COMP 6721 Project

AI Face Mask Detector - CNN

Phase-1

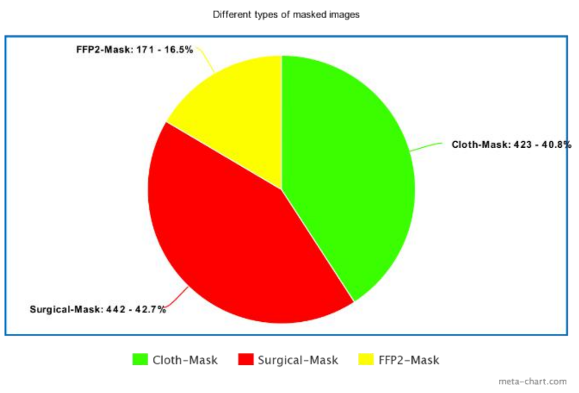
TEAM NAME: **NS_09**

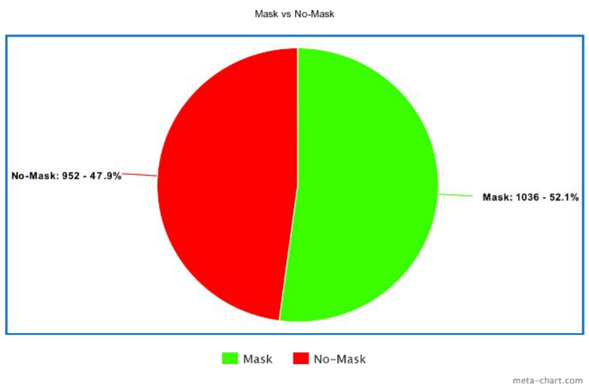
Name	**ID**	**Specification**
Smit Desai	40120178	Data collection & preprocessing. Model building and training
Aayush Khandelwal	40156633	Model building and training Evaluation
Mansajan Singh	40160310	Evaluation

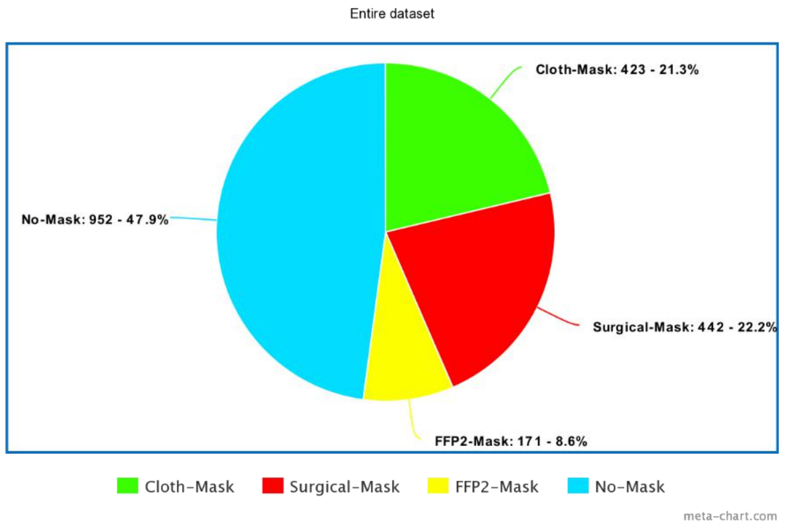
1. DATASET

1.1 COLLECTING DATA

As outlined in the project description, the model had to classify images into 4 different classes: Cloth-Mask, Surgical-Mask, FFP2-Mask, and No-Mask. The dataset has a total of 1988 images. Here are some statistics about the dataset:







Source:

1) Cloth-Mask, Surgical-Mask images were collected from Google images.
2) FFP2-Mask images were collected from Shutterstock.
3) No-Mask images were collected from 2 different Kaggle datasets
    a. https://www.kaggle.com/vinaykudari/facemask by Vinay Kudari
    b. https://www.kaggle.com/spandanpatnaik09/face-mask-detectormask-not-mask-incorrect-mask by Spandanpatnaik

(Please see the attached reference files to view every image source)

1.2 PREPROCESSING

The following preprocessing steps were taken:

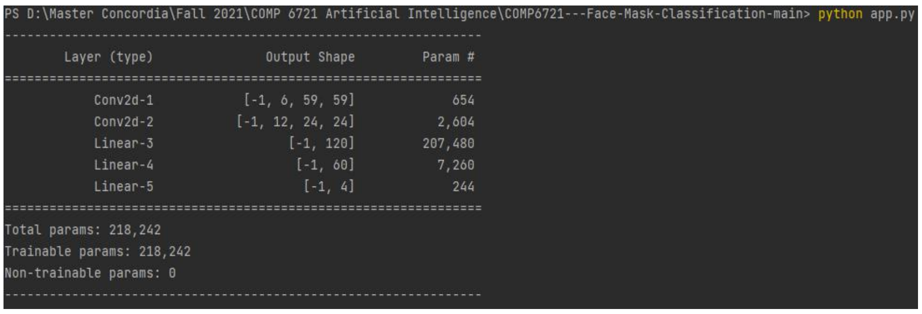
1) **Resize:** In order to feed the images to the CNN network, they were resized (64 px, 64 px)
2) **Horizontal-flips:** To increase randomness, images were flipped horizontally with a probability of 0.5
3) **Conversation to tensor:** Converts the images into an array of numbers, called torch tensor. Each pixel of the input RGB image is divided into three different pixels- red, blue, and green. This creates three different images. For each generated image, the pixel value is divided by 255 to range the pixel range from [0 255] to [0 1]
4) **Train-Test split:** The data set is split into 2 categories during runtime. The training dataset has 1690 images (85%) whereas the test dataset has 298 images (15%)
5) **Batch-loading:** Finally, images from both categories are loaded into a batch of size 32 and are randomly shuffled.

2. CNN ARCHITECTURE

Taking account of CNN architecture we have created a Convolution Neural Network which primarily consists of 3 non-linear convolution layers and 2 linear fully connected layers. The architecture initiates with producing a feature map when the input tensor of shape [32, 3, 64,64] is passed through this layer. The stack normalization is then applied to all the feature maps. This helps reduce a wider weight range by normalizing with the standard deviation.

Here, PyTorch describes the beta and gamma hyperparameters of batch normalization. For this architecture, we have chosen Relu as our activation function because it sets the negative value to 0. After that pass the output tensor through the dropout layer which will deactivate randomly selected activations according to probability. This gives the model considerable robustness and reduces the possibility of overfitting. At Last, performing MaxPooling with a kernel size of 2X2 to reduce network complexity. This model used a 3x3 size kernel to extract features. To further optimize the memory, we implemented the Relu feature directly on the output sensor without allocating additional memory. For reshaping the formula we have used is (width_of_input –kernel_size + 1)/stride so placing the value in the above formula gives us the output 32 after applying operation.

Looping the above-mentioned process for the next 2 convolution layers produces an output tensor of shape which then passes it to the Fully connected linear layer, followed by flatting the output tensor first, producing the linear output tensor. The output is shown in Figure 3 given below:



```
PS D:\Master Concordia\Fall 2021\COMP 6721 Artificial Intelligence\COMP6721---Face-Mask-Classification-main> python app.py
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 6, 59, 59]             654
            Conv2d-2           [-1, 12, 24, 24]           2,604
            Linear-3                  [-1, 120]         207,480
            Linear-4                   [-1, 60]           7,260
            Linear-5                    [-1, 4]             244
================================================================
Total params: 218,242
Trainable params: 218,242
Non-trainable params: 0
----------------------------------------------------------------
```

Here, all layers are classes in PyTorch's nn module, and model classes are inherited from nn. Module classes that handle the entire complexity of initializing model parameters (weighting), manipulating them, and storing them in memory.

3. CNN Model Training

Here, we have selected five model training epochs and used batch input processing to overcome the customization of large amounts of data in memory. (batch size for input during training is 32).

CNN model training has two phases: forward pass and backward pass. Repeated at each epoch, the input batch goes through a series of layers defined by the forward method of the CNN model class. After performing this forward pass, the loss value is calculated for the specified prediction (at this point model.train() allows gradient manipulation so you can update the model weights). The loss value or gradient is typically the following error calculation: Each output node and inner layer node. This is done using the CrossEntropyLoss function of the Nn module. This function finally uses the SoftMax activation function to narrow the output probability between 0 and 1 and calculate the loss value. Next, loss.backward() performs the backpropagation phase. In this phase, the difference in total weighting is calculated and optimized and optimized. Step function updates the model weights. Here, Adam is used as the optimizer and a learning rate of 0.01 is selected for backward execution. Here, the output prediction gets the probabilities of all four output classes. Among them, the class with the highest probability value was selected as the class predicted by the model.

We trained the model for 5 Epoch and the results of each epoch are as follows:

Epoch 1: Correct predictions: 7
Accuracy: 0.46094674556213017
Loss: 8.298946174855768

Epoch 2: Correct predictions: 801/1690
Accuracy: 0.47396449704142013
Loss: 0.038740938302327894

Epoch 3: Correct predictions: 801/1690
Accuracy: 0.47396449704142013
Loss: 0.038791874174535626

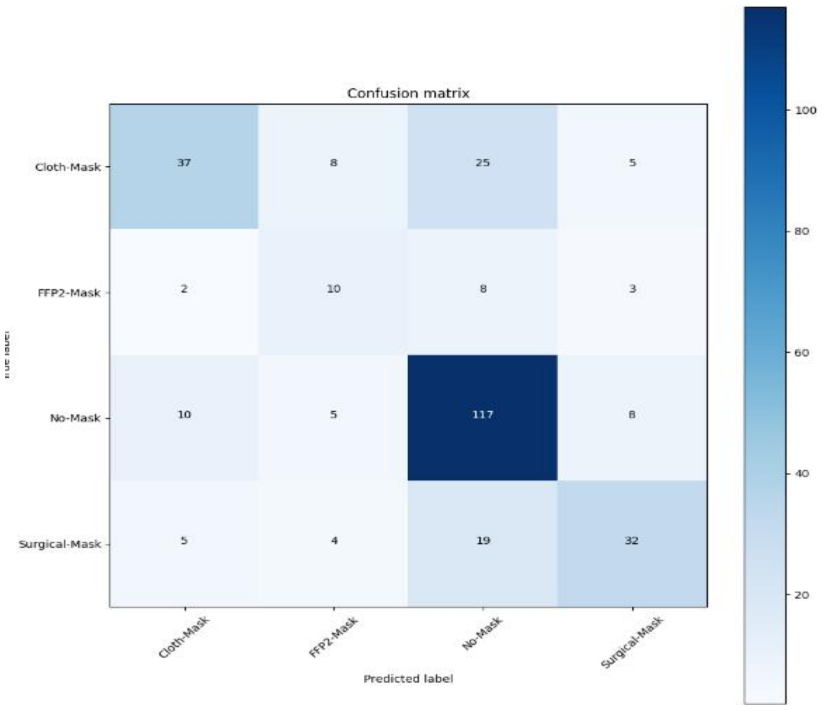
Epoch 4: Correct predictions: 801/1690
Accuracy: 0.47396449704142013
Loss: 0.03877918755514382

Epoch 5: Correct predictions: 801/1690
Accuracy: 0.47396449704142013
Loss: 0.03880976057616917

4. EVALUATION

In the evaluation phase, we applied our generated CNN model to the test dataset and observed the predictions made by the model for the images in the test dataset. The test dataset consists of 298 images divided into 4 classes namely “cloth-mask” class, “Surgical-mask” class, “FFP2-mask” class, and “No-mask” class. We generated a confusion matrix and a classification report for the predictions made by our model on the test dataset.

Observing the Confusion Matrix, we found that our model falsely classifies the images into the “no-mask” classes and the possible explanation is the use of an imbalance dataset where the no-mask class has almost the same number of images as the other 3 classes combined. The given confusion matrix is the normalized and beautified version of the original confusion matrix we generated for better, clear, and simple understanding and visualization. We used the comprehensive matplotlib library of Python for this purpose.



When model was applied for testing, it predicted 151 images as correct class out of 298. **Accuracy comes out to be 0.5067114093959731 and loss was 0.04058477662553723.**

Precision, Recall and f1-score for all the four classes is given below:

	precision	recall	f1-score	support
Cloth mask	0.68	0.49	0.56	75
FFP2 mask	0.37	0.43	0.39	23
No mask	0.69	0.83	0.75	140
Surgical mask	0.67	0.53	0.59	60

The values in above table are calculated with following formulas:

Precision = (Correctly predicted positive observations / Total predicted positive observations)
Recall = (Correctly predicted positive observations/ All observations in actual class)
f1-score = 2*(Precision*recall) / (Recall + precision)

No mask class gives best results in terms of precision, recall and f1-score but still needs improvement in phase 2. All other classes are low on precision, recall and f1-score and need alterations for better results as confusion matrix clearly indicates that there is a lot of confusion among classes during predictions.

REFERENCES

[1] https://medium.com/nybles/a-brief-guide-to-convolutional-neural-network-cnn-642f47e88ed4

[2]https://medium.com/@kohlishivam5522/understanding-a-classification-report-for-your-machine-learning-model-88815e2ce397

[3] https://machinelearningmastery.com/confusion-matrix-machine-learning/

[4] https://prvnk10.medium.com/cnn-architectures-ecefaa2359ff ---improve CNN architecture

[5] https://deeplizard.com/learn/video/0LhiS6yu2qQ

[6] https://medium.com/thecyphy/train-cnn-model-with-pytorch-21dafb918f48

[7] PyTorch_tutorials - https://github.com/gaurav67890/Pytorch_Tutorials/blob/master/cnn-scratch-training.ipynb

[8] https://medium.com/secure-and-private-ai-writing-challenge/loading-image-using-pytorch-c2e2dcce6ef2 --Preprocessing using torchvision

[9] https://pytorch.org/vision/stable/transforms.html