# ECE763 Computer Vision

## Project Report

Submitted by:

Apoorva Kamalakar Gosavi, Bhagya Thulasidharan Nair, Hritwik Shukla
(agosavi@ncsu.edu, bthulas@ncsu.edu, hshukla@ncsu.edu)
(200344977, 200421890, 200425661)

Under the supervision of
Dr. Tianfu Wu

**ELECTRICAL AND COMPUTER ENGINEERING DEPARTMENT**
**NORTH CAROLINA STATE UNIVERSITY**
Raleigh, NC 27695
May 1, 2022

# Contents

# List of Figures

# Chapter 1

## 1.1 Abstract

Improving the accuracy of a model is an important and tedious task especially while using a large amount of dataset to train the model. Thus, adding more data, tuning different parameters and hyperparameters used in the algorithm of a classification task can improve the performance of the classifier and output a better result. In this work, some of the proven methods to improve accuracy are implemented and the different variations in the output are observed while inputting a dataset to the LeNet5 architecture.

## 1.2 Introduction

Convolutional Neural Network(CNN) is used for image classification tasks as they are effective in reducing the number of parameters without disturbing the model quality. The CNN used in this project is LeNet5, which is one of the pretrained models[1]. This architecture contains two sets of CNN and maximum pooling layers then a flattening convolutional layer followed by two fully connected layers with a softmax classifier. The image dataset used for training and testing consists of face and non-face images. The performance of the classification of this dataset is influenced by different parameters. Therefore, adjusting these parameters while training the model will bring more efficient output by the architecture. Thus, in this project, parameter values and techniques used in the model are altered, analyzed and improved by observing the output at each stage and is shown in the result. The following sections give a brief introduction of the data preparation, hyper parameter, model selection and the optimization and regularization used in this project.

## 1.3 Data Preparation

The dataset used for this project is FDDB dataset[2]. Each image in the dataset is preprocessed and saved as a face dataset. The non-face dataset is generated from the same FDDB dataset by creating non-face images from face images after some processing.. This face and non-face dataset is used for the testing and training purpose of this model. Data augmentation and data normalization is applied on the dataset and the result is observed before and after the application of these techniques.The code used in this project for splitting testing and training is shown in Fig 1.1.

```
# Form the training, validation, and testing dataset using DataLoader class
x_train, x_val, y_train, y_val =train_test_split(training_imgs,
                                                 training_labels, train_size=0.8)
train_loader = DataLoader(dataset = ([[x_train[i], y_train[i]]
                                      for i in range(len(y_train))]),
                                           batch_size = batch_size,
                                           shuffle = True)

validation_loader = DataLoader(dataset = ([[x_val[i], y_val[i]]
                                           for i in range(len(y_val))]),
                                           batch_size = batch_size,
                                           shuffle = True)

test_loader = DataLoader(dataset = ([[testing_imgs[i], testing_labels[i]]
                                     for i in range(len(testing_labels))]),
                                           batch_size = batch_size,
                                           shuffle = True)
```

Figure 1.1: Data Preparation

### 1.3.1 Training and Testing

The number of data in the training dataset is 300 and the testing dataset is 100 which contains both face and non face data. The images in the dataset are resized to 32x32 so as to input to LeNet 5 architecture.

### 1.3.2 Data Augmentation

This technique is employed for increasing the number of data in the dataset used for training and testing. So, this technique uses the existing dataset and makes simple alterations on data such as padding, re-scaling, cropping, flipping, blurring and creates new data. The implementation of data augmentation is shown in fig 1.2.

```python
# Function to perform Data Augmentation
def dataAugmentation(data):
  num_imgs = data.shape[0]
  temp_tensor = torch.empty((num_imgs*3,3,32,32), dtype=torch.float64)
  #print(temp_tensor.shape)
  for i in range(num_imgs):
    temp_tensor[i,:,:,:] = torch.flip(data[i,:,:,:], (1,2))
    temp_tensor[i+num_imgs,:,:,:] = T.GaussianBlur(kernel_size=(7,7),
                                                   sigma=(0.1, 0.2))(data[i,:,:,:])
    temp_tensor[i+num_imgs*2,:,:] = T.Grayscale(3)(data[i,:,:,:])
  return torch.cat((data,temp_tensor))
```

Figure 1.2: Data Augmentation

### 1.3.3 Data Normalization

This technique is used for standardizing the dataset by changing the values to a common scale without distorting differences in the range of values. So, the final value of each data point will be between the maximum and minimum values of that dataset. Data normalization implementation is shown in fig 1.3.

```python
#==================================================================================================#
                              Plotting Training Data : with normalization
#==================================================================================================#
fig, (ax1, ax2, ax3) = plt.subplots(1, 3)
fig.suptitle('Plotting training Data (with normalization)',color = "green",
             fontsize=13)
ax1.scatter(training_imgs_normalized_flattened[:,0],
            training_imgs_normalized_flattened[:,1], color="red",
            facecolor='none', s=8)
ax1.set_xlim(-10, 10)
ax1.set_ylim(-10, 10)
ax1.set_title('original data')

# Subtracting mean from each image to give us a zero-centred data
training_imgs_zeroCentred = training_imgs_normalized_flattened - mean_data[0]

ax2.scatter(training_imgs_zeroCentred[:,0],
            training_imgs_zeroCentred[:,1], color='green',
            facecolor='none', s=8)
ax2.set_xlim(-10, 10)
ax2.set_ylim(-10, 10)
ax2.set_title('zero-centered data')

# Dividing by standard deviation to give a zero-centred normalized data
training_imgs_Normalized = training_imgs_zeroCentred/std_dev_data[0]

ax3.scatter(training_imgs_Normalized[:,0],
            training_imgs_Normalized[:,1], color='blue',
            facecolor='none', s=8)
ax3.set_xlim(-25, 25)
ax3.set_ylim(-25, 25)
ax3.set_title('normalized data')
plt.show()
```

Figure 1.3: Data Normalization

## 1.4 Model Selection

LeNet 5 (LeNet): This is a convolutional neural network(CNN) architecture proposed by Yann LeCun et al. [1] in 1989. CNN is a kind of feed-forward neural network used for large-scale image processing. This architecture consists of the following parts, a convolutional encoder with two sets of convolutional layers and max pooling layers. One flattening convolutional layer and two fully-connected layers followed by a softmax classifier.

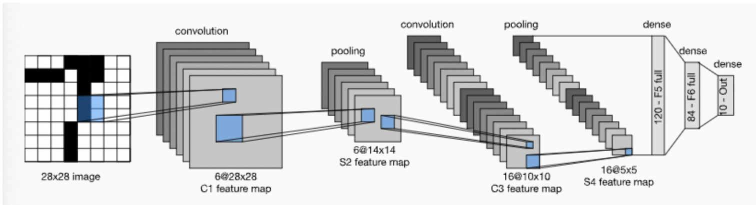



Figure 1.4: LeNet5 Architecture

First Layer: The first layer is a convolutional layer with filters or feature maps with size 6 having size 5x5 and stride of 1. The image dimension changes to 28x28x6. Also, this layer reduces the width and height of the image but increases the depth.

Second Layer: This is the pooling layer or sub-sampling layer with filter size 2x2 and a stride of two. Thus the feature map dimension is reduced to half which is 14x14x6.

Third Layer: This is the second convolutional layer where there are 16 feature maps with size 5x5. The stride is 1.

Fourth Layer: This is the second pooling layer, with filter size of 2x2 and stride of 2. Thus, this layer outputs the image with size 5x5x16, where 16 is the number of feature maps.

Fifth Layer: This is a convolutional layer which is fully connected with a feature map of size 1x1 and the number of input feature maps is 400. These 400 values are then passed to a fully connected layer made up of 120 neurons.

Sixth Layer: This is a fully connected layer which flattens the feature map further to 120 values which are then passed to a fully connected layer made up of 84 neurons.

Seventh Layer: This is a fully connected layer with 84 neurons. From this layer, the output layer is a fully connected softmax layer which outputs the desired result.

## 1.5 Optimization

The process of training the model iteratively which results in the maximum and minimum function evaluation. The optimization technique used for this project is stochastic gradient descent(SGD), where we use a small set of data points for calculating the local minimum of the function. The data points selected from the dataset are random. Thus, it will bring a parameter update for each training input and output example.

## 1.6 Regularization

This method is used to reduce the error caused by overfitting by adding a function on the given training dataset. Also, it will add an additional penalty term in the error function while tuning the function. Thus, the additional term prevents fluctuation so that the coefficients will not take extreme values. The technique employed in this project is batch normalization: This technique is used for training the neural network that standardizes the input to a layer for each batch of input data. This stabilizes the learning process by reducing the number of epochs and speeding up the training process.

## 1.7 Hyperparameter

Hyper parameters are variables whose values influence the learning process and affect the model parameters that a learning algorithm learns. The hyper parameters used are batch size, learning rate, Number of epochs, activation function and cost function.

Learning rate: It is the tuning parameter in the optimization algorithm which determines the step size during each iteration.

Batch size: It is the number of training samples utilized in one iteration.

Number of epochs: It is the number of passes through the dataset. This should be greater than or equal to one or less than or equal to the number of datapoints in the training dataset.

Activation Function: This determines whether a neuron should be active or not.

Sigmoid Activation: A weighted sum of inputs is passed through an activation function and this serves as the input to the next layer.

Rectified Linear Unit(ReLU) Activation: This is a piecewise linear function in which the input is shown as the output if the input is positive else zero.

Loss function: It computes the distance between the current and expected output of the algorithm and the average of all the loss functions over the total training data is called cost function. In cross entropy loss, the performance of a classification model is analyzed whose output is a value between 0 and 1.

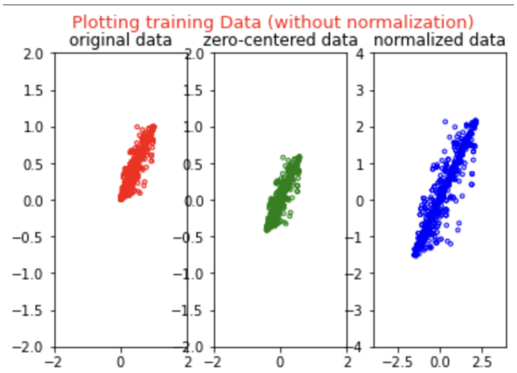
## 1.8 Results

Step1: Implemented data pre-processing. The result of the model architecture after classification is observed with and without data augmentation and normalization. Using data augmentation increases the number of training samples and thereby improves the accuracy and overall performance of the model.

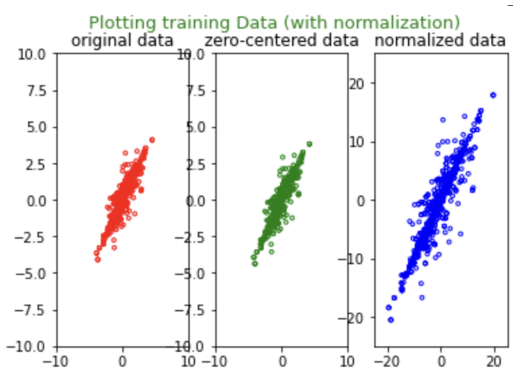
The plot of training data with and without the normalization is shown in fig 1.5, which brings the entire dataset in the range of maximum and and minimum data points in the dataset and it is standardized.

Step 2: The architecture-LeNet is preferred over other convolutional neural networks because it is less complex and easy to understand when we change the value of parameters used.

Figure 1.6 shows the summary of the model used in the project.




(a) Without Normalization




(b) With Normalization

Figure 1.5: Data Normalization

```
net = LeNet5(2).to(device)
print(net)
```

```
LeNet5(
  (layer1): Sequential(
    (0): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
    (1): BatchNorm2d(6, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc): Linear(in_features=400, out_features=120, bias=True)
  (relu): ReLU()
  (fc1): Linear(in_features=120, out_features=84, bias=True)
  (relu1): ReLU()
  (fc2): Linear(in_features=84, out_features=2, bias=True)
)
```

Figure 1.6: Summary of the model used

Step 3: In this step we are checking if the loss is reasonable. For two classes the loss should be approximately equal to 0.693.The loss calculated without regularization is shown in Fig 1.7(a).

Step 4: In this step we are adding a small regularization and checking if the loss increases. The weight_decay parameter applies L2 regularization while initialising optimizer. This adds regularization term to the loss function, with the effect of shrinking the parameter estimates, making the model simpler and less likely to overfit. The training loss has increased by a small value after adding a small amount of regularization as shown in Fig 1.7(b).

```
Printing loss for first epoch
Epoch [1/20], Step [960/960], Training Loss: 0.6954
Epoch [1/20], Step [240/960], Validation Loss: 0.6655
```

(a) Without Regularization

```
Printing loss for first epoch
Epoch [1/20], Step [960/960], Training Loss: 0.7217
Epoch [1/20], Step [240/960], Validation Loss: 0.6364
```

(b) With Regularization

Figure 1.7: Comparison of Loss Function with and without Regularization

Step 5: The learning rate is set to 0.03 to observe the overfitting. The result is shown in Fig 1.8, at loss value = 0, accuracy = 1.

Step 6: The value of learning rate is changed so as to find the good value that minimizes the loss. The learning rate is set to 0.0001 and observed that the loss is

```
Epoch [17/20], Step [20/960], Training Accuracy: 1.0000, Training Loss: 0.0000
Epoch [17/20], Step [20/960], Validation Accuracy: 1.0000, Validation Loss: 0.0007
17
Epoch [18/20], Step [20/960], Training Accuracy: 1.0000, Training Loss: 0.0000
Epoch [18/20], Step [20/960], Validation Accuracy: 1.0000, Validation Loss: 0.0000
18
Epoch [19/20], Step [20/960], Training Accuracy: 1.0000, Training Loss: 0.0012
Epoch [19/20], Step [20/960], Validation Accuracy: 1.0000, Validation Loss: 0.0000
19
Epoch [20/20], Step [20/960], Training Accuracy: 1.0000, Training Loss: 0.0017
Epoch [20/20], Step [20/960], Validation Accuracy: 1.0000, Validation Loss: 0.0005
```

Figure 1.8: Overfitting at Loss value =0 and Accuracy=1

barely changing as the learning rate is too small. Then, learning rate is changed to 0.0005 and the loss value starting decreasing and when learning rate is 3e-6, loss becomes NaN, which can be observed in Fig 1.9(a),(b) and Fig 1.10.

```
Epoch [1/20], Step [960/960], Training Loss: 0.6875
Epoch [1/20], Step [240/960], Validation Loss: 0.7123
Epoch [2/20], Step [960/960], Training Loss: 0.7072
Epoch [2/20], Step [240/960], Validation Loss: 0.6897
Epoch [3/20], Step [960/960], Training Loss: 0.7053
Epoch [3/20], Step [240/960], Validation Loss: 0.6739
Epoch [4/20], Step [960/960], Training Loss: 0.7002
Epoch [4/20], Step [240/960], Validation Loss: 0.6906
Epoch [5/20], Step [960/960], Training Loss: 0.6763
Epoch [5/20], Step [240/960], Validation Loss: 0.6889
Epoch [6/20], Step [960/960], Training Loss: 0.6994
Epoch [6/20], Step [240/960], Validation Loss: 0.6784
Epoch [7/20], Step [960/960], Training Loss: 0.6924
Epoch [7/20], Step [240/960], Validation Loss: 0.6995
Epoch [8/20], Step [960/960], Training Loss: 0.6758
Epoch [8/20], Step [240/960], Validation Loss: 0.6837
Epoch [9/20], Step [960/960], Training Loss: 0.6754
Epoch [9/20], Step [240/960], Validation Loss: 0.6790
Epoch [10/20], Step [960/960], Training Loss: 0.6838
Epoch [10/20], Step [240/960], Validation Loss: 0.6745
Epoch [11/20], Step [960/960], Training Loss: 0.6641
Epoch [11/20], Step [240/960], Validation Loss: 0.6866
Epoch [12/20], Step [960/960], Training Loss: 0.6946
Epoch [12/20], Step [240/960], Validation Loss: 0.6545
Epoch [13/20], Step [960/960], Training Loss: 0.7044
Epoch [13/20], Step [240/960], Validation Loss: 0.6319
Epoch [14/20], Step [960/960], Training Loss: 0.6744
Epoch [14/20], Step [240/960], Validation Loss: 0.5946
Epoch [15/20], Step [960/960], Training Loss: 0.6456
Epoch [15/20], Step [240/960], Validation Loss: 0.6479
Epoch [16/20], Step [960/960], Training Loss: 0.6635
Epoch [16/20], Step [240/960], Validation Loss: 0.6234
Epoch [17/20], Step [960/960], Training Loss: 0.6887
Epoch [17/20], Step [240/960], Validation Loss: 0.6429
Epoch [18/20], Step [960/960], Training Loss: 0.7650
Epoch [18/20], Step [240/960], Validation Loss: 0.7036
Epoch [19/20], Step [960/960], Training Loss: 0.7623
Epoch [19/20], Step [240/960], Validation Loss: 0.6469
Epoch [20/20], Step [960/960], Training Loss: 0.6461
Epoch [20/20], Step [240/960], Validation Loss: 0.6556
```

(a) Loss barely changing

```
Epoch [1/20], Step [960/960], Training Loss: 0.6424
Epoch [1/20], Step [240/960], Validation Loss: 0.6364
Epoch [2/20], Step [960/960], Training Loss: 0.6532
Epoch [2/20], Step [240/960], Validation Loss: 0.7316
Epoch [3/20], Step [960/960], Training Loss: 0.6857
Epoch [3/20], Step [240/960], Validation Loss: 0.6312
Epoch [4/20], Step [960/960], Training Loss: 0.5951
Epoch [4/20], Step [240/960], Validation Loss: 0.7327
Epoch [5/20], Step [960/960], Training Loss: 0.5606
Epoch [5/20], Step [240/960], Validation Loss: 0.7242
Epoch [6/20], Step [960/960], Training Loss: 0.4790
Epoch [6/20], Step [240/960], Validation Loss: 0.5112
Epoch [7/20], Step [960/960], Training Loss: 0.4441
Epoch [7/20], Step [240/960], Validation Loss: 0.5847
Epoch [8/20], Step [960/960], Training Loss: 0.8147
Epoch [8/20], Step [240/960], Validation Loss: 1.1722
Epoch [9/20], Step [960/960], Training Loss: 0.5418
Epoch [9/20], Step [240/960], Validation Loss: 1.7321
Epoch [10/20], Step [960/960], Training Loss: 0.2260
Epoch [10/20], Step [240/960], Validation Loss: 0.0735
Epoch [11/20], Step [960/960], Training Loss: 0.0177
Epoch [11/20], Step [240/960], Validation Loss: 0.0817
Epoch [12/20], Step [960/960], Training Loss: 0.0419
Epoch [12/20], Step [240/960], Validation Loss: 0.0245
Epoch [13/20], Step [960/960], Training Loss: 0.0245
Epoch [13/20], Step [240/960], Validation Loss: 0.0193
Epoch [14/20], Step [960/960], Training Loss: 0.0008
Epoch [14/20], Step [240/960], Validation Loss: 0.0162
Epoch [15/20], Step [960/960], Training Loss: 0.0204
Epoch [15/20], Step [240/960], Validation Loss: 0.1481
Epoch [16/20], Step [960/960], Training Loss: 0.0453
Epoch [16/20], Step [240/960], Validation Loss: 0.2014
Epoch [17/20], Step [960/960], Training Loss: 0.0004
Epoch [17/20], Step [240/960], Validation Loss: 0.1565
Epoch [18/20], Step [960/960], Training Loss: 0.0001
Epoch [18/20], Step [240/960], Validation Loss: 0.0029
Epoch [19/20], Step [960/960], Training Loss: 0.1373
Epoch [19/20], Step [240/960], Validation Loss: 0.0013
Epoch [20/20], Step [960/960], Training Loss: 0.0006
Epoch [20/20], Step [240/960], Validation Loss: 0.0004
```

(b) Loss going down

Figure 1.9: Comparison of change in loss

Step 7: As part of testing the model performance on FDDB dataset,the results were observed with and without batch normalization. The accuracy value change with and without batch normalization is shown in Fig 1.11. Results of training with weight decay of 0.001 and learning rate of 0.0003. The above results were observed without batch normalization and got an accuracy of 82.75. When batch normalization is included, Test Accuracy accuracy is 91.75.

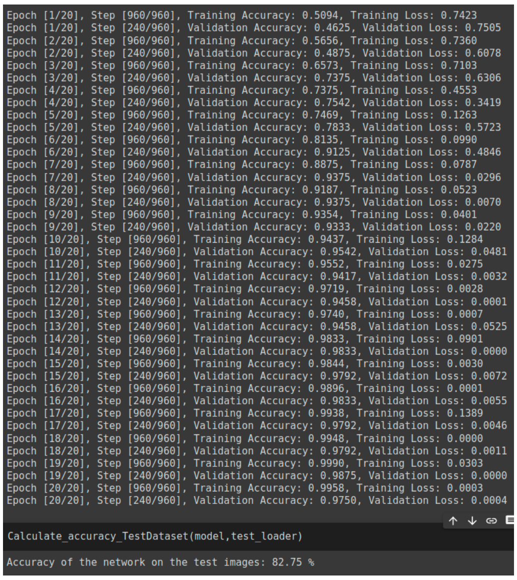
Step 8: In the final step for cross validation, the starting parameters used are shown in Fig 1.12. ReduceLROnPlateau scheduler is used to update the learning rate. The training rate updates itself if the validation loss doesn't decrease till the number of epochs equal to patience(passed as an argument to ReduceLROn Plateau Function. The patience = 5.

```
Epoch [1/20], Step [960/960], Training Loss: nan
Epoch [1/20], Step [240/960], Validation Loss: nan
Epoch [2/20], Step [960/960], Training Loss: nan
Epoch [2/20], Step [240/960], Validation Loss: nan
Epoch [3/20], Step [960/960], Training Loss: nan
Epoch [3/20], Step [240/960], Validation Loss: nan
Epoch [4/20], Step [960/960], Training Loss: nan
Epoch [4/20], Step [240/960], Validation Loss: nan
Epoch [5/20], Step [960/960], Training Loss: nan
Epoch [5/20], Step [240/960], Validation Loss: nan
Epoch [6/20], Step [960/960], Training Loss: nan
Epoch [6/20], Step [240/960], Validation Loss: nan
Epoch [7/20], Step [960/960], Training Loss: nan
Epoch [7/20], Step [240/960], Validation Loss: nan
Epoch [8/20], Step [960/960], Training Loss: nan
Epoch [8/20], Step [240/960], Validation Loss: nan
Epoch [9/20], Step [960/960], Training Loss: nan
Epoch [9/20], Step [240/960], Validation Loss: nan
Epoch [10/20], Step [960/960], Training Loss: nan
Epoch [10/20], Step [240/960], Validation Loss: nan
Epoch [11/20], Step [960/960], Training Loss: nan
Epoch [11/20], Step [240/960], Validation Loss: nan
Epoch [12/20], Step [960/960], Training Loss: nan
Epoch [12/20], Step [240/960], Validation Loss: nan
Epoch [13/20], Step [960/960], Training Loss: nan
Epoch [13/20], Step [240/960], Validation Loss: nan
Epoch [14/20], Step [960/960], Training Loss: nan
Epoch [14/20], Step [240/960], Validation Loss: nan
Epoch [15/20], Step [960/960], Training Loss: nan
Epoch [15/20], Step [240/960], Validation Loss: nan
Epoch [16/20], Step [960/960], Training Loss: nan
Epoch [16/20], Step [240/960], Validation Loss: nan
Epoch [17/20], Step [960/960], Training Loss: nan
Epoch [17/20], Step [240/960], Validation Loss: nan
Epoch [18/20], Step [960/960], Training Loss: nan
Epoch [18/20], Step [240/960], Validation Loss: nan
Epoch [19/20], Step [960/960], Training Loss: nan
Epoch [19/20], Step [240/960], Validation Loss: nan
Epoch [20/20], Step [960/960], Training Loss: nan
Epoch [20/20], Step [240/960], Validation Loss: nan
```

Figure 1.10: Getting NaN values for loss

## 1.9 Observation and Conclusion

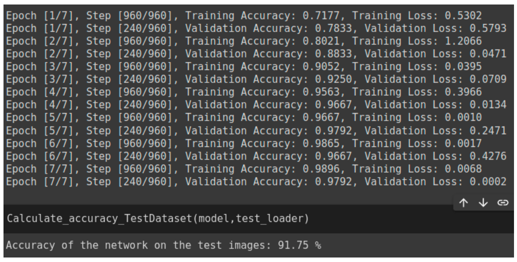
Learned the architecture of LeNet5 and the different layers in the model. Different hyper parameters were defined and tested on the training and testing dataset. Analyzed the accuracy, loss function and different changes during the evaluation and shown in the result. Thus, if we add more number of data using data augmentation and standardize the data using data normalization, the accuracy increases. Regularization and optimization also plays a major part in the model evaluation. Therefore, the machine learning performance is supported by other parameters. So, the improvement in the parameters help in getting the best result.


```
Epoch [1/20], Step [960/960], Training Accuracy: 0.5094, Training Loss: 0.7423
Epoch [1/20], Step [240/960], Validation Accuracy: 0.4625, Validation Loss: 0.7505
Epoch [2/20], Step [960/960], Training Accuracy: 0.5656, Training Loss: 0.7360
Epoch [2/20], Step [240/960], Validation Accuracy: 0.4875, Validation Loss: 0.6078
Epoch [3/20], Step [960/960], Training Accuracy: 0.6573, Training Loss: 0.7103
Epoch [3/20], Step [240/960], Validation Accuracy: 0.7375, Validation Loss: 0.6306
Epoch [4/20], Step [960/960], Training Accuracy: 0.7375, Training Loss: 0.4553
Epoch [4/20], Step [240/960], Validation Accuracy: 0.7542, Validation Loss: 0.3419
Epoch [5/20], Step [960/960], Training Accuracy: 0.7469, Training Loss: 0.1263
Epoch [5/20], Step [240/960], Validation Accuracy: 0.7833, Validation Loss: 0.5723
Epoch [6/20], Step [960/960], Training Accuracy: 0.8135, Training Loss: 0.0990
Epoch [6/20], Step [240/960], Validation Accuracy: 0.9125, Validation Loss: 0.4846
Epoch [7/20], Step [960/960], Training Accuracy: 0.8875, Training Loss: 0.0787
Epoch [7/20], Step [240/960], Validation Accuracy: 0.9375, Validation Loss: 0.0296
Epoch [8/20], Step [960/960], Training Accuracy: 0.9187, Training Loss: 0.0523
Epoch [8/20], Step [240/960], Validation Accuracy: 0.9375, Validation Loss: 0.0070
Epoch [9/20], Step [960/960], Training Accuracy: 0.9354, Training Loss: 0.0401
Epoch [9/20], Step [240/960], Validation Accuracy: 0.9333, Validation Loss: 0.0220
Epoch [10/20], Step [960/960], Training Accuracy: 0.9437, Training Loss: 0.1284
Epoch [10/20], Step [240/960], Validation Accuracy: 0.9542, Validation Loss: 0.0481
Epoch [11/20], Step [960/960], Training Accuracy: 0.9552, Training Loss: 0.0275
Epoch [11/20], Step [240/960], Validation Accuracy: 0.9417, Validation Loss: 0.0032
Epoch [12/20], Step [960/960], Training Accuracy: 0.9719, Training Loss: 0.0028
Epoch [12/20], Step [240/960], Validation Accuracy: 0.9458, Validation Loss: 0.0001
Epoch [13/20], Step [960/960], Training Accuracy: 0.9740, Training Loss: 0.0007
Epoch [13/20], Step [240/960], Validation Accuracy: 0.9458, Validation Loss: 0.0525
Epoch [14/20], Step [960/960], Training Accuracy: 0.9833, Training Loss: 0.0901
Epoch [14/20], Step [240/960], Validation Accuracy: 0.9833, Validation Loss: 0.0000
Epoch [15/20], Step [960/960], Training Accuracy: 0.9844, Training Loss: 0.0030
Epoch [15/20], Step [240/960], Validation Accuracy: 0.9792, Validation Loss: 0.0072
Epoch [16/20], Step [960/960], Training Accuracy: 0.9896, Training Loss: 0.0001
Epoch [16/20], Step [240/960], Validation Accuracy: 0.9833, Validation Loss: 0.0055
Epoch [17/20], Step [960/960], Training Accuracy: 0.9938, Training Loss: 0.1389
Epoch [17/20], Step [240/960], Validation Accuracy: 0.9792, Validation Loss: 0.0046
Epoch [18/20], Step [960/960], Training Accuracy: 0.9948, Training Loss: 0.0000
Epoch [18/20], Step [240/960], Validation Accuracy: 0.9792, Validation Loss: 0.0011
Epoch [19/20], Step [960/960], Training Accuracy: 0.9990, Training Loss: 0.0303
Epoch [19/20], Step [240/960], Validation Accuracy: 0.9875, Validation Loss: 0.0000
Epoch [20/20], Step [960/960], Training Accuracy: 0.9958, Training Loss: 0.0003
Epoch [20/20], Step [240/960], Validation Accuracy: 0.9750, Validation Loss: 0.0004

Calculate_accuracy_TestDataset(model,test_loader)

Accuracy of the network on the test images: 82.75 %
```

(a) Without Batch Normalization

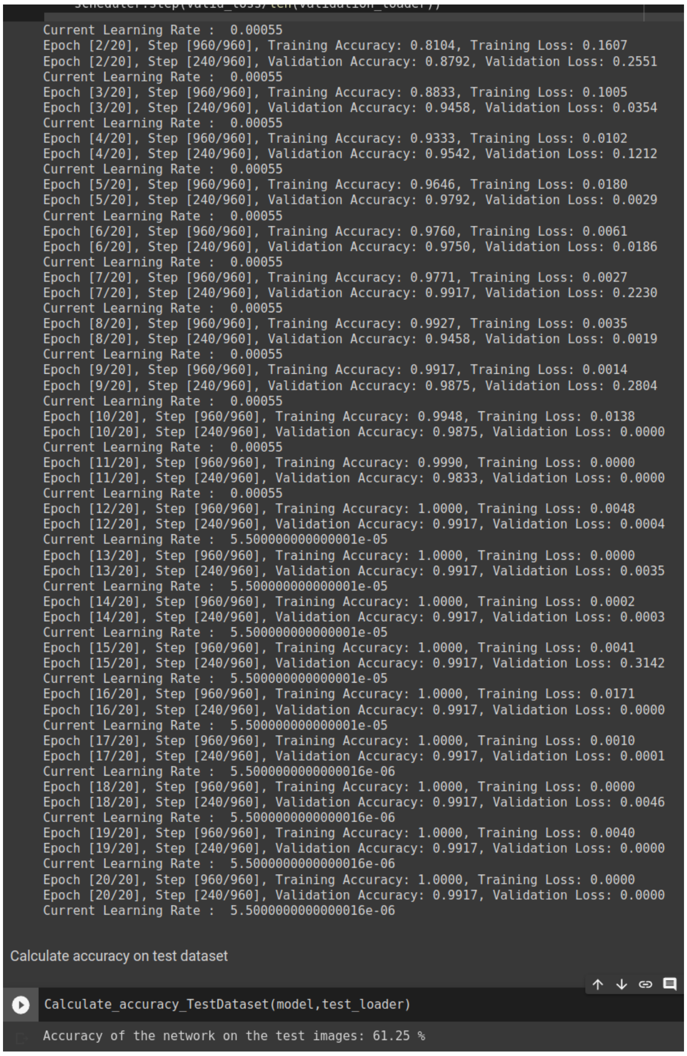

```
Epoch [1/7], Step [960/960], Training Accuracy: 0.7177, Training Loss: 0.5302
Epoch [1/7], Step [240/960], Validation Accuracy: 0.7833, Validation Loss: 0.5793
Epoch [2/7], Step [960/960], Training Accuracy: 0.8021, Training Loss: 1.2066
Epoch [2/7], Step [240/960], Validation Accuracy: 0.8833, Validation Loss: 0.0471
Epoch [3/7], Step [960/960], Training Accuracy: 0.9052, Training Loss: 0.0395
Epoch [3/7], Step [240/960], Validation Accuracy: 0.9250, Validation Loss: 0.0709
Epoch [4/7], Step [960/960], Training Accuracy: 0.9563, Training Loss: 0.3966
Epoch [4/7], Step [240/960], Validation Accuracy: 0.9667, Validation Loss: 0.0134
Epoch [5/7], Step [960/960], Training Accuracy: 0.9667, Training Loss: 0.0010
Epoch [5/7], Step [240/960], Validation Accuracy: 0.9792, Validation Loss: 0.2471
Epoch [6/7], Step [960/960], Training Accuracy: 0.9865, Training Loss: 0.0017
Epoch [6/7], Step [240/960], Validation Accuracy: 0.9667, Validation Loss: 0.4276
Epoch [7/7], Step [960/960], Training Accuracy: 0.9896, Training Loss: 0.0068
Epoch [7/7], Step [240/960], Validation Accuracy: 0.9792, Validation Loss: 0.0002

Calculate_accuracy_TestDataset(model,test_loader)

Accuracy of the network on the test images: 91.75 %
```

(b) With Batch Normalization

Figure 1.11: Accuracy comparison with and without Batch Normalization


```
Current Learning Rate :  0.00055
Epoch [2/20], Step [960/960], Training Accuracy: 0.8104, Training Loss: 0.1607
Epoch [2/20], Step [240/960], Validation Accuracy: 0.8792, Validation Loss: 0.2551
Current Learning Rate :  0.00055
Epoch [3/20], Step [960/960], Training Accuracy: 0.8833, Training Loss: 0.1005
Epoch [3/20], Step [240/960], Validation Accuracy: 0.9458, Validation Loss: 0.0354
Current Learning Rate :  0.00055
Epoch [4/20], Step [960/960], Training Accuracy: 0.9333, Training Loss: 0.0102
Epoch [4/20], Step [240/960], Validation Accuracy: 0.9542, Validation Loss: 0.1212
Current Learning Rate :  0.00055
Epoch [5/20], Step [960/960], Training Accuracy: 0.9646, Training Loss: 0.0180
Epoch [5/20], Step [240/960], Validation Accuracy: 0.9792, Validation Loss: 0.0029
Current Learning Rate :  0.00055
Epoch [6/20], Step [960/960], Training Accuracy: 0.9760, Training Loss: 0.0061
Epoch [6/20], Step [240/960], Validation Accuracy: 0.9750, Validation Loss: 0.0186
Current Learning Rate :  0.00055
Epoch [7/20], Step [960/960], Training Accuracy: 0.9771, Training Loss: 0.0027
Epoch [7/20], Step [240/960], Validation Accuracy: 0.9917, Validation Loss: 0.2230
Current Learning Rate :  0.00055
Epoch [8/20], Step [960/960], Training Accuracy: 0.9927, Training Loss: 0.0035
Epoch [8/20], Step [240/960], Validation Accuracy: 0.9458, Validation Loss: 0.0019
Current Learning Rate :  0.00055
Epoch [9/20], Step [960/960], Training Accuracy: 0.9917, Training Loss: 0.0014
Epoch [9/20], Step [240/960], Validation Accuracy: 0.9875, Validation Loss: 0.2804
Current Learning Rate :  0.00055
Epoch [10/20], Step [960/960], Training Accuracy: 0.9948, Training Loss: 0.0138
Epoch [10/20], Step [240/960], Validation Accuracy: 0.9875, Validation Loss: 0.0000
Current Learning Rate :  0.00055
Epoch [11/20], Step [960/960], Training Accuracy: 0.9990, Training Loss: 0.0000
Epoch [11/20], Step [240/960], Validation Accuracy: 0.9833, Validation Loss: 0.0000
Current Learning Rate :  0.00055
Epoch [12/20], Step [960/960], Training Accuracy: 1.0000, Training Loss: 0.0048
Epoch [12/20], Step [240/960], Validation Accuracy: 0.9917, Validation Loss: 0.0004
Current Learning Rate :  5.500000000000001e-05
Epoch [13/20], Step [960/960], Training Accuracy: 1.0000, Training Loss: 0.0000
Epoch [13/20], Step [240/960], Validation Accuracy: 0.9917, Validation Loss: 0.0035
Current Learning Rate :  5.500000000000001e-05
Epoch [14/20], Step [960/960], Training Accuracy: 1.0000, Training Loss: 0.0002
Epoch [14/20], Step [240/960], Validation Accuracy: 0.9917, Validation Loss: 0.0003
Current Learning Rate :  5.500000000000001e-05
Epoch [15/20], Step [960/960], Training Accuracy: 1.0000, Training Loss: 0.0041
Epoch [15/20], Step [240/960], Validation Accuracy: 0.9917, Validation Loss: 0.3142
Current Learning Rate :  5.500000000000001e-05
Epoch [16/20], Step [960/960], Training Accuracy: 1.0000, Training Loss: 0.0171
Epoch [16/20], Step [240/960], Validation Accuracy: 0.9917, Validation Loss: 0.0000
Current Learning Rate :  5.500000000000001e-05
Epoch [17/20], Step [960/960], Training Accuracy: 1.0000, Training Loss: 0.0010
Epoch [17/20], Step [240/960], Validation Accuracy: 0.9917, Validation Loss: 0.0001
Current Learning Rate :  5.5000000000000016e-06
Epoch [18/20], Step [960/960], Training Accuracy: 1.0000, Training Loss: 0.0000
Epoch [18/20], Step [240/960], Validation Accuracy: 0.9917, Validation Loss: 0.0046
Current Learning Rate :  5.5000000000000016e-06
Epoch [19/20], Step [960/960], Training Accuracy: 1.0000, Training Loss: 0.0040
Epoch [19/20], Step [240/960], Validation Accuracy: 0.9917, Validation Loss: 0.0000
Current Learning Rate :  5.5000000000000016e-06
Epoch [20/20], Step [960/960], Training Accuracy: 1.0000, Training Loss: 0.0000
Epoch [20/20], Step [240/960], Validation Accuracy: 0.9917, Validation Loss: 0.0000
Current Learning Rate :  5.5000000000000016e-06

Calculate accuracy on test dataset

Calculate_accuracy_TestDataset(model,test_loader)

Accuracy of the network on the test images: 61.25 %
```

Figure 1.12: Cross Validation Strategy

## 1.10 References

[1] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791

[2] http://vis-www.cs.umass.edu/fddb/

Websites:
1. https://pytorch.org/tutorials/beginner/introyt/modelsyt$_t$$utorial.html?highlight = lenet$5
2. https://discuss.pytorch.org/t/simple-l2-regularization/139

CONTRIBUTION:
Bhagya- data preprocessing (augment, normalized, regularization others), report
Apoorva- model exploration, selection and implemention, adding standard regularization such as batch normalization, report.
Hritwik-hyparameter tuning, cross validation checking, report