EUROPEAN HIGHER EDUCATION INSTITUTE

Deep Learning - Final Exam Submission

Prepared by: Çiçek Akkaya

1.Functional Jupyter Notebook workflow with the CNN adopted

Jupyter Notebook with all Python codes applied which is in the format of .ipynb file:

- https://drive.google.com/file/d/1BK-YDwSvvBSr9VnKtWuQJYn79E2p6sto/view?usp=sharing

CNN model with Keras which extracted from related Jupyter Notebook Python codes which is in the format of .keras file:

- https://drive.google.com/file/d/1PCRZmTfZfsGnnNeHy54mwjsXcdJK6lmt/view?usp=sharing

2. Report

a. Explore the data by making use of Visualisation tools.

I have used python libraries such as Numpy, Matplotlib for visualising data. All the figures in this report and more are created in the Jupyter Notebook file which is specified in the first chapter.

We can see in the figure 1 there are 10 digits in total which are 0 to 9 as handwritten digits in the Mnist dataset as a sample.

Figure 1 - Sample Handwritten Digits from Dataset

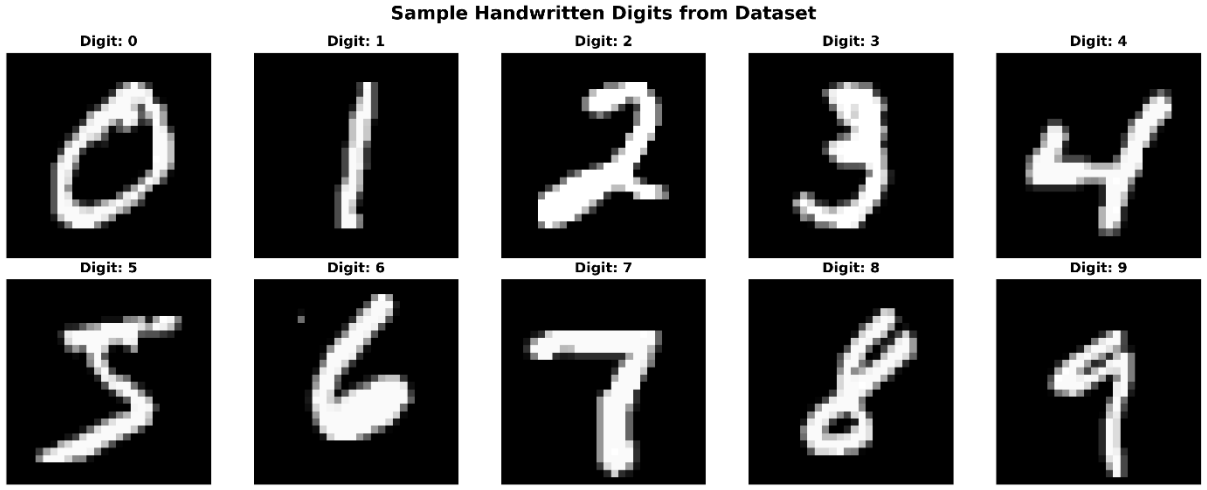


Figure 2 - Class Distribution

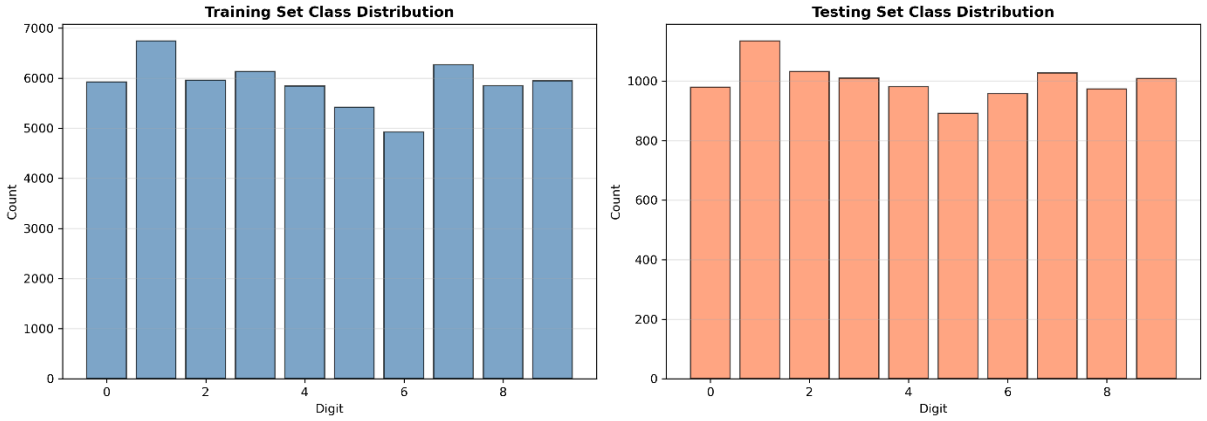


Figure 2 shows the class distributions of the training set and testing set. Therefore, we can specify how much obligation there is per digit. If we compare the obligations of the digits in the training set, the digit 1 has the largest sample size, while the digit 6 has the smallest sample size. If we compare the obligations of the digits in the testing set, the digit 1 has the largest sample size, while the digit 5 has the smallest sample size. However, in the both datasets there is no meaningful difference between the size of inner digits which means it doesn't affect the result.

In the model we used 2 separate datasets as training set and testing set. The split was made in the zip file as two datasets for the training model. Because the size of the training set and testing set were compatible for the appropriate rate. For example, in the training set digit 0 has 6000 obligations while in the testing set there are approximately 1000 obligations in it, which means there are approximately 85% of training set obligations and there are 15% of testing set obligations. Therefore, there is compatibility of training for these two separate datasets.

b. Discuss why partitioning the data into train and test is important.

Partitioning data into training and testing sets is crucial for building reliable machine learning models. The main goal of this process is to accurately estimate how well the model will perform on unseen data, which is the data it will encounter in the real world.

Why Data Partitioning is Important

The necessity of partitioning boils down to two core concepts: Generalization and Detecting Overfitting.

Generalization (Real-World Performance):

The ultimate goal of a machine learning model is generalization—the ability to make accurate predictions on new data that it wasn't trained on.

- Training Set: This set is used to *teach* the model, allowing it to learn the patterns, features, and relationships in the data. The model adjusts its weights and biases based on the errors it makes on this set.
- Test Set: This set is held back and is only used *after* the model has been trained. Since the model has never seen this data before, the metrics (like accuracy or loss) derived from the

test set provide an unbiased estimate of the model's true real-world performance. This is the only way to know if your model learned general rules or just memorized the training examples.

Detecting Overfitting and Underfitting:

- Partitioning is the primary method for diagnosing two critical problems in model training: Overfitting and underfitting.
- In overfitting, the model learns the training data too well, including noise and specific data points. A large gap between the high training score and the low test score is the unmistakable sign of overfitting.
- In underfitting, the model is too simple to capture the underlying patterns in the data. Both scores are low, indicating the model needs more complexity or features.
- Without a separate test set, you would only see the training score, which can be misleadingly high if the model is overfit. This would lead you to believe you have a perfect model when, in reality, it performs poorly on new data.

The standard practice is to randomly split the data, often using a ratio like 70/30, 80/20, or 90/10, where the larger portion is for training and the smaller portion is reserved for testing. For rigorous evaluation and hyperparameter tuning, a third set, the validation set, is often used (or cross-validation is performed) to tune hyperparameters without contaminating the final test set.

c. Give some detail regarding the inner workings of the CNN architecture adopted.

1. Feature Detectors (Convolutional Blocks)

The first three blocks act as feature detectors, systematically examining the 28 times 28 input image:

- Convolution (Conv2D): These layers use small 3 times 3 filters to scan the image, looking for simple patterns like straight lines, curves, and corners. The process is repeated, so later blocks look for combinations of those simple patterns (e.g., the loop of an '8').
    - ReLU Activation after convolution introduces non-linearity, allowing the model to detect complex relationships beyond simple lines.
- Max Pooling (MaxPooling2D): This layer shrinks the feature map by taking the highest value in a small window. This makes the model robust to small shifts or variations in the handwriting—if a digit is drawn slightly to the left, the important features are still captured.
- Batch Normalization (BatchNormalization): This stabilizes the training process by ensuring the features remain at a consistent scale, which helps the model learn faster.
- Dropout (Dropout): This is a defense against overfitting (memorization). It randomly "shuts off" neurons during training, forcing the network to rely on multiple combinations of features, not just one specific path.

2. Decision Maker (Dense Layers)

Once the features are extracted, they are prepared for the final classification:

- Flattening (Flatten): This converts the 3D feature grid into a single, long 1D list of numbers.
- Dense Layers: These fully connected layers act as the final voting system. They take the flattened features and weigh them to make a final decision.

- Softmax Output (10 units): The final layer has 10 outputs, one for each digit (0-9). The Softmax function turns these outputs into probabilities that sum up to 1. The digit with the highest probability is the network's prediction. For example, an output of [0, 0, 0.99, 0, ...] means the network is 99% certain the image is the digit '2'.

d. Visualise and analyse the results, provide a confusion matrix and other accuracy metrics that you deem fit.

In the print_detailed_report named method created in the related Jupyter Notebook file, the output has shown as figure 3 which specifies accuracy metrics for all digits and the ultimate accuracy. For all digits the accuracy of the model is changing between 99-100% which means the model has high performance for each digit. And as we can see in the output as figure 3, the ultimate accuracy of the model is 99.52% which means the model has high performance for predicting class of the digits.

Figure - Detailed Accuracy Metrics

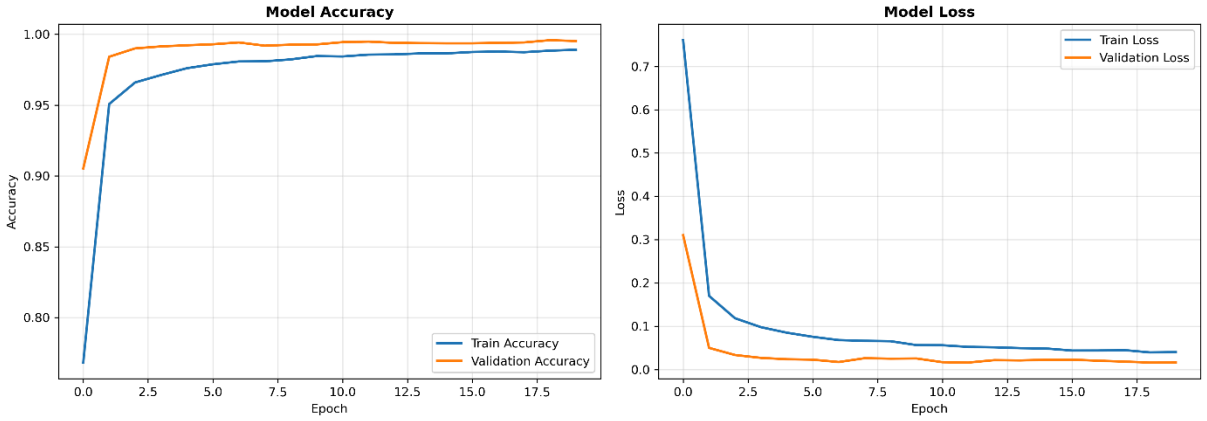
```
                                                                    ====
DETAILED CLASSIFICATION REPORT
======================================================================
              precision    recall  f1-score   support

           0       1.00      0.99      1.00       980
           1       1.00      1.00      1.00      1135
           2       1.00      1.00      1.00      1032
           3       0.99      1.00      1.00      1010
           4       1.00      0.99      1.00       982
           5       0.99      0.99      0.99       892
           6       0.99      0.99      0.99       958
           7       0.99      1.00      0.99      1028
           8       1.00      1.00      1.00       974
           9       0.99      1.00      0.99      1009

    accuracy                           1.00     10000
   macro avg       1.00      1.00      1.00     10000
weighted avg       1.00      1.00      1.00     10000


======================================================================
OVERALL METRICS
======================================================================
Accuracy:        0.9952 (99.52%)
Precision (Macro): 0.9951
Recall (Macro):    0.9952
F1-Score (Macro):  0.9952
======================================================================
```

Figure 4 - Model Accuracy & Loss



Analysis of Model Accuracy (Left Plot for fig.4)

- Initial Epochs (0 to ~5): Both Training Accuracy (blue line) and Validation Accuracy (orange line) increase rapidly. The Validation Accuracy starts higher and increases faster initially, reaching 98% very quickly. The Training Accuracy lags slightly but also increases steadily.
- Later Epochs (~5 onwards): Both accuracies continue to increase, but the rate of increase slows down significantly.
    - The Validation Accuracy plateaus are very high, around 99% or slightly below, and remain very consistent.
    - The Training Accuracy continues to climb and slowly approaches the validation accuracy, finishing at approximately 99%.
- Key Observation: The final Training Accuracy and Validation Accuracy are both very high ($\sim 99\%$). This indicates that the model is performing exceptionally well on both the training data and the unseen validation data. The minimal gap between the two suggests low variance and that the model has not significantly overfit to the training data.

Analysis of Model Loss (Right Plot for fig.4)

- Initial Epochs (0 to ~5): Both Train Loss (blue line) and Validation Loss (orange line) decrease very rapidly. The Validation Loss drops more steeply initially.
- Later Epochs (~5 onwards): Both losses continue to decrease but the rate of decrease is minimal, suggesting the optimization process is nearing convergence.
    - The Train Loss stabilizes at a very low value, approximately 0.04.
    - The Validation Loss stabilizes at an even lower value, approximately 0.03 or less, for most of the training.
- Key Observation: The final losses are extremely low. For a classification task like MNIST, low loss is desirable. The fact that the Validation Loss is lower than the Train Loss for most of the run is slightly unusual but not necessarily a cause for concern, especially when the validation accuracy is so high. It sometimes happens due to the way loss is calculated (e.g., if dropout or regularisation is used during training but not during validation/testing) or if the validation batch size/dataset is very small, but in the context of the excellent accuracy, it primarily signifies a very well-fit model with high generalization capability.

Conclusion

The results, as depicted in the plots, show a highly successful training outcome on the MNIST dataset:

1. High Performance: Both accuracy metrics approach %99 which is excellent for the MNIST task.
2. Good Generalization: The Validation Accuracy is nearly identical to the Training Accuracy, and the losses are very low and close, suggesting the model has learned the patterns well without over-memorizing the training data (low overfitting).
3. Convergence: The model appears to have converged after about 10 epochs, as the metrics change very little after that point. Further training would likely yield negligible benefit.

The model is well-trained and generalizes effectively to unseen data.

Model Accuracy Analysis (Left Plot for fig. 4)

The left plot shows the Accuracy over the training Epochs:

- Rapid Improvement: Both Train Accuracy (blue) and Validation Accuracy (orange) increase very quickly in the initial epochs (0-5).
- High Final Accuracy: Both lines plateau at a very high level, approaching %99 or slightly above, by the end of 20 epochs.
- Strong Generalization: Crucially, the Validation Accuracy is very close to the Train Accuracy throughout the training process. This minimal gap suggests the model has not significantly overfit to the training data and will perform nearly as well on unseen data (the validation set). The model demonstrates low variance.

Model Loss Analysis (Right Plot)

The right plot shows the Loss over the training Epochs:

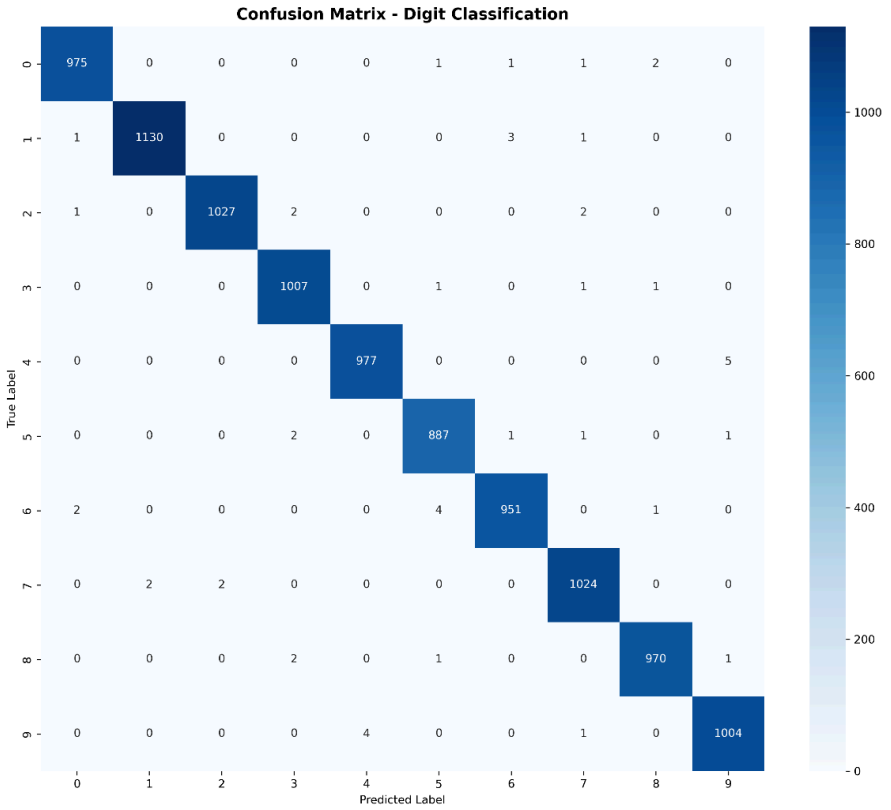
- Rapid Decrease: Both Train Loss (blue) and Validation Loss (orange) decrease dramatically in the first few epochs (0-5).
- Very Low Final Loss: Both losses stabilize at extremely low values, with the Validation Loss finishing around 0.03 and the Train Loss slightly higher at around 0.04.
- Convergence: The loss curves flatten out significantly after about 10 epochs, suggesting the model has largely converged and further training may yield minimal benefit. The very low final loss is consistent with the near-perfect accuracy observed.

Overall Conclusion

The model exhibits exceptional performance on the MNIST dataset:

1. High Performance: Achieving approximately %99 accuracy which is compatible with the figure 3 as accuracy of %99.52.
2. Well-Generalized: The closeness of the training and validation curves (both accuracy and loss) confirms the model is well-fit and has excellent generalization capability, successfully learning the underlying patterns of the digits rather than memorizing the training examples.

Figure 5 - Confusion Matrix



In the figüre 5 there is the confusion matrix which shows the distribution of correct and incorrect predictions for each digit (0 through 9).

The matrix shows excellent overall performance.

- High Accuracy: The vast majority of predictions fall along the main diagonal (where the True Label equals the Predicted Label). This indicates a very high number of correct classifications and a model with high accuracy.
- Total Samples: To estimate the total number of samples, we sum the counts in the matrix. Given the counts are generally around 970 to 1130 per class, the total test/validation set size is approximately 10,000 samples (typical for MNIST test set). The total number of correct predictions (sum of the diagonal) is 975 + 1130 + 1027 + 1007 + 977 + 887 + 951 + 1024 + 970 + 1004 = 9952. This suggests an estimated accuracy of 99.52% (9952/10000), which is exceptional.

Key Misclassifications (Model Biases): The off-diagonal entries (the errors) reveal the specific digits the model struggles to differentiate:

- 4 vs. 9: The model confused the digit 4 for 9 five times.
- 0 vs. 8: The digit 0 was mistaken for 8 twice, and 6 for 8 once, which are common errors due to similar closed loop shapes.
- 5 vs. 3/6/9: The digit 5 has few errors, but they are scattered among 3, 6, and 9.

In summary, the model is highly effective and accurate. The errors are minimal and largely involve visually similar digits, which is expected even for a high-performing classifier.