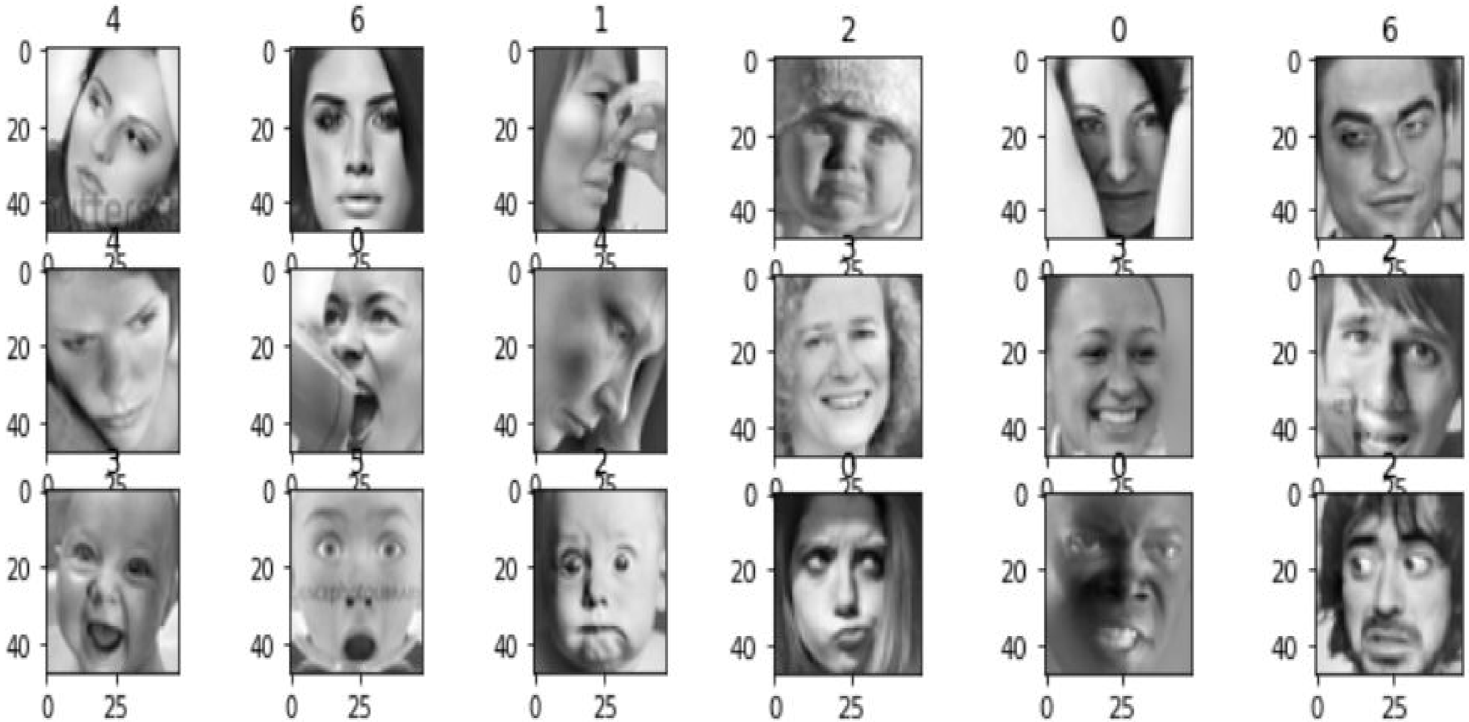
COL774- MACHINE LEARNING
ASSIGNMENT 4

DIWAKAR PRAJAPATI - 2018CS10330
UTSAV DEEP - 2018CS10396

DATA SET

The dataset contains 7 types of facial expressions on grayscale. A few of the samples with the class labels are given below.



NON COMPETITIVE PART

We have used Pytorch Library for the neural network.

A. VANILLA NEURAL NETWORK

No of Epochs	Learning Rate	Weight decay	Activation function	Training Cost	Training Time	Train Accuracy	Test Accuracy
200	0.005	0.0001	ReLU	0.033493	4min 10secs	99.77%	39.45%
200	0.005	0.0001	Sigmoid	0.728391	4min 27secs	82.71%	38.26%
200	0.005	0.0001	Tanh	0.052944	4 min 28secs	99.73%	36.72%

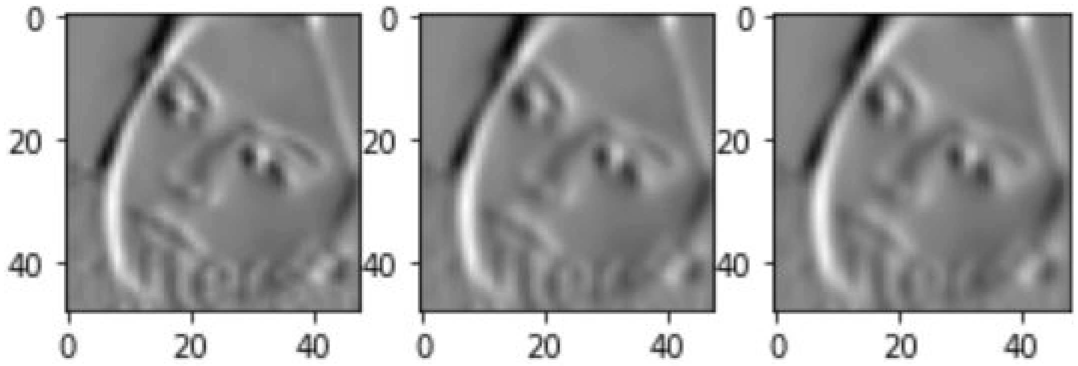
> The specifications of the layers were as given in the problem statement.
> We tried 3 different activation functions - ReLU, Sigmoid, Tanh. Cross entropy loss is used to train the data.
> From the results we can see that among various activation functions, ReLU activation gives the best accuracy. Next, best is by Sigmoid and last by Tanh.
> Training accuracy of both ReLU and Tanh were very high ~ 100% but the training accuracy for sigmoid was very low.
> Training time was almost the same for all parts.
> ReLU function has been used for final submission. In all the subsequent parts also ReLU will only be used as an activation function unless specified explicitly.

B. FEATURE ENGINEERING

The filters were implemented using the skimage library for image processing.

I. Applying Gabor filters :

Applying different Gabor filters to an image resulted in different outputs as shown below:



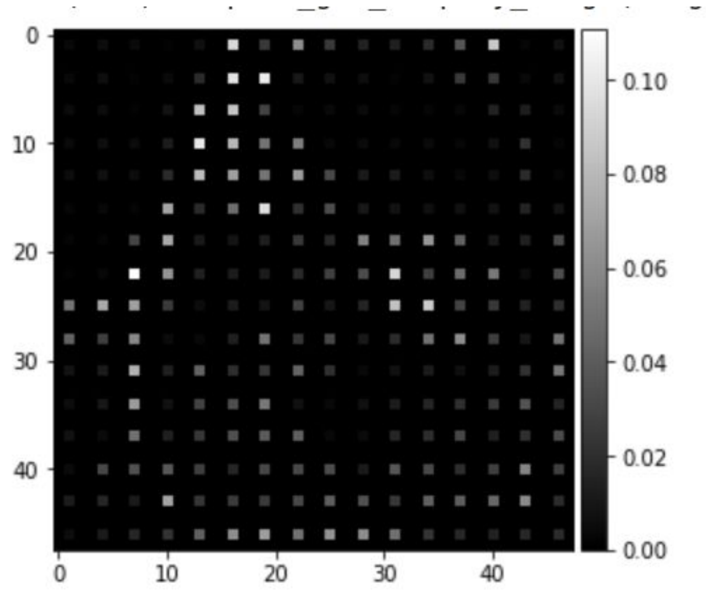
Parameters for first filter: `frequency=0.9, theta = 0, sigma_x=0.5, sigma_y=0.5`
Parameters for second filter: `frequency=0.9, theta = 0, sigma_x=0.8, sigma_y=0.8`
Parameters for third filter: `frequency=0.9, theta = 0, sigma_x=0.9, sigma_y=0.9`

> We passed each image through each of the above filters and then trained the model.
> We did this to reduce the overfitting in the model. Thus each input image would transform into three images from the Gabor filters.

Accuracy values after passing through Gabor filter:
Number of epochs: 200
Learning rate: 0.005
Training accuracy: 99.82 %
Test accuracy: 40.12 %

ii. Histogram of Oriented Gradients Filter(HOG filter):

Applying hog filter to an image resulted in the following



The parameters used were:

```
orientations=16, pixels_per_cell=(3, 3),cells_per_block=(2, 2), visualize=True, multichannel=False
```

> The hog filter was taking a lot of time to train.

Accuracy values after passing through HOG filter:
Number of epochs: 200
Learning rate: 0.005
Training accuracy: 99.80 %
Test accuracy: 39.94 %

> The time to train has increased as time is taken to apply filter on each image. Also hog filter was taking a lot more time compared to gabor filter.
> We observe that using the image filters has helped improve the accuracy, however that increase is only little and not very huge.
> The best performing feature engineering was by Gabor filter and that has been used in the final submission.

C. CONVOLUTIONAL NEURAL NETWORK

Optimizer	No of Epochs	Learning Rate	Weight Decay	Training Cost	Train Accuracy	Test Accuracy
SGD	200	0.001	0.0001	0.102990	97.50%	41.59%
Adam	200	0.001	0.0001	0.103110	97.48%	40.46%

> We used SGD optimiser on the following network structure:

```
digit_model_cnn = torch.nn.Sequential(
               torch.nn.Conv2d(kernel_size=(3,3), stride=3, padding=0, in_channels=1, out_channels=64),
               torch.nn.BatchNorm2d(64),
               torch.nn.ReLU(),
               torch.nn.MaxPool2d(kernel_size=2, stride=2, padding=0),
               torch.nn.Conv2d(kernel_size=(2,2), stride=2, padding=0, in_channels=64, out_channels=128),
               torch.nn.BatchNorm2d(128),
               torch.nn.ReLU(),
               torch.nn.MaxPool2d(kernel_size=2, stride=2, padding=0),
               torch.nn.Flatten(start_dim=1),
               torch.nn.Linear(128,32),
               torch.nn.ReLU(),
               torch.nn.BatchNorm1d(32),
               torch.nn.Linear(32,10)
       )
digit_model_cnn
```

> Inference time taken by Neural Networks is lesser than that of CNN.

> However, f1-score is almost the same and there is not a lot of difference.

> Best accuracy after a lot of submissions was seen when we first applied the Gabor filter on the image and then apply the CNN network defined in question 3 which resulted in about 42 % accuracy.

COMPETITIVE PART

> We used CNN for this part without a gabor filter or hog filter.
> We have augmented the data before training.
> The architecture we used for this part was as follows:

```
model = torch.nn.Sequential(
               torch.nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1),
               torch.nn.ReLU(inplace=True),
               torch.nn.BatchNorm2d(32),
               torch.nn.MaxPool2d(kernel_size=2, stride=2),
               torch.nn.Dropout(p=0.25),
               torch.nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
               torch.nn.ReLU(inplace=True),
               torch.nn.BatchNorm2d(64),
               torch.nn.MaxPool2d(kernel_size=2, stride=2),
               torch.nn.Dropout(p=0.25),
               torch.nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
               torch.nn.ReLU(inplace=True),
               torch.nn.BatchNorm2d(128),
               torch.nn.MaxPool2d(kernel_size=2, stride=2),
               torch.nn.Dropout(p=0.25),
               torch.nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
               torch.nn.ReLU(inplace=True),
               torch.nn.BatchNorm2d(128),
```

```
            torch.nn.MaxPool2d(kernel_size=2, stride=2),
            torch.nn.Dropout(p=0.25),
            torch.nn.Flatten(start_dim=1),
            torch.nn.Linear(1152, 512),
            torch.nn.ReLU(inplace=True),
            torch.nn.Dropout(),
            torch.nn.Linear(512, 256),
            torch.nn.ReLU(inplace=True),
            torch.nn.Dropout(),
            torch.nn.Linear(256,7),
      )
```

> Before passing through the filter the image was augmented with the following transforms:

1) 45-degree anti-clockwise rotation.
2) Flip up-down.
3) Flip left-right.
4) Random noise.

The final accuracy we obtained was:
54% on the test data
85% on training data.

THANK YOU!