The George Washington University

Machine Learning 2

DATS 6203

Fall 2022

Instructor: Dr. Amir Jafari

Face Mask Detection

Authors:

Mahtab Barkhordarian

Nusrat Nawshin

Submission Date:

December 13, 2022

CONTENTS

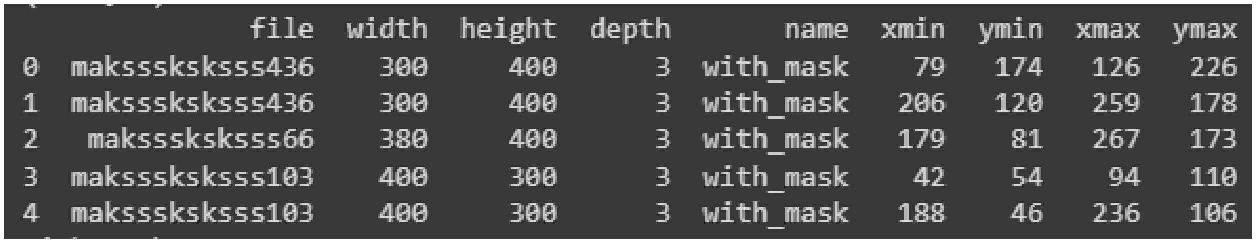
Introduction

The CDC continues to monitor the spread of COVID-19 and advises people who are completely vaccinated as well as those who are not fully vaccinated to wear face masks. When visiting the doctor's office, hospitals, or long-term care institutions, the CDC recommends wearing masks and keeping a safe distance.

Manually monitoring people entering such institutions is tedious and requires a workforce. In this project, we will learn how we can automate this process through deep learning techniques which will automatically detect people not wearing masks to prevent their entry.

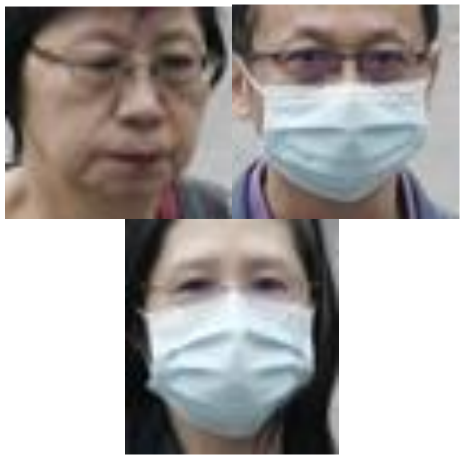
Dataset & Data Preparation

We use the dataset from Kaggle. The dataset contains two folders containing images in .png format and the annotations of each image in .xml format. There are a total 853 images belonging to the 3 classes: with mask, without mask, wearing masks incorrectly. For data preparation, we extract each individual face from the images and store bounding box information from the XML file in a Dataframe. Each XML file contains the individual face information of each image, image file name, image width, image height, image depth and contains each face's class name, bounding box information such as xmin, ymin, xmax, ymax. The final dataframe few records are shown below.



```
              file  width  height  depth       name  xmin  ymin  xmax  ymax
0  makssskskss436    300     400      3  with_mask    79   174   126   226
1  makssskskss436    300     400      3  with_mask   206   120   259   178
2   makssskskss66    380     400      3  with_mask   179    81   267   173
3  makssskskss103    400     300      3  with_mask    42    54    94   110
4  makssskskss103    400     300      3  with_mask   188    46   236   106
```

Figure 1: Dataframe with face object information

Full Image	Cropped faces
	

After extracting faces number of images are now:

with_mask: 3231
without_mask: 715
mask_weared_incorrect: 123

As we can observe that mask-weared incorrectly data is too low, we are merging it to the without mask category and now we have 838 images of 'without_mask' category and 3231 images 'with_mask' category. Clearly the dataset is very imbalanced. We are going to implement image augmentations before training the model for face mask detection. We are dividing all the individual face images into Train Validation and Test set with 80-10-10 split and saving the images containing the single faces into the directory in separate folders.

Number of images in Train Set:
With Mask: 2585
Without Mask: 671

Number of images Test Set:
With Mask: 323
Without Mask: 84

Number of imagesValidation Set:

With Mask: 324
Without Mask: 84

Data Preprocessing and Augmentation

There are several functions in our preprocessing file which are going to be explained here:

preprocessing(): The main processing py file starts running from this function. This function is going to read the image, annotation folders and create our data by mapping annotation and image together, split the dataset into train-validation-test set and finally save the face images into the directory.

parse_annotation_object(annotation_object): Detects the exact faces and returns a dictionary containing image class name, xmin, ymin, xmax, ymax.

parse_annotation(path): In this function it's trying to read the annotation file which includes xml files. Each XML file contains the individual face information of each image, image file name, image width, image height, image depth and contains each face's class name, bounding box information such as xmin, ymin, xmax, ymax. It returns a list containing image name, image width, image height, image depth, classname, xmin, ymin, xmax, ymax.

extract- faces: Extract faces from full image

crop image: Crops images from face objects

save image: The faces which are extracted are going to be saved as png files in the image folder. In the preprocessing python file, we divide the data into train, validation, and test. For each train, validation and test, it is the helper function to create two folders: with masks and without masks.

create_datagen(): It implements Tensorflow ImageGenerator method that generates batches of tensor image data with real-time data augmentation. The images are loaded as a batch, and it generates images with the following augmentation parameters.

- rescale=1.0 / 255,

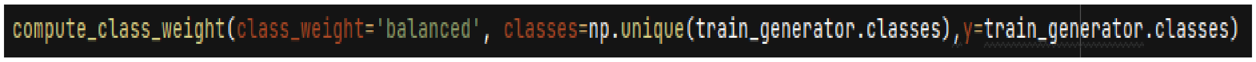
- horizontal_flip=True,
- zoom_range=0.1,
- shear_range=0.2,
- width_shift_range=0.1,
- height_shift_range=0.1,
- rotation_range=4,
- vertical_flip=False

create_valDatagen(): Rescales validation and test set images values in range of 0 to 1. As the original images consist of RGB coefficients in the 0-255, but such values would be too high for the model to process, so it targets values between 0 and 1 instead by scaling with a 1/255.

create_dataset_generator(datagen, batch_size, path,shuffle): Reads images in batches, encodes the target to binary, reshapes the images to 32x32. The shuffle option is applied on the training and validation set only.

prediction_label(predicts): Set threshold for prediction. Here we are using the standard 0.5 threshold value.

Class balancing: For learning in a balanced way we have added the class_weights argument in model.fit(), which is used to make the model learn more from the minority class. Using the Scikit learn module's compute_class_weight method we calculated the weights required for the balancing train set.



```
compute_class_weight(class_weight='balanced', classes=np.unique(train_generator.classes),y=train_generator.classes)
```

Which gives us the following weights:

{0: 0.6297872340425532, 1: 2.4262295081967213}

Our dataset has been balanced using class weight.

Here, we can see that class 0 has more data than class 1, thus our model will give this class an estimated weight of 0.63, while class 1 has less data than class 0 and will receive an approximate weight of 2.43. Simply put, the model will take into account around 4 (2.43/0.63) images of class 1 for every 1 image of class 0.

Modeling Architectures:

Convolution Neural Network (CNN):

A convolutional neural network (CNN or convnet) is a subset of machine learning. It is one of the various types of artificial neural networks which are used for different applications and data types. A CNN is a kind of network architecture for deep learning algorithms and is specifically used for image recognition and tasks that involve the processing of pixel data.

It has 4 main layers; we may add some more layers to normalize our network.

1- Convolution Layer: This convolution layer represents CNN layers where they interact with filters and Kernels. by using a sliding window, we apply small units across the input and these units are known as filters.

2- pooling layer: Pooling includes down-sampling of features with the goal that we must train less features. with this layer we avoid overfitting

3- Batch Normalization layer: During training if there is any instability in any layers of our neural network, we do batch normalization to that layer.

4- Fully connected layer: The fully connected layer is where image classification happens in the CNN based on the future extracted in the previous layers.

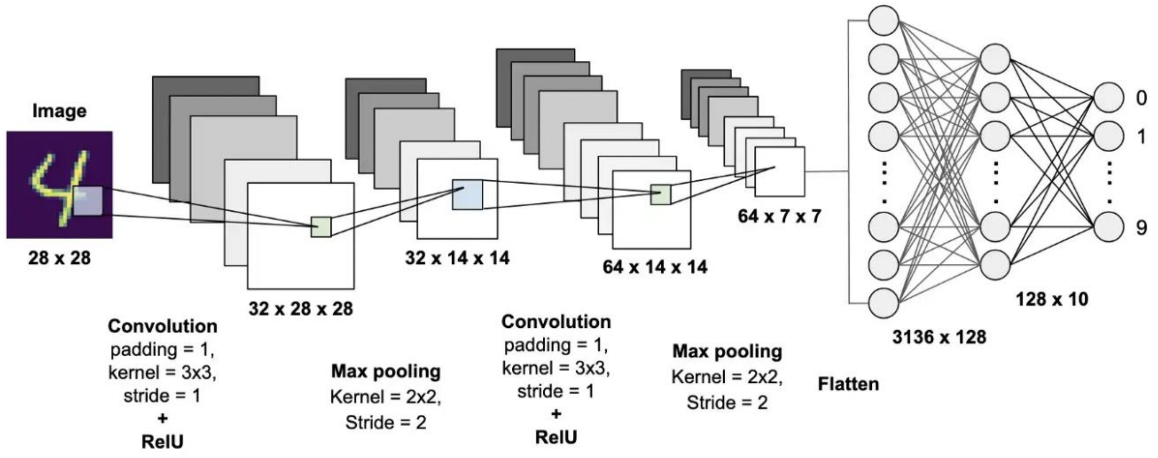


Figure 2: CNN Architecture

LSTM:

Long short-term memory (**LSTM**) is an artificial neural network used in the fields of artificial intelligence and deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. Such a recurrent neural network (RNN) can process not only single data points (such as images), but also entire sequences of data (such as speech or video). For example, LSTM is applicable to tasks such as unsegmented, connected handwriting recognition speech recognition,machine translation, robot control, video games,and healthcare.

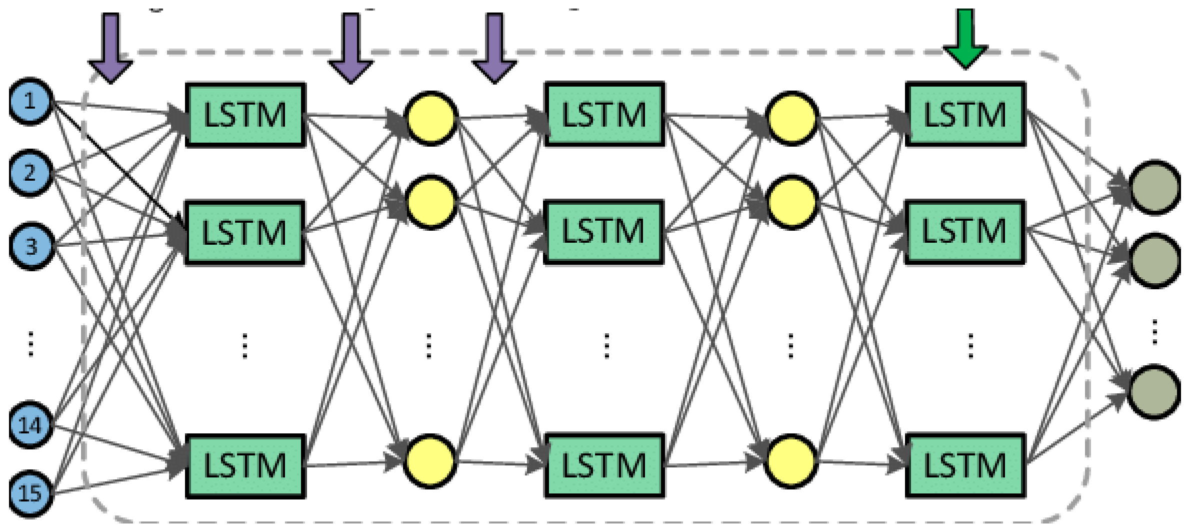


Figure 3: LSTM architecture

Base Model (CNN):

We used CNN which has 4 convolution layers. For each convolution layer we added a Batch normalization layer to catch the instability. for the CNN activation layer, we used 'Relu'. In comparison to sigmoid and tanh, ReLU is more reliable and accelerates the convergence by six times. In the input convolution layer, we reshaped all the images to 32*32* 3 to be the same size. We got the size from our preprocessing and 3 channels because the data are pictures. We used a 2D convolution layer because the data is picture not text, audio or video. After we were done with the CNN as our model the F1 increased to 88.

Framework & Optimizer:

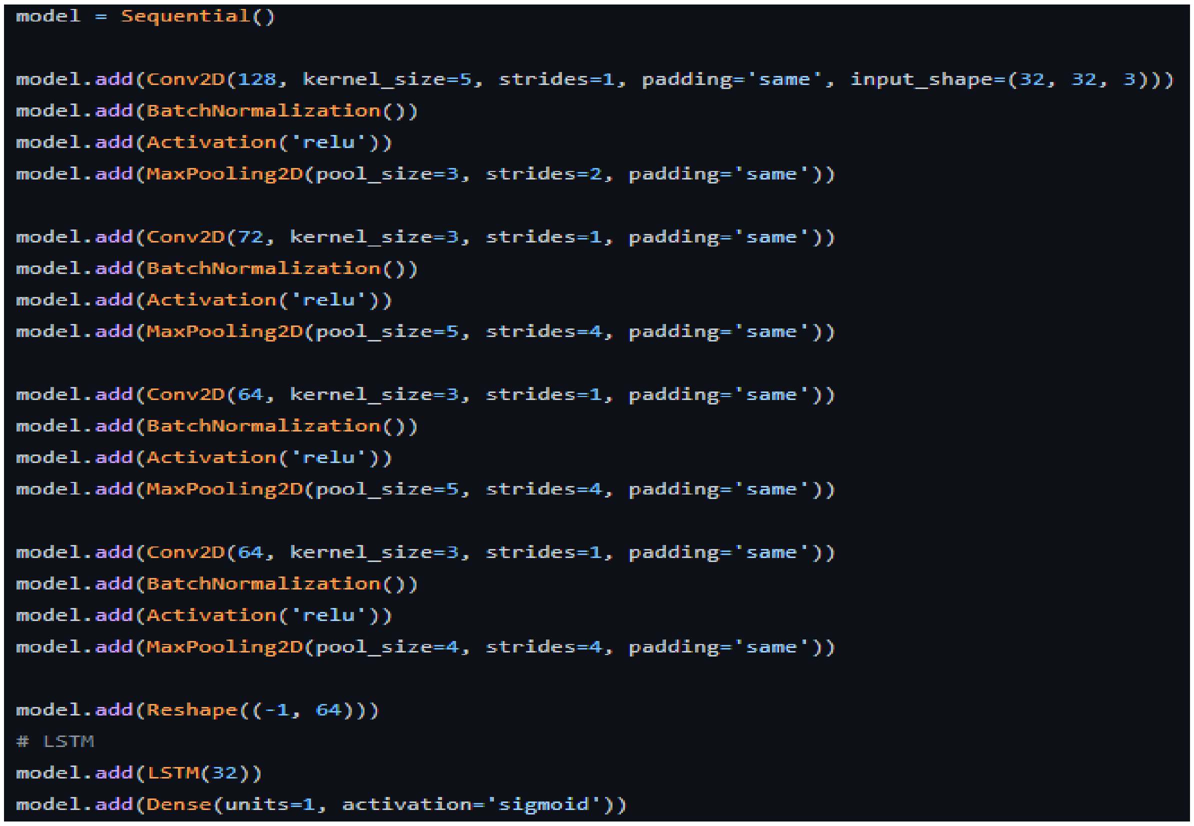
The Framework we used is Tensorflow.keras is one of the most popular frameworks for deep learning. It is used for rapid deployment of new algorithms. It also provides integration with other TensorFlow models. TensorFlow is one of the most preferred deep learning frameworks as it is Python-based, supported by Google, and comes loaded with top-notch documentation and walkthroughs to guide. Keras is easy to understand, and it has built in support for understanding and learning. The optimizer we used in the final model is Adam. Adam utilizes the concept of momentum by adding fractions of previous gradients to the current one; it is mostly accepted in many projects during training neural networks.[7] As our classification problem in binary classification, we used 'Binary Cross Entropy' as the loss function. For the performance metric, we are using Accuracy, Recall and Precision.

Final Model: CNN+LSTM

In this study, the structure of this architecture was designed by combining CNN and LSTM networks, where the CNN is used to extract complex features from images, and LSTM is used as a classifier. We have used 4 convolution layers each with batch normalization and activation function of 'ReLU', followed by MaxPooling layer. After that we reshaped the data to feed it into the LSTM model followed by a fully connected layer to get our target output with activation function 'Sigmoid'.

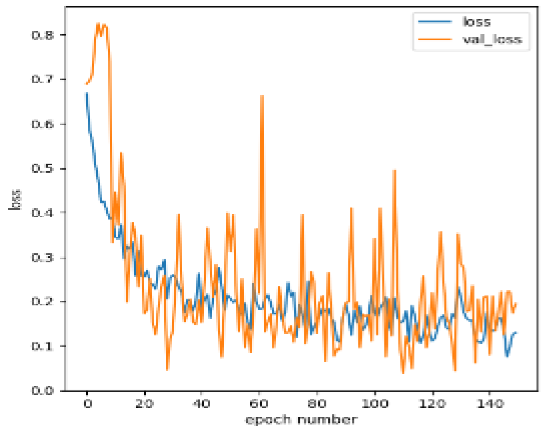
Model Architecture:

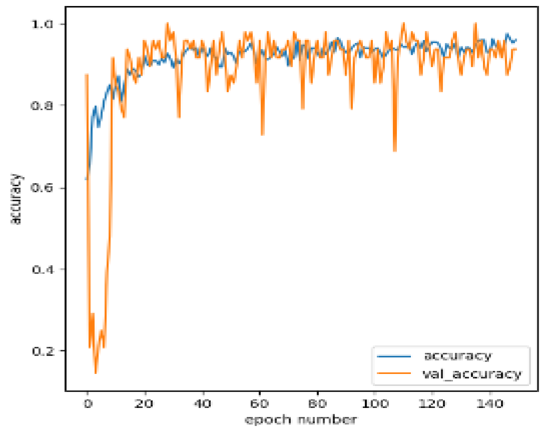
```
Model: "sequential"
_________________________________________________________________________________
 Layer (type)                                   Output Shape                Param #
=================================================================================
 conv2d (Conv2D)                               (None, 32, 32, 128)            9728
 batch_normalization (BatchNormalization)      (None, 32, 32, 128)             512
 activation (Activation)                       (None, 32, 32, 128)               0
 max_pooling2d (MaxPooling2D)                  (None, 18, 18, 128)               0

 conv2d_1 (Conv2D)                              (None, 18, 18, 72)           83016
 batch_normalization_1 (BatchNormalization)     (None, 18, 18, 72)             288
 activation_1 (Activation)                      (None, 18, 18, 72)               0
 max_pooling2d_1 (MaxPooling2D)                  (None, 5, 5, 72)                0

 conv2d_2 (Conv2D)                               (None, 5, 5, 64)            41536
 batch_normalization_2 (BatchNormalization)      (None, 5, 5, 64)              256
 activation_2 (Activation)                       (None, 5, 5, 64)                0
 max_pooling2d_2 (MaxPooling2D)                  (None, 2, 2, 64)                0

 conv2d_3 (Conv2D)                               (None, 2, 2, 64)            36928
 batch_normalization_3 (BatchNormalization)      (None, 2, 2, 64)              256
 activation_3 (Activation)                       (None, 2, 2, 64)                0
 max_pooling2d_3 (MaxPooling2D)                  (None, 1, 1, 64)                0

 reshape (Reshape)                                (None, 1, 64)                  0
 lstm (LSTM)                                       (None, 32)                12416
 dense (Dense)                                      (None, 1)                   33

=================================================================================
Total params: 184,969
Trainable params: 184,313
Non-trainable params: 656
_________________________________________________________________________________
Found 3256 images belonging to 2 classes.
Found 408 images belonging to 2 classes.
Found 407 images belonging to 2 classes.
```



```
model = Sequential()

model.add(Conv2D(128, kernel_size=5, strides=1, padding='same', input_shape=(32, 32, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=3, strides=2, padding='same'))

model.add(Conv2D(72, kernel_size=3, strides=1, padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=5, strides=4, padding='same'))

model.add(Conv2D(64, kernel_size=3, strides=1, padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=5, strides=4, padding='same'))

model.add(Conv2D(64, kernel_size=3, strides=1, padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=4, strides=4, padding='same'))

model.add(Reshape((-1, 64)))
# LSTM
model.add(LSTM(32))
model.add(Dense(units=1, activation='sigmoid'))
```

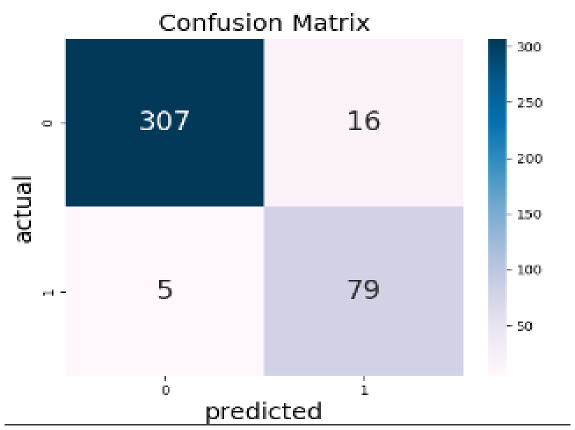
Results:



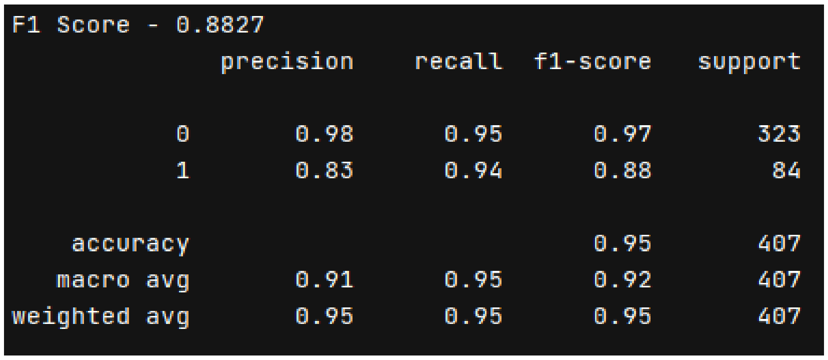


The loss is getting less after each epoch and the accuracy is increasing since the model starts running. The reason we have lots of instability during each epoch is because the learning rate is big.

The model trains label 0 as ‘With Mask’ and 1 as ‘Without Mask’



From this confusion matrix, we can get the result that 307 people who have masks are predicted correctly. 79 people who are not wearing masks are predicted correctly. There are 16 people who are wearing masks but predicted not wearing the masks and 5 people who are not wearing masks but predicted as wearing masks.



```
F1 Score - 0.8827
              precision    recall  f1-score   support

           0       0.98      0.95      0.97       323
           1       0.83      0.94      0.88        84

    accuracy                           0.95       407
   macro avg       0.91      0.95      0.92       407
weighted avg       0.95      0.95      0.95       407
```

Here we can observe that the model can predict people wearing masks more accurately, it has higher precision, recall and f1 score. But overall, both classes are getting predicted well.

Other Models attempted before finalizing the best model:

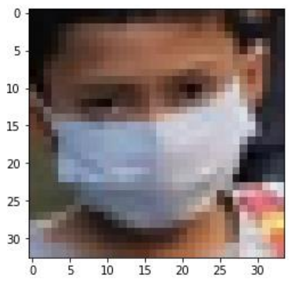
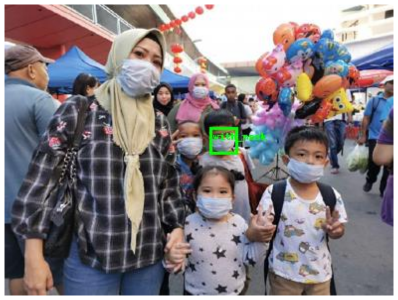
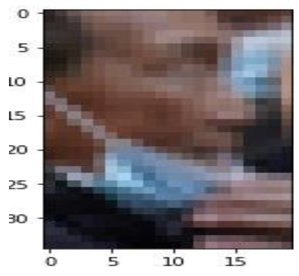
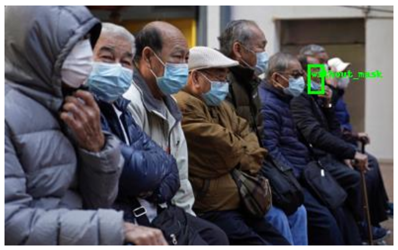
Model	Optimizer	Epoch	Batch Size	Learning Rate	Train Accuracy	Validation Accuracy	Test Accuracy
CNN	Adam	100	8	0.001	66	63	53
CNN + MLP	RMSprop	100	8	0.00001	92	91	94
MLP	Adam	150	8	0.001	90.25	89.58	93.61

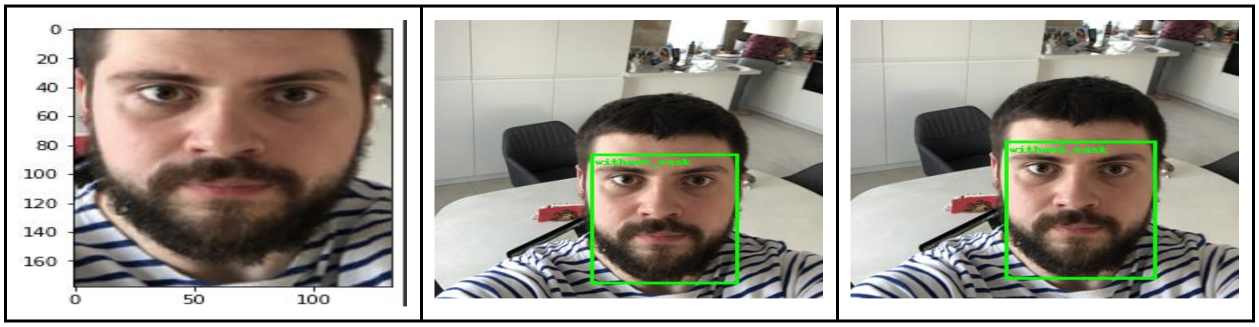
CNN + MLP	Adam	50	8	0.001	94.75	91.67	94.59
CNN + LSTM	**Adam**	**150**	**8**	**0.0009**	**96.00**	**93.75**	**94.84**
CNN + LSTM	Adam	150	16	0.001	93.23	1.00	95.82
CNN + LSTM	Adam (Early stopping)	63	16	ReduceLROnPlateau	92.19	93.75	91.646
VGG + CNN	Adam	50	8	ReduceLROnPlateau	92.23	91.64	91.72

All the models had a good performance overall. But in our problem set, we are preferring a model with stable F1 score as the dataset is imbalanced. Among all the model's 5th model with CNN + LSTM had the highest accuracy along with a good F1 score of 0.88.

Predictions:

Here are some randomly picked faces from the test set and their label at the original image and the model predicted image.

Cropped face Image	Original full Image	Predicted Image Classification
		
		



Conclusion:

We used CNN + LSTM architecture for your final model and it gave us a better model with 96% training accuracy and 94% testing accuracy. The testing f1 score is 88.27%. As our dataset is imbalanced a better F1 score indicates the model performance with more insight. The F-Measure is a popular metric for imbalanced classification. Even though LSTM is mostly used for more sequenced data, using it along with CNN made the mode more consistent on true positive and true negative values and lesser false negative and false positive value. LSTMs have the capacity to selectively remember patterns for a long duration of time and CNNs are able to extract the important features out of it.

Percentage of code written:

Approximately 60% of the code was referred from the internet and modified according to the needs of our project. The remaining 40% was done to create helper functions or automating certain processes like downloading data, models, etc.

References:

[1]https://www.kaggle.com/code/chinmayaudupa/face-mask-detection-using-cnn-and-deep-learning

[2]https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network

[3]https://www.upgrad.com/blog/basic-cnn-architecture/

[4]https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939

[5]https://www.researchgate.net/figure/The-proposed-deep-LSTM-network-with-three-LSTM-layers-and-two-feedforward-layers-For_fig1_283748673

[6]https://en.wikipedia.org/wiki/Long_short-term_memory

[7]https://analyticsindiamag.com/guide-to-tensorflow-keras-optimizers/

Dataset Link: https://www.kaggle.com/datasets/andrewmvd/face-mask-detection
https://becominghuman.ai/building-a-convolutional-neural-network-cnn-model-for-image-classification-116f77a7a236

Reference Code Link: face-mask-detection CNN | Kaggle

Project GitHub Link: https://github.com/NusratNawshin/ML2ImageClassification