Applied Artificial Intelligence

COMP 6721

Dr. René Witte, P.Eng. (Associate Professor)

(Winter 2021)

Project AI Face Mask Detector

Part I

Haroon Aziz Mirza (Student ID: 40183707)
Neelofer Shama (Student ID: 40121559)
Tejinder Singh (Student ID: 40114377)

CONTENTS

Project Introduction

The corona virus COVID-19 pandemic triggers a global health crisis and in result, according to the World Health Organization (WHO), the different protection methods have been approved and wearing a face mask in public areas is one of them. Almost all the countries around the globe made it mandatory to wear a face covering in public areas to alleviate the risks of spreading this virus.

The basic purpose of the project is to develop an AI that can analyze face images and to differentiate between faces with masks and without masks and for random pictures.
In this project, we strived to develop a Deep Learning Convolutional Neural Network (CNN) for face detecting using PyTorch[1] (an open-source library widely used to develop Convolutional Neural Network) and train it to recognize three different classes in the images.

1) Person wearing a mask
2) Person not wearing a mask
3) Not a person as random image

In this project, before feeding input to our Convolutional neural network model, certain preprocessing, and transformation of 3 different types of datasets images are mandatory and once done then trained model would be evaluated based on four metrics which are accuracy to measure the losses as well precision, recall and F1-measure.

Experimental results show that the trained CNN model performs well on the test dataset images with almost 94% accuracy and for the remaining 3 evaluation metrics; precision, recall and F1-measure almost 92% respectively.

1. Methodologies of proposed Project:

1.1 Dataset Used

All the images are actual images extracted from Kaggle datasets [2]. We divided the dataset into three different classes i.e., face with masks, face without mask and random pictures for training, testing and validation purposes. The idea is to split the dataset to make our model more generalized to avoid overfitting so that model performs with optimization to unseen testing data which will be fed after training the neural network.

The proportion of dataset has been chosen in a way to maintain the balance between images of all three different classes and for that roughly 2000 images have been removed from the downloaded dataset from Kaggle.

To train the model purpose, Dataset chosen consists of a minimum of 1200 (400 images of masked faces and 400 images of non-masked faces and approximately 400 random pictures of different objects were chosen). After training, the model optimizes its parameters.

For testing purpose, Dataset chosen consists of a minimum of 600 (200 images of masked faces and 200 images of non-masked faces and 200 random pictures of different objects were chosen).
In our project approximately 70% of images were dedicated for training purpose and the rest of 30% data used for testing purpose so the splitting ratio was almost 0.70 to 0.30.

The dataset must have the following structure:

```
Train_Model
├── data
│   ├── Face Mask Dataset         # dataset folder
│   │   └── Train                 # to train data
│   │       ├── WithMask
│   │       ├── WithoutMask
│   │       └── Random
│   │   └── Test                  # to test model
│   │       ├── WithMask
│   │       ├── WithoutMask
│   │       └── Random
│
└── Train_Model.py
```

Below are the sources reference table where the datasets have been taken from Kaggle accordingly.

For Training			
	WithMask	Withoutmask	Random
Dataset sources	https://www.kaggle.com/ashishjangra27/face-mask-12k-images-dataset https://www.kaggle.com/omkargurav/face-mask-dataset	https://www.kaggle.com/ashishjangra27/face-mask-12k-images-dataset https://www.kaggle.com/omkargurav/face-mask-dataset	https://www.kaggle.com/chetankv/dogs-cats-images https://www.kaggle.com/sdevkota007/vehicles-nepal
Images	400	400	400
For Testing			
Dataset sources	https://www.kaggle.com/ashishjangra27/face-mask-12k-images-dataset https://www.kaggle.com/omkargurav/face-mask-dataset	https://www.kaggle.com/ashishjangra27/face-mask-12k-images-dataset https://www.kaggle.com/omkargurav/face-mask-dataset	https://www.kaggle.com/chetankv/dogs-cats-images https://www.kaggle.com/sdevkota007/vehicles-nepal
Images	200	200	200

Table 1: Sources reference table

1.2 Data pre-processing and Transformation

Datasets used in this project have been downloaded from Kaggle website. After downloading dataset to computer hard drive, unzipping it, and then creating pandas [3] data frame and finally saving it as a pickle [4] file where the python object is converted into a byte stream also called flattening so that we can save our time by not reading all dataset images again and again rather simply load the pickle file which is less than 1Mbit. All this functionality has been included in the Data_setup.py file. This includes a custom Dataset class.

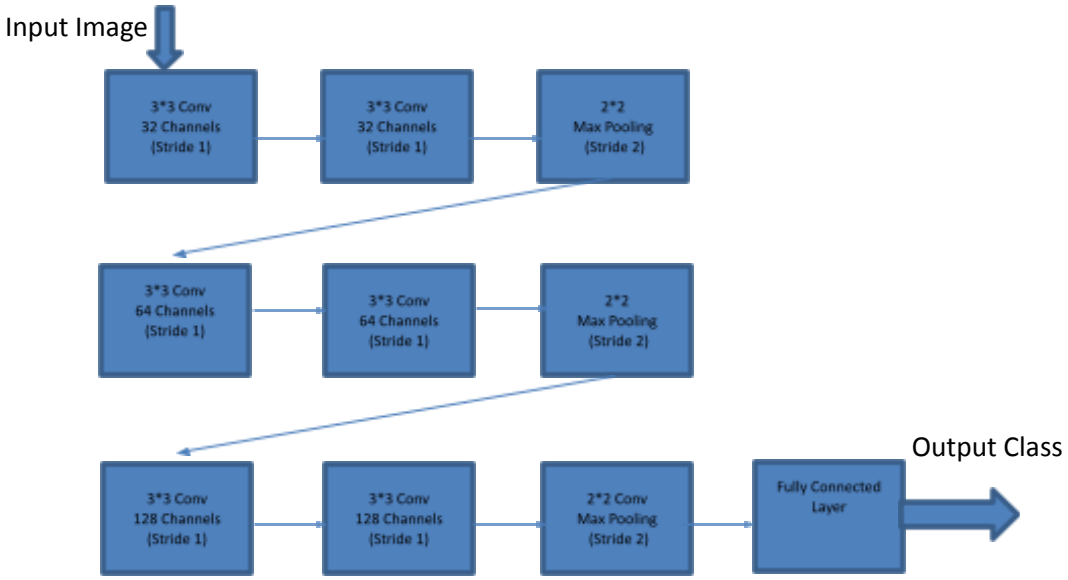
Below tasks were performed to all the dataset to transform them before the process and feeding them to a neural network learning model. This will allow the model to train well and work faster.

1. Resizing the images to 32*32 since the neural network expects a square image.
2. To ensure each data pixel has a similar distribution we are normalizing them.
3. Finally Converting them into Pytorch tensors.

For the input data we found various datasets on the Kaggle website. But for the random sets we combined various datasets. Since a street view camera comes across many things, we tried to include cats, dogs, trees, and vehicles images in our dataset.

2. CNN Model Architecture

Here in this project, we are using ***Sequential CNN model*** [7] with consisting of various layers such as Convolutional Layers, LeakyReLU activation function, Max-Pooling, and at the end fully connected layer for the final classification.



Process Flow diagram of the CNN model

In CNN architecture, nn.Sequential module [8] used to create sequentially ordered layers of convolution layers, activation function ReLU and max pooling sequence. First convolution layer extracting more ground features whereas the second Convolutional Layer extracts more advanced features from images. CNN uses convolution kernels (activation map) to convolve with the original images or feature maps to extract higher-level features

To accelerate the training process in each convolution layer it is a good approach to use BatchNorm2d [9] as well as LeakyRelu for the activation function.

In the training module, we define a class name FaceMaskDetectorCNN, which inherits nn.Module module. We have chosen the hyperparameters as follows: initial learning rate is taken as 0.001, batch size is taken to be 32 and number of epochs as 7. One epoch is when the whole dataset goes

through both forward and backward pass in the neural network. Hyperparameters were selected such that we would achieve better performance of our model during evaluation and testing phase. We have tried with a greater number of epochs for example 20 but it will decrease the accuracy, possibly due to the overfitting. So, we finalized with 7 epochs.

For training, we call three neural network sequential layers objects and details of training process is as follows:

2.1.1 NN – First Sequential Layer

In the first Sequential layer, we give the input to first convolutional layer [6] where we give the input channel of 3 layers as RGB and we get output channel of 32, kernel size (feature extractor) of 3X3 weight matrix and use padding 1 layer in order to maintain the image size. After the convolutional, apply batch normalization which accelerates training by internal covariate shift Since in the neural network during training the model, input changes at each layer which slows down the training by lowering the learning rate. Batch Normalization allows us to use much higher learning rates and be less careful about initialization [10].

Next step is to apply activation function, we use 'LeakyRelu' (Rectified Linear Unit function). This layer extracts generic features of images. Then the output would be fed to the next convolutional layer+LeakyRelu by maintaining the image size and parameters used for the first layer, here we get again 32 output channels. After that Max pooling function applies with the feature filter of 2X2 matrix and the stride of 2. Here it is worth to mention that to avoid Overfitting (increased capacity to learn by heart by Neuron/perceptron OR Overfitting is when Neurons try to memorise instead of generalizing which we do not need during training) we used Dropout value of 0.1 which avoid the network to dependent on any one neuron.

The training process goes on and after the 6th convolutional layer we get 64 output channels. This layer learns higher-level features of images.

2.1.2 NN – Second Sequential Layer

Then apply linear transformation to the 64 input channels [6] where 1000 hidden neurons are used and we get a smaller resulting image with 8X8X64 output channels, this would considerably shrink our input data. After that pass the data through Relu + dropout and finally passing through linear transformation consisting of 512 neurons, we get 3 outputs.

In the training process, we define forward class in which the input passed through the convolutional layers and once we get output after max-pooling then need to reshape it or flatten it in 1-dimensional array and for that we use x.view (Pytorch function) which copies all values and convert it into 1 big array which then forwarded to fully connected layer which give us 3 predicted output values which correspond to our three classes.

2.1.3 NN - Third Sequential Layer

Similarly, we used the third sequential layer with input channels = 64 and output channels = 128.

3. Training

As mentioned above, approximate 70% of the whole dataset is used for training the model and the rest of 30% put aside for testing our CNN model. The Dataloader splits them into batches and each batch is forwarded to the CNN in one iteration. Here we defined batch size to be 32.

In the training, we loop the number of epochs and in each loop, we iterate through each input from the batch and forward it to CNN. The CNN returns predicted class for the given input. We then calculate the loss using cross entropy loss function by using the input class label and the predicted label. Then the weights and biases need to be modified. This is calculated using Adam optimizer. We use Adam [8] (short for Adaptive Moment Estimation) optimization algorithm which is used to train deep learning neural networks and is a method that computes adaptive learning rates for each parameter and to change the weight to minimize the losses. In recent studies Adam optimization has given better results so we chose this optimizer.

Cross entropy loss is handy in training classification images especially when one has an unbalanced training set [12]. The output for 3 different classes which we get from our CNN model, we pass to the Cross entropy loss function and measure the losses in forward pass and once the losses would be calculated then we call the method optimizer.step() to perform optimization on the training model which updates the weights by first resetting the optimizer using function optimizer.zero_grad(). We keep track of loss for each iteration using train_loss.

4. Evaluation

Evaluation is a very crucial part to check the performance of any AI CNN model. After training the model with several epochs, we used the rest of the dataset which is 20% for testing the model and fed this data to our model to evaluate how well-trained our network model is and how good is the accuracy of the AI model on unseen images. Like the training process, each image in test data has been performed transformations and passed to the NN. Unlike training we do not need to drop-out any of the neurons in the network. For this purpose we call model.eval() to let the network know that training has been finished.

For evaluating any model, the important metrics which we need to look at are accuracy, precision, recall and F1 which we can observe from seeing the classification graph generated because of CNN model classification.

For this we implement the class named test_model() inside which we are passing the input image to the NN and getting its predicted class label. Unlike training we neither calculate any loss nor perform any optimization. The predicted class label is compared with the actual class label to determine the model's accuracy. The test accuracy of the CNN model which we get from testing on dataset images is 91.45833333333333 %.

This predicted result of the model is satisfactory. However, we may improve this accuracy in the next phase of project assignment which mainly consists of extended evaluation of our AI network model.

The following images show some of the statistics obtained on the test data. We used a classification report module of scikit-learn [18] package for this purpose.

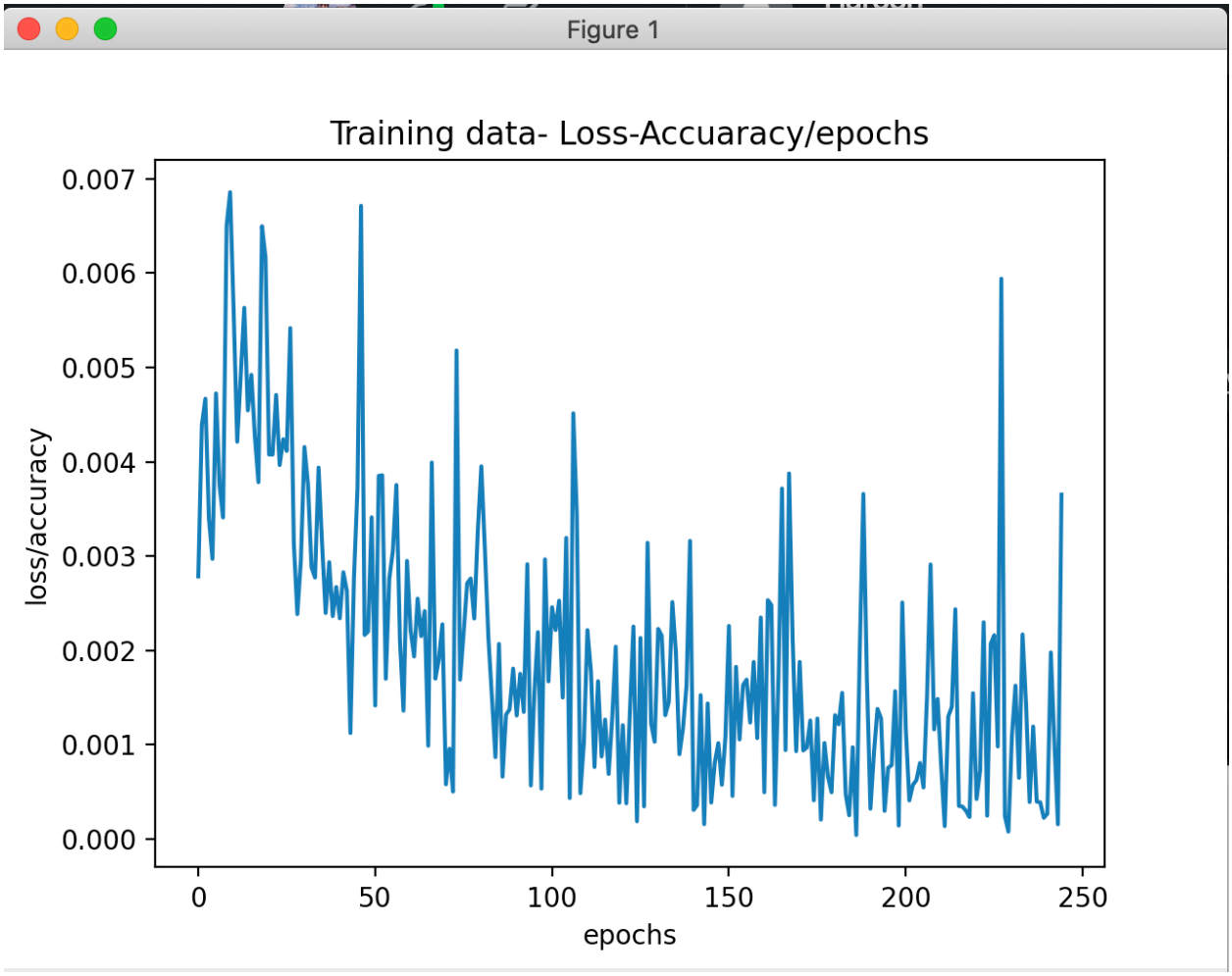
Classification Report for the test data

	precision	recall	f1-score	support
With Mask	0.95	0.95	0.95	150
Without Mask	0.84	0.97	0.90	150
Not a Person	0.97	0.84	0.90	180
accuracy			0.91	480
macro avg	0.92	0.92	0.92	480
weighted avg	0.92	0.91	0.91	480

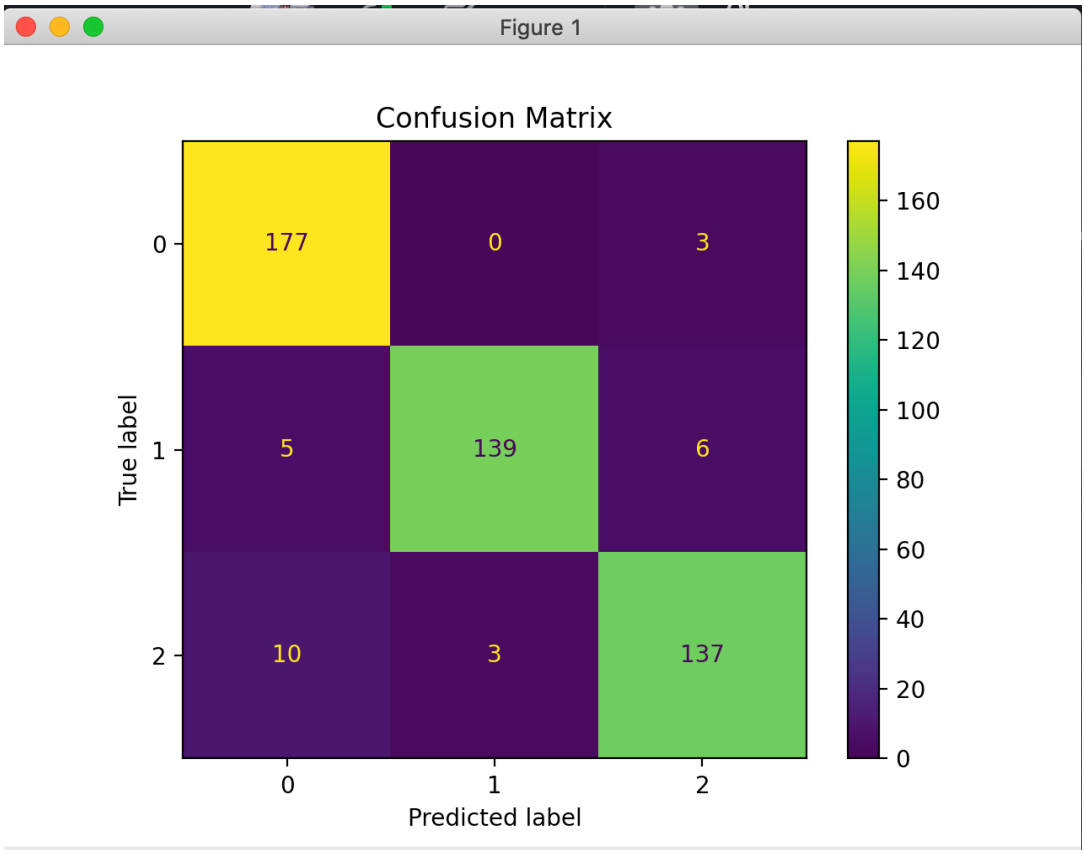
We can see that out of 95% of images predicted as With Mask were actually images with a person with a mask. Similarly, precision for without mask class is 84% and for random images it is 97%. Also, we can observe that recall for class without mask is 97% implying that 97% of the actually without mask data was classified as without mask. Similarly, the recall for class with mask and random. However, precision and recall are not good measures alone hence they can be combined by taking their harmonic mean which is given by f1-measure.

The following graph shows the training loss-accuracy vs epochs.

We can observe that loss has reduced over the epochs.



The following depicts the confusion matrix of our test data.



The following are the labels and their numeric representation.

With mask - 0

Without mask -1

Random -2

So the confusion matrix says that out of 192 images of class 0 i.e."With Mask":

- 177 were classified as with masks.
- 5 were classified as without mask
- 2 were classified as random

Future-Work

In future we would like to improve the efficiency of this project by trying to correctly classify images with multiple people to ensure there are no biases.

5. Biasing for Gender Detection:

In the second phase of this project, we are going to evaluate biasing on the basis of gender in our AI model, first by proving that model is biased [19] and afterwards get rid of this possible bias either partially or fully however possible.

The important thing for testing bias is a complete set of data available for both male and female respectively. The whole dataset has been downloaded from kaggle. In project phase-II, we analyse our trained CNN model for the category, gender detection. For this we divided datasets and trained the model as mentioned in following rounds:

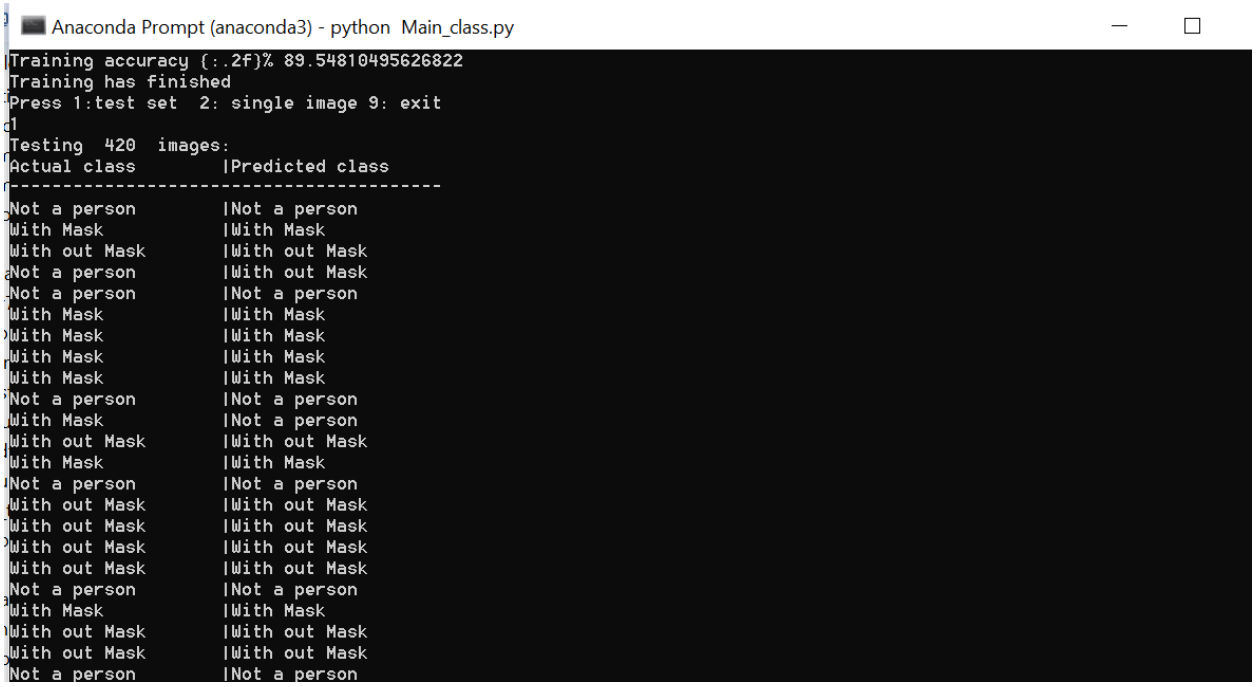
1. In the first round, we trained the model against the datasets for male category and for this a sound number of images were required. Initially we tried with 200 images which resulted in accuracy output which had bias of around 16% in favour of Male Dataset. Therefore, in order to decrease/remove that bias, we increased the number of images and split data into three subclasses: almost 500 images of men with masks, 500 images without masks and 400 random images. This resulted in 89.54% and 91.43% accuracy for training and testing respectively.

2. Similarly, in the second round, we trained the model against the datasets for female category and datasets for the female category split into three further subclasses; almost 500 images of women with masks, 500 images without masks and 400 random images resulting in 90.64% and 95.23% accuracy for training and testing respectively.
3. In the third round, we mixed up all the images of men, women and a new set of random images to train the model and evaluate the model performance with respect to gender category only and compared the evaluation metrics: precision, recall, F1-measure with our already trained model developed and tested in project phase-1.

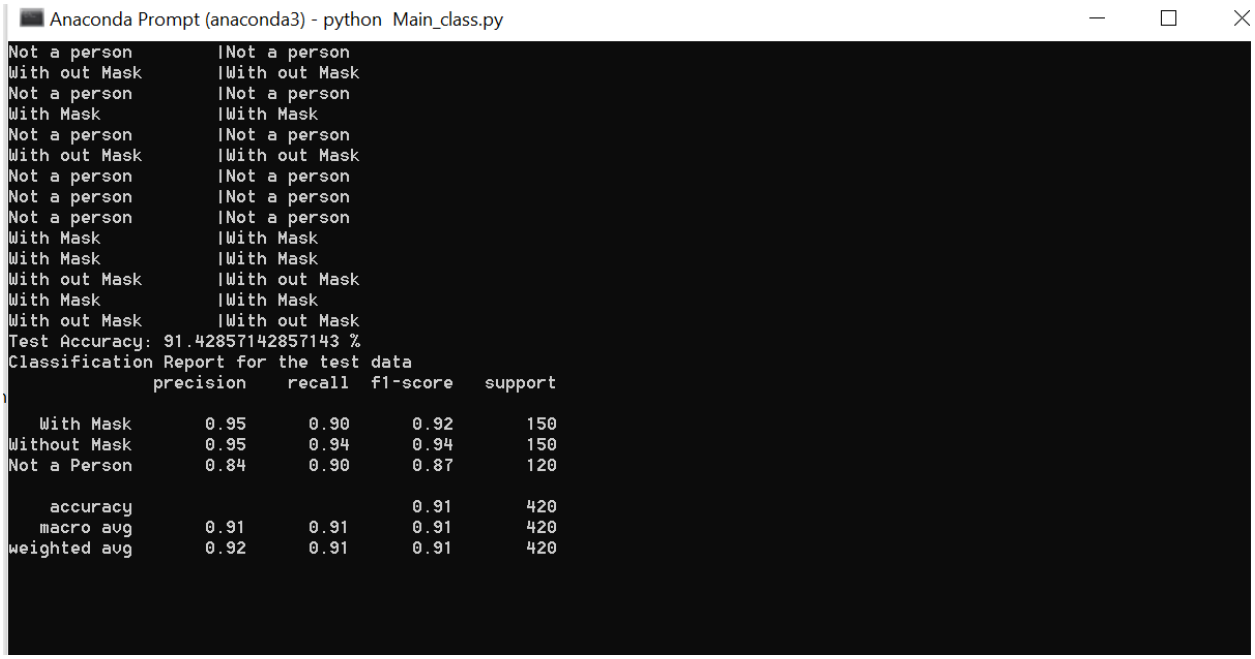
During the training of our CNN model, we found that even AI systems can be gender biased. That means the performance is not consistent for both male and female faces corresponding to all three different classes (mask, no-mask and for random data class). We found better accuracy for the female dataset as compared to the male dataset.

In order to circumvent this gender biasing effect, rebalanced the training data and fed it to AI. The results are as below:

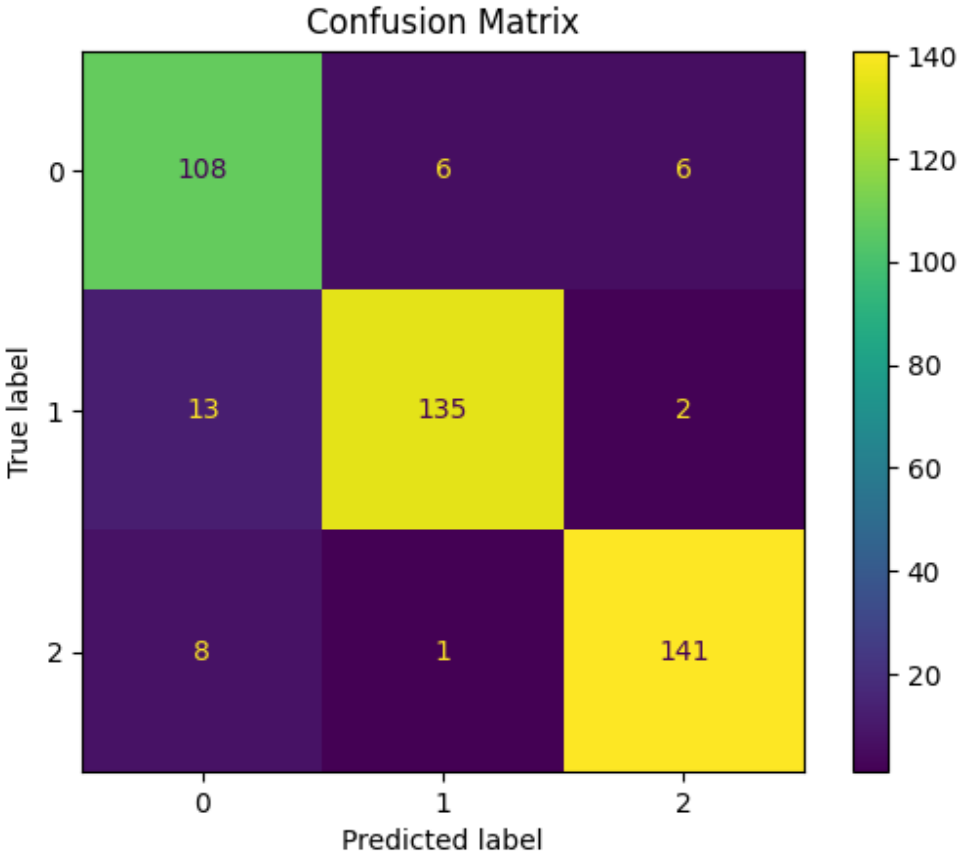
Male Dataset Training Accuracy:



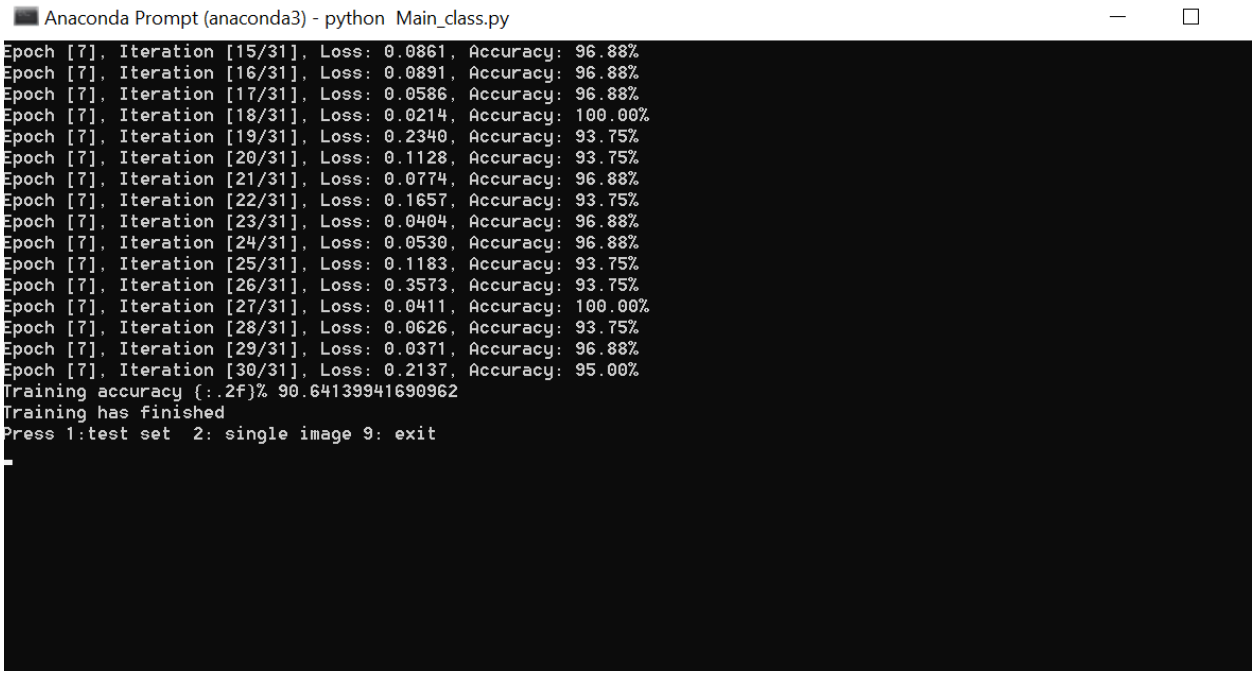
Male Dataset Testing Accuracy:



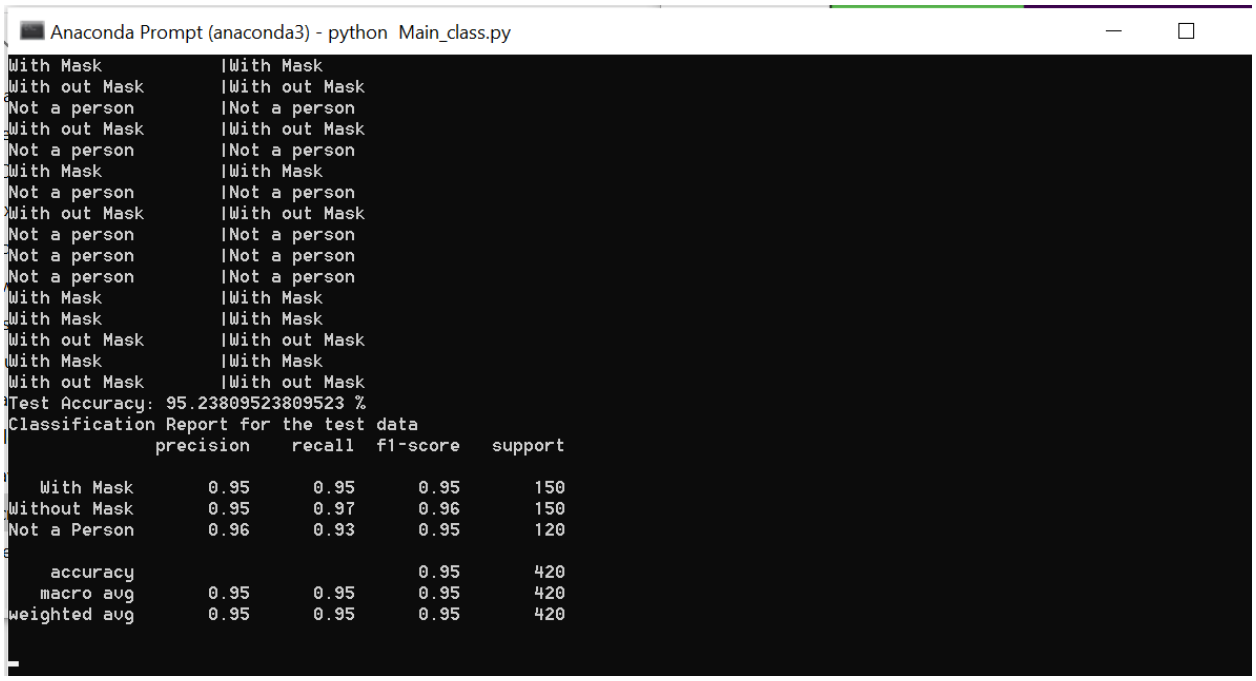
Male Dataset Confusion Matrix:



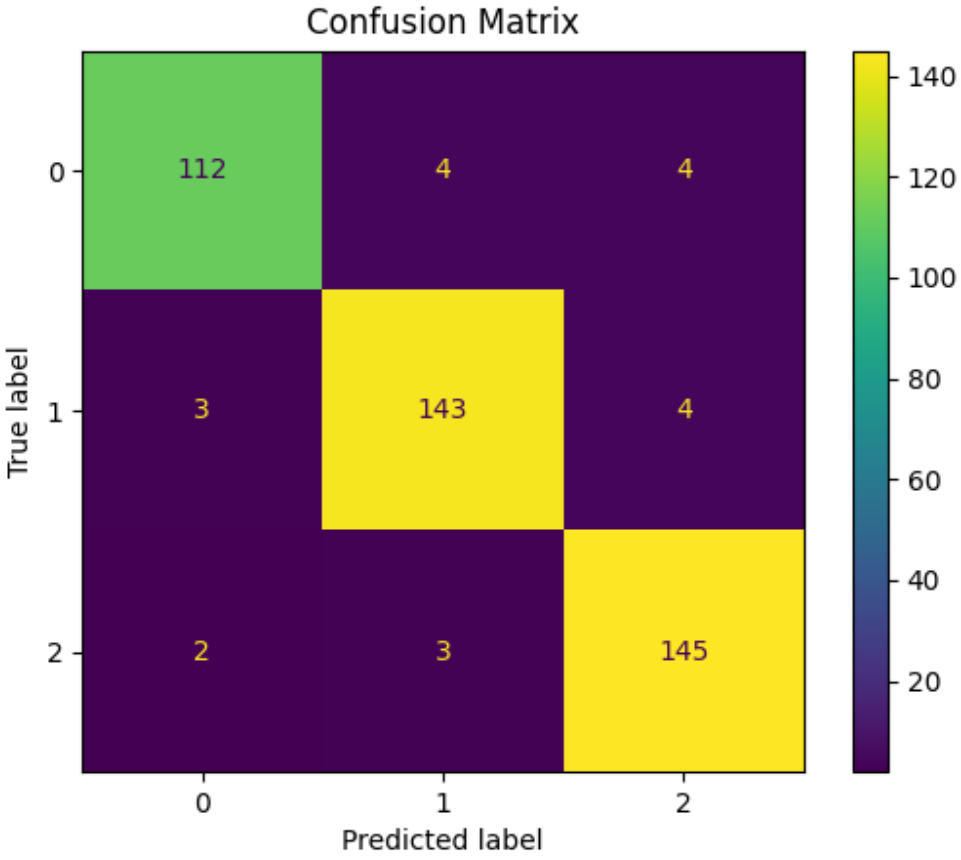
Female Dataset Training Accuracy:



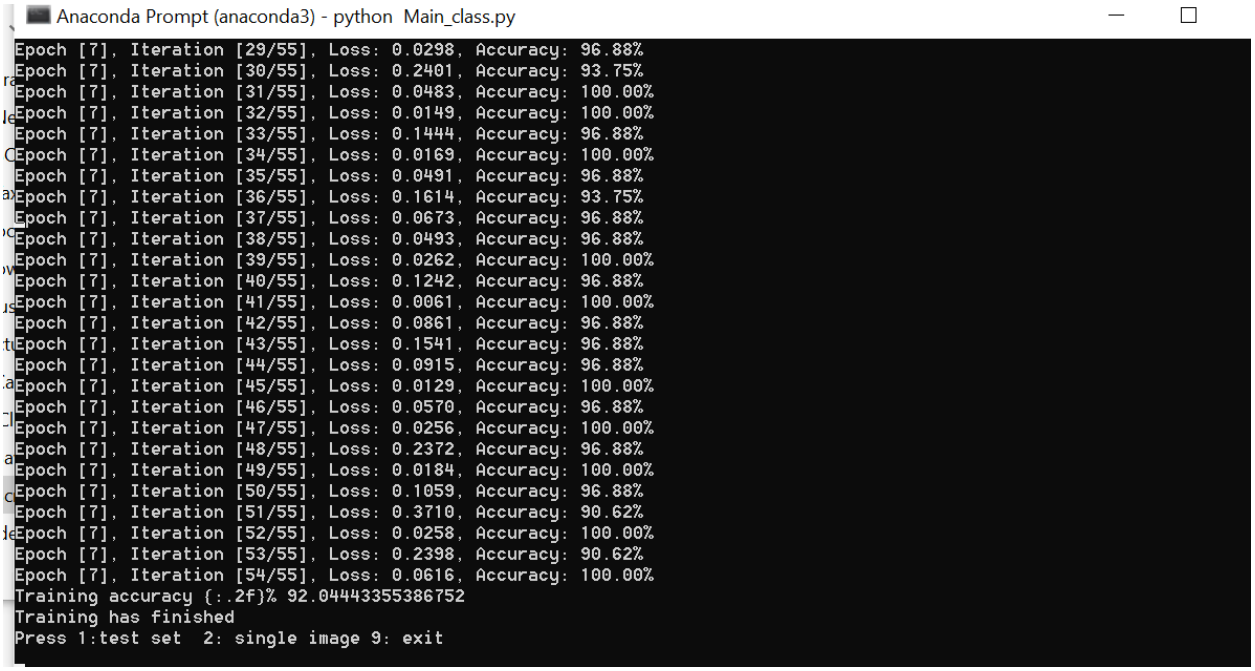
Female Dataset Testing Accuracy:



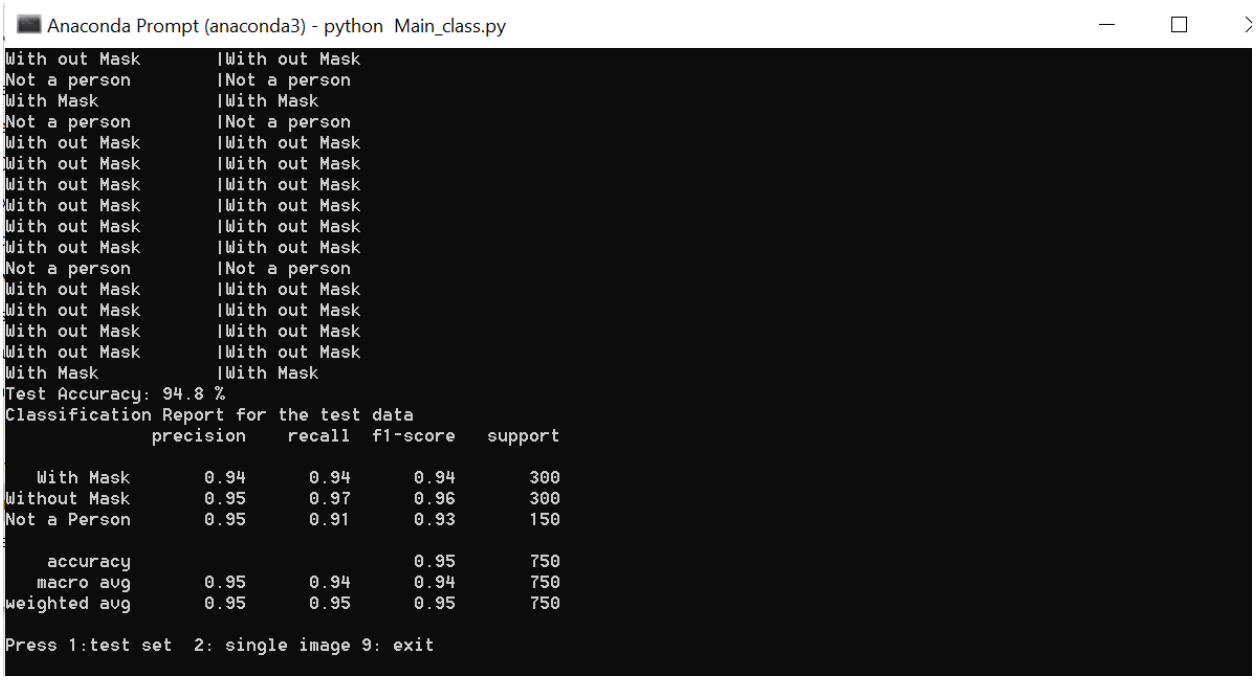
Female Dataset Confusion Matrix:



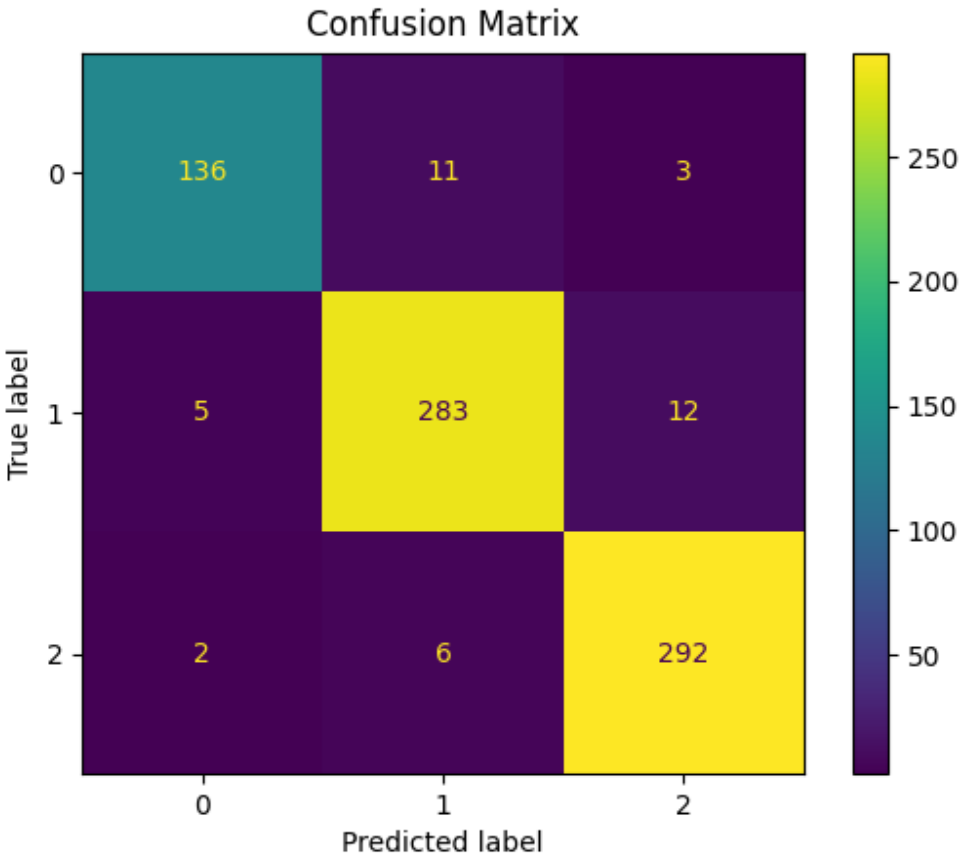
Mixed Dataset Training Accuracy:



Mixed Dataset Testing Accuracy:



Mixed Dataset Confusion Matrix:



6. Evaluation: K-fold Cross-validation

For the cross-validation we are using the standard K-Fold split[17] function defined in the scikit-learn implementation[18]. We are implementing this in the prepare_data_kfold () method of Data_Setup.py class. Here we are defining the number of folds to be 10 with random shuffling. So we have set parameter n_splits =10 with shuffle = True and random_state=42 since the common values for random_state is 0 or 42, we chose 42.

We are creating a list of lists of training data where each list consists of images of 1 fold. We are then iterating over this list and passing each fold to train_model method of Train class. For each

fold we are calculating the accuracy. Also, we are showing the aggregate accuracy of all the folds after training the model on all folds.

Training accuracy for Female dataset :

```
Training has finished
Accuracy for each K-Fold:
 1 Fold accuracy: 91.780045
 2 Fold accuracy: 97.199546
 3 Fold accuracy: 98.605442
 4 Fold accuracy: 98.809524
 5 Fold accuracy: 99.115646
 6 Fold accuracy: 99.319728
 7 Fold accuracy: 99.297052
 8 Fold accuracy: 99.195011
 9 Fold accuracy: 99.092971
 10 Fold accuracy: 99.206349
Training accuracy 98.16%
```

Training accuracy for Male dataset :

```
Training has finished
Accuracy for each K-Fold:
 1 Fold accuracy: 90.306122
 2 Fold accuracy: 96.303855
 3 Fold accuracy: 98.015873
 4 Fold accuracy: 98.696145
 5 Fold accuracy: 98.832200
 6 Fold accuracy: 98.752834
 7 Fold accuracy: 99.263039
 8 Fold accuracy: 99.070295
 9 Fold accuracy: 99.263039
 10 Fold accuracy: 99.455782
Training accuracy 97.80%
```

Training Accuracy for Combined Dataset:

```
Training has finished
Accuracy for each K-Fold:
 1 Fold accuracy: 92.390269
 2 Fold accuracy: 97.230515
 3 Fold accuracy: 98.418345
 4 Fold accuracy: 98.920155
 5 Fold accuracy: 98.945563
 6 Fold accuracy: 99.167884
 7 Fold accuracy: 99.529950
 8 Fold accuracy: 99.726863
 9 Fold accuracy: 99.504542
 10 Fold accuracy: 99.447619
Training accuracy 98.33%
```

From the given screenshot above we can observe that accuracy has been improving as we train the model on each fold. The aggregate accuracy obtained is 98.33% with K-Fold split whereas it was only 92.04% with train_test_split in the last assignment. Thus we can conclude the training accuracy has improved with K-Fold split. Coming to the Test accuracy, previously our accuracy was 94.8% with train_test_split but after checking for bias and retraining the model we have achieved 100% accuracy. Following screenshots depict the accuracies obtained with K-Fold split.

Test Accuracy for Female dataset:

```
Test Accuracy: 100.0 %
Classification Report for the test data
              precision    recall  f1-score   support

   With Mask       1.00      1.00      1.00        55
Without Mask       1.00      1.00      1.00        59
Not a Person       1.00      1.00      1.00        26

    accuracy                           1.00       140
   macro avg       1.00      1.00      1.00       140
weighted avg       1.00      1.00      1.00       140
```

Test Accuracy for Male dataset:

```
Test Accuracy: 100.0 %
Classification Report for the test data
              precision    recall  f1-score   support

   With Mask       1.00      1.00      1.00        55
Without Mask       1.00      1.00      1.00        59
Not a Person       1.00      1.00      1.00        26

    accuracy                           1.00       140
   macro avg       1.00      1.00      1.00       140
weighted avg       1.00      1.00      1.00       140
```

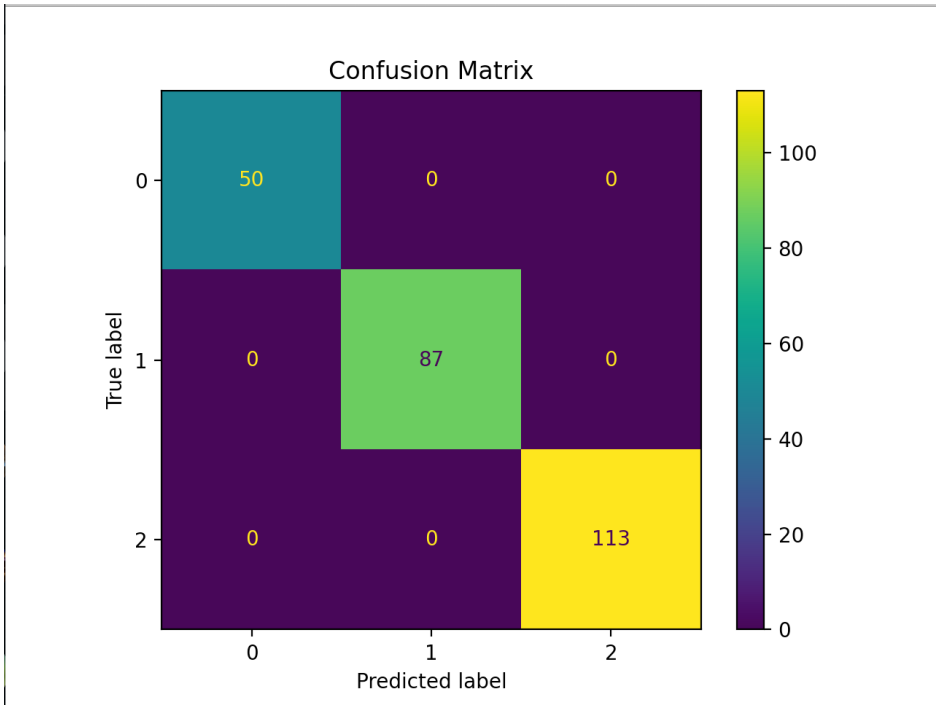
Testing Accuracy for Combined Dataset:

```
Test Accuracy: 100.0 %
Classification Report for the test data
              precision    recall  f1-score   support

   With Mask       1.00      1.00      1.00        87
Without Mask       1.00      1.00      1.00       113
Not a Person       1.00      1.00      1.00        50

    accuracy                           1.00       250
   macro avg       1.00      1.00      1.00       250
weighted avg       1.00      1.00      1.00       250
```

Confusion Matrix for Combined Dataset:



7. Conclusion and Future Work

	Train_Test_Split			K-Fold		
	Male	Female	Combined	Male	Female	Combined
Training	89.54	90.64	92.04	97.80	98.16	98.33
Testing	91.43	95.23	94.8	100	100	100

The results of all our experiments have been summarised in the above table. Following observations can be made:

1. K-Fold split has increased the accuracy of our model compared to the train_test_split accuracy.

2. Our model shows more accurate results in case of Female images. Earlier the accuracy difference between male and female was very large but retraining the data has reduced the difference. But it is still biased towards female images.

References

1. https://pytorch.org/
2. https://www.kaggle.com
3. https://pandas.pydata.org/
4. https://docs.python.org/3/library/pickle.html#module-pickle

5. https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html
6. https://pytorch.org/docs/stable/nn.html#linear-layers
7. https://pytorch.org/docs/stable/generated/torch.nn.Sequential.html
8. https://en.wikipedia.org/wiki/Stochastic_gradient_descent#cite_note-Adam2014-26
9. https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html#torch.nn.BatchNorm2d
10. https://arxiv.org/abs/1502.03167
11. https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html
12. https://pytorch.org/docs/stable/generated/torch.max.html
13. https://github.com/JadHADDAD92/covid-mask-detector/tree/0eab0b5ffd218b64108960e9bd3789fc775af485
14. https://github.com/whitetig3r/face-mask-detector-cnn/blob/master/face_mask_detector_cnn.ipynb
15. https://github.com/PacktPublishing/Deep-learning-with-PyTorch-video/blob/master/2.training.your.first.neural.network.ipynb
16. https://becominghuman.ai/image-data-pre-processing-for-neural-networks-498289068258
17. https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-iterators
18. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html
19. https://www.media.mit.edu/projects/gender-shades/overview/