# COMP 6721

# Applied Artificial Intelligence

Project Report (Phase-1)

Submitted to: Prof. Dr. René Witte

**Team Id: NS-05**

Gaurang Dobariya (40184707) – Evaluation Specialist
Vickykumar Patel (40185238) – Training Specialist
Parth Navsariwala (40178800) – Data Specialist

# Table of Content

# Table of Images

## 1. Dataset:

Collection of data called Dataset. In AI Based Face Mask Detection our main aim is to classify whether person wearing a certain type of mask which fall in one of five category or not person wearing no mask at all.

For this project, we need to collect data for **5 different categories** namely Person "**without a mask**", Person with a "**cloth mask**", Person with a "**Surgical mask**" , Person with a "**N95 mask**" and Person with "**N95 mask with valve**".

**Cloth Mask** **N95 Mask** **N95 with Valve Mask**

**Surgical Mask** **Without Mask**

**1.1 General Examples of Five Categories of Mask**

The most difficult part of collecting enough data is ensuring that all training and testing photos for each class only contain data for that class. The majority of public available dataset contain two datasets, dataset where people wearing mask and where people not wearing mask. However, a closer examination reveals that the dataset containing person waring mask includes a variety of face masks, including N95, N95 with valve, Cloth Mask, Surgical Mask and so on are examples. As a result, spend a significant amount of time gathering the relevant information and We used both publicly available data and data that we created ourselves Another crucial point to remember is that the photographs should only show one face, hence we didn't offer any training or testing images with many faces. Only a single face of a person wearing a (particular type of) mask or not must be portrayed in the image. Finally, we discovered that the background has a significant impact on our model's performance. To put it another way, photos with particular objects or environments in the background had a negative impact on our model training. As a result, we had to ensure that background noise was eliminated, which we accomplished by cropping our photographs such that only the face (along with some minor background components) is visible.

To build **our data set** we collect all images from **mainly 3 sources:**

a. **Kaggle Dataset**
   There are many available datasets for mask detection on Kaggle we refer all the dataset and then we filter all dataset one by one and download images as per our categories.
b. **Google Images**
   To create our own data set we research on google using different key words which are as follow: **'Person wearing N95 face mask,' 'Person wearing cloth face mask', 'silk cloth face mask,' or 'cotton cloth face mask', 'Person without face mask' , 'Person wearing N95 face mask with valve'.** We've already begun planning the second phase of our project.

**Size of each category or label in the dataset**

| | |
|---|---|
| Cloth Mask | 401 |
| N95 Mask | 404 |
| N95 Mask with Valve | 401 |
| Surgical Mask | 647 |
| Without Mask | 999 |

So, total **we have 2852 Images** that divide in different categories as per above mentioned table.

| | | |
|---|---|---|
| Training Set | 75% | 2139 |
| Testing Set | 25% | 713 |
| Total | 100% | 2852 |

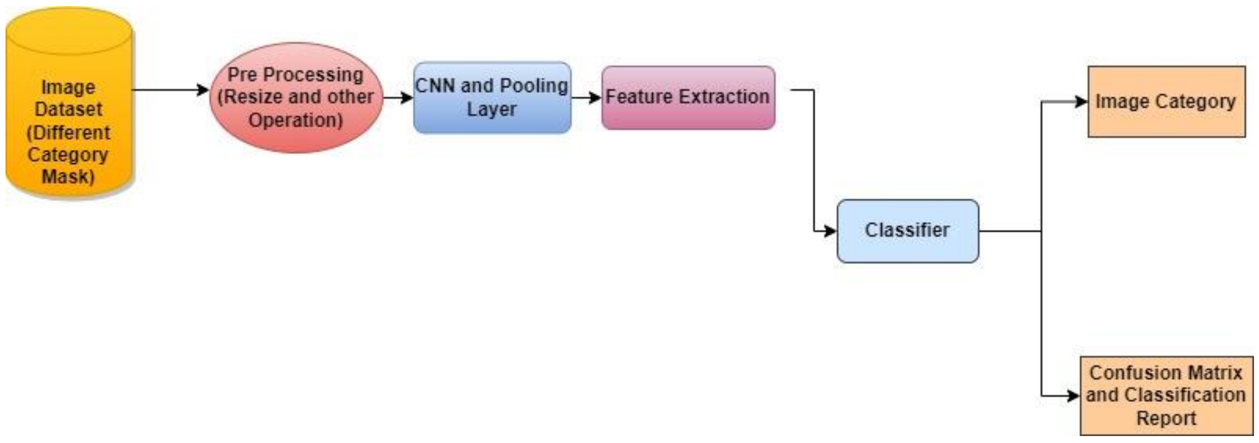


Image Dataset (Different Category Mask)
Pre Processing (Resize and other Operation)
CNN and Pooling Layer
Feature Extraction
Classifier
Image Category
Confusion Matrix and Classification Report

**1.2 Project Overview**

## 2. CNN Architecture:

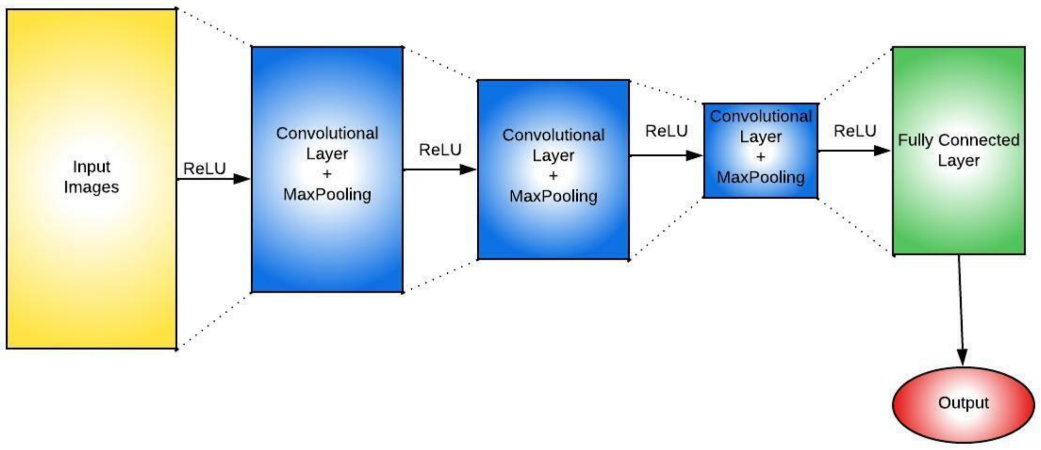
Convolutional Neural Network (CNN) are special kind of multi-layer neural networks, designed to extract key features from an image like visual patterns with very low pre-processing of data.

We used CNN to train our model for specific categories. The dimensions of all the photographs we had were not in the right size because we had gathered data from various sources. Some were too big, while others were too small. We had to apply some **pre-processing** to make all of these **photographs look the same.**

We transformed all of the photos in the dataset to the **same size of (224 * 224)** while loading them. We also modify the pixel intensity by normalizing the data using standard deviation and image means. We also shuffled the data before passing it on to the training phase to add some randomization.

There are mainly **3 layers** in our CNN model which are as follow:

a. **Convolutional layers:** Convolutional layers can be used to extract specific features from the image. Each convolutional layer learns specific features of image.
b. **Pooling layers:** These layers are used after convolutional layers to reduce the size of feature map generated by the convolutional layer so that the model can generalize on other data apart from training data and also to reduce the complexity of computation.
c. **Fully connected layers:** These layers use the features extracted by the convolutional layers and use the extracted features for decision making.

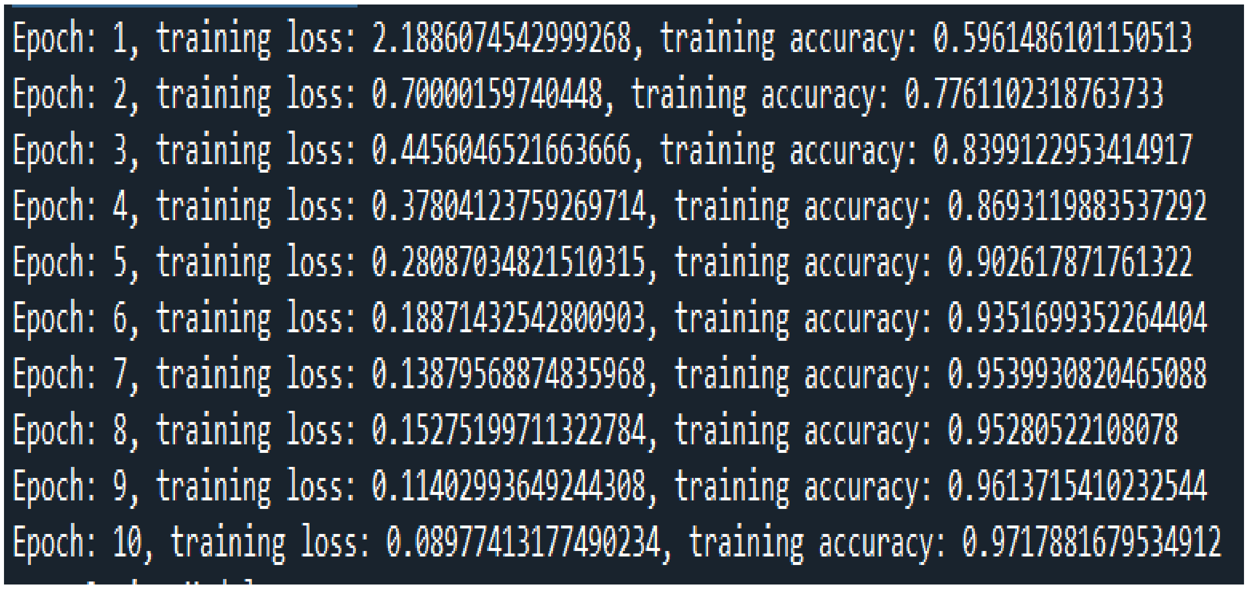

Input Images
ReLU
Convolutional Layer + MaxPooling
ReLU
Convolutional Layer + MaxPooling
ReLU
Convolutional Layer + MaxPooling
ReLU
Fully Connected Layer
Output

**2.1 CNN Model**

To extract features from the dataset, we used a very simple CNN architecture with only 3 convolutional layers. First, an **image of size (224 * 224)** is passed to the first convolution 2D layer with **kernel size (3 * 3)** and stride and **padding equal to 1**. The output channel for this layer is 64. The data is then normalized and passed as input to the **ReLU activation function**. From here, the output of the ReLU activation function serves as an input to MaxPooling with **stride 2 and kernel size (2 * 2).** This output is passed to the next convolutional 2D layer.

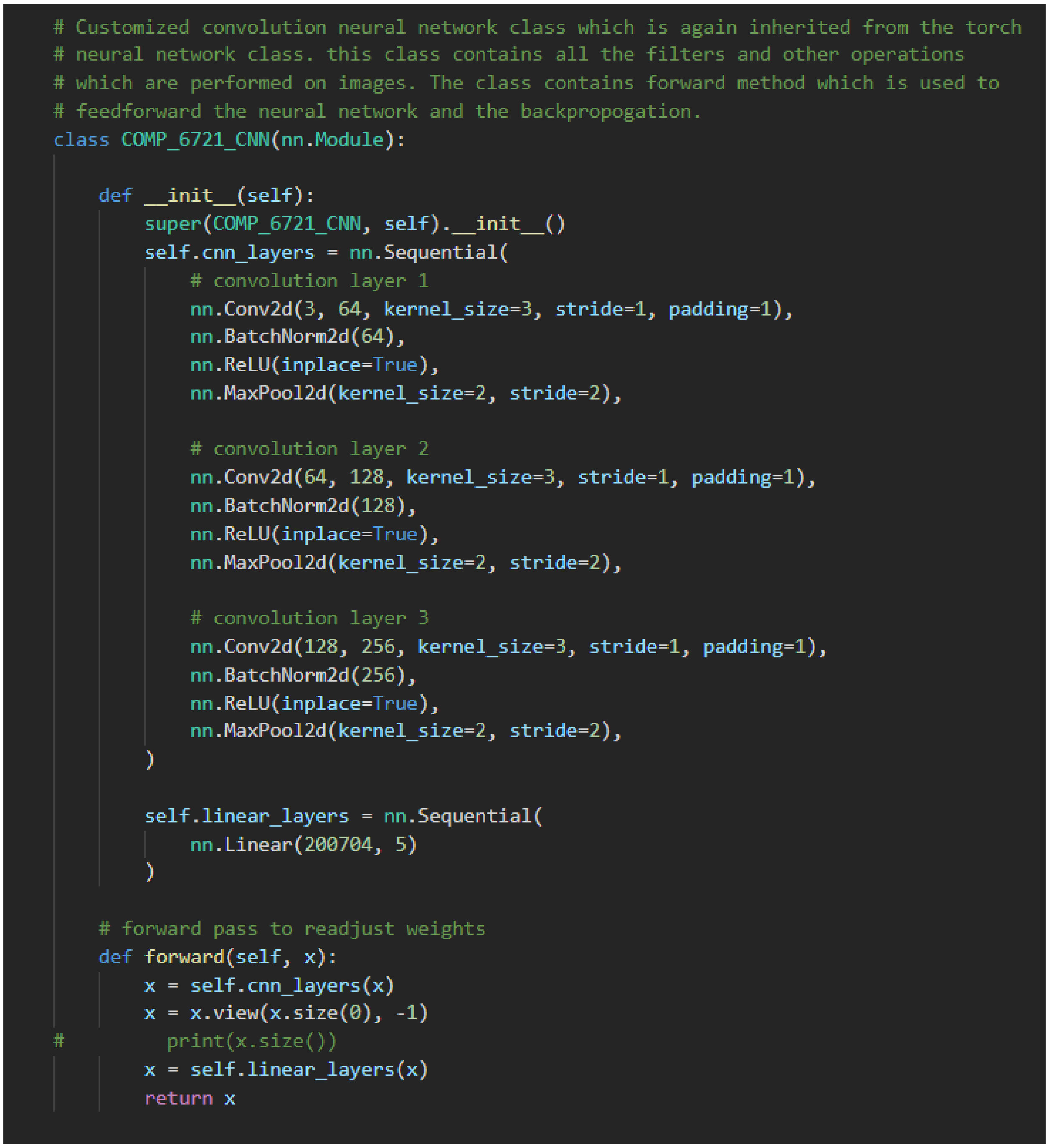
Here, the input channel is 64, and all other parameters are the same as for the first convolution 2D layer. The next step is the same as before. That is, normalization, the **ReLU** activation function, and MaxPooling. The output is then input to a third convolutional 2D layer with 128 input channels and 256 output channels. Again, all steps such as normalization, ReLU activation function, MaxPooling, etc. are performed. **Adam, a gradient-based stochastic optimization, is used in this project.**

Then, we used a fully connected dense layer that layers use the features extracted by the convolutional layers and use the extracted features for decision making.



```
Epoch: 1, training loss: 2.1886074542999268, training accuracy: 0.5961486101150513
Epoch: 2, training loss: 0.70000159740448, training accuracy: 0.7761102318763733
Epoch: 3, training loss: 0.4456046521663666, training accuracy: 0.8399122953414917
Epoch: 4, training loss: 0.37804123759269714, training accuracy: 0.8693119883537292
Epoch: 5, training loss: 0.28087034821510315, training accuracy: 0.902617871761322
Epoch: 6, training loss: 0.18871432542800903, training accuracy: 0.9351699352264404
Epoch: 7, training loss: 0.13879568874835968, training accuracy: 0.9539930820465088
Epoch: 8, training loss: 0.15275199711322784, training accuracy: 0.95280522108078
Epoch: 9, training loss: 0.11402993649244308, training accuracy: 0.9613715410232544
Epoch: 10, training loss: 0.08977413177490234, training accuracy: 0.9717881679534912
```

**2.2 Epoch Output**



```
# Customized convolution neural network class which is again inherited from the torch
# neural network class. this class contains all the filters and other operations
# which are performed on images. The class contains forward method which is used to
# feedforward the neural network and the backpropogation.
class COMP_6721_CNN(nn.Module):

    def __init__(self):
        super(COMP_6721_CNN, self).__init__()
        self.cnn_layers = nn.Sequential(
            # convolution layer 1
            nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            # convolution layer 2
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            # convolution layer 3
            nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )

        self.linear_layers = nn.Sequential(
            nn.Linear(200704, 5)
        )

    # forward pass to readjust weights
    def forward(self, x):
        x = self.cnn_layers(x)
        x = x.view(x.size(0), -1)
#         print(x.size())
        x = self.linear_layers(x)
        return x
```

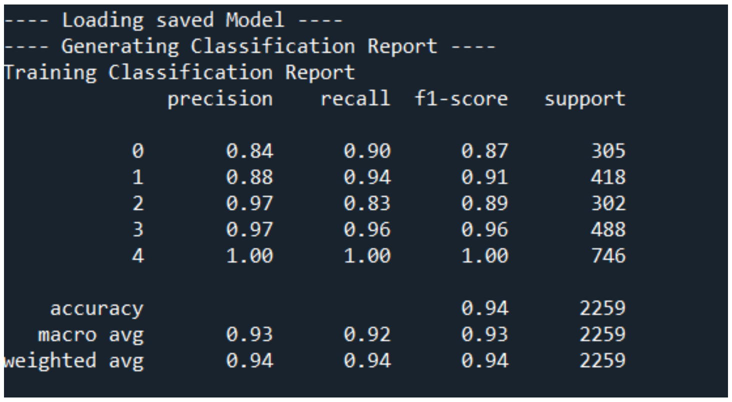
**2.3 CNN Code**

## 3. Evaluation:

After training the model, we need to check whether the model was trained well and for that we can evaluating various factors like accuracy, precision and f1score and support. **The model is trained for 10 epochs** and in each epoch, the **learning rate was 0.001**. We have calculated all this with factors with the help of **classification_report()** method. At last, we build the confusion matrix using **confusion_matrix()** method. The confusion matrix of the model can be useful to know the precision which in our case shows that many testing set images were misclassified due to imbalanced data. The confusion matrix of the model is useful for knowing the accuracy. In this case, **the training dataset is 94% accurate and the test dataset is 96% accurate.**

We also randomly fetched **100 different category images** and tried to predict those categories by a trained model. The file name (image type) is displayed on the left side, and the expected category is displayed on the right side.

**Training Classification Report:**



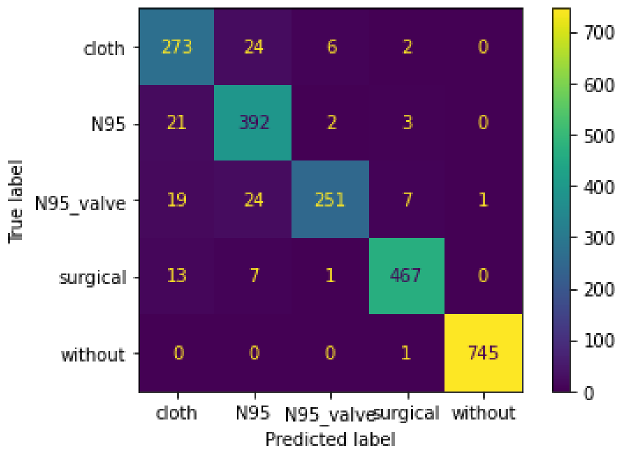
```
---- Loading saved Model ----
---- Generating Classification Report ----
Training Classification Report
              precision    recall  f1-score   support

           0       0.84      0.90      0.87       305
           1       0.88      0.94      0.91       418
           2       0.97      0.83      0.89       302
           3       0.97      0.96      0.96       488
           4       1.00      1.00      1.00       746

    accuracy                           0.94      2259
   macro avg       0.93      0.92      0.93      2259
weighted avg       0.94      0.94      0.94      2259
```

**3.1 Training Classification Report**

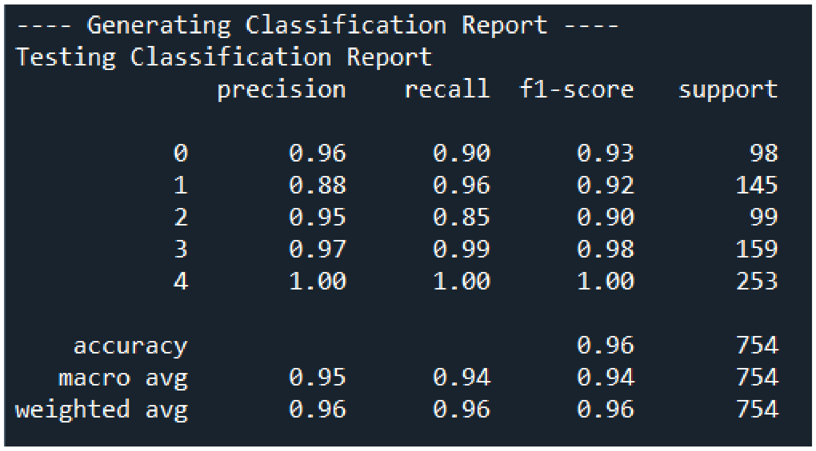
**Training Confusion Matrix:**



| True label \ Predicted label | cloth | N95 | N95_valve | surgical | without |
|---|---|---|---|---|---|
| cloth | 273 | 24 | 6 | 2 | 0 |
| N95 | 21 | 392 | 2 | 3 | 0 |
| N95_valve | 19 | 24 | 251 | 7 | 1 |
| surgical | 13 | 7 | 1 | 467 | 0 |
| without | 0 | 0 | 0 | 1 | 745 |

**3.2 Training Classification Report**

**Testing Classification Report:**



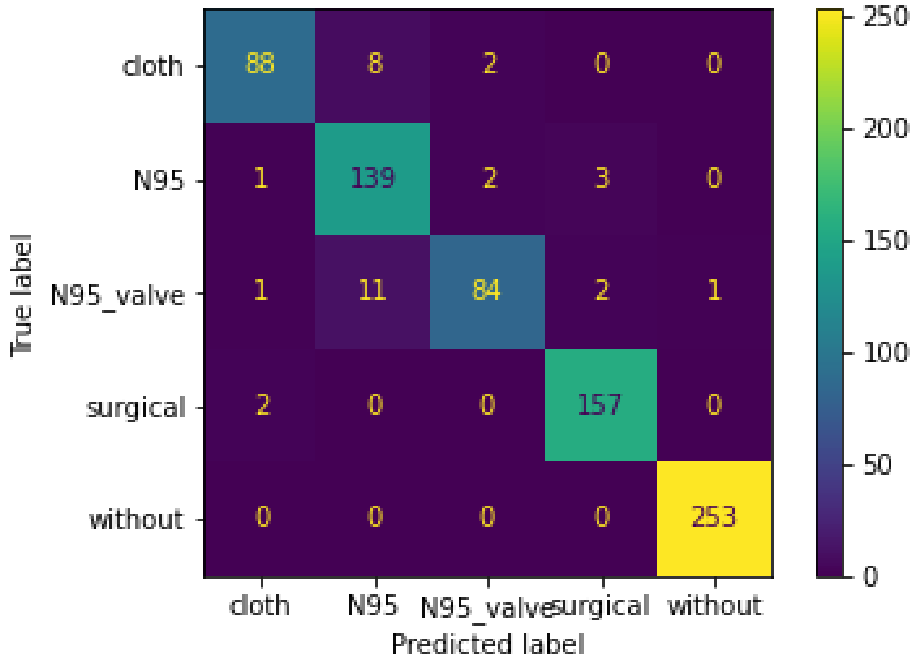
```
---- Generating Classification Report ----
Testing Classification Report
              precision    recall  f1-score   support

           0       0.96      0.90      0.93        98
           1       0.88      0.96      0.92       145
           2       0.95      0.85      0.90        99
           3       0.97      0.99      0.98       159
           4       1.00      1.00      1.00       253

    accuracy                           0.96       754
   macro avg       0.95      0.94      0.94       754
weighted avg       0.96      0.96      0.96       754
```

**3.3 Testing Classification Report**

**Testing Confusion Matrix:**



**3.4 Testing Classification Report**

## 4. References:

1. Kaggle Dataset
2. CNN Model
3. Different Type of CNN Architecture
4. Without Mask
5. Surgical Mask
6. Surgical Mask
7. N95 Valve Mask
8. Cloth Mask
9. Cloth Mask
10. N95 Mask
11. N95 Mask
12. Confusion Matrix
13. Confusion Matrix Display
14. Neural Network Model
15. Image Transformation