CMP SCI 5390 Deep Learning
Face Mask Detection using Deep Learning Model

Keerthi Vakkalagadda
18222774

30th April 2022

Contents

1 Abstract

Everyone is required to wear a mask during the pandemic for preventing the spread of CORONA virus. for this purpose a basic Deep Learning model is built using TensorFlow, Keras, Scikit-learn to make the algorithm as accurate as possible. The proposed work contains two stages: (i) pre-processing, (ii) Training a CNN. The first part is the Pre-processing section involves using "image resizing and applying data augmentation techniques". Then the proposed CNN, classifies faces with and without masks, mask worn incorrectly as the output layer of proposed CNN architecture contains three neurons with 'Softmax' activation for classification.The proposed model has Validation accuracy of 53 percent.

1.1 Motivation

Following the rapid spread of a novel Coronavirus (COVID-19) case in Wuhan, China in December 2019, the World Health Organization (WHO) determined that this is a deadly virus that can transmit from human to human via droplets and airborne transmission. Wearing a face mask while going outside or socializing with others plays a crucial role for prevention. After all these months, face mask has become as an everyday essential device to carry along with us. Whether it's office or classroom or theater or any other place you will see a person with mask on. Isn't it cool that there is a computer to monitor and detect whether a person is wearing mask or not, instead assigning as human being to do this task. The thought of detecting whether a person's face mask is on or not using my learnings stirred interest in me and I decided to take up this problem. So, the project's objective is to create a convolution neural network that could detect and classify whether a person is wearing a mask or not or the person is wearing it incorrectly. As there are classes to classify it is a multi-class classification problem.

2 Dataset Preparation

2.1 Collecting Images

I've gathered a total of 815 photos of people with mask on, no mask and who wore it incorrectly (either the nose is not covered with mask or mouth or both). Only the people connected with the image's label was included in each folder. Images containing people with another classification other than the image's label are included in the dataset to avoid the problem of confusion; for instance an image with one person's mask on and other who does not have mask at all. Multiple instances of the same class are allowed to be included in the dataset; Example, group of people not wearing mask .

((a)) Person with no mask on

((b)) Person with mask on

((c)) Person wearing mask incorrectly

2.2 Image Pre-processing

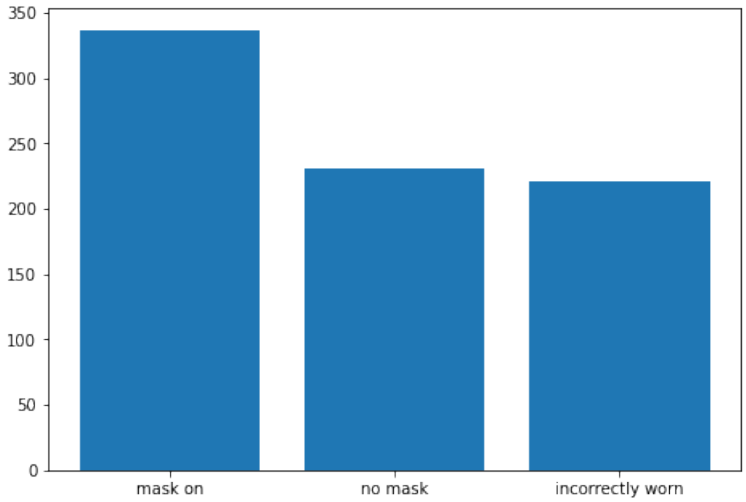
The photos were selected in a way to add emphasis to their respective labels. Images with multiple classes were resized to ensure that only one class was visible in the image. I have tried to produce some samples from a single image by cropping it. Only the images that are capable of producing an additional sample of its appropriate class are chosen. I've utilized PILLOW library to modify the samples, and performed rescaling, height/width shift, rotation, flip operations with customized values on images, using Image Augmentation from Image data generator. With the help of Image augmentation, all of the samples were modified to fit the 250 x 250 dimensions

2.3 Data Normalization

As there are various techniques like Rescaling, Standardization, Min-Max, Mean to normalize data, I've used Rescaling method by dividing the RGB values of all the input images with "255". We can prevent any potential skewing that may occur due to the presence of grayscale images within the datasets by normalizing RGB channels, and with normalized values, the model should be able to quickly converge to the appropriate trainable parameters. .

2.4 Data Distribution

The following plot depicts the distribution of how many images belong to which categories. A data imbalance exists in the datasets as there are more samples of people with mask on than those of other two classes. The favoring of mask on samples is due to the variability in the masks, like I've tried to include conditions like people wearing masks that are transparent, different coloured (not limited to blue/white) and has prints of nose and mouth on mask.



A total of 798 images in which 338 images belong to the people wearing mask correctly(both nose and mouth are covered), 230 images belong to people who wore mask incorrectly(mask does not cover either nose/mouth or both) and 231 belongs to images of people with no face mask on.

3 Implementing a Deep Learning Model

3.1 Build an Overfitting model

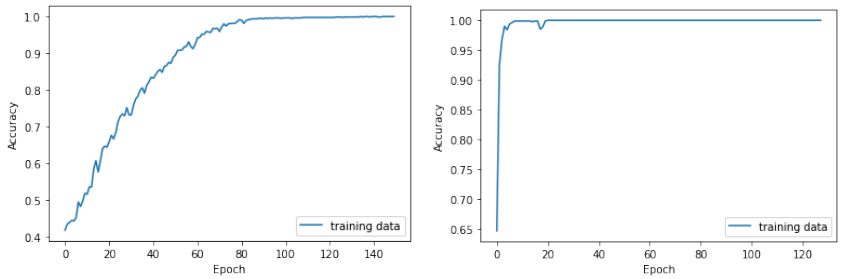
For this task I've performed few experiments by building CNN models of different architectures to overfit the data. The constrain here is to find a model that has small number of parameters (no.of layers, neurons count etc) and it should be able to achieve around accuracy 100 percent. For this I've tried to reduce the count of total parameters and make the model to memorize the data by introducing "Maxpool2d" layer in between Conv2d layers. This reduces the parameters count to half, however, by using it heavily will hold/pull the accuracy down. So I've limited it's presence in my models. Another approach to overfit the model is by passing the output as input and traning the model on that data. following table shows the results of my experiments

Total params	**Accuracy obtained**
935	77%
3,079 - (More neurons in dense layer)	81%
3,619	82.5%
5,939	100%
5,939 (Passing Output as input)	100%
6,411	99%
7,479	99.87%
30,683	100%

By seeing the results of each of model I can say the smallest possible overfitting model for my data should minimum have 5500 paramters and anything less than that will not be able to deliver an accuracy close to 100. Below is the architecture of the smallest possible over fitted CNN model.

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_17 (Conv2D)          (None, 147, 147, 16)      784

 max_pooling2d_13 (MaxPoolin  (None, 36, 36, 16)       0
 g2D)

 conv2d_18 (Conv2D)          (None, 33, 33, 8)         2056

 max_pooling2d_14 (MaxPoolin  (None, 8, 8, 8)          0
 g2D)

 flatten_5 (Flatten)         (None, 512)               0

 dense_10 (Dense)            (None, 6)                 3078

 dense_11 (Dense)            (None, 3)                 21

=================================================================
Total params: 5,939
Trainable params: 5,939
Non-trainable params: 0
_________________________________________________________________
```

I used the same architecture to provide output as input along with other input values for training the model to overfit the data, as it is the smallest model of my experiments that is able to yield accuracy close to 100 percent. Learning curves for the above CNN model are attached below:



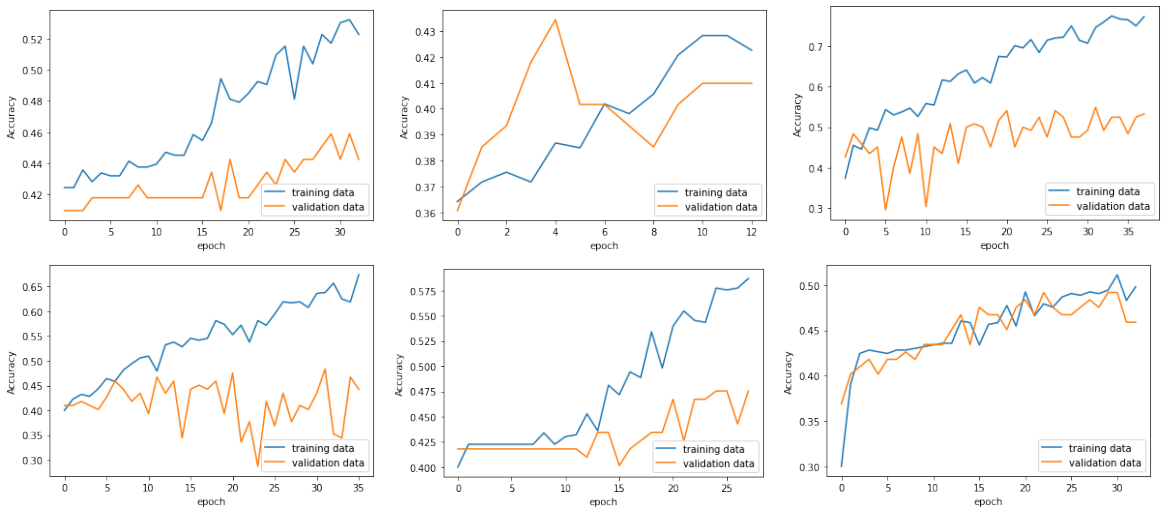
The second learning curve is obtained by feeding output/labels column as input and training the model on it

3.2 Split and evaluate on test set

To achieve this task, I've created three folders as Train, Valid, Test and placed 60 percent of the images in Train folder, 15 percent in valid folder and 25 percent images in Test folder. After splitting the images among folders I've used Image Data Generators for loading the data. In the process of conducting series of experiments to find a best model that can produce highest possible accuracy I started with a base model that has 4- Conv2d layers with (64,32,32,16) as neuron count, filter size of (5,5), 2 - dense layers (64, 3 neurons in respective layers) and trained this base model using Train set and evaluated the trained model on validation set. During the training phase I included "Early stopping and Model checkpoint" to monitor accuracy of model on validation set and used "patience = 32", where the model stops the training if "val accuracy" is not improved for straight 32 epochs. I've used "adadelta" as optimizer which helped my model to converge faster than "adam" and used "softmax" as activation function in last layer.

By observing the learning curve of the base model it's clear that the model is overfitting the data. In order to limit overfitting and achieve accuracy on validation set I started tuning hyper parameters by decreasing the count of neurons, conv2d - layers, filter size and introduced maxpool2d.

3.2.1 Learning Curves for Convolution Neural Networks



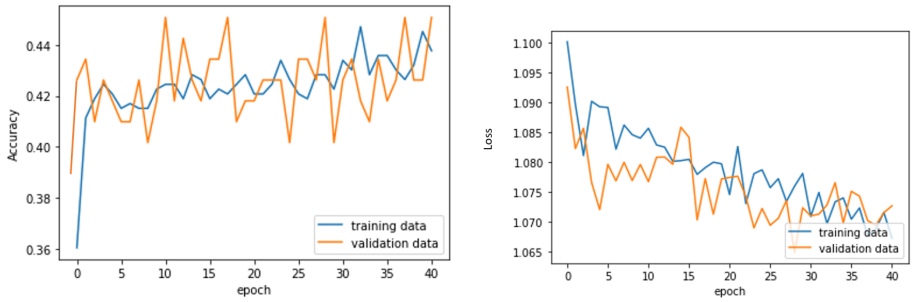
In all the experiments, one model is able to achieve 45.08 percent accuracy on validation set with minimized overfitting problem and evaluating this model on test set produced 46.875 percent accuracy. Below are the pictures of learning curves, summary of the model and final evaluation results

Layer (type)	Output Shape	Param #
conv2d_14 (Conv2D)	(None, 253, 253, 42)	2058
max_pooling2d_10 (MaxPoolin g2D)	(None, 50, 50, 42)	0
conv2d_15 (Conv2D)	(None, 47, 47, 28)	18844
max_pooling2d_11 (MaxPoolin g2D)	(None, 9, 9, 28)	0
flatten_7 (Flatten)	(None, 2268)	0
dense_14 (Dense)	(None, 32)	72608
dense_15 (Dense)	(None, 3)	99

Total params: 93,609
Trainable params: 93,609
Non-trainable params: 0

```
p = model7.predict(x)
accuracy = model7.evaluate(x,y)
print("Model accuracy:", accuracy[1]*100)
```

```
1/1 [==============================] - 1s 579ms/step - loss: 1.0588 - accuracy: 0.4688
Model accuracy: 46.875
```



3.3 Effects of Augmentation

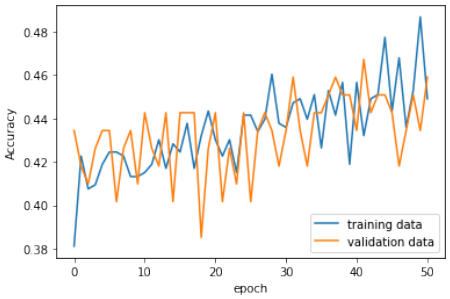
For studying the effects of using Data Augmentation I've utilized Image generators and applied augmentation techniques like rotation-range, width-shift-range, height-shift-range, shear-range, zoom-range, horizontal-flip, and rescaling on Development set and on Test set. Then trained the best model obtained from phase-3 on these sets to study the improvement in accuracy. Below is the architecture of the best obtained model of phase-3.

Model-#	Values range in Image generators	Accuracy on Validation set	Accuracy on Test set
Model1	rotation_range=40, width_shift_range=0.2, height_shift_range=0.2, shear_range=0.2, zoom_range=0.2, horizontal_flip=True, rescale=1./255,	46.7 (resulted in Overfittingl)	43.9%
Model2	rotation_range=25, width_shift_range=0.3, height_shift_range=0.3, shear_range=0.4, zoom_range=0.1, horizontal_flip=False, rescale=1./255	49.2%	46.099%
Model3	rotation_range=60, width_shift_range=0.7, height_shift_range=0.7, shear_range=0.4, zoom_range=0.5, horizontal_flip=False, rescale=1./255	43.44%	34.75%
Model4	rotation_range=10, width_shift_range=0.3, height_shift_range=0.4, shear_range=0.2, zoom_range=0.15, horizontal_flip=True, rescale=1./255	45.09	43.97%

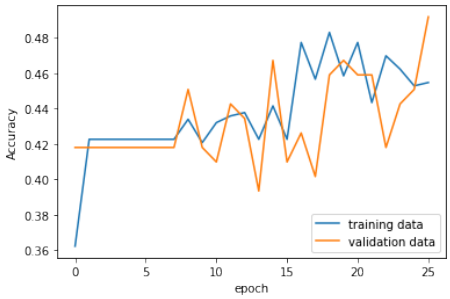
Figure 3.1:
Tabular-data of Augmentation Techniques used

From the above data you can clearly see the accuracy for model-2 has been improved. The highest accuracy I got from last phase is **45 percent** and this is increased to **49.2 percent**. When I used very high/very small values for augmentation techniques the model's accuracy is either not improved or the model got overfitted. Reason for accuracy not being improved can possibly be; the images might be augmented (too much zoom-in, rotated, etc) such way that model is not able to learn/ find patterns and overfitting issue is raised when very small values are used as the model will start seeing most similar or duplicate images.

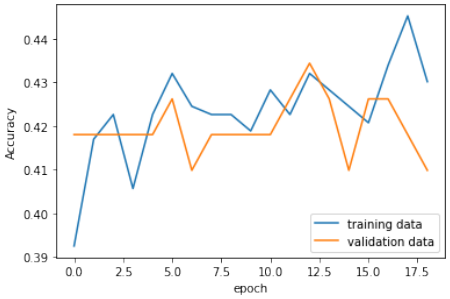
Following are the learning curves for the experiments I conducted.



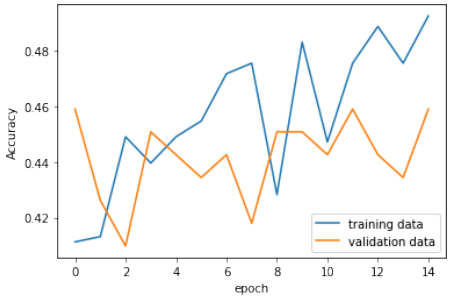
((a)) Model1



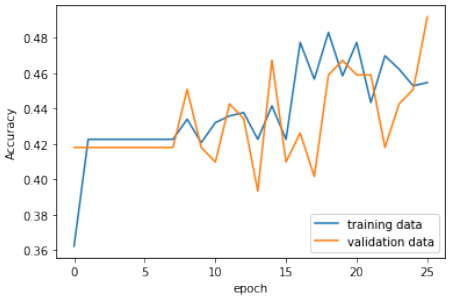
((b)) Model2



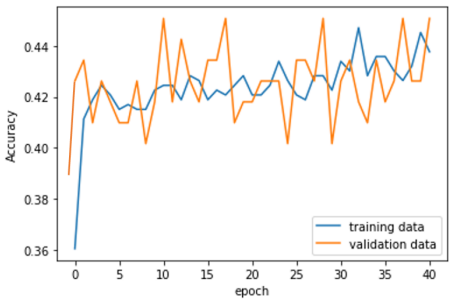
((a)) Model3



((b)) Model4



((a)) After Data Augmentation



((b)) Before Data Augmentation

3.4 Effects of Regularization

I've applied various regularization techniques like reducing model size, Batch Normalization, L1 and L2 Regularization, Dropout technique at different dropout rates, on the best model that I achieved from the last phase (using best achieved augmentation values) and studied the model's performance. In all the techniques 'Batch Normalization' augmented with 'Dropout' at drop rate of 0.5 achieved the highest accuracy, where as remaining all achieved moderate results. This combination yielded the best accuracy of 53.28 percent which is higher than accuracy obtained in last phase (49 percent). Following tabular data shows the various experiments I conducted.

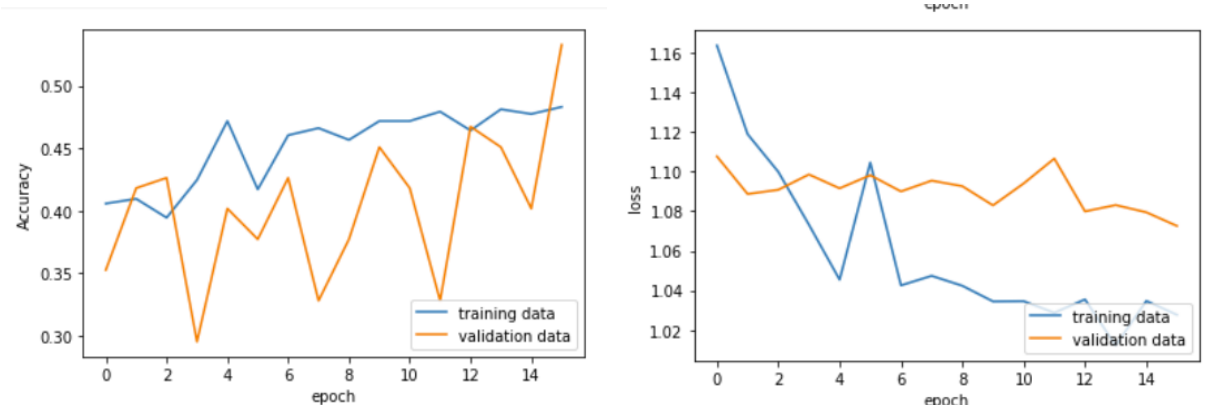
In all the experiments I conducted, validation loss of the model increased terribly after applying L1-regularization, it raised to 17 percent from 1.18 percent. I tried to

Regularization Technique	Accuracy on Training set	Accuracy on Validation set
Reducing Network's size	40%	45%
Dropout (dropout rate = 0.25)	42%	43%
Dropout (dropout rate = 0.5)	41.7%	45.902%
Batch Normalization	53.16%	48.26%
Batch Normalization with drop out	48.30%	53.28%
L1 Regularization	42.45%	43.443%
L2 Regularization	42.84%	41.18%

Figure 3.2: Results after applying different Regularization Techniques

reduce the network's size from 600k to 73k parameters by adding more maxpool layers, but no improvement is seen in accuracy because the number of parameters might be insufficient for the model as it might not be able to learn different patterns during training phase, with Dropout technique, in between 0.25 and 0.5 values, dropout rate with 0.5 got better results. Batch Normalization without any other techniques made the model overfit, however it boosted the model's performance when used it with dropout.

3.4.1 Learning curves of Model when Batch Normalization is augmented with Dropout



3.5 Experimenting with Pre-Trained Models

ResNet and VGG16 networks are built to see whether the model's performance can be improvised or not. The results are: Basic Conv2D model achieved 46 percent where as VGG16 model yeilded 51 percent which is a pre-trained model and ResNet model achieved 47 percent accuracy. For VGG16 model I've used 'imagenet' as pre-trained weights and it is the biggest model so far, which has 14,722,919 parameters in total, with 5-maxpool2d layers. It took almost two hour for executing the model and took **165 seconds** for completing one epoch. With ResNet, I've tried various models and finalized the one with less overfitting problem. The final ResNet architecture has '859,251' parameters with one batch Normalization layer and Dropout layer with drop rate as 0.25. One of my observation with ResNet model is as number of layers and parameters are increased, the model got bigger and it easily got overfitted. ResNet model took 105 seconds to complete single epoch. Below are the learning curves of Basic CNN model, VGG16 Model and ResNet model:

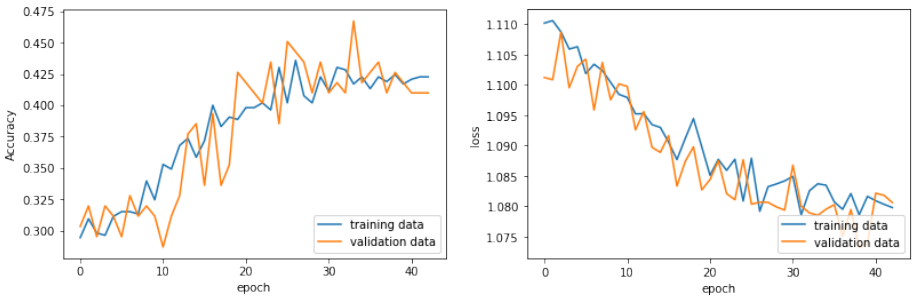


Figure 3.3: Basic Conv2D Network model's Performance

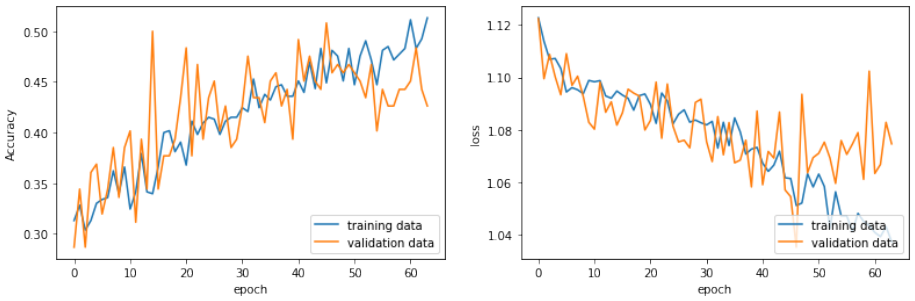


Figure 3.4: VGG16 Network model's Performance

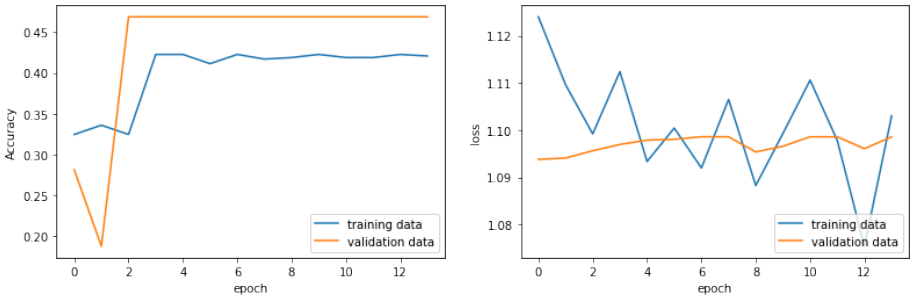


Figure 3.5: ResNet model's Performance

Pre-trained model and ResNet networks achieved decent results and the overfitting problem is minimized when compared with basic CNN model. One more interesting observation from the experiments is, the ResNet model which has smaller architecture than best-performed CNN model from previous phases is able to produce similar results.

4 Conclusion

I've used the dataset which contains 815 pictures and used 530 images for training the model, 122 images for validating the model and 141 images for testing the model's accuracy. These pictures were individually downloaded from internet. During initial stages of work I've reviewed the basic concepts of deep CNN models and built the base model with 'adadelta' as optimizer, because it converged faster than 'rmsprop' and used 'softmax' as last layer's activation function, 'categorical crossentropy' as loss function. Metrics like accuracy,loss precision, f1-score are used in selecting the base model in which best model achieved 43 percent accuracy. Later by applying augmentation techniques, regularization techniques the model's performance has been increased from 43 to 53 percent. Finally, I implemented the most used and powerful pre-trained CNN models ResNet-50 and VGG-16 on the dataset which produced decent results with minimized overfitting problem.

4.1 References

https://www.hindawi.com/journals/sp/2021/8340779/

https://ieeexplore.ieee.org/document/9402670