COMP 6721 Applied Artificial Intelligence (Winter 2022)

Project Assignment, Part I

AI Face Mask Detector

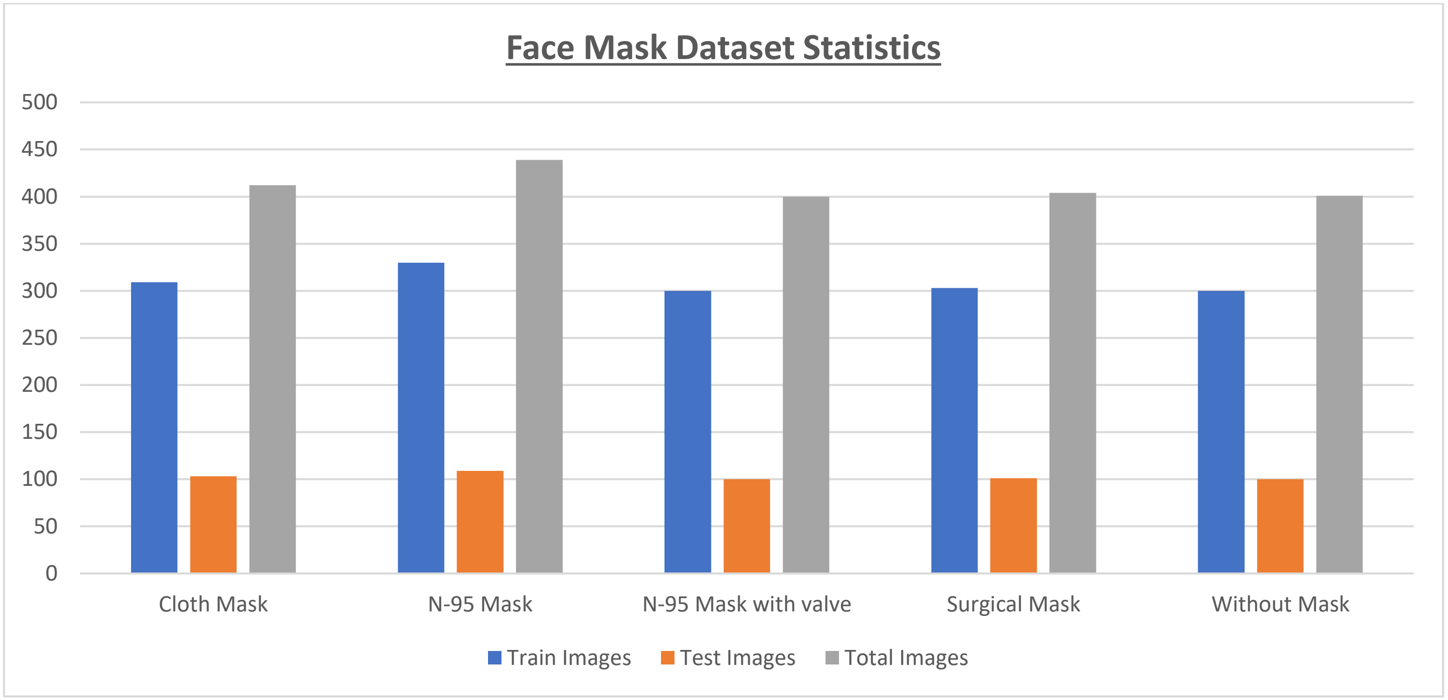
Group name: **NS_13**

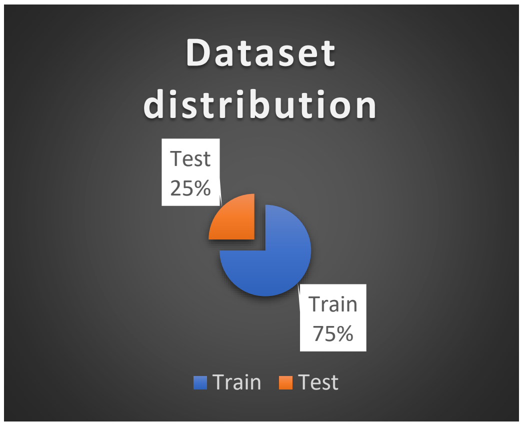
[1]

Role Assignments

Role	**Group Member**	**Student id**
Data Specialist	Shubham Bhanderi	40156448
Training Specialist	Parthiv Akbari	40155566
Evaluation Specialist	Jigar Borad	40155320

[1] Above images are taken from GitHub and Google Images.





Dataset References:

- Author, Larxel(2020) Face Mask Detetction (version 1) from Kaggle
- Author, Omkar Gurav(2020) Face Mask Detection Dataset (version 1) from Kaggle
- Deb, C. (2022). Face Mask Detection (Version v1.0.0) [Computer software]. https://github.com/chandrikadeb7/Face-Mask-Detection
- Google Images
- https://humansintheloop.org/resources/datasets/
- GitHub
- Author, Prithwiraj Mitra (2020) COVID Face Mask Detection Dataset (Version 1) from Kaggle
- Author, Ashish Jangra (2020) Face Mask Detection ~12K Images Dataset (Version 1) from Kaggle

For the preprocessing of the data images, Pytorch transform function is used to resize all input images. Particularly in this project we transformed all images to 50 x 50 size[2]. For dividing the dataset randomly, Pytorch random split function is used to split randomly data from each class. As per diagram, it shows that data is split to the ratio of 75% train and 25% test images. There are total 2098 images across all class.

[2] https://towardsdatascience.com/how-to-train-an-image-classifier-in-pytorch-and-use-it-to-perform-basic-inference-on-single-images-99465a1e9bf5

CNN Architecture

Layers:

(0): Conv2d(3, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU(inplace=True)
(2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
(3): Conv2d(12, 24, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU(inplace=True)
(5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
(6): Conv2d(24, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(7): ReLU(inplace=True)
(8): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
(9): Dropout(p=0.3, inplace=False)
(10): Flatten(start_dim=1, end_dim=-1)
(11): Linear(in_features=1200, out_features=196, bias=True)
(12): Linear(in_features=196, out_features=5, bias=True)

Diagram[3]:

Layer (Type)	Input shape	Output shape
Conv2d_1 (Conv2D)	(100,3,50,50)	(100,12,50,50)
ReLU – Activation	(100,12,50,50)	(100,12,50,50)
MaxPool2D	(100,12,50,50)	(100,12,24,24)
Conv2d_2 (Conv2D)	(100,12,24,24)	(100,24,24,24)
ReLU – Activation	(100,24,24,24)	(100,24,24,24)
MaxPool2D	(100,24,24,24)	(100,24,11,11)
Conv2d_3 (Conv2D)	(100,24,11,11)	(100,48,11,11)
ReLU – Activation	(100,48,11,11)	(100,48,11,11)
MaxPool2D	(100,48,11,11)	(100,48,5,5)
Dropout (0.3)	(100,48,5,5)	(100,48,5,5)
Flatten	(100,48,5,5)	(100,1200)
Linear_1 (Linear)	(100,1200)	(100,196)
Linear_2 (Linear)	(100,196)	(100,5)

Table 1: CNN network workflow

*Format of input and output tensor is (batch size, channel , image height , image width)

Above table shows the output the generates from each layer after performing operations.

[3] https://github.com/pytorch/pytorch/issues/2001

The above diagram of network layers shows how layers are implemented in CNN[4] network along with activation functions. Specifically, in this project, three convolutional layers and one linear layer are used. Diagram also refers to the input feature and output feature dimensions, when and how it changes while going to the next layer. ReLU activation function is used between each preceding layer to normalize the negative values. Maxpool2D is also used to down sample the image based on kernel size, padding and stride value. The equation for maxpool2D is as follows:[5]

Convolution Output dimension = [(I - F +2 *P) / S] +1 x D

Ex: in 1st conv2d layer we have dimension, 100x12x50x50 so image dimension is 50x50. **I=50.** Kernel size in maxpooling2d is **F = 3**. Padding is **P= 0**. Stride is **S = 2**. D means output features of the previous layer that is **D = 12**.

Apply all values to equation, [(50-3 + 2*0)/2] + 1 x 12 = **12 x 24 x 24**

More importantly, when input image(tensor) goes from Convolution layer to Linear layer, we need to flatten the input tensor value. Hence, we multiply input channel with image width and heigh which you can see in table 1.

Ex: in above table before linear layer tensor is **(100,48,5,5).** After applying flatten, it converts input tensor to **(100,48*5*5) = (100,1200).**

Training of CNN Model:[6]

Training the CNN model basically consist two parts:

1. A forward pass, in which the input image throughout the neural network layers.
2. A backward pass, in which gradients are backpropagated and weights and bias are updated.

In model, we are processing **20 epochs** to train the model**. 75%** of dataset is used to train the model and rest of **25%** data is used to test the model. Adam optimizer is used to update the gradients value while training the model. All the tensor values are uploaded to single device that is CPU. Learning rate is set to 0.001. CrossEntropyLoss function is used to calculate the loss during the training and used that value in backpropagation of the network. Things to consider while training model, if the gap between training and testing accuracy widens with subsequent epochs that means model is overfitting. To address this issue, we will add dropout layer in out network. So, we force layers to learn from other parameters too. One more thing, a large batch size reduces the time to train the model, but batch size should not be extremely large.

[4] https://www.analyticsvidhya.com/blog/2019/10/building-image-classification-models-cnn-pytorch/
https://pyimagesearch.com/2021/07/19/pytorch-training-your-first-convolutional-neural-network-cnn/
https://medium.com/thecyphy/train-cnn-model-with-pytorch-21dafb918f48
https://medium.com/thecyphy/train-cnn-model-with-pytorch-21dafb918f48
[5] https://kvirajdatt.medium.com/calculating-output-dimensions-in-a-cnn-for-convolution-and-pooling-layers-with-keras-682960c73870
[6] https://towardsdatascience.com/training-a-convolutional-neural-network-from-scratch-2235c2a25754

Evolution:

For the evolution part of the project, classification report is calculated for test data which includes precision, recall and f1 score for each class separately. Report also includes the how many numbers of images are in each class for testing the model. Accuracy of the model is also shown in the classification report.

	precision	**recall**	**f1-score**	**No. of images in testing**
Cloth mask	0.67	0.55	0.58	91
N-95 mask with valve	0.43	0.41	0.55	112
N95_Mask	0.52	0.63	0.55	112
No Face Mask	0.82	0.79	0.80	91
Surgical Mask	0.71	0.70	0.70	132
		accuracy	**0.70**	**522**
macro avg	0.65	0.66	0.64	522
weighted avg	0.71	0.70	0.70	522

Table 2: Model evaluation report[7]

The above table shows the report of the model of trained model. we gave 500 testing images across all classes. We also derived confusion matrix with the help of scikit learn library. By considering all evaluation report and confusion matrix, we can see the for the second class that is N-95 mask with valve, we are getting less accuracy something near about 55%-60%, so in the second phase, we try to achieve to more for that class by modifying dataset images and network parameters.

69	19	4	5	7
15	54	19	1	14
5	16	68	4	13
12	4	3	88	2
5	7	10	2	78

Confusion matrix[8]

By looking at above matrix, we can verify that first 2 classes have moderate accuracy, so we improve that in next phase. At end, in next phase, our focus will be improving model performance and modifying dataset images by looking at different features.

[7] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
[8] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html