Faculty of Engineering & Technology – Electrical & Computer Engineering Department

First Semester 2022 – 2023

Machine learning and data science

ENCS5341

Course project

Name: Amal Ziad

ID: 1192141

Instructor: Dr. Yazan AbuFarha

Date: 10th Feb 2023

Introduction:

The abstract of the project is to classify fashion mnist training and testing data in many ways. First, I made a baseline of k-nearest neighbor, as it will be shown later. After choosing two additional models, I have tracked each model to see what metric is better to tell me the model is getting better. Then I tried to get incorrected predictions, and have changed the feature I used to make the model better in performance. The models I used besides Knn:

1. **<u>Convolutional neural network (CNN) model</u>**: I have chosen this model for these reasons:

- <u>Image classification</u>: Fashion MNIST is an image classification problem, and CNNs are designed specifically for image classification tasks. They have proven to be very successful in this type of problem, especially when the input images have some spatial structure, as in the case of Fashion MNIST.
- <u>Feature extraction</u>: CNNs are capable of automatically extracting features from the images, reducing the need for manual feature engineering. In the case of Fashion MNIST, the images are 28x28 grayscale images, and a CNN can learn to extract meaningful features from these images that can be used to make predictions.
- <u>Efficient training</u>: CNNs have a well-defined architecture, making them relatively easy to train. They also have fewer parameters compared to fully connected networks, which makes the training process more efficient.

2. **SVM (support vector machine) model**: I have chosen this model for these reasons:

- <u>Linear seperablity</u>: SVMs are well-suited for classification problems where the classes are linearly separable. In some cases, the Fashion MNIST dataset may exhibit linear separability, and an SVM can perform well in this scenario.
- <u>Computational efficiency</u>: SVMs are relatively efficient to train, especially for smaller datasets. This can be an advantage if you have limited computational resources or if you need to train the model quickly.
- <u>Handle high-dimensional data</u>: SVMs can handle high-dimensional data, such as images, without difficulty. In the case of Fashion MNIST, the images are 28x28 grayscale images, which can be represented as 784-dimensional vectors.

Technical Details:

1. Knn model (baseLine):

Used parameters:

I. <u>“k” parameter</u>: This is the number of nearest neighbors used for prediction. A smaller value of k means that the prediction is based on fewer neighbors and is therefore more sensitive to outliers, while a larger value of k means that the prediction is based on more neighbors and is therefore less sensitive to outliers.

II. <u>Distance Metric</u>: This is the metric used to calculate the distance between instances. Common distance metrics include Euclidean distance, Manhattan distance.

In this model, it was suggested for those combinations of k and distance metric respectively: (1,”manhattan”), (3,”manhattan”), (1,”euciladean”), (3,”euciladean”). I have chosen (1,”euciladean”) depending on the accuracy of each combination. The accuracy of each combination are shown in appendix 1.1 and 1.2.

2. CNN model:

Used parameters:

I. Number of Convolutional Layers: This determines the depth of the network and the number of feature maps produced. I used 2 Dense layers. In a dense layer of a cnn, the "units" hyper parameter represents the number of neurons or nodes in the layer. It determines the size of the output produced by the layer, and is one of the most important hyper parameters for controlling the capacity or expressiveness of the network. I sat it to 10 (according to how much the model outputs are (10 items)). Both layers are 128 and 10 units respectively.

II. Activation Function: The activation function is another hyper parameter in a dense layer. It is applied element-wise to the output of each neuron, and determines the mapping from the inputs to the outputs of the layer. I used reLU function. The importance of ReLU lies in its ability to introduce non-linearity into the network, which is crucial for solving complex problems. ReLU is also computationally efficient and does not require any complex mathematical operations.

III. Optimizer: The optimizer updates the weights of the model based on the gradients of the loss with respect to the weights, and the learning rate specified by the user, during each training iteration. By choosing an appropriate optimizer, you can significantly impact the speed and quality of the training process. I used “adam” algorithm because it is computationally efficient, has little memory requirement, and is invariant to diagonal rescale of the gradients. It has been found to work well on a wide range of problems.

IV. Sparse Categorical Crossentropy loss: It is used to measure the dissimilarity between the true labels and the predicted probabilities. The logits are the output of the last layer of the neural network before the activation function is applied. When "from_logits=True", the loss function will apply a softmax activation function to the logits, converting them into predicted probabilities, and then compute the cross-entropy loss between the predicted probabilities and the true labels.

V. Accuracy: In my code, the sparse categorical crossentropy loss function is specified as the loss function in the model's compile function, along with the "Adam" optimizer and the "accuracy" evaluation metric. By selecting the appropriate loss function, I can influence the optimization process and the quality of the model's predictions.

VI. Training parameter: epochs, determines the number of times the entire training dataset is passed through the model during training. In other words, an epoch is a single forward and backward pass of all the training samples. I sat it to 10. This means that the model will be trained for 10 complete passes through the training data.

3. SVM model:

Used parameters:

I. Kernel: "linear", this is the type of kernel function to be used in the SVM algorithm. The "linear" option means that the SVM will use a linear kernel, which is suitable for linearly separable data.
II. There are many other hyperparameters that can be passed to the SVM algorithm, such as C, gamma, degree, etc. However, they are not specified in the code, so their default values will be used.

Experiments and Results:

1. Knn (Base Line):

Results of the model are shown in appendix 2.1 and 2.2.

Discussion: the base Line here is used to compare between other models, as we consider this model to be the best one.

Having an accuracy of 100% on the training data means that the model is able to correctly classify or predict the output for all the examples in the training set. However, having a high accuracy on the training data does not always indicate that the model will perform well on new, unseen data. This is because the model may have overfitted to the training data and has learned the noise or random fluctuations in the training data, rather than the underlying patterns that determine the relationship between the inputs and outputs. This is what we saw in other accuracies (they are not 100%).

Confusion matrixes: If there is just one rectangle in the top left corner, it means that the model is classifying all instances into one specific class and all of those predictions are correct. This might be good in some cases, but it's also possible that the model is overfitting and has not learned to generalize well to new data.

2. SVM:

First results of the model are shown in appendix 4.1 and 4.2. Noticed that run time had taken the longest of the other models.

Discussion: The results show the accuracies compared to baseline are acceptable in general. The confusion matrixes shows variant square, the radius shows the correctly guessed data, the color indicates the number of each square. The missed data's' colors show the few outliers. In general, the results are acceptable.

The appendix 4.3 shows 20 random samples chosen from incorrectly predicted data.

Discussion: it should be a common pattern between these data or most of them. I see that most of them are look like T-shirts.

Comment: This could be for reasons such as:

1. Overfitting: The model may have memorized the training data and is not generalizing well to the test data. This could cause the model to perform poorly on new, unseen data, including misclassifying t-shirts.
2. Lack of features: The SVM algorithm is highly dependent on the features that are used for training. If the features do not adequately represent the underlying patterns in the data, the model will not perform well. In the case of the Fashion MNIST dataset, the images may need to be transformed into features that are more meaningful before training the SVM model. This is what have been done in my code.

3. CNN:

First results of the model are shown in appendix 3.1.

Discussion: I noticed that each time I run the code, the numbers of accuracy gets better, also the number of loss reduces. As shown in appendix 3.1. In the left side, the accuracies are considered bad, but in the right side, the accuracies are somewhat acceptable. However, compared to the baseline it should perform better and that what has been done in next results.

Second results after getting the common feature of 20 random missed data as shown in appendix 3.2

Discussion: The random data has in common that almost the entire set shown are boots!

Comment: this is because of the following reasons:

1. Lack of diversity in the Boot images: The dataset may not contain enough diversity in the images of boots, leading the model to not learn the full range of variations for this category.
2. Overfitting: The model may be overfitting to the training data, leading to poor generalization performance on the test data. This can be due to the model having too many parameters, or training for too many epochs.

To improve the model's performance on boots, it can be tried:

1. Increase the size of the dataset with additional boot images or by using data augmentation techniques. However, this cannot be done by me but the provider.
2. Increase the model's capacity by adding more layers or filters, or using architectures that are more advanced. This was applied to my model which adds a new layer of a common feature and was applied it to testing data only (shorten the time).

It should be mentioned that Fashion MNIST dataset is a relatively simple dataset, and more complex and realistic datasets may have different challenges.

Third results after taking a feature of the missed data as shown in appendix 3.3

Discussion: As shown, the testing data accuracy was improved from 0.88 to 0.94, and also loss was reduced from 0.46 to 0.16. The improvement is obvious.

Comment: the improvement done after making the evaluation of the algorithm on different feature, and this feature is taken from the common feature between boots as shown before.

Forth results after taking 20 incorrected predictions as shown in appendix 3.4

Discussion: As shown, the incorrected predictions data do not contains boots. This is a sign for big improvement.

Conclusions and Discussions:

After dealing with each of the used models, it should be said that the best model to deal with mnist data is CNN model because of its high accuracy and self-improvements, even if I didn't enter a new feature layer to deal with. The least run time model is knn model which is the baseLine. Evaluation metrics are chosen different from model to another depending on the best to lead to the right decisions.

For knn model, even of its high accuracy and efficient confusion matrix which are both good metrics, but its limitations are as follows:

1. Sensitivity to Outliers: KNN with k=1 is highly sensitive to outliers. A single outlier can greatly affect the prediction of the nearest neighbor, leading to incorrect classifications.
2. The KNN algorithm becomes less effective as the number of dimensions increases (curse of dimensionality). This is because the Euclidean distance between two points increases as the number of dimensions increases, making it harder to find close neighbors.

For SVM model, both accuracy and confusion matrix are good, but the better one for svm is confusion matrix because it provides a more detailed view of the model's performance by breaking down the predictions into different categories. SVM limitations are as follows:

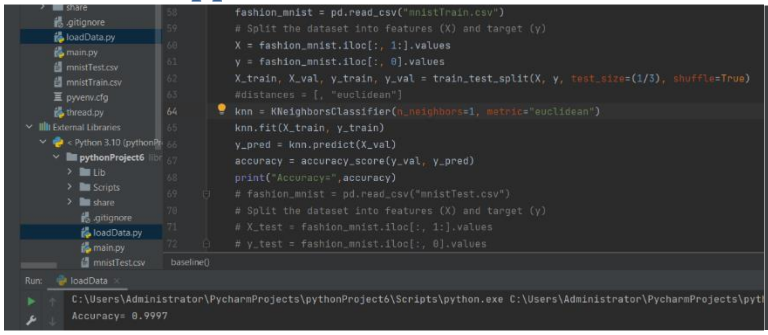
1. Computational Complexity: especially for large datasets. The time complexity of the training phase is typically O(n^3), where n is the number of samples in the training set. This can make SVMs infeasible for very large datasets. This explains why it has taken many time.
2. Overfitting: especially when the number of features is large compared to the number of samples.
3. Difficulty in Handling Multi-Class Problems: SVMs are designed for binary classification problems, but I liked to try it in my code.

For CNN, accuracy and loss metrics are the best metric because it do not need confusion matrix since this model has achieved the best performance. However, its limitations are as follows:

1. Computational Complexity: This can make it difficult to use CNNs in real-time applications, but it still better than SVM in this point.
2. Overfitting: especially when there is a small amount of training data compared to the number of parameters in the model.
3. High Sensitivity to Hyperparameters: Finding the optimal hyperparameters can be a time-consuming process, and even small changes in hyperparameters can have a big impact on performance.

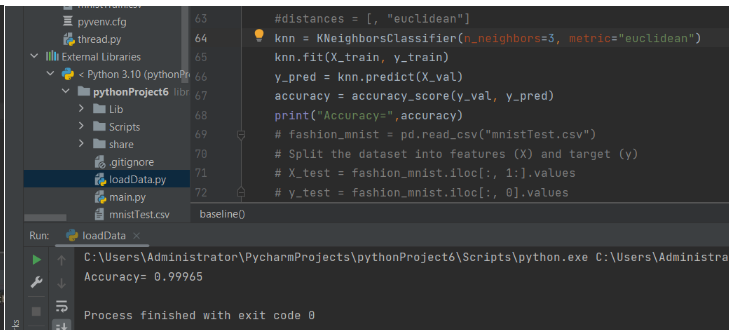
Future view: I hope to make models like the tested ones better in performing data.

Appendix:



```
fashion_mnist = pd.read_csv("mnistTrain.csv")
# Split the dataset into features (X) and target (y)
X = fashion_mnist.iloc[:, 1:].values
y = fashion_mnist.iloc[:, 0].values
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=(1/3), shuffle=True)
#distances = [, "euclidean"]
knn = KNeighborsClassifier(n_neighbors=1, metric="euclidean")
knn.fit(X_train, y_train)
y_pred = knn.predict(X_val)
accuracy = accuracy_score(y_val, y_pred)
print("Accuracy=",accuracy)
# fashion_mnist = pd.read_csv("mnistTest.csv")
# Split the dataset into features (X) and target (y)
# X_test = fashion_mnist.iloc[:, 1:].values
# y_test = fashion_mnist.iloc[:, 0].values
```

```
C:\Users\Administrator\PycharmProjects\pythonProject6\Scripts\python.exe C:\Users\Administrator\PycharmProjects\pyt
Accuracy= 0.9997
```

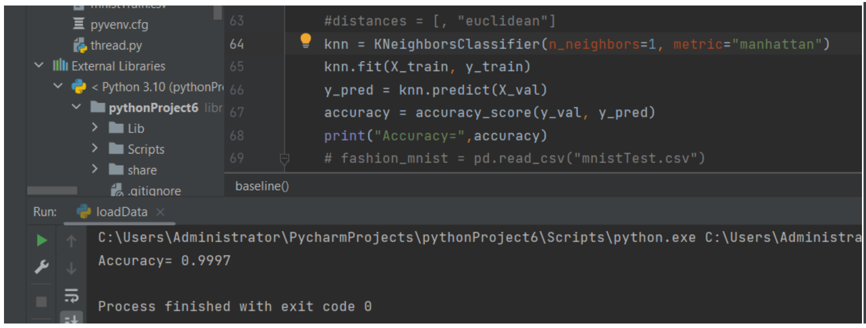


```
#distances = [, "euclidean"]
knn = KNeighborsClassifier(n_neighbors=3, metric="euclidean")
knn.fit(X_train, y_train)
y_pred = knn.predict(X_val)
accuracy = accuracy_score(y_val, y_pred)
print("Accuracy=",accuracy)
# fashion_mnist = pd.read_csv("mnistTest.csv")
# Split the dataset into features (X) and target (y)
# X_test = fashion_mnist.iloc[:, 1:].values
# y_test = fashion_mnist.iloc[:, 0].values
```

```
C:\Users\Administrator\PycharmProjects\pythonProject6\Scripts\python.exe C:\Users\Administra
Accuracy= 0.99965

Process finished with exit code 0
```

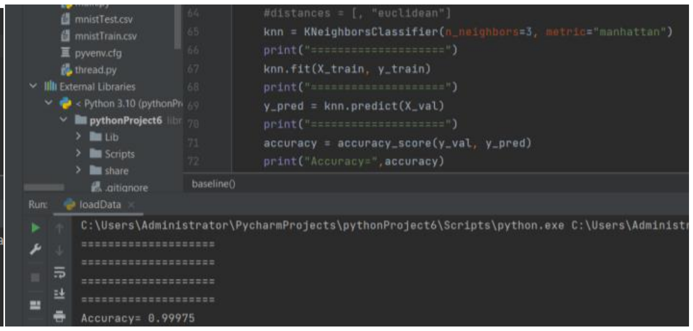
Appendix 1.1: Accuracy for euclidean distance with k=1 and k=3



```
#distances = [, "euclidean"]
knn = KNeighborsClassifier(n_neighbors=1, metric="manhattan")
knn.fit(X_train, y_train)
y_pred = knn.predict(X_val)
accuracy = accuracy_score(y_val, y_pred)
print("Accuracy=",accuracy)
# fashion_mnist = pd.read_csv("mnistTest.csv")
```

```
C:\Users\Administrator\PycharmProjects\pythonProject6\Scripts\python.exe C:\Users\Administra
Accuracy= 0.9997

Process finished with exit code 0
```

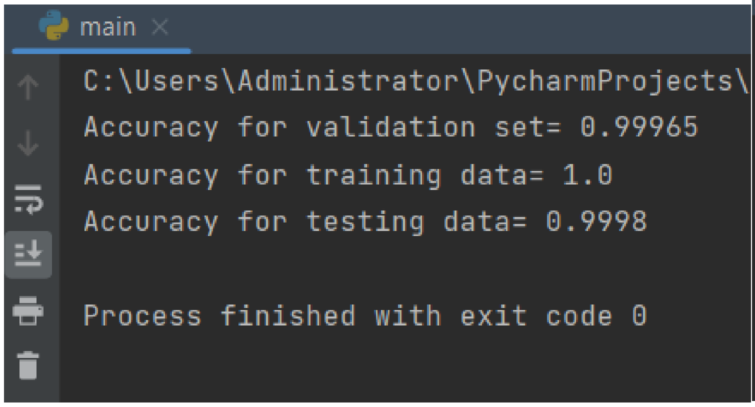


```
#distances = [, "euclidean"]
knn = KNeighborsClassifier(n_neighbors=3, metric="manhattan")
print("====================")
knn.fit(X_train, y_train)
print("====================")
y_pred = knn.predict(X_val)
print("====================")
accuracy = accuracy_score(y_val, y_pred)
print("Accuracy=",accuracy)
```

```
C:\Users\Administrator\PycharmProjects\pythonProject6\Scripts\python.exe C:\Users\Administr
====================
====================
====================
====================
Accuracy= 0.99975
```

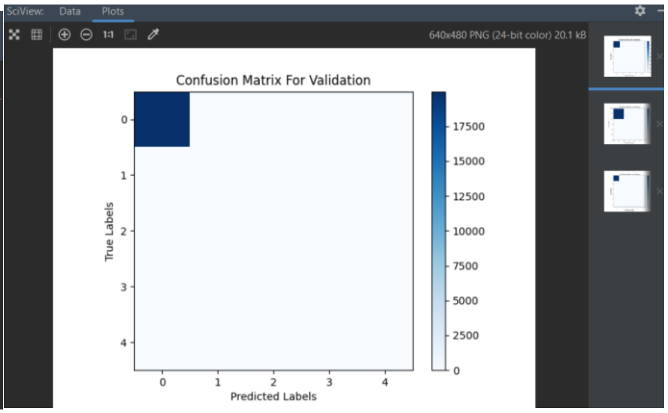
Appendix 1.2: Accuracy for manhattan distance with k=1 and k=3

- **Results:**

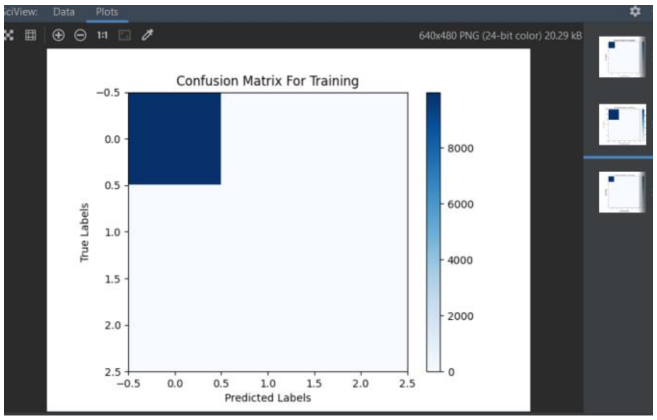


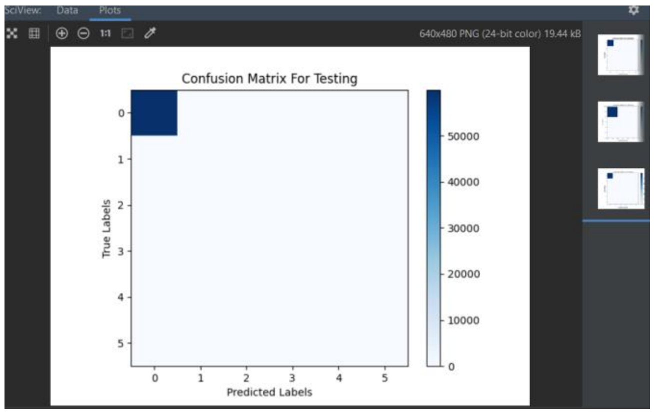
```
C:\Users\Administrator\PycharmProjects\
Accuracy for validation set= 0.99965
Accuracy for training data= 1.0
Accuracy for testing data= 0.9998

Process finished with exit code 0
```

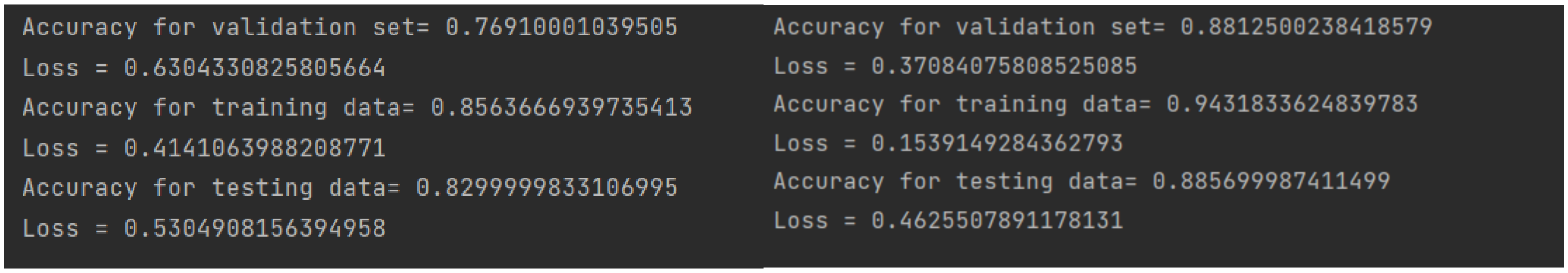


Appendix 2.1 : Results for knn baseline model





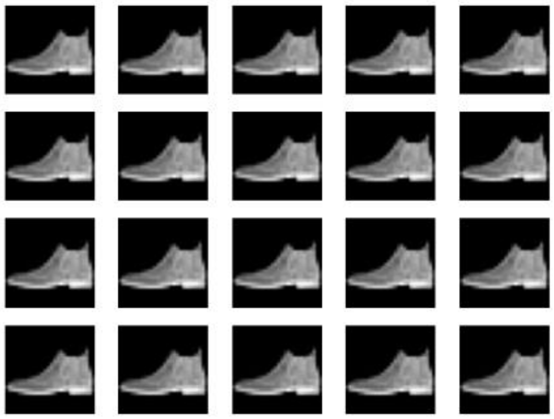
Appendix 2.2 Results for knn baseline model



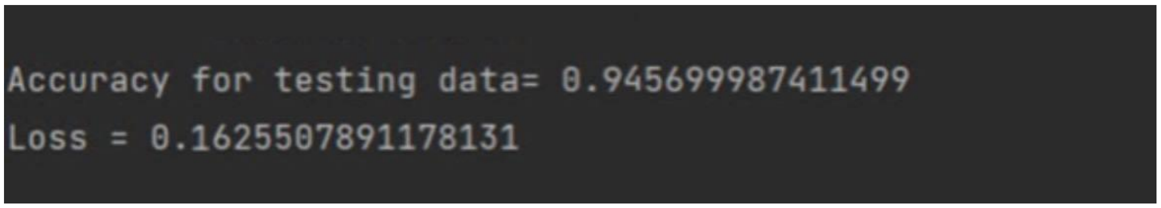
```
Accuracy for validation set= 0.76910001039505
Loss = 0.6304330825805664
Accuracy for training data= 0.8563666939735413
Loss = 0.4141063988208771
Accuracy for testing data= 0.8299999833106995
Loss = 0.5304908156394958
```

```
Accuracy for validation set= 0.8812500238418579
Loss = 0.37084075808525085
Accuracy for training data= 0.9431833624839783
Loss = 0.1539149284362793
Accuracy for testing data= 0.885699987411499
Loss = 0.4625507891178131
```

Appendix 3.1: Results for CNN runned two times



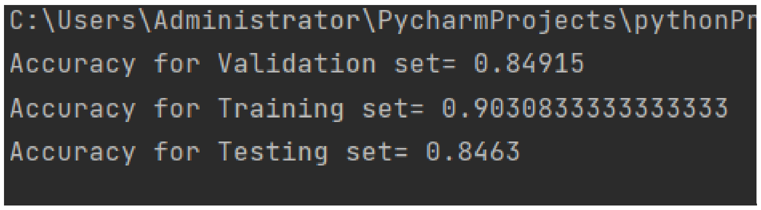
Appendix 3.2 (20) random incorrectly predicted samples



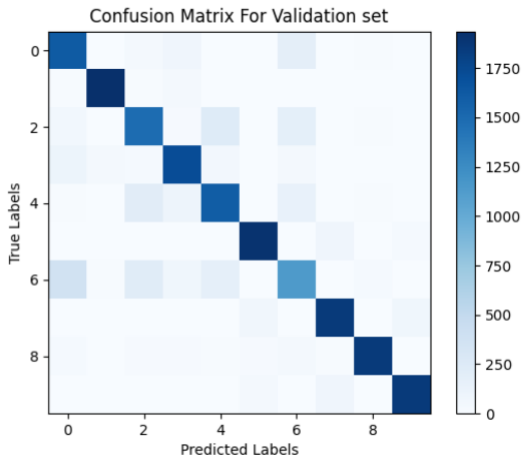
```
Accuracy for testing data= 0.945699987411499
Loss = 0.1625507891178131
```

Appendix 3.3 New Accuracy after improving CNN

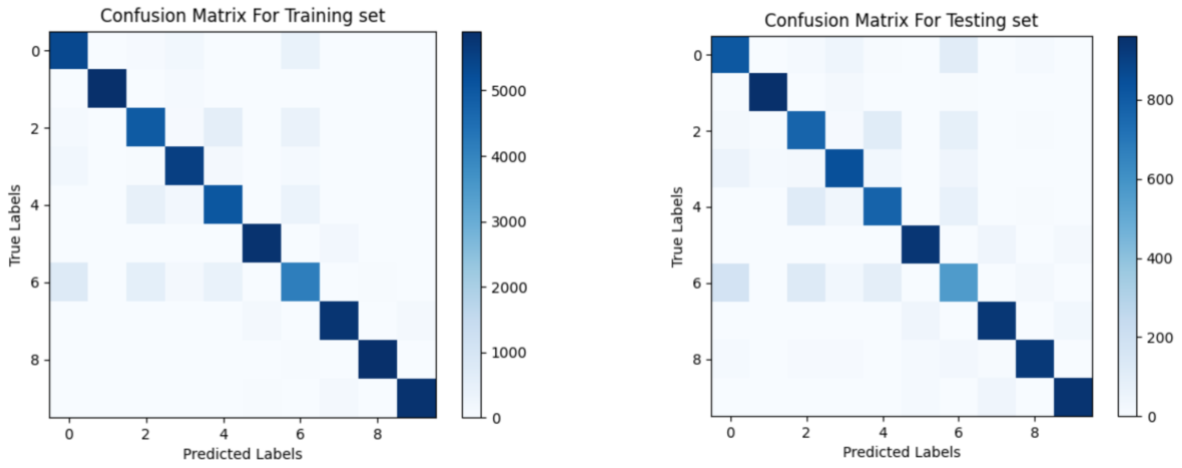
Appendix 3.4 (20) samples output after improving



```
C:\Users\Administrator\PycharmProjects\pythonPr
Accuracy for Validation set= 0.84915
Accuracy for Training set= 0.9030833333333333
Accuracy for Testing set= 0.8463
```



Appendix 4.1 Results for SVM



Appendix 4.2 Results for SVM

Appendix 4.3 (20) random incorrectly predicted samples for SVM