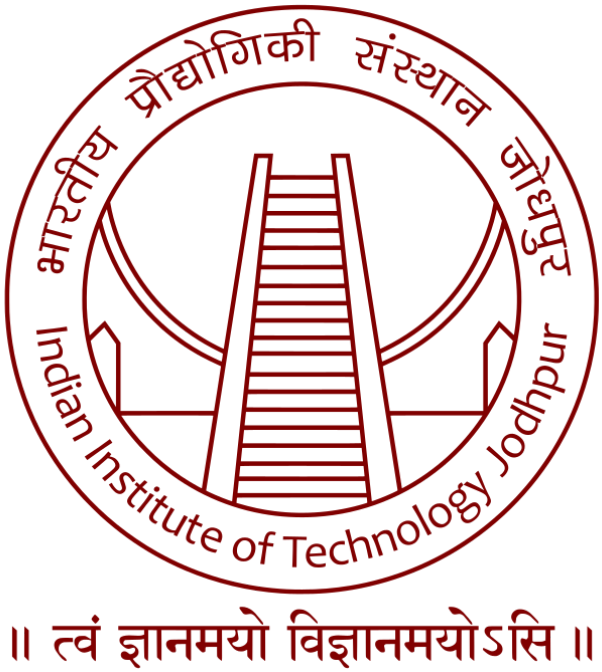
Assignment 3

Machine Learning



Submitted By
Ratnesh Kumar Tiwari
M23MAC011

Submitted to
Dr. Deepak Mishra

Please visit this link to view the code:
https://colab.research.google.com/drive/1OkO3xrsLLG5t7iFJMYZTUsjodc-scNLa?usp=sharing

In this assignment, I have incorporated a neural network from scratch in python. The network architecture followed is-

1. The network contains 3 hidden layers. Excluding input and output layers.
2. Network architecture is set as:
   a. Input layer = set to the size of the dimensions
   b. Output layer = set to the size of the number of classes
   c. Hidden layer 1 = 128
   d. Hidden layer 2 = 64
   e. Hidden layer 3 = 32

3. The weights are initialized randomly using seed value as 11 (M23MAC011).
4. Train-test splits are as randomized 70:30, 80:20 and 90:10.
5. Batch size is 23 (M23MAC011)
6. 'Sigmoid' is used as an activation function for hidden layers and 'softmax' for output layers.
7. Gradient Descent is used for optimization. Cross entropy is used as Loss Function
8. Trained for 25 epochs. Plotted accuracy and loss per epoch.
9. Prepared a Confusion matrix for all the combinations of the network.
10. Used L2 regularization to prevent overfitting

Methodology:

1. **Neural Network Architecture Setup (__init__ and initialize_weights):**
   __init__(self, input_size, hidden_layers, output_size):
   a. Initializes the neural network with specified input size, hidden layer configurations, and output size.
   b. Stores these parameters and initializes weights and biases for each layer.

   initialize_weights(self):
   a. Sets up weights and biases for each layer in the neural network.
   b. Generates random initial weights with seed value 11 using a normal distribution for each layer and initializes biases to ones.

2. **Activation Functions (sigmoid and softmax):**
   sigmoid(self, x):
   a. Implements the sigmoid activation function used in the hidden layers of the neural network.
   b. Squashes input values between 0 and 1.

softmax(self, x):

a. Implements the softmax activation function used in the output layer for multi-class classification.
b. Calculates probabilities for each class ensuring they sum up to 1.

3. **Forward and Backward Propagation (forward_propagation and backward_propagation):**

forward_propagation(self, X):

a. Conducts forward propagation through the neural network.
b. Computes activations and intermediate outputs for each layer using weights and biases.

backward_propagation(self, X, y, activations, learning_rate, reg_lambda):

a. Implements backward propagation to update weights and biases based on calculated gradients.
b. Computes errors, gradients, and updates weights and biases using gradient descent.
c. Incorporates regularization to prevent overfitting.

4. **Training the Neural Network (train):**

train(self, X_train, y_train, epochs, learning_rate, batch_size):

a. Trains the neural network using the provided training data and specified hyperparameters.
b. Conducts training for the specified number of epochs, updating weights and biases using batches of data.
c. Calculates and stores training losses and accuracies per epoch.

5. **Loss Calculation (calculate_loss):**

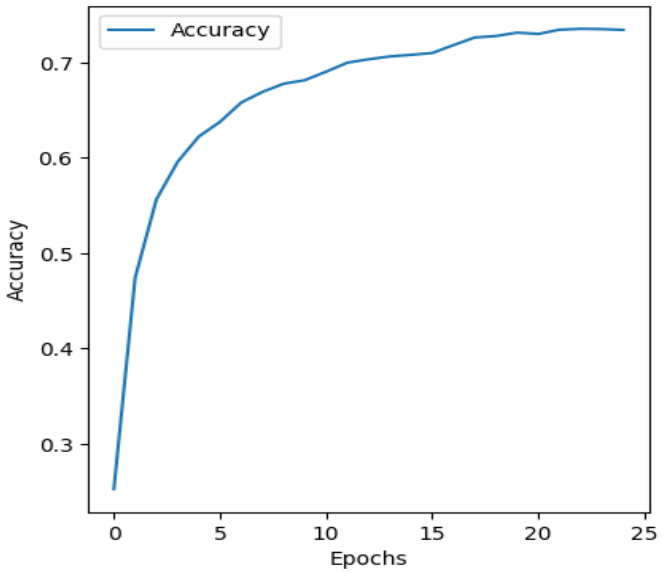
calculate_loss(self, y_true, y_pred):

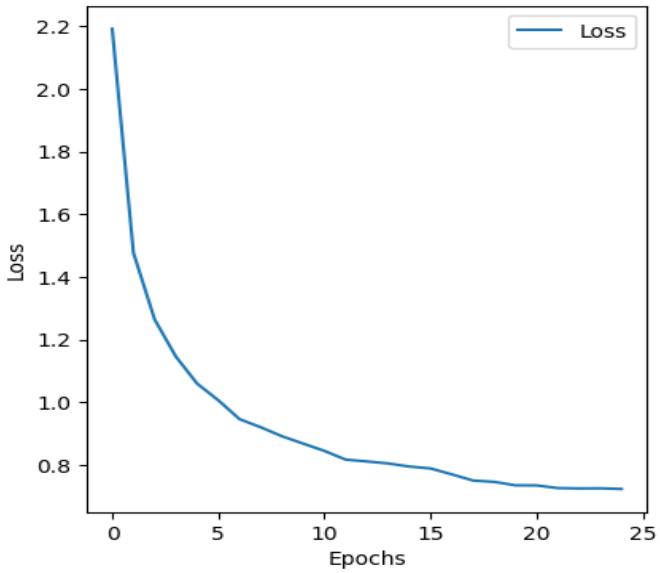
a. Computes the loss/error between predicted and true labels.
b. Utilizes cross-entropy loss for multi-class classification.

Results:

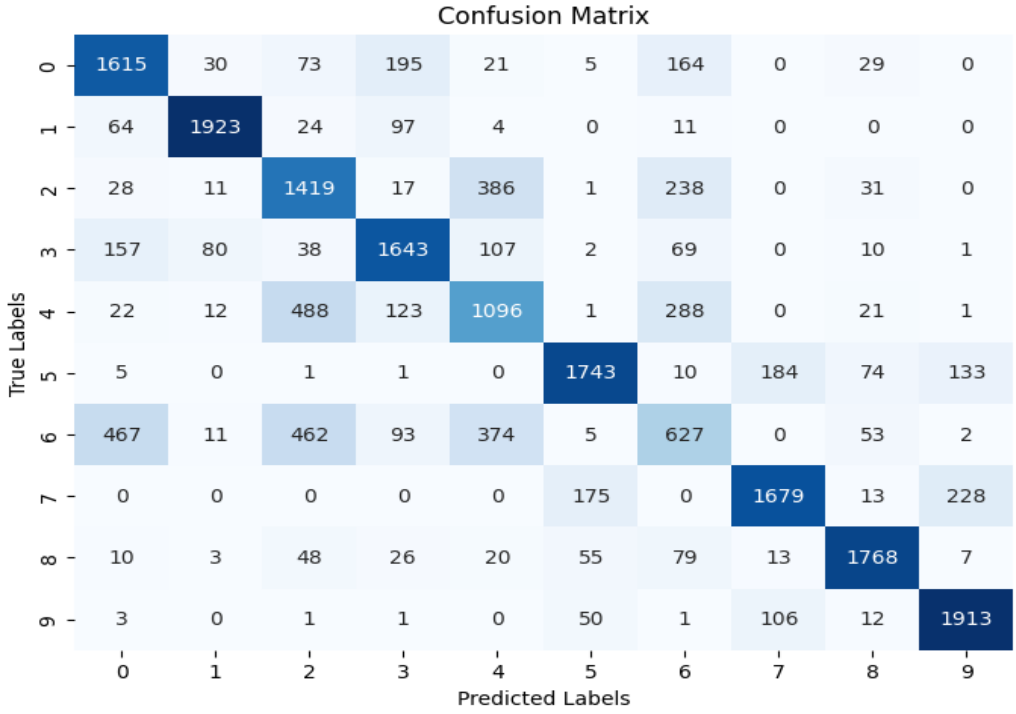
1. Train Test Split ratio as 70:30

a. Loss and Accuracy vs Epoch graph



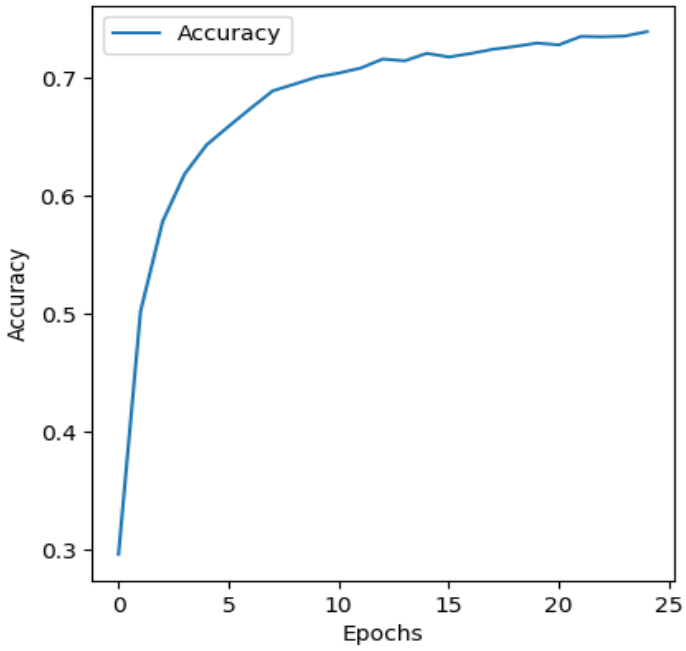


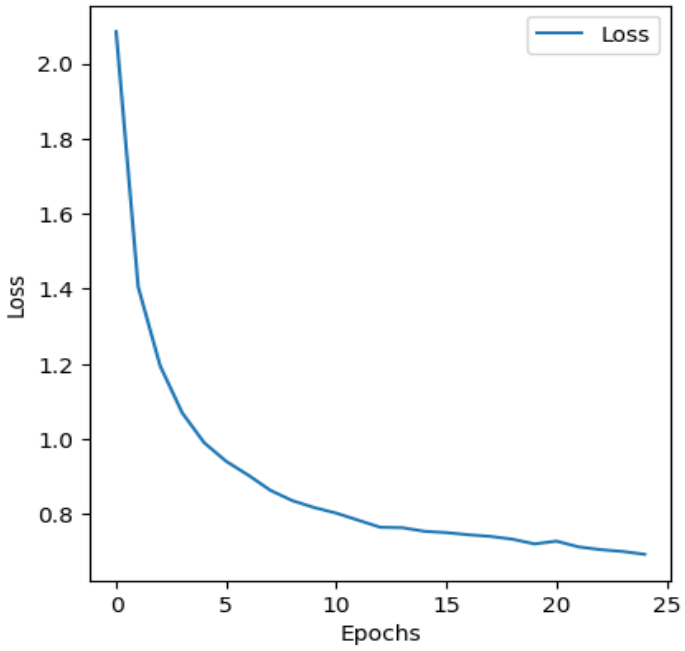
b. Confusion Matrix



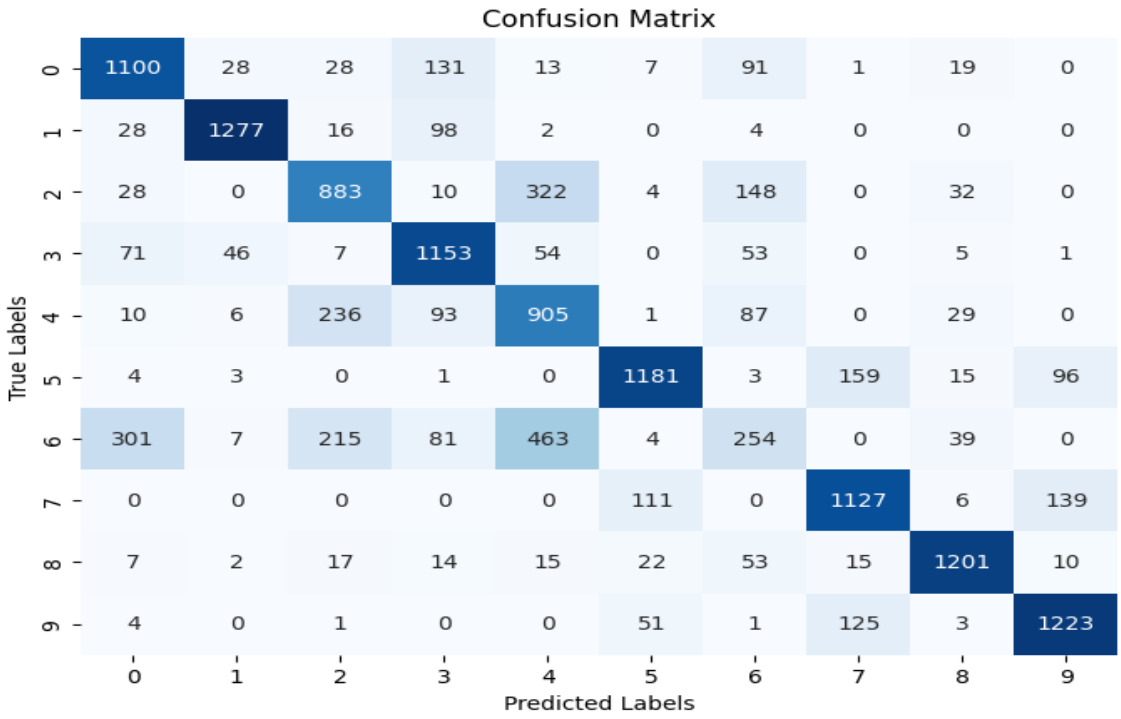
2. Train Test Split ratio as 80:20

a. Accuracy and Loss vs Epochs



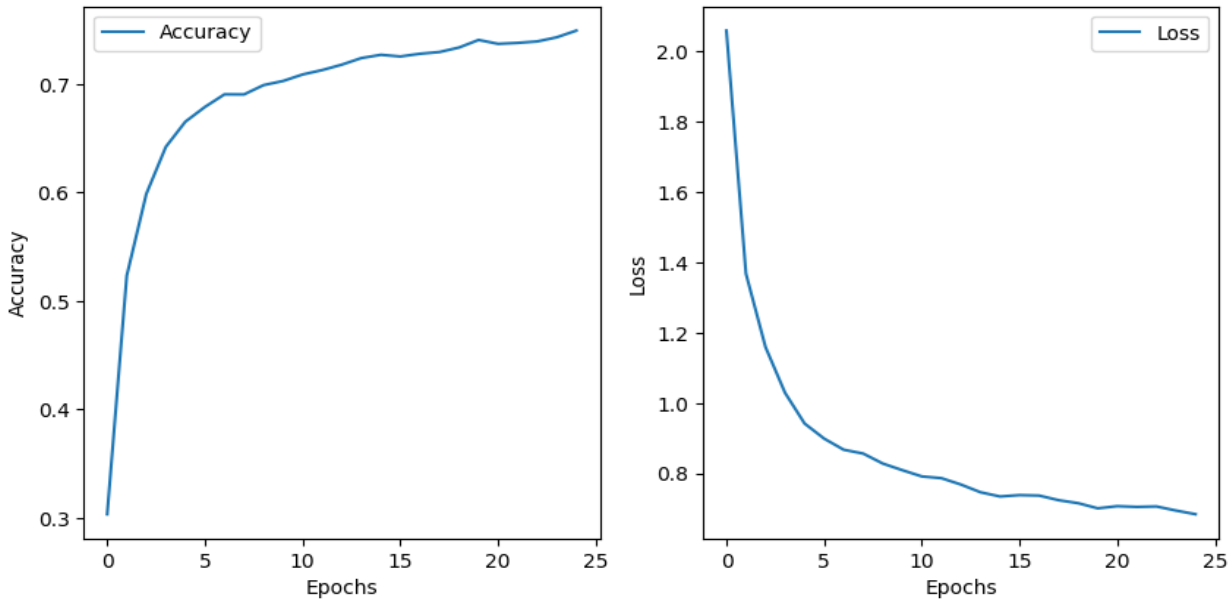


b. Confusion Matrix

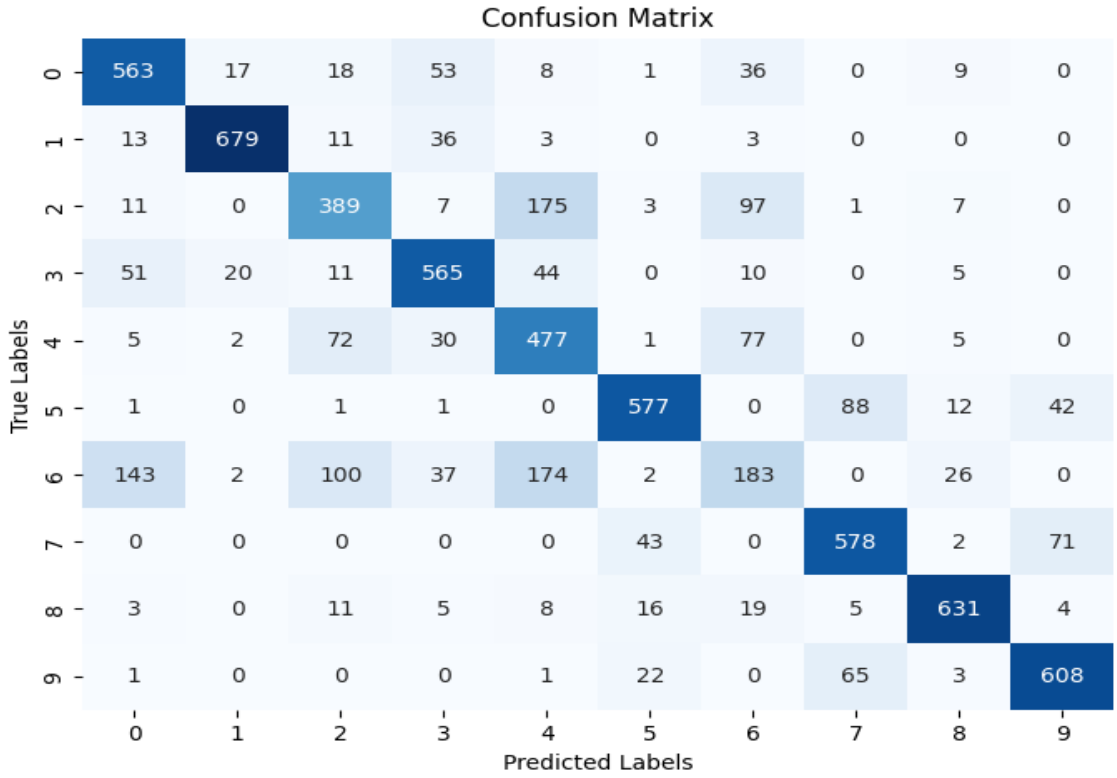


3. Train Test Split ratio as 90:10

a. Accuracy and Loss vs Epochs



b. Confusion matrix



4. Number of Trainable and Non-Trainable Parameters

Trainable parameters: Weights and biases in the neural network layers.
Non-trainable parameters: Direct connections from input to output.

In our case, the number of trainable parameters is equal to 111146 and the number of non-trainable parameters is equal to 7840.
This is observed in all of the three cases of split i.e, 70%, 80% and 90%.

5. Train and Test Accuracy

a. In 70:30 split

The training accuracy is 73.4211104990286 % and the testing accuracy is 73.45364506452073 %.

b. In 80:20 split

The training accuracy is 73.9128450832763 % and the testing accuracy is 73.6 %

c. In 90:10 split

The training accuracy is 74.94499100814555 % and the testing accuracy is 75.0 %.

Observations:

1. In each of the cases we can see that the Loss curve decreases sharply and after epoch 6, it decreases with a slower rate. After epoch 20,the loss curve starts to saturate.
2. Similarly, in all of the cases we can see that the accuracy curve sharply increases in epoch 0 to 5 and after that it increases at a slower rate. After epoch 20, the accuracy curve starts to saturate.
3. In the split of 70:30, the training accuracy observed is 73.4211104990286 %, while the test accuracy observed is 73.45364506452073 %.
   Similarly, in the split of 80:20, the training accuracy observed is 73.9128450832763 %, while the test accuracy observed is 73.6 %.
   And in the split of 90:10, the training accuracy observed is 74.94499100814555 %, while the test accuracy observed is 75.0 %.
4. We can clearly see that there is not any significant difference between the accuracies of train set and test set. This suggests that our model is not overfitted. This is because of the implementation of L2 regularization.
5. The diagonal element of the confusion matrix in each case shows the number of test data points that are correctly classified in each of the classes. While all elements represent misclassified data points in each class.
6. The diagonal element is abruptly higher than other elements in the confusion matrix. This suggests that our model is classifying correctly in most of the cases.