Concordia University

Engineering and Computer Science

COMP 6721- **Applied Artificial Intelligence**

Part I-Project: AI Face Mask Detector

Submitted By: Team(AK_10)

Aruna Devi Pala (40184469) – Data Specialist

Divyanatha Reddy Dantu (40199280) – Training Specialist

Anusha Reddy Lattupally (40163526) – Evaluation Specialist

GitHub Link: https://github.com/Aruna-Pala/AI_FaceMaskDetection

Contents

1. Dataset:

The data for the project is gathered from Kaggle (https://www.kaggle.com) by using different references which will be mentioned by the end of this document. Initially, we gathered images based on the given 5 different categories which are "Cloth mask, No face mask, Surgical mask, N95 mask, and Mask worn incorrectly". While validating the data we found many limitations in the collected data such as face angle, lightning conditions etc., All such limitations are normalized in the data by sorting out manually and removing noisy data from collected datasets.

1.1. Statistics:

Collected data consists of 1522 Training images and 507 Testing images. Both Training and testing data are being balanced, which means in each category we have collected $1/5^{th}$ images.

Prediction data consists of 100 images.

Training and Testing data are divided as shown below:

Training Dataset			Testing Dataset		
S.No	**Category**	**#Data**	**S.No**	**Category**	**#Data**
1	Cloth Mask	300	1	Cloth Mask	100
2	No face Mask	300	2	No face Mask	100
3	Surgical Mask	300	3	Surgical Mask	101
4	N95 Mask	305	4	N95 Mask	103
5	Mask Worn incorrectly	317	5	Mask Worn incorrectly	103
	Total Count	**1522**		**Total Count**	**504**

Among the data mentioned above we have selected 20% of the data to generate Prediction model. We split the remaining images between training and Validation sets (75% training and 25% Validation).

1.2 Data Pre-Processing:

We have performed the following steps before feeding the data into our Convolutional Neural Network.

1. **Conversion of images:** We have converted NumPy images to Tensor images by using the ToTensor() function which converts 0-255 NumPy images to 0-1 Tensor images.
2. **Resizing images:** We have resized all the testing and training images to 180*180.
3. **Normalization:** We have normalized complete data to transform the features to be on a similar scale which helps us to improve the performance and stability of the model developed.

1.3 Loading Dataset:

Dataset which is being split into 75% training and 25% Validation is transformed using data preprocessing which is explained in section 1.2., then finally this will be fed to the network in batches of 12 considering batch size of 128 using PyTorch DataLoader. To Split training and testing data we have used torch.utils.data.dataset.random_split function and data will be automatically split with 1500 training images and 500 testing images.

2. CNN Architecture:

CNN is nothing but Convolution Neural Network where the unique feature of the image data is being extracted, the most complex features are also being detected using this architecture. There are many other applications which use this architecture such as face detection and video recognition.

There are three main types of CNN network:

1. **Convolution Layer:** The convolution layer transforms to statistical values from the image during the image extraction process.
2. **Pooling Layer:** Also called down sampling as it decreases the number of input samples.
3. **Fully connected layer:** Here each node in the output layer connects to the node in the previous layer and executes the tasks which depend on the features taken from the previous layer.

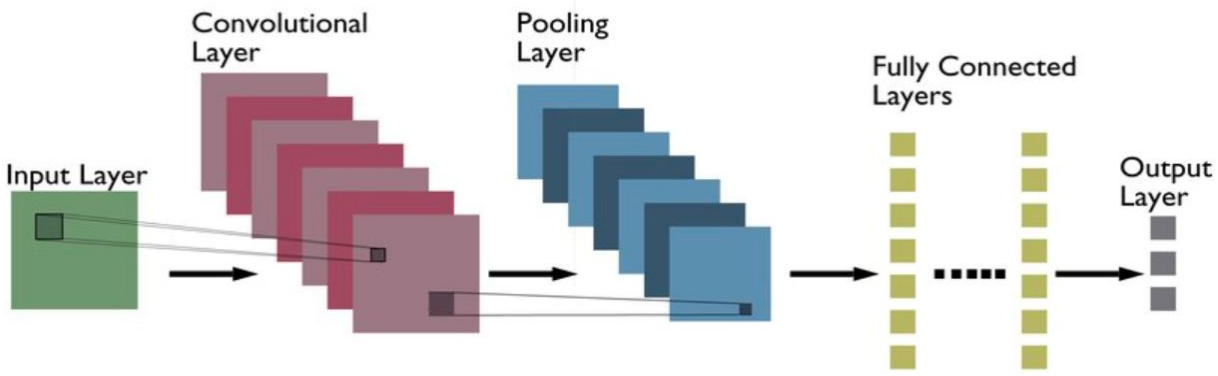

Figure 1: Model Architecture

2.1 Model Layers:

Here there are five convolution layers in the architecture,

Convolutional Layer 1: Here the parameters taken were the number of input channels 3, output channels 12, kernel size as 3, stride as 1 and padding 1. The output channel layer will be the input to the next upcoming layers. We use batch normalization layer with activation function which is used for nonlinear transformation. This is followed by Max Pooling Layer with kernel size 2 which minimizes the input dimensions.

Convolutional Layer 2: Second layer, the parameters taken were the number of input channels 12, output channels 20, kernel size as 3, stride as 1 and padding 1 which is same as layer 1. We use batch normalization layer with features 20 with activation function which is used for nonlinear transformation.

Convolutional Layer 3: Third layer, the parameters taken were the number of input channels 20, output channels 32, kernel size as 3, stride as 1 and padding 1 which is same as layer 1. We use batch normalization layer with a few features 32 with activation function which is used for nonlinear transformation.

Convolutional Layer 4: Fourth layer, the parameters taken were the number of input channels 32, output channels 46, kernel size as 3, stride as 1 and padding 1 which is same as layer 1. We use batch normalization layer with a few features 46 with activation function which is used for nonlinear transformation.

Convolutional Layer 5: Fifth layer, the parameters taken were the number of input channels 46, output channels 64, kernel size as 3, stride as 1 and padding 1

which is same as layer 1. We use batch normalization layer with a few features 64 with activation function which is used for nonlinear transformation.

2.2. Proposed CNN Architecture Model:

2.2.1 Final Model Architecture:

self.conv1 = nn.Conv2d(in_channels=3, out_channels=12, kernel_size=3, stride=1, padding=1)

self.pool1 = nn.MaxPool2d(kernel_size=2)

self.conv2 = nn.Conv2d(in_channels=12, out_channels=20, kernel_size=3, stride=1, padding=1)

self.conv3 = nn.Conv2d(in_channels=20, out_channels=32, kernel_size=3, stride=1, padding=1)

self.conv4 = nn.Conv2d(in_channels=32, out_channels=46, kernel_size=3, stride=1, padding=1)

self.conv5 = nn.Conv2d(in_channels=46, out_channels=64, kernel_size=3, stride=1, padding=1)

self.fc1 = nn.Linear(in_features= 64 * 64 * 64, out_features=num_classes)

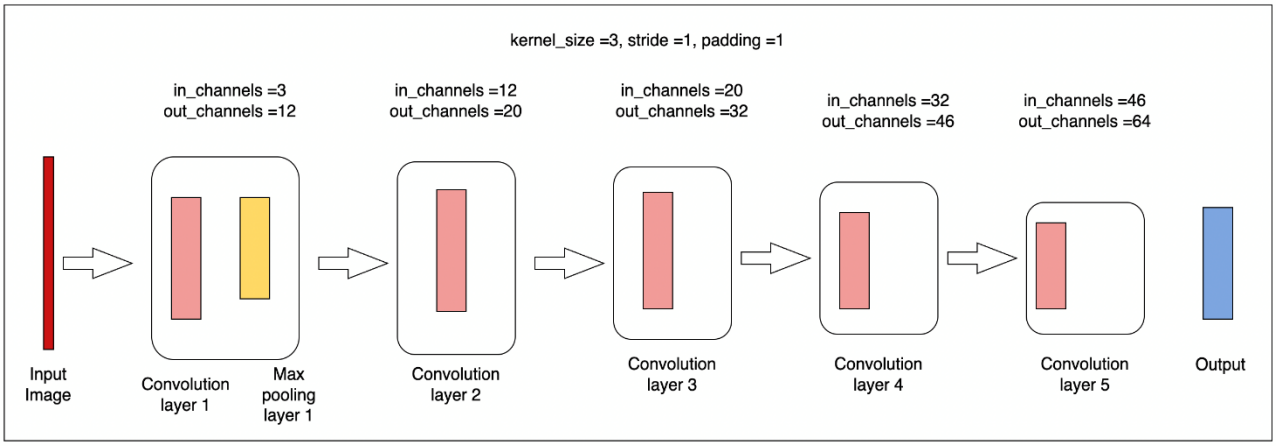

Figure 2: Final Model Architecture Flow

2.2.2 Variant 1:

self.conv1 = nn.Conv2d(in_channels=3, out_channels=12, kernel_size=3, stride=1, padding=1)

self.conv2 = nn.Conv2d(in_channels=12, out_channels=20, kernel_size=3, stride=1, padding=1)

self.pool1 = nn.MaxPool2d(kernel_size=2)

self.conv3 = nn.Conv2d(in_channels=20, out_channels=28, kernel_size=3, stride=1, padding=1)

self.conv4 = nn.Conv2d(in_channels=28, out_channels=36, kernel_size=3, stride=1, padding=1)

self.pool2 = nn.MaxPool2d(kernel_size=2)

self.conv5 = nn.Conv2d(in_channels=36, out_channels=40, kernel_size=3, stride=1, padding=1)

self.conv6 = nn.Conv2d(in_channels=40, out_channels=48, kernel_size=3, stride=1, padding=1)

self.pool3 = nn.MaxPool2d(kernel_size=2)

self.fc1 = nn.Linear(in_features=16 * 16 * 48, out_features=64)

self.fc2 = nn.Linear(in_features=64, out_features=32)

self.fc3 = nn.Linear(in_features=32, out_features=num_classes)

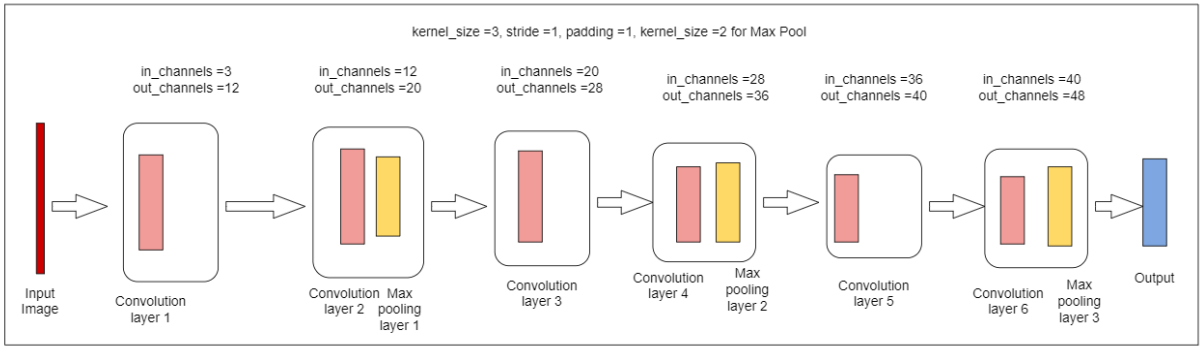

Figure 3: First Variant Model Architecture Flow

2.2.3.Variant 2:

self.conv1 = nn.Conv2d(in_channels=3, out_channels=12, kernel_size=3, stride=1, padding=1)

self.pool1 = nn.MaxPool2d(kernel_size=2)

self.conv2 = nn.Conv2d(in_channels=12, out_channels=20, kernel_size=3, stride=1, padding=1)

self.conv3 = nn.Conv2d(in_channels=20, out_channels=28, kernel_size=3, stride=1, padding=1)

self.conv4 = nn.Conv2d(in_channels=28, out_channels=36, kernel_size=3, stride=1, padding=1)

self.conv5 = nn.Conv2d(in_channels=36, out_channels=40, kernel_size=3, stride=1, padding=1)

self.conv6 = nn.Conv2d(in_channels=40, out_channels=48, kernel_size=3, stride=1, padding=1)

self.fc1 = nn.Linear(in_features= 64 * 64 * 48, out_features=num_classes)

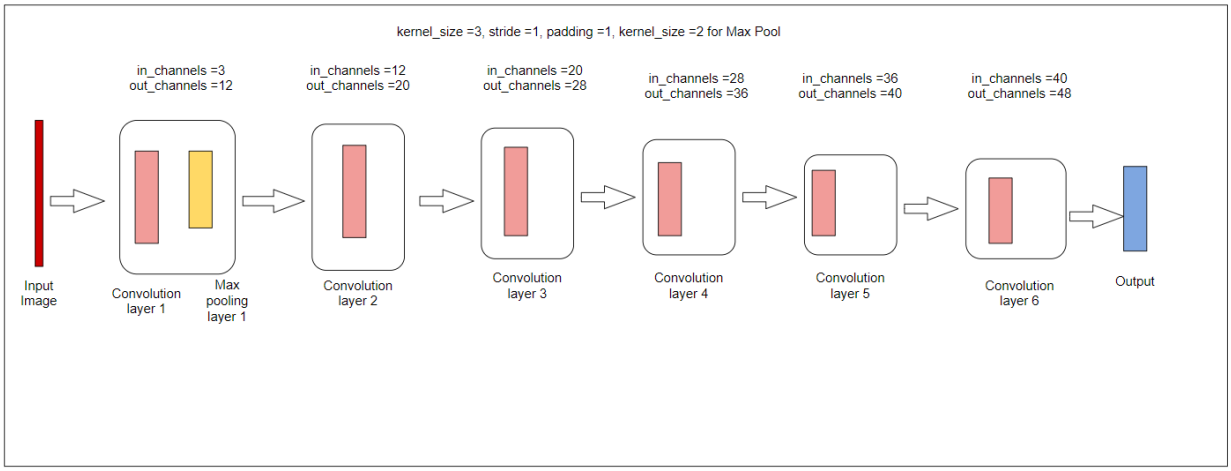

Figure 4: Second Variant Model Architecture Flow

2.3 Model Parameters:

There are four parameters used in this architecture such as Optimizer, Loss Function, Batch size, Epochs.

1. Optimizer: Generally used to reduce error by adjusting model parameters, in this case Adam optimizer is being used for optimization purposes and it is initialized by registering model parameters and learning rate parameters is passed where the weights are equally updated.

2.Loss Function: It measures the differences between the target and result of output, and Loss function should be reduced in the training. By which weights can be adjusted to reduce the loss function. Cross entropy is used as Loss Function here. and true data can be compared with the input value so that loss can be predicted and calculated.

3.Batch size: It defines the data samples count which was propagated before the parameters being updated. The batch size taken in this model is 128 with 12 batches as it gives better results.

4.Epochs: As it defines the count of iteration, model trained for 40 epochs.

3.Evaluation

Convolutional Neural Network model that is being proposed in this project was trained with the following hyperparameters:

Parameter	Value
Learning Rate	0.00001
Loss Function	Cross Entropy
Optimizer	Adam Optimizer
Batch Size	128
#Epochs	40

For every epoch, we calculated Train_Accuracy, Test_Accuracy and Train_loss by using the below formula:

```
    loss = loss_criteria(output, target)

    train_loss += loss.item()

    loss.backward()
    optimizer.step()

    print('\tTraining batch {} Loss: {:.6f}'.format(batch_idx + 1, loss.item()))

avg_loss = train_loss / (batch_idx+1)
print('Training set: Average loss: {:.6f}'.format(avg_loss))
```

3.1. Final Architecture evaluation:

Results below shows the first 2 epochs that contains batch size of 12:

Training on mps

Epoch: 1

Training batch 1 Loss: 1.609718

Training batch 2 Loss: 1.606394

Training batch 3 Loss: 1.618221

Training batch 4 Loss: 1.605311

Training batch 5 Loss: 1.614035

Training batch 6 Loss: 1.602707

Training batch 7 Loss: 1.598986

Training batch 8 Loss: 1.606015

Training batch 9 Loss: 1.613507

Training batch 10 Loss: 1.615316

Training batch 11 Loss: 1.602336

Training batch 12 Loss: 1.612136

Training set: Average loss: 1.608723

Validation set: Average loss: 1.606135, Accuracy: 107/507 (21%)

Epoch: 2

Training batch 1 Loss: 1.605789

Training batch 2 Loss: 1.602179

Training batch 3 Loss: 1.605586

Training batch 4 Loss: 1.605678

Training batch 5 Loss: 1.605076

Training batch 6 Loss: 1.599398

Training batch 7 Loss: 1.595878

Training batch 8 Loss: 1.602357

Training batch 9 Loss: 1.601828

Training batch 10 Loss: 1.604716

Training batch 11 Loss: 1.603858

Training batch 12 Loss: 1.598956

Training set: Average loss: 1.602608

Validation set: Average loss: 1.599492, Accuracy: 107/507 (21%)

Below figure explains the difference between Accuracy and loss of various epochs

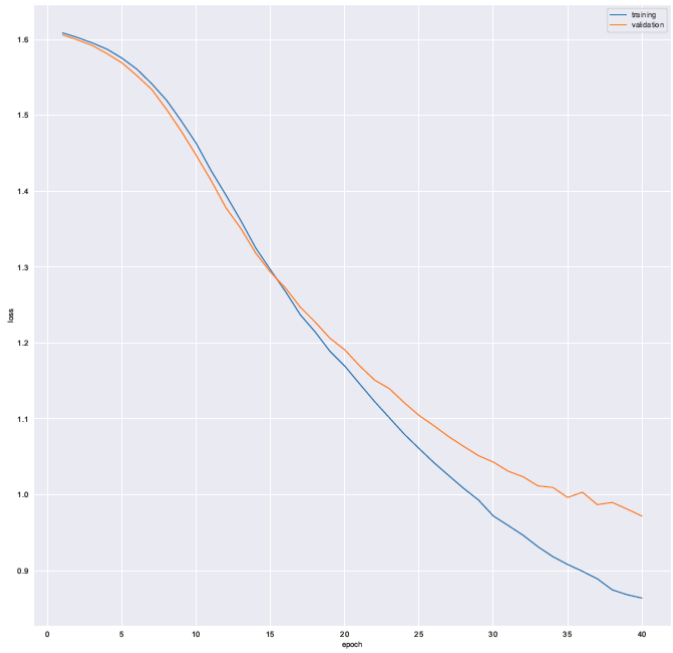

Figure 5: Training and Validation loss Plot for final Architecture

3.1.1. Model Evaluation using Confusion Matrix:

Confusion matrix is used to show the accuracy of the data which will be evaluated in our architecture. During evaluation of test data, the values of predicted and actual will be computed. Using the two classes the Confusion matrix will be calculated. Confusion matrix is as shown below:

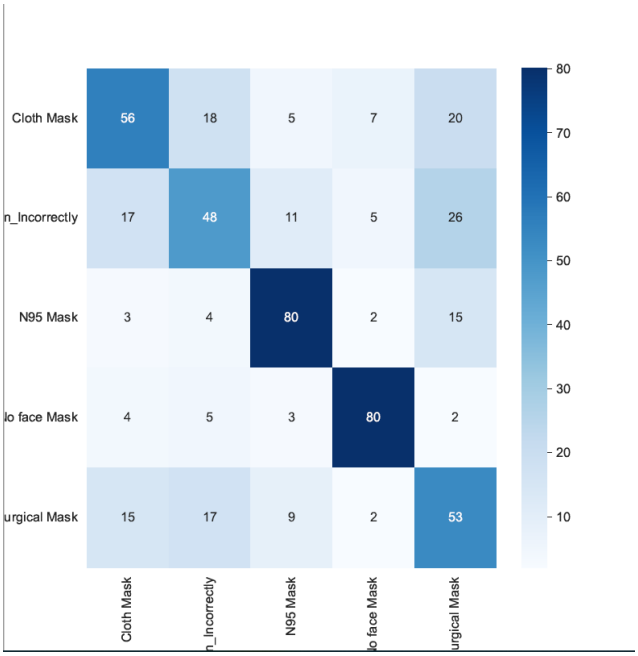

Figure 6: Confusion matrix for Final Architecture

3.1.2. Classification Report:

Below is the detailed report of the precision, recall, and f1-score of the test data for 40 epochs:

	precision	**recall**	**f1-score**	**support**
Cloth Mask	0.589473684	0.528301887	0.55721393	106
Mask_Worn_Incorrectly	0.52173913	0.448598131	0.48241206	107
N95 Mask	0.740740741	0.769230769	0.754716981	104
No face Mask	0.833333333	0.85106383	0.842105263	94
Surgical Mask	0.456896552	0.552083333	0.5	96
accuracy	0.625246548	0.625246548	0.625246548	0.625246548
macro avg	0.628436688	0.62985559	0.627289647	507
weighted avg	0.626317035	0.625246548	0.623927471	507

3.2 First Variant Architecture:

Below figure explains the difference between Accuracy and loss of various epochs

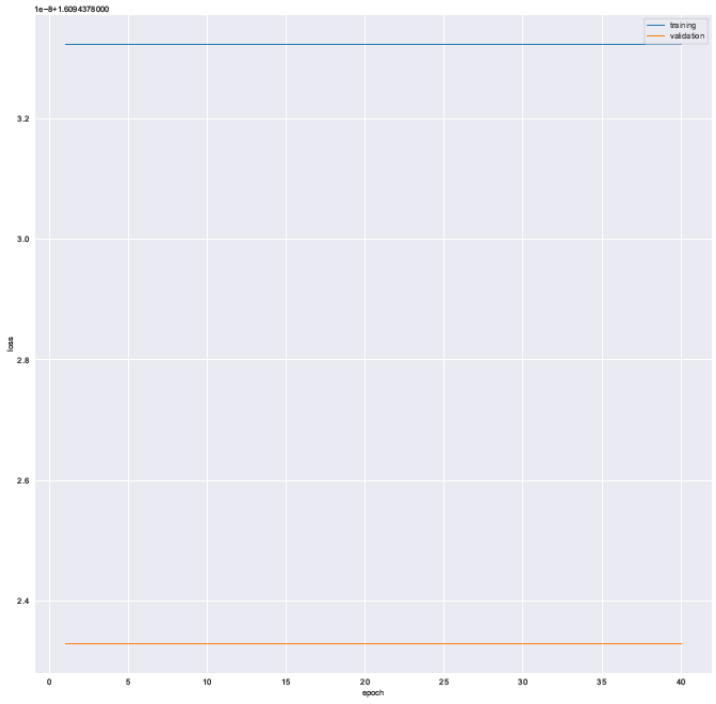

Figure 7: Training and Validation loss Plot for Variant 1

3.2.1 First Model Evaluation using Confusion Matrix:

Confusion matrix is used to show the accuracy of the data which will be evaluated in our first version of architecture. During evaluation of test data, the values of predicted and actual will be computed. Using the two classes the Confusion matrix will be calculated. Confusion matrix is as shown below:

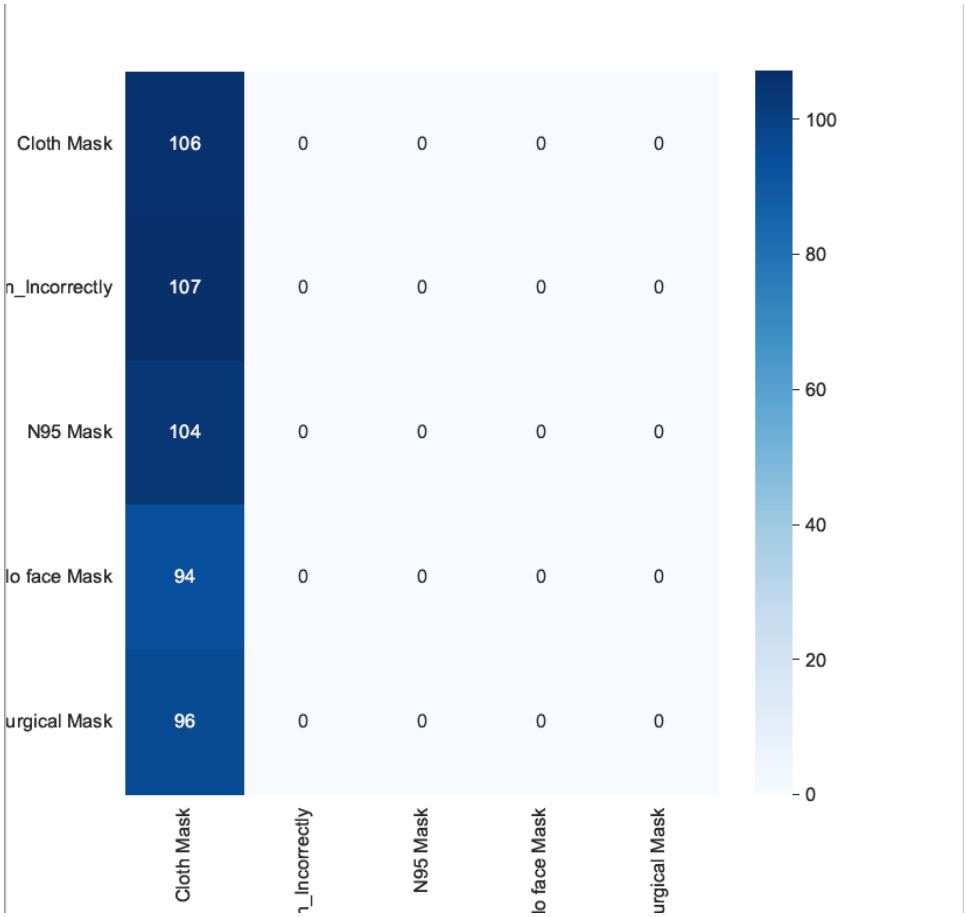

Figure 8: Confusion matrix for Variant 1

3.2.2 First model Classification report

Below is the detailed report of the precision, recall, and f1-score of the test data for 40 epochs:

	precision	recall	f1-score	support
Cloth Mask	0.209072978	1	0.345840131	106
Mask_Worn_Incorrectly	0	0	0	107
N95 Mask	0	0	0	104
No face Mask	0	0	0	94
Surgical Mask	0	0	0	96

accuracy	0.209072978	0.209072978	0.209072978	0.209072978
macro avg	0.041814596	0.2	0.069168026	507
weighted avg	0.04371151	0.209072978	0.072305826	507

3.3 Second Variant Architecture:

Below figure explains the difference between Accuracy and loss of various epochs.

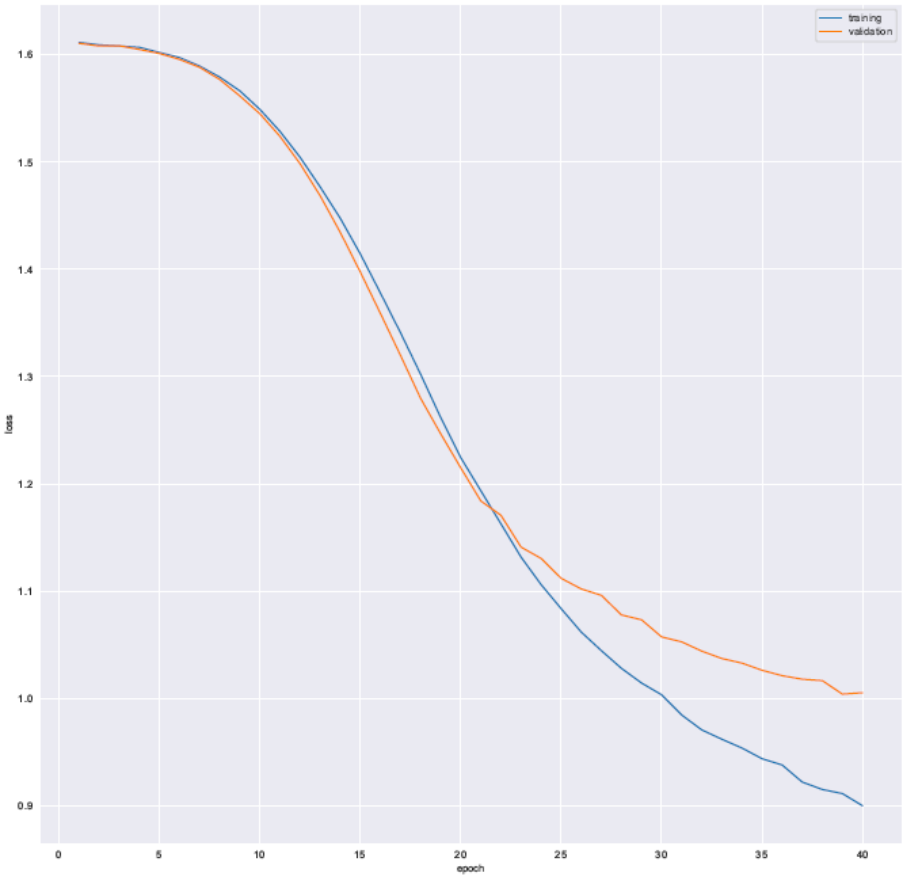

Figure 9: Training and Validation loss Plot for Variant 2 Architecture

3.3.1. Second Model Evaluation using Confusion Matrix:

Confusion matrix is used to show the accuracy of the data which will be evaluated in our Second architecture. During evaluation of test data, the values of predicted and actual will be computed. Using the two classes the Confusion matrix will be calculated. Confusion matrix is as shown below:

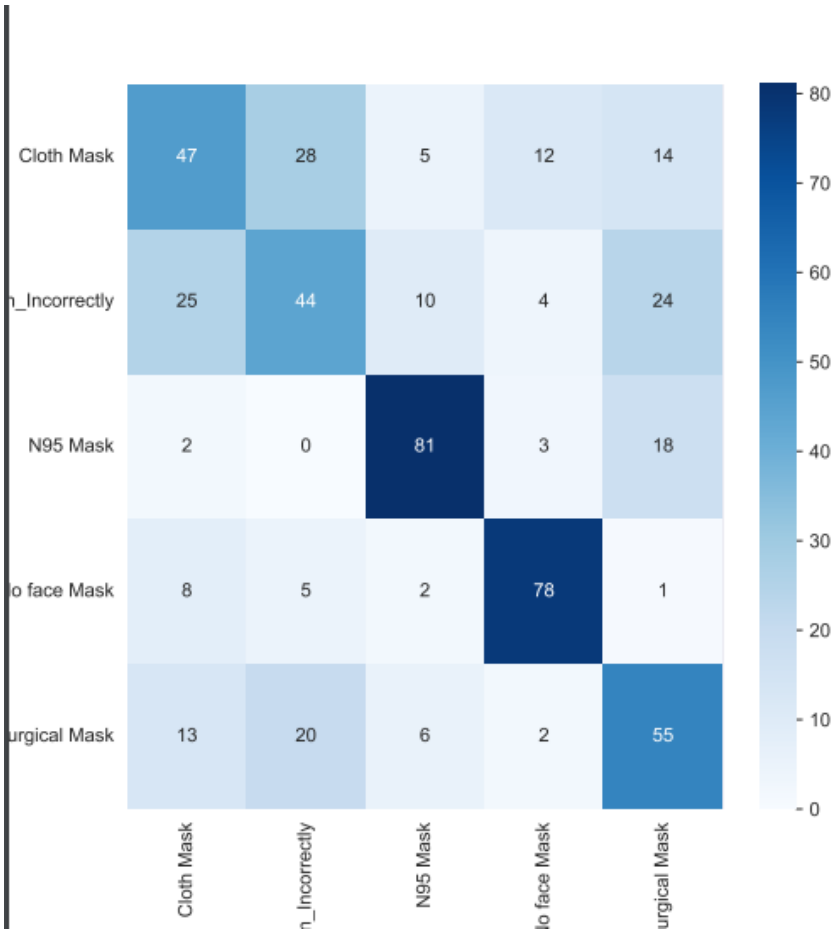

Figure 10: Confusion matrix for Variant 2

3.3.2 Second model Classification report

Below is the detailed report of the precision, recall, and f1-score of the test data for 40 epochs:

	precision	recall	f1-score	support
Cloth Mask	0.494736842	0.443396226	0.467661692	106
Mask_Worn_Incorrectly	0.453608247	0.411214953	0.431372549	107
N95 Mask	0.778846154	0.778846154	0.778846154	104
No face Mask	0.787878788	0.829787234	0.808290155	94
Surgical Mask	0.491071429	0.572916667	0.528846154	96
accuracy	0.601577909	0.601577909	0.601577909	0.601577909
macro avg	0.601228292	0.607232247	0.603003341	507
weighted avg	0.597991422	0.601577909	0.598574965	507

4.Output of Prediction model:

Below is the sample output that is generated by the face_mask_prediction.py using the best trained model(.py).

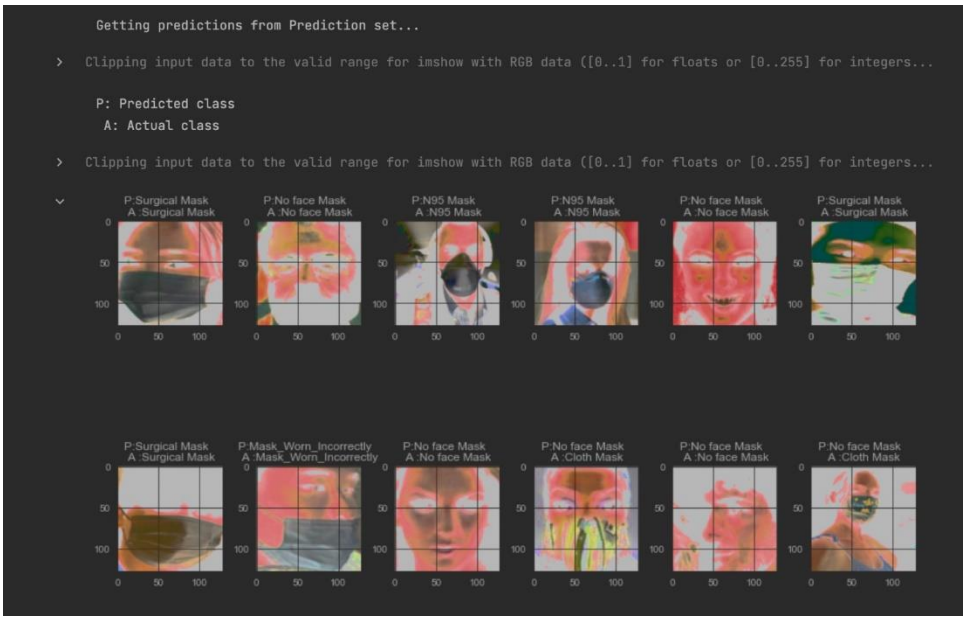

Figure 11: Trained Model

5. Improvements for Part II:

1. We will increase the accuracy for the upcoming second phase.
2. We will perform the evaluation for possible bias and try to eliminate it.
3. We will introduce two classifying models using two parameters i.e., age and gender.

6.References

1. https://www.kaggle.com/datasets/sumansid/facemask-dataset
2. https://www.kaggle.com/datasets/akibhosenkhan/facemask-detection-datasets
3. https://www.kaggle.com/datasets/coffee124/facemaskn95
4. https://www.kdnuggets.com/2022/05/image-classification-convolutional-neural-networks-cnns.html
5. https://www.ibm.com/cloud/learn/convolutional-neural-networks

6. https://pytorch.org/tutorials/beginner/basics/optimization_tutorial.html
7. https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy
8. https://vitalflux.com/wp-content/uploads/2022/04/Typical-CNN-architecture.png
9. https://vitalflux.com/different-types-of-cnn-architectures-explained-examples/
10. https://www.researchgate.net/figure/Proposed-adopted-Convolutional-Neural-Network-CNN-model_fig2_332407214
11. https://www.kaggle.com/datasets/harnalashok/distinguish-between-correct-and-incorrect-masks:
12. https://www.kaggle.com/datasets/vinaykudari/facemask:
13. https://www.kaggle.com/datasets/pratyushjanghel/face-mask-detection
14. https://www.kaggle.com/datasets/hadjadjib/facemask-recognition-dataset
15. https://www.kaggle.com/datasets/muhammadahsan026/facemask-dataset-covid1910k-images-2-folders
16. https://www.kaggle.com/datasets/nasser61010/facemask-datasets4
17. https://www.kaggle.com/datasets/prithwirajmitra/covid-face-mask-detection-dataset
18. https://www.kaggle.com/code/kholoudsayed/face-mask-detection-using-cnn/data
19. https://www.kaggle.com/datasets/perke986/face-mask-segmentation-dataset