**Comparative Analysis of Machine Learning, Deep Learning and Convolutional Neural Network on MNIST Data**

**Problem Definition**
The purpose of this project is to apply and evaluate different traditional machine learning algorithms, neural network with multiple layers and convolutional neural network using MNIST Dataset. The problem this project addresses is to determine which model is most effective for this dataset considering the accuracy and other metrics score. This project also analyzes the computational resources and time taken for each model training. In addition, it also points how to increase the performance of the models.

**Objectives**
- To apply and evaluate different machine learning algorithms and combine the result of all through voting to improve the results.
- To implement and train deep neural network with multiple hidden layers.
- To analyze specialized deep learning algorithm named Convolutional Neural Network (CNN) for image data.
- To compare these models based on their training time, metrics score such as accuracy, macro-f1.

**Data Description**
MNIST is an image dataset loaded from Keras by default containing 60,000 train images and 10,000 test images. These images are in grayscale with pixel values between 0 and 255. The labels are digits between 0 and 9. In the preprocessing step, the values are normalized to be on the same scale and faster convergence. Data augmentation is applied by rotation of images of all original images (50,000) and the rotated images added to original dataset making it total 100,000 images for training. Additionally, data is further split to validation 10000 records from training data for parameter tuning. For machine learning and MLP (Multi layers perceptron) input shape compatibility, each image is flattened to 1D array.

**Model Description**
In this project mainly three algorithms from Machine learning are used. These include Decision Tree, Random Forest, KNN (K-nearest neighbors). For Deep learning, a multi-layer perceptron and convolutional neural network are used.

**1). Machine Learning Models**
To analyze the performance of machine learning models on image dataset, random forest, decision tree and KNN are applied.

**1a) Decision Tree**

Decision Tree splits the data based on the feature value to create the tree like structure for decision. It is used as baseline to see the performance of the model. It splits the data on the basis of Gini-index, entropy. The purer node is selected as head node. The Decision tree is fast to train and easy to implement but cannot capture the pixel relationship.

**1b). Random Forest**

Random forest combines multiple trees and then predict the output by voting from each tree. The reason to choose Random Forest is it works better on large and tabular data and produce better result by combining the output class prediction from all trees instead of a single tree. However, it is slower to train if number of trees are more and it also lacks spatial understanding of image data.

**1c) K-Nearest Neighbors (KNN)**

KNN classifies a sample based on the majority label of its k closest neighbors. It can perform well on clean and separated data but sensitive to noise.

In the end Voting Classifier of the three machine learning models is used to improve final result. This will collectively vote for the label and based on the majority, the output is predicted. Which leads to better performance compare to individual machine learning model.

**2. Deep Learning (MLP)**

The second algorithm used is a multi-layer deep neural network. For this project, two hidden layers with 256 and 128 nodes respectively are used. 1st hidden layer will capture edges and 2nd hidden layer will capture high level feature like shapes. Learning rate 0.001 is used as default for faster convergence and stability, the more increase in the learning rate will make the model to oscillate and overshoot while the lower value makes the model training slow. weight decay 0.0001 is used to regularize the weight to reduce overfitting, if higher value used, model can't learn and may underfit while for larger values, the model can't prevent overfitting. Batch size of 64 is used to balance training speed and model generalization. This size provides stable gradient updates. Additionally, we used Relu (Rectified Linear Unit) activation function which avoid vanishing gradient and fast training. In the end, to predict the class, we used SoftMax which will assign each class probability. Final layer has 10 nodes which is our class labels. Adam is used as optimizer for faster convergence, Spare categorical cross-entropy is the loss function suited for multi classification and does not require one-hot encoding. Early stopping and model checkpoints are used to see if the model is overfitting and did not improve, then it will

save the best model weights. It can learn non-linear boundaries by hidden layers and activation functions. However, it also lacks spatial relationship with nearby pixels.

### 3). Convolutional Neural Network (CNN)

The final algorithm used is convolutional neural network. Learning rate with other batch size and weight decay are set similar to MLP. CNN captures the patterns in image using different convolutional filters. Here we used 128 filters of size 3 x 3 in first layer and 64 filters with 3 x 3 size. After convolution, max pooling is applied with 2 x 2 matrix and then flatten with one other hidden layer. It is also robust to translation, rotation as it captures the shape easily using filters. However, it requires more computational resources.

### Experimental Environment

Platform and major libraries used in the project are given below.

| Name | Specification / Version |
|------|-------------------------|
| Operating System | Window 10 |
| Python | 3.13.5 |
| Tensorflow | 2.20.0-rc0 |
| scikit-learn | 1.6.1 |
| NumPy | 2.1.3 |
| Matplotlib | 3.10.0 |
| torchsummary | 1.5.1 |
| torch | 2.8.0 |
| torchvision | 0.23.0 |
| Hardware | Intel Core i5-6200 @ 2.30 GHz, RAM 12 GB, CUDA/GPU=0, |

*Table 1 Experimental Environment*

### Results and Analysis

The Decision Tree perform poorly because they treat each pixel as an independent feature, without considering the spatial context. This is the reason their performance is limited. However Random Forest has the advantage of multiple tree classification result and perform somehow better instead of individual tree. KNN also can calculate and assign the nearest pixel class which help in better result. Finally, The Voting Classifier showed that combining the strengths of different traditional models can improve performance.
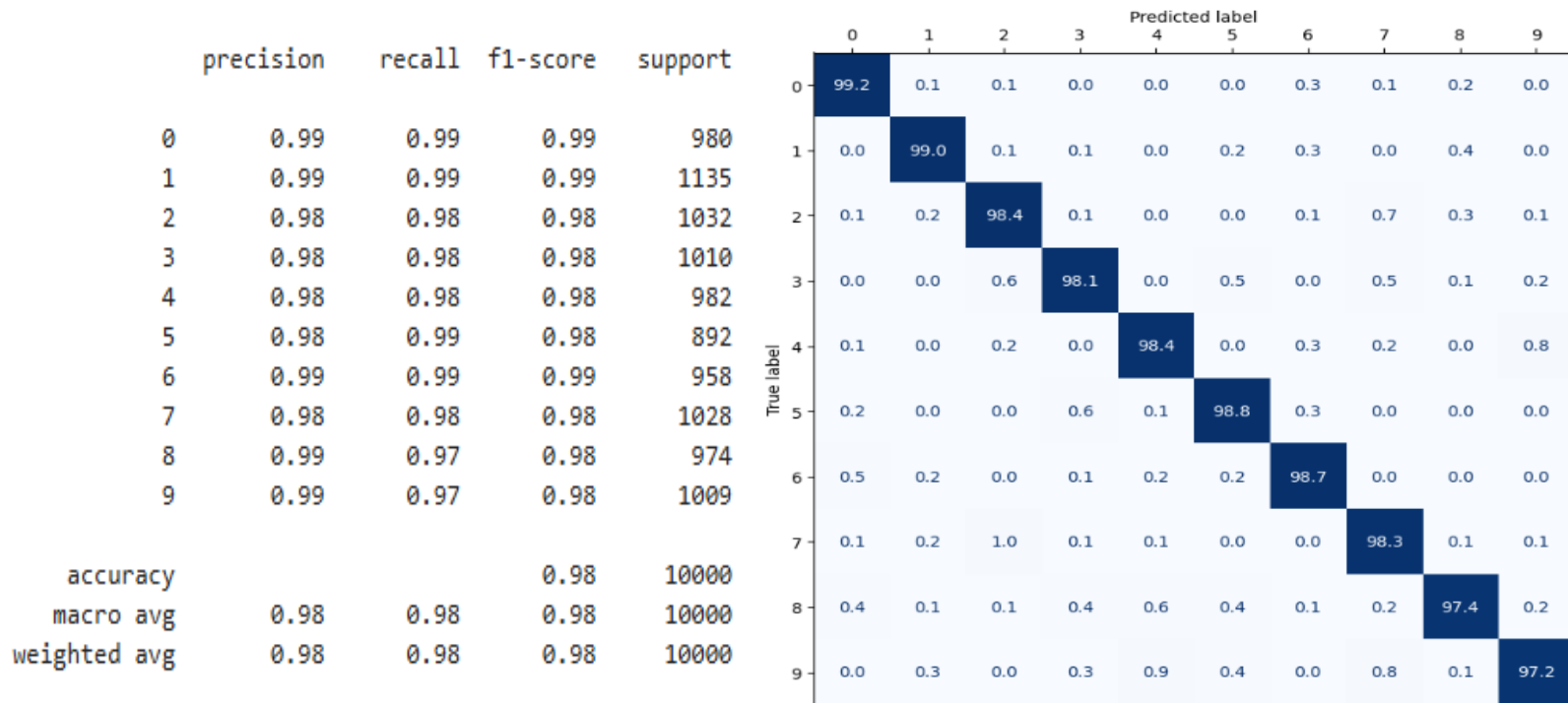
**Figure 1 Voting Classifier Metrics Score and Confusion Matrix**

|   | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.98 | 0.99 | 0.99 | 980 |
| 1 | 0.98 | 1.00 | 0.99 | 1135 |
| 2 | 0.99 | 0.98 | 0.98 | 1032 |
| 3 | 0.97 | 0.97 | 0.97 | 1010 |
| 4 | 0.98 | 0.97 | 0.98 | 982 |
| 5 | 0.98 | 0.97 | 0.97 | 892 |
| 6 | 0.98 | 0.99 | 0.98 | 958 |
| 7 | 0.96 | 0.97 | 0.97 | 1028 |
| 8 | 0.99 | 0.95 | 0.97 | 974 |
| 9 | 0.96 | 0.96 | 0.96 | 1009 |
| accuracy |  |  | 0.98 | 10000 |
| macro avg | 0.98 | 0.98 | 0.98 | 10000 |
| weighted avg | 0.98 | 0.98 | 0.98 | 10000 |

Predicted label

|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 99.5 | 0.2 | 0.0 | 0.0 | 0.0 | 0.1 | 0.1 | 0.1 | 0.0 | 0.0 |
| 1 | 0.0 | 99.6 | 0.2 | 0.2 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 |
| 2 | 0.6 | 0.1 | 97.7 | 0.1 | 0.1 | 0.0 | 0.0 | 1.4 | 0.1 | 0.0 |
| 3 | 0.0 | 0.0 | 0.1 | 97.2 | 0.0 | 0.6 | 0.0 | 0.7 | 0.9 | 0.5 |
| 4 | 0.1 | 0.1 | 0.0 | 0.0 | 97.5 | 0.0 | 0.3 | 0.3 | 0.0 | 1.7 |
| 5 | 0.3 | 0.0 | 0.0 | 1.0 | 0.1 | 96.7 | 1.1 | 0.1 | 0.1 | 0.4 |
| 6 | 0.5 | 0.3 | 0.0 | 0.0 | 0.2 | 0.2 | 98.6 | 0.0 | 0.1 | 0.0 |
| 7 | 0.0 | 1.1 | 0.7 | 0.0 | 0.1 | 0.0 | 0.0 | 97.4 | 0.0 | 0.8 |
| 8 | 0.1 | 0.2 | 0.4 | 1.5 | 0.8 | 0.6 | 0.1 | 0.2 | 95.4 | 0.6 |
| 9 | 0.3 | 0.5 | 0.1 | 0.6 | 0.8 | 0.5 | 0.1 | 1.0 | 0.2 | 95.9 |

From above Figure 1, we can see the model did well by obtaining accuracy and macro-f1 (0.98). Macro-f1 shows how the model peform for imbalanced dataset, however in our case the MNIST data is balanced, so It is similar to accuracy. For "0" digit High Precision (0.98): the model predicts "0", it is correct 98% of the time. Very few false positives (other digits mistakenly predicted as 0). Recall (0.99): Out of all the true "0"s in the dataset, the model only catches 99%. some number of "0"s were missed (classified as another digit). F1-score is balance of precision and recall obtained by harmonic mean of precision and recall. It means the model did not confused with predicted 0 as other digit 99% time. However digit 9 is consfused with other digits (like 7) reducing the f1-score to 0.96. From confusion matrix, we see that model predicted "0" correctly compare to other digits and the most misclassified digits for voting classifier are "8" which is confused with digit "3" and "9" which is predicted digit"7" most of them.

The second algorithm, MLP is more accurate because its hidden layers can learn non-linear combinations of features, making it able to capture the underlying patterns in the data. However, it still processes the image as a flattened array, which means it loses the pixel relationship between adjacent pixels and also make the model parameter heavy. Like, a small shift in the digit's position can change the input vector leading to misclassification.

## Figure 2 MLP Metrics Score and Confusion Matrix

|  | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.99 | 0.99 | 0.99 | 980 |
| 1 | 0.99 | 0.99 | 0.99 | 1135 |
| 2 | 0.98 | 0.98 | 0.98 | 1032 |
| 3 | 0.98 | 0.98 | 0.98 | 1010 |
| 4 | 0.98 | 0.98 | 0.98 | 982 |
| 5 | 0.98 | 0.99 | 0.98 | 892 |
| 6 | 0.99 | 0.99 | 0.99 | 958 |
| 7 | 0.98 | 0.98 | 0.98 | 1028 |
| 8 | 0.99 | 0.97 | 0.98 | 974 |
| 9 | 0.99 | 0.97 | 0.98 | 1009 |
| accuracy |  |  | 0.98 | 10000 |
| macro avg | 0.98 | 0.98 | 0.98 | 10000 |
| weighted avg | 0.98 | 0.98 | 0.98 | 10000 |

Confusion Matrix (Predicted label columns 0–9, True label rows 0–9):

| True \ Pred | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 99.2 | 0.1 | 0.1 | 0.0 | 0.0 | 0.0 | 0.3 | 0.1 | 0.2 | 0.0 |
| 1 | 0.0 | 99.0 | 0.1 | 0.1 | 0.0 | 0.2 | 0.3 | 0.0 | 0.4 | 0.0 |
| 2 | 0.1 | 0.2 | 98.4 | 0.1 | 0.0 | 0.0 | 0.1 | 0.7 | 0.3 | 0.1 |
| 3 | 0.0 | 0.0 | 0.6 | 98.1 | 0.0 | 0.5 | 0.0 | 0.5 | 0.1 | 0.2 |
| 4 | 0.1 | 0.0 | 0.2 | 0.0 | 98.4 | 0.0 | 0.3 | 0.2 | 0.0 | 0.8 |
| 5 | 0.2 | 0.0 | 0.0 | 0.6 | 0.1 | 98.8 | 0.3 | 0.0 | 0.0 | 0.0 |
| 6 | 0.5 | 0.2 | 0.0 | 0.1 | 0.2 | 0.2 | 98.7 | 0.0 | 0.0 | 0.0 |
| 7 | 0.1 | 0.2 | 1.0 | 0.1 | 0.1 | 0.0 | 0.0 | 98.3 | 0.1 | 0.1 |
| 8 | 0.4 | 0.1 | 0.1 | 0.4 | 0.6 | 0.4 | 0.1 | 0.2 | 97.4 | 0.2 |
| 9 | 0.0 | 0.3 | 0.0 | 0.3 | 0.9 | 0.4 | 0.0 | 0.8 | 0.1 | 97.2 |

In **Figure 2**, we can see the MLP able to predict 0 correct most of the times. Compare to Machine Learning models, it also did better on digit "9" where the f1 score is increased from 0.96(Machine learning model) to 0.98(MLP).

The final model is CNN, the high performance of CNN is due to its use of convolutional layers, which apply filters to small sections of the image. This method allows the network to learn local features, such as edges and curves, and then combine them to recognize more complex patterns. This approach makes CNNs translation independent, where it can correctly classify a digit even if its position in the image is slightly shifted. The data augmentation technique further improves the CNN's ability to generalize by exposing it to a variety of rotated images during training, which makes it more robust to variations in image. Here is the for comparison of each model.

**Figure 3 CNN Metrics scores and Confusion Matrix**

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.99 | 0.99 | 0.99 | 980 |
| 1 | 0.99 | 1.00 | 1.00 | 1135 |
| 2 | 0.99 | 0.99 | 0.99 | 1032 |
| 3 | 1.00 | 0.99 | 0.99 | 1010 |
| 4 | 0.99 | 0.99 | 0.99 | 982 |
| 5 | 0.98 | 0.99 | 0.99 | 892 |
| 6 | 1.00 | 0.99 | 0.99 | 958 |
| 7 | 0.99 | 0.99 | 0.99 | 1028 |
| 8 | 0.99 | 0.99 | 0.99 | 974 |
| 9 | 0.99 | 0.98 | 0.99 | 1009 |
| | | | | |
| accuracy | | | 0.99 | 10000 |
| macro avg | 0.99 | 0.99 | 0.99 | 10000 |
| weighted avg | 0.99 | 0.99 | 0.99 | 10000 |

Predicted label

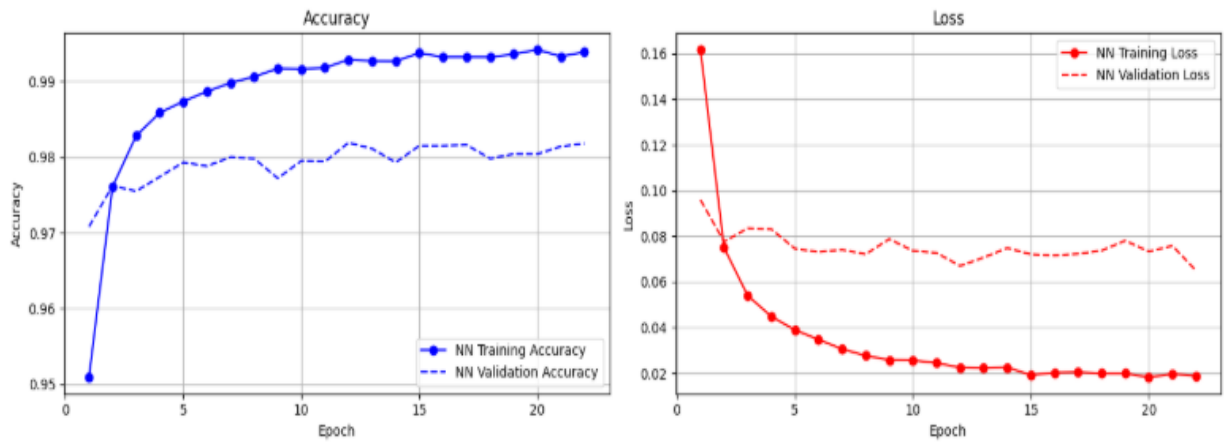| True label | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 99.5 | 0.1 | 0.1 | 0.0 | 0.0 | 0.0 | 0.1 | 0.1 | 0.1 | 0.0 |
| 1 | 0.0 | 99.9 | 0.0 | 0.0 | 0.0 | 0.1 | 0.0 | 0.0 | 0.0 | 0.0 |
| 2 | 0.1 | 0.0 | 99.1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.8 | 0.0 | 0.0 |
| 3 | 0.0 | 0.0 | 0.2 | 99.2 | 0.0 | 0.4 | 0.0 | 0.1 | 0.1 | 0.0 |
| 4 | 0.1 | 0.0 | 0.0 | 0.0 | 99.5 | 0.0 | 0.0 | 0.1 | 0.0 | 0.3 |
| 5 | 0.0 | 0.0 | 0.0 | 0.2 | 0.0 | 99.4 | 0.2 | 0.0 | 0.1 | 0.0 |
| 6 | 0.3 | 0.2 | 0.0 | 0.1 | 0.1 | 0.5 | 98.6 | 0.0 | 0.1 | 0.0 |
| 7 | 0.0 | 0.4 | 0.7 | 0.0 | 0.1 | 0.0 | 0.0 | 98.6 | 0.1 | 0.1 |
| 8 | 0.0 | 0.0 | 0.4 | 0.1 | 0.0 | 0.0 | 0.0 | 0.0 | 99.3 | 0.2 |
| 9 | 0.0 | 0.0 | 0.0 | 0.0 | 0.7 | 0.4 | 0.0 | 0.4 | 0.3 | 98.2 |

In above **Figure 3**, the CNN perform better compare to the voting classifier and MLP. Even the digit "9" which was misclassified by other is correctly predicted 98.2% time. This is because it can capture the shape by using different filters.
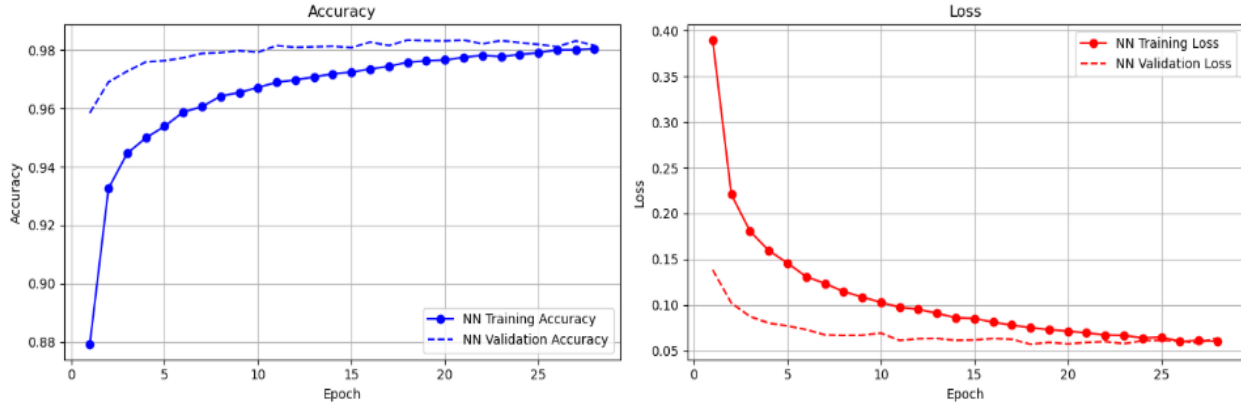
*Figure 4 NN performance by epoch*

From the NN accuracy and loss **Figure 4**, we can see that validation accuracy is higher than training accuracy across all epochs. This is because dropout is active in training which reduces training performance but improves generalization on validation. At validation, dropout is off, so the full model is evaluated and accuracy appears higher compare to training. The result without dropout is shown above. On the other hand, for CNN **Figure 5**, the training accuracy is higher because CNN can capture image shapes easily and fit the training data well due to multiple filters at training time
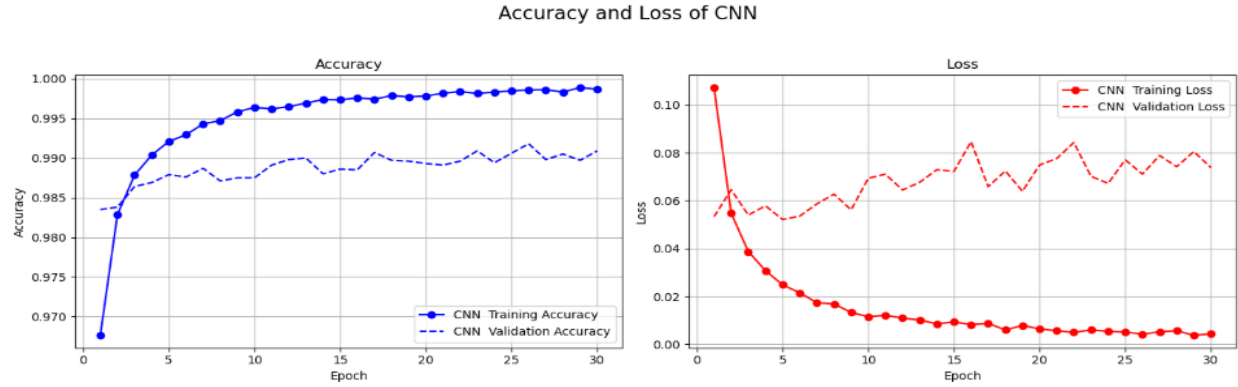
*Figure 5 CNN performance by epoch*

By looking through misclassification patterns, the MLP was more likely to confuse digits that are visually similar, such as a 5 and a 6 as shown in below **Figure 6**. Due to bottom filled circle, NN unable to predict it 5, however CNN did due to various filters.



*Figure 6 NN vs CNN Prediction*

The overall accuracy score is shown in below **Table 2**, we can see from the above results that CNN performed well on image. However, the resources consumption and training time for CNN is also very high. Considering the resources provided in **Table 1**, here is the summarized overview of the major three models.

| | Model | #Params | Training Time (min) | Test Acc % | Macro-F1 | Weighted-F1 | avg time per epoch (min) |
|---|---|---|---|---|---|---|---|
| 0 | ML | N/A | 6.72 | 97.58 | 0.98 | 0.98 | N/A |
| 1 | MLP | 235914 | 6.79 | 98.37 | 0.98 | 0.98 | 0.24 |
| 2 | CNN | 1219614 | 437.72 | 99.15 | 0.99 | 0.99 | 14.59 |

*Table 2 Models Summary (TensorFlow)*

8

Similarly, the pytorch performance is similar, however the training time is more compared to TensorFlow.

| | Model | #Params | Training Time (min) | Test Acc % | Macro-F1 | Weighted-F1 | avg time per epoch (min) |
|---|---|---|---|---|---|---|---|
| 0 | MLP | 235914 | 10.31 | 98.07 | 0.98 | 0.98 | 0.43 |
| 1 | CNN | 1632286 | 723.27 | 99.05 | 0.99 | 0.99 | 34.44 |

*Table 3 Models Summary (Pytorch)*

From the summary, we see that Macro-F1 is similar to accuracy due to balanced dataset. If macro-F1 is low while accuracy is high, it means the model is ignoring one or more minority classes. But here model performs fairly well across all classes. Further, weighted-F1 gives higher edge to majority classes, but similar for balanced dataset.

**Conclusion and Future work**

This project compared the three algorithms on MNIST Image dataset where CNN achieved the highest accuracy 99.15% followed by Multi-Layer Perceptron MLP 98.37% while machine learning voted classifier obtained 97.58%. MLP are less effective and more parameter heavy for image data but still performed better due to gradient descent optimization method. In the end, CNN takes the highest accuracy as it is most appropriate for image data due to considering the neighboring pixel and also less parametric. Further, even if the pixel shifts slightly, the CNN captures those patterns inside the image, However, in case of MLP, the output can be different as it can be affected by pixel shift unlike CNN.

We trained the model with limited number of epochs and without using pretrained models. However, in future, the results can be further improved by using more epochs, transfer learning – pre trained model, data augmentation including translation, zooming and rotation of image. Further the number of hidden layers can be increased to capture complex shapes in the image.