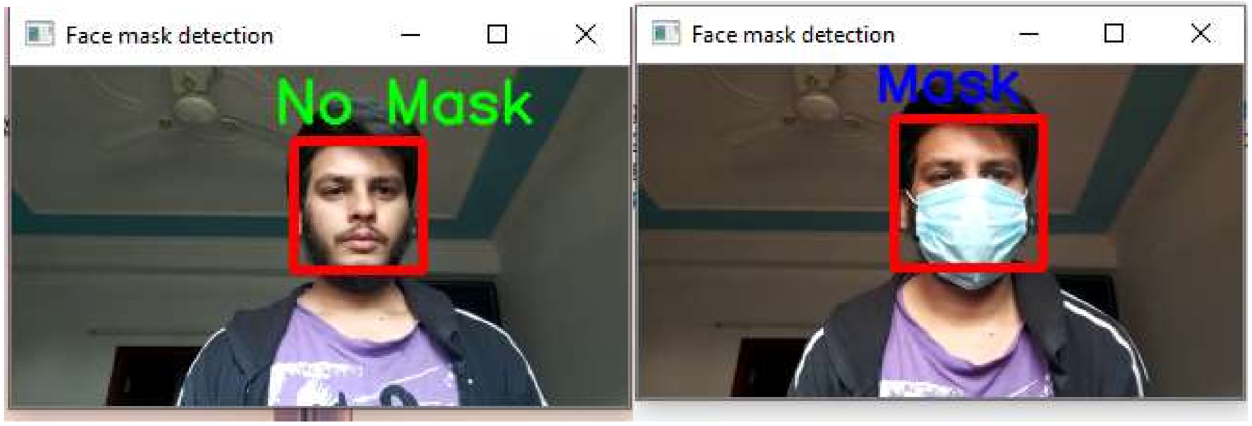


Figure 24 Working Application Model

5.2.4. Comparison with related works

In [5] the work presented by Z.Wang, he used the datasets which included masked and non-masked dataset which included 5000 masked images and 90000 non-masked images. Along with authors they achieved testing accuracy ranging from 50% to 95%. While the presented model used a dataset that included a total of 1006 images distributed in 3 subcategories of Train, Test, and Validate set having 50, 300, and 153 images of masked and no masked images in each set, the data set size has been reduced to a great extent. The training accuracy has reached around 97% & the validation accuracy has reached around 96%, and the testing accuracy has reached a total of 93% using the image data generator in the model.

5.2.5. Funding

The following research did not received any funding from anyone

5.2.6. Declaration of Competing Interest

There was no personal or social relationship that could have appeared to influence the work for the following paper

6. References

1. **Hu, G., Yang, Y., Yi, D., Kittler, J., Christmas, W., Li, S. Z., & Hospedales, T. (2015). When face recognition meets with deep learning: an evaluation of convolutional neural networks for face recognition. In *Proceedings of the IEEE international conference on computer vision workshops* (pp. 142-150).**

2. Wang, W., Yang, J., Xiao, J., Li, S., & Zhou, D. (2014, November). Face recognition based on deep learning. In *International Conference on Human Centered Computing* (pp. 812-820). Springer, Cham.

3. Cheng, V. C., Wong, S. C., Chuang, V. W., So, S. Y., Chen, J. H., Sridhar, S., ... & Yuen, K. Y. (2020). The role of community-wide wearing of face mask for control of coronavirus disease 2019 (COVID-19) epidemic due to SARS-CoV-2. *Journal of Infection*.

4. Loey, M., Manogaran, G., Taha, M. H. N., & Khalifa, N. E. M. (2020). Fighting against COVID-19: A novel deep learning model based on YOLO-v2 with ResNet-50 for medical face mask detection. *Sustainable Cities and Society*, 102600.

5. Wang, Z., Wang, G., Huang, B., Xiong, Z., Hong, Q., Wu, H., ... & Chen, H. (2020). Masked face recognition dataset and application. *arXiv preprint arXiv:2003.09093*.

6. https://www.kaggle.com/prithwirajmitra/covid-face-mask-detection-dataset

7. https://www.cs.ryerson.ca/~aharley/vis/conv/flat.html