Implementation and evaluation of MLP, CNN and KNN

Goutham Reddy Kallem

Introduction

In this project, three different classifiers were implemented and evaluated their performance on the MNIST dataset. Specifically, we will implement a Multi-Layer Perceptron (MLP), k-nearest neighbors (KNN) classifier, and a convolutional neural network (CNN) using MATLAB functions.

Implemented the MLP and KNN classifiers from scratch and assess their performance on the MNIST database. Will vary the number of units in the hidden layer of the MLP and the value of k for KNN to determine the optimal hyperparameters. Compute the confusion matrix and accuracy to evaluate the performance of the classifiers.

Implemented CNN using existing MATLAB functions, and evaluated its performance on the MNIST dataset

Methods implemented

MLP

activation_fn = @sigmoid
An activation function that maps any input value to a value between 0 and 1.

Main MLP algorithm to update weights for various hidden layer sizes

CNN

Trains the CNN using the training data and the defined architecture with the stochastic gradient descent with momentum (SGDM) optimizer. Specifies the number of epochs, learning rate, mini-batch size, validation data, and other training options.

Once the CNN is trained, test its performance using the test data. It uses the trained CNN model to classify the test data and calculates the accuracy of the model by comparing the predicted labels with the true labels. Finally, plot the confusion matrix to visualize the model's performance in classifying each digit.

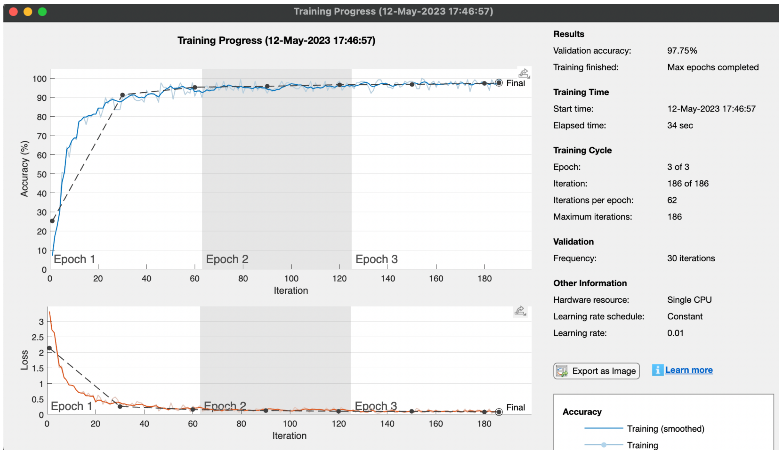
KNN

Method knn(train_data, train_labels, test_data, test_labels, k)

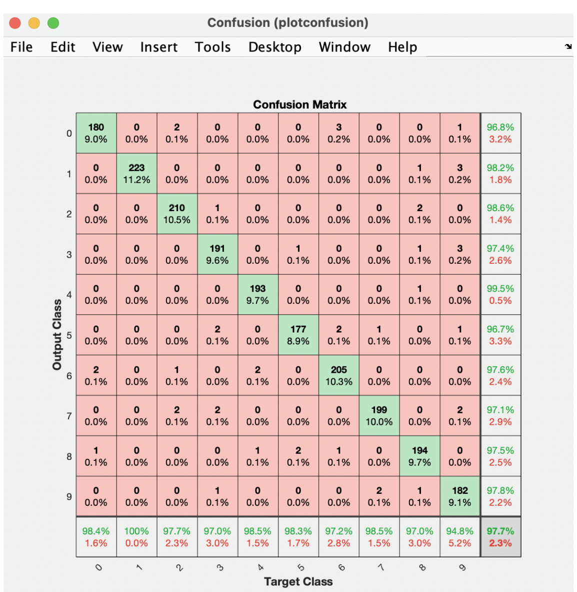
This is a function that implements the K-Nearest Neighbors (KNN) algorithm for classification. The function takes as input the training data, training labels, test data, test labels, and the number of neighbors to consider (k). It returns the predicted labels for the test data, as well as the accuracy of the predictions.

Results

Convolutional Neural Network(CNN)



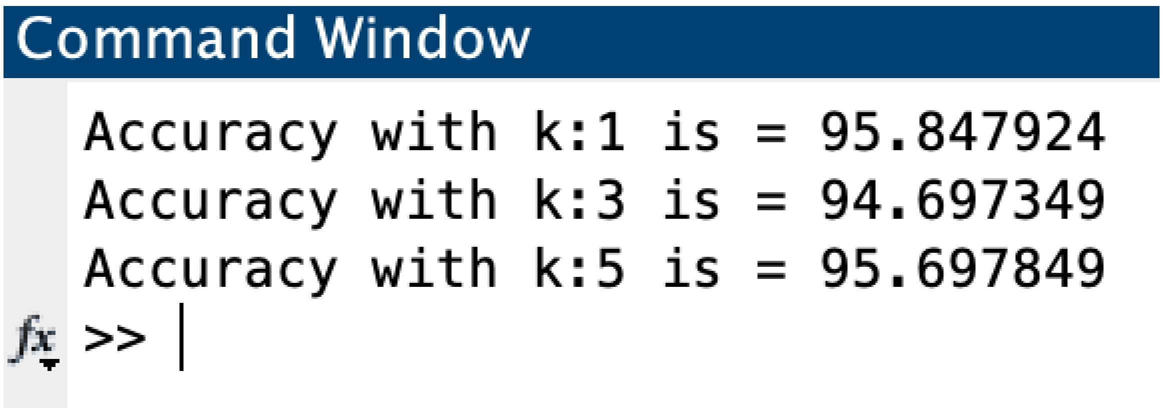

```
>> CNN
Training on single CPU.
Initializing input data normalization.
|===============================================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Validation  |  Mini-batch  |  Validation  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |   Accuracy   |     Loss     |     Loss     |      Rate       |
|===============================================================================================================|
|       1 |           1 |       00:00:07 |        7.03% |       25.21% |       3.3174 |       2.1350 |          0.0100 |
|       1 |          30 |       00:00:12 |       89.84% |       91.35% |       0.3300 |       0.2556 |          0.0100 |
|       1 |          50 |       00:00:14 |       96.88% |              |       0.1285 |              |          0.0100 |
|       1 |          60 |       00:00:16 |       90.62% |       95.35% |       0.2667 |       0.1535 |          0.0100 |
|       2 |          90 |       00:00:20 |       92.19% |       95.80% |       0.1520 |       0.1227 |          0.0100 |
|       2 |         100 |       00:00:21 |       96.09% |              |       0.1283 |              |          0.0100 |
|       2 |         120 |       00:00:24 |       98.44% |       96.65% |       0.0641 |       0.1004 |          0.0100 |
|       3 |         150 |       00:00:28 |       98.44% |       96.80% |       0.1189 |       0.0937 |          0.0100 |
|       3 |         180 |       00:00:32 |       97.66% |       97.30% |       0.0689 |       0.0811 |          0.0100 |
|       3 |         186 |       00:00:33 |       98.44% |       97.55% |       0.0736 |       0.0780 |          0.0100 |
|===============================================================================================================|
Training finished: Max epochs completed.
Test Accuracy: 97.75%
>>
```



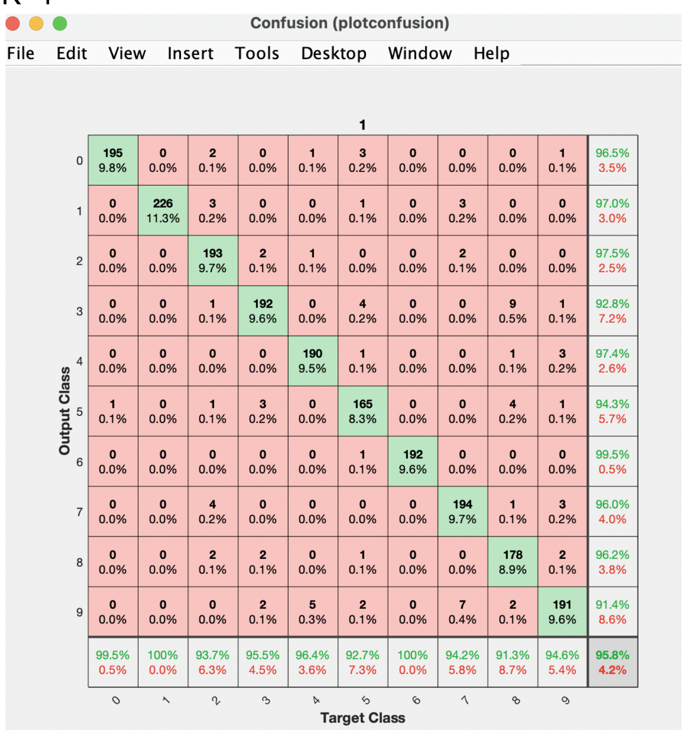
Got accuracy of 97.75

Values along the diagonal indicate the model predicted the correct labels most of the time, with 100% prediction for number **“1”** and the least success for numbers **“8 and 3”**

The k-nearest neighbors (KNN)



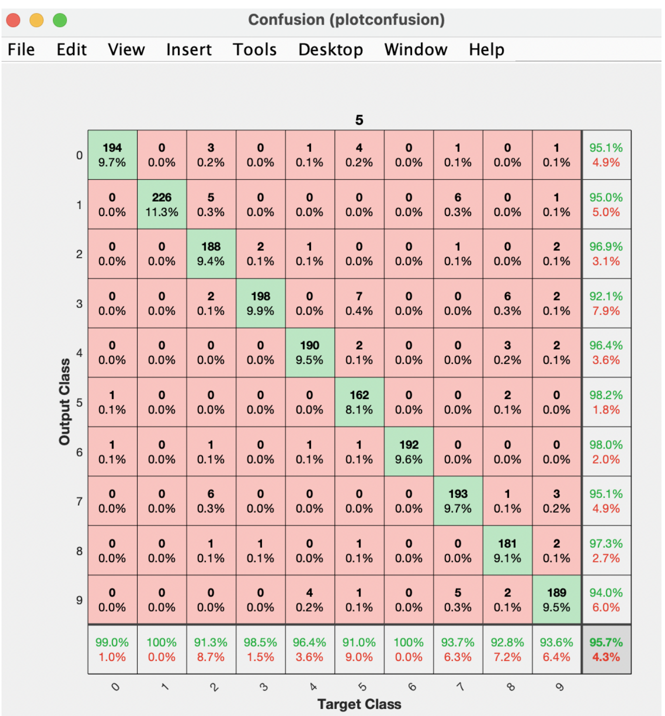
K=1



K=3

Output \ Target	0	1	2	3	4	5	6	7	8	9	
0	196 9.8%	0 0.0%	3 0.2%	0 0.0%	1 0.1%	3 0.2%	2 0.1%	0 0.0%	1 0.1%	1 0.1%	94.7% 5.3%
1	0 0.0%	226 11.3%	5 0.3%	0 0.0%	0 0.0%	1 0.1%	0 0.0%	7 0.4%	0 0.0%	1 0.1%	94.2% 5.8%
2	0 0.0%	0 0.0%	194 9.7%	4 0.2%	1 0.1%	0 0.0%	0 0.0%	1 0.1%	4 0.2%	0 0.0%	95.1% 4.9%
3	0 0.0%	0 0.0%	0 0.0%	195 9.8%	0 0.0%	11 0.6%	0 0.0%	0 0.0%	16 0.8%	3 0.2%	86.7% 13.3%
4	0 0.0%	0 0.0%	0 0.0%	0 0.0%	194 9.7%	2 0.1%	1 0.1%	2 0.1%	3 0.2%	6 0.3%	93.3% 6.7%
5	0 0.0%	0 0.0%	0 0.0%	1 0.1%	0 0.0%	160 8.0%	1 0.1%	0 0.0%	3 0.2%	1 0.1%	96.4% 3.6%
6	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	188 9.4%	0 0.0%	0 0.0%	1 0.1%	99.5% 0.5%
7	0 0.0%	0 0.0%	3 0.2%	1 0.1%	0 0.0%	0 0.0%	0 0.0%	193 9.7%	1 0.1%	7 0.4%	94.1% 5.9%
8	0 0.0%	0 0.0%	1 0.1%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	166 8.3%	1 0.1%	98.8% 1.2%
9	0 0.0%	0 0.0%	0 0.0%	0 0.0%	1 0.1%	1 0.1%	0 0.0%	3 0.2%	1 0.1%	181 9.1%	96.8% 3.2%
	100% 0.0%	100% 0.0%	94.2% 5.8%	97.0% 3.0%	98.5% 1.5%	89.9% 10.1%	97.9% 2.1%	93.7% 6.3%	85.1% 14.9%	89.6% 10.4%	94.7% 5.3%

K=5



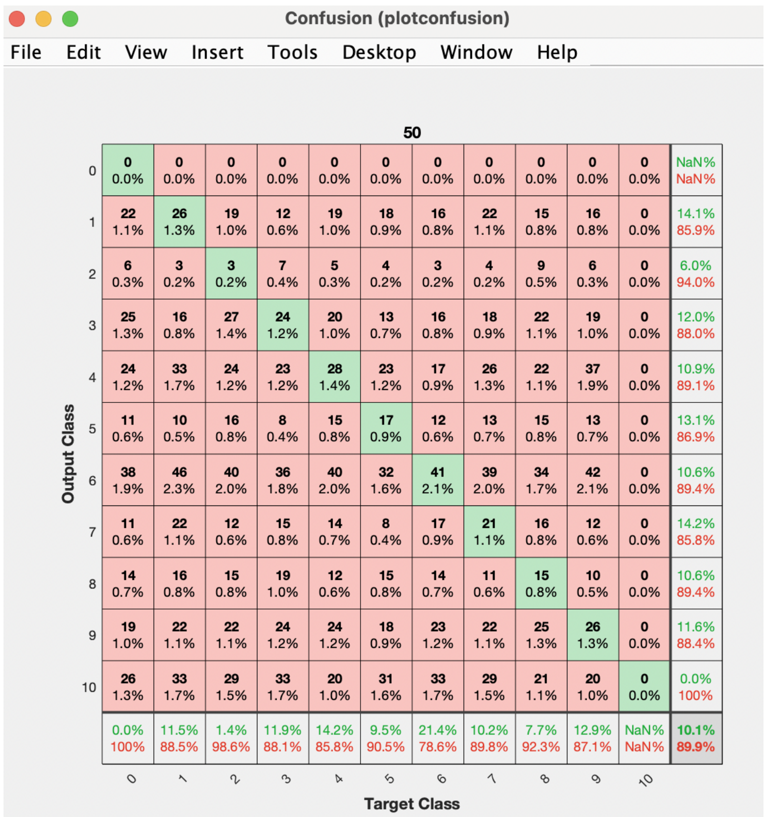
Output \ Target	0	1	2	3	4	5	6	7	8	9	
0	194 9.7%	0 0.0%	3 0.2%	0 0.0%	1 0.1%	4 0.2%	0 0.0%	1 0.1%	0 0.0%	1 0.1%	95.1% 4.9%
1	0 0.0%	226 11.3%	5 0.3%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	6 0.3%	0 0.0%	1 0.1%	95.0% 5.0%
2	0 0.0%	0 0.0%	188 9.4%	2 0.1%	1 0.1%	0 0.0%	0 0.0%	1 0.1%	0 0.0%	2 0.1%	96.9% 3.1%
3	0 0.0%	0 0.0%	2 0.1%	198 9.9%	0 0.0%	7 0.4%	0 0.0%	0 0.0%	6 0.3%	2 0.1%	92.1% 7.9%
4	0 0.0%	0 0.0%	0 0.0%	0 0.0%	190 9.5%	2 0.1%	0 0.0%	0 0.0%	3 0.2%	2 0.1%	96.4% 3.6%
5	1 0.1%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	162 8.1%	0 0.0%	0 0.0%	2 0.1%	0 0.0%	98.2% 1.8%
6	1 0.1%	0 0.0%	1 0.1%	0 0.0%	1 0.1%	1 0.1%	192 9.6%	0 0.0%	0 0.0%	0 0.0%	98.0% 2.0%
7	0 0.0%	0 0.0%	6 0.3%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	193 9.7%	1 0.1%	3 0.2%	95.1% 4.9%
8	0 0.0%	0 0.0%	1 0.1%	1 0.1%	0 0.0%	1 0.1%	0 0.0%	0 0.0%	181 9.1%	2 0.1%	97.3% 2.7%
9	0 0.0%	0 0.0%	0 0.0%	0 0.0%	4 0.2%	1 0.1%	0 0.0%	5 0.3%	2 0.1%	189 9.5%	94.0% 6.0%
	99.0% 1.0%	100% 0.0%	91.3% 8.7%	98.5% 1.5%	96.4% 3.6%	91.0% 9.0%	100% 0.0%	93.7% 6.3%	92.8% 7.2%	93.6% 6.4%	95.7% 4.3%

Overall model performed well for all the k values, but performed best with k =1.
Diagonal values indicate that the predictions are mostly successful for all k values.

A Multi-Layer Perceptron (MLP)

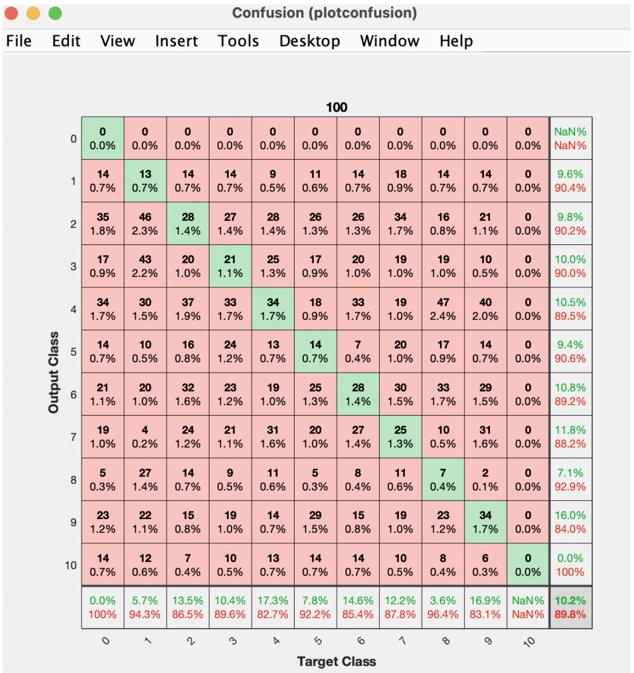
```
Command Window
Accuracy with hidden unit 50 is = 10.0550
Accuracy with hidden unit 100 is = 10.2051
Accuracy with hidden unit 200 is = 10.0550
>>
```

Units = 50

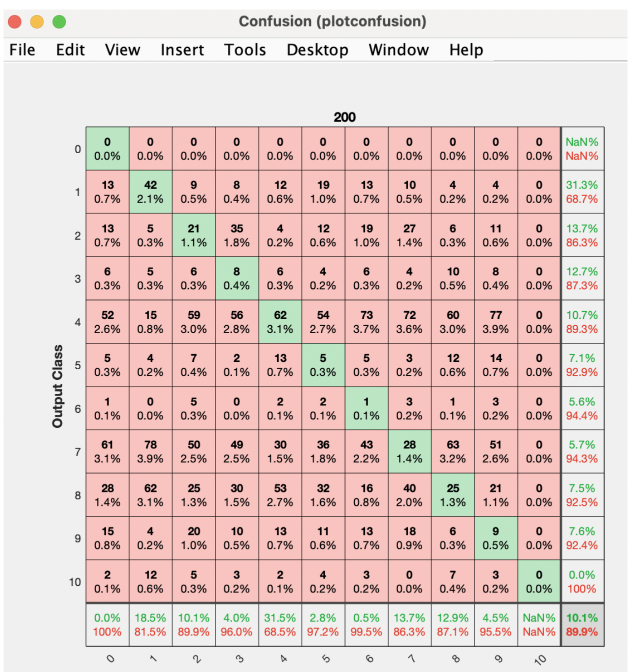


Output \ Target	0	1	2	3	4	5	6	7	8	9	10	
0	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN% NaN%
1	22 1.1%	26 1.3%	19 1.0%	12 0.6%	19 1.0%	18 0.9%	16 0.8%	22 1.1%	15 0.8%	16 0.8%	0 0.0%	14.1% 85.9%
2	6 0.3%	3 0.2%	3 0.2%	7 0.4%	5 0.3%	4 0.2%	3 0.2%	4 0.2%	9 0.5%	6 0.3%	0 0.0%	6.0% 94.0%
3	25 1.3%	16 0.8%	27 1.4%	24 1.2%	20 1.0%	13 0.7%	16 0.8%	18 0.9%	22 1.1%	19 1.0%	0 0.0%	12.0% 88.0%
4	24 1.2%	33 1.7%	24 1.2%	23 1.2%	28 1.4%	23 1.2%	17 0.9%	26 1.3%	22 1.1%	37 1.9%	0 0.0%	10.9% 89.1%
5	11 0.6%	10 0.5%	16 0.8%	8 0.4%	15 0.8%	17 0.9%	12 0.6%	13 0.7%	15 0.8%	13 0.7%	0 0.0%	13.1% 86.9%
6	38 1.9%	46 2.3%	40 2.0%	36 1.8%	40 2.0%	32 1.6%	41 2.1%	39 2.0%	34 1.7%	42 2.1%	0 0.0%	10.6% 89.4%
7	11 0.6%	22 1.1%	12 0.6%	15 0.8%	14 0.7%	8 0.4%	17 0.9%	21 1.1%	16 0.8%	12 0.6%	0 0.0%	14.2% 85.8%
8	14 0.7%	16 0.8%	15 0.8%	19 1.0%	12 0.6%	15 0.8%	14 0.7%	11 0.6%	15 0.8%	10 0.5%	0 0.0%	10.6% 89.4%
9	19 1.0%	22 1.1%	22 1.1%	24 1.2%	24 1.2%	18 0.9%	23 1.2%	22 1.1%	25 1.3%	26 1.3%	0 0.0%	11.6% 88.4%
10	26 1.3%	33 1.7%	29 1.5%	33 1.7%	20 1.0%	31 1.6%	33 1.7%	29 1.5%	21 1.1%	20 1.0%	0 0.0%	0.0% 100%
	0.0% 100%	11.5% 88.5%	1.4% 98.6%	11.9% 88.1%	14.2% 85.8%	9.5% 90.5%	21.4% 78.6%	10.2% 89.8%	7.7% 92.3%	12.9% 87.1%	NaN% NaN%	10.1% 89.9%

units=100



Units = 200



Got 9.6548 accuracy with 50 hidden units
Got 9.8549 accuracy with 100 hidden units
Got 7.5538 accuracy with 200 hidden units