Using Neural Network Model for Handwritten Images Classification

Xiao Jun Lin

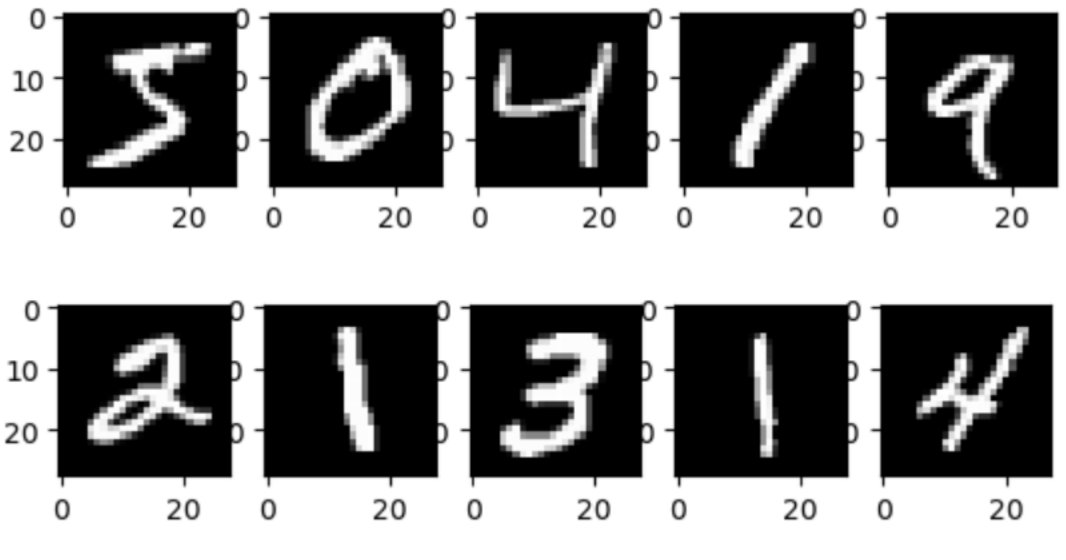
1. Introduction

The objective of this study is to develop a deep learning model that effectively extracts features from MNIST handwritten images and accurately classifying them into their respective categories. To achieve this goal, we will utilize two types of deep learning architectures: the Simple Neural Network and the Convolutional Neural Network. These models will be implemented in Python using TensorFlow package and trained on the MNIST dataset. Additionally, we will conduct structural enhancements and optimize hyperparameters to enhance the accuracy of the models.

2. Data Description and Preprocessing

Data used in this test is MNIST dataset. This dataset consists of a total of 70,000 images. These images are divided into two sets: 60,000 for the training set and 10,000 for the test set. Each image is a grayscale 28x28 pixel representation associated with a label indicating the digit it represents. The labels range from 0 to 9. Below are the ten sample images from the dataset.

Figure 1: Plot the first 10 images in the MNIST dataset



The images data, denoted as “x” in our model, serves as the input for our training process. To ensure stability and faster convergence of the training models, we conduct normalization on the input data. All image data is converted into floating-point numbers and then normalized by dividing each pixel value by 255.0, resulting in all inputs data being scaled down to the range of 0 to 1.

Subsequently, one-hot encoding will be applied on the label data (category data), denoted as “y”. The purpose of one-hot encoding is to convert categorical variables into a numerical format that can be fed into machine learning models, thereby, improving prediction accuracy.

3. Methodology

As mentioned above, the two deep learning models investigated in this study are: Simple Neural Network, and its extension, Convolution Neural Network. They both are supervised learning model for multi-class classification problem.

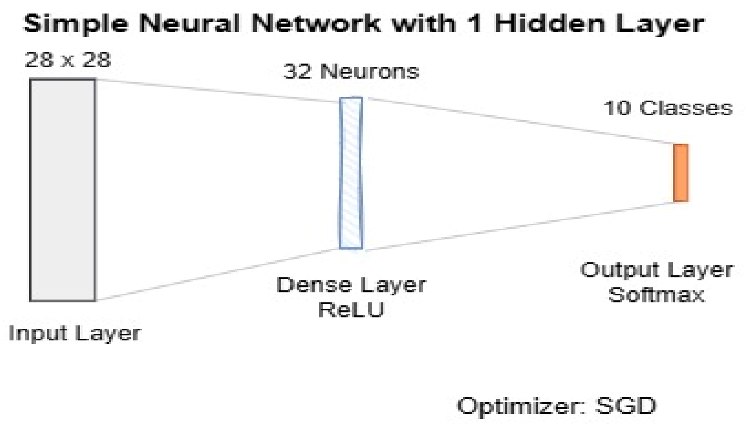
3.1 Simple Neural Network

Simple Neural Network

A Simple Neural Network, also known as a Multi-Layer Perceptron (MLP) or Fully Connected Neural Network (FCNN), comprises three main types of layers: an input layer, one or more hidden layers, and an output layer. Except for input nodes, all nodes in the hidden layers and output layer, also referred to as neuron, require the application of an activation function to enable the model to capture non-linear feature relationships within the data.

Below is a structure diagram of a basic neural network with one hidden layer. The activation function used in the hidden layer is ReLU, while the activation function for output layer is Softmax. Softmax is a specialized activation function typically applied to the output layer in classification problems.

Figure 2: Simple Neural Network with 1 Hidden Layer



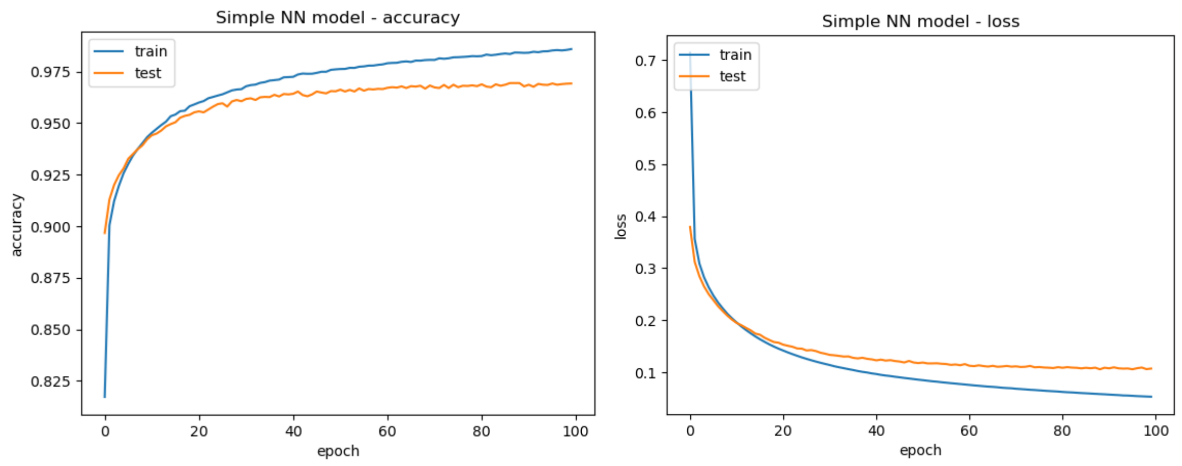
The optimization algorithm used in the model is Stochastic Gradient Descent (SGD). The loss function utilized in this model is cross entropy, which aims to minimize the dissimilarity between the true labels (which were one-hot encoded in preprocessing stage) and the predicted probabilities generated by the model. It is worth noting that Cross Entropy is specifically tailored for classification tasks, and hence it is employed as the loss function for all the models tested in this study.

For performance evaluation, the prediction accuracy metric is used to assess the model's performance.

$$\text{Accuracy} = \frac{\text{Number of correctly classified samples}}{\text{Total number of samples}}$$

To avoid overfitting, the model is also evaluated on validation data during training. It serves as a proxy for unseen data and helps to monitor the model's performance and generalization ability. However, due to the limited amount of data in the study, test data is used as validation data.

Chart 1: Accuracy Metric & Loss of the Simple NN model over Epochs



From Chart 1 above, we can see that a simple neural network (NN) model with just one dense layer did a pretty good job performed quite well in this handwriting classification task. After 100 epochs, its accuracy rate for training data is 98.64% and for validation data is 96.98%.

Enhanced Simple Neural Network

To further boost classification accuracy, enhancements were made to the initial simple neural network. Additional dense layers were added, with the number of neurons in the first dense layer increased to 128 and 64 neurons in the second layer. Both layers now use "sigmoid" activation function.

Figure 3: Enhanced Neural Network with 2 Hidden Layers

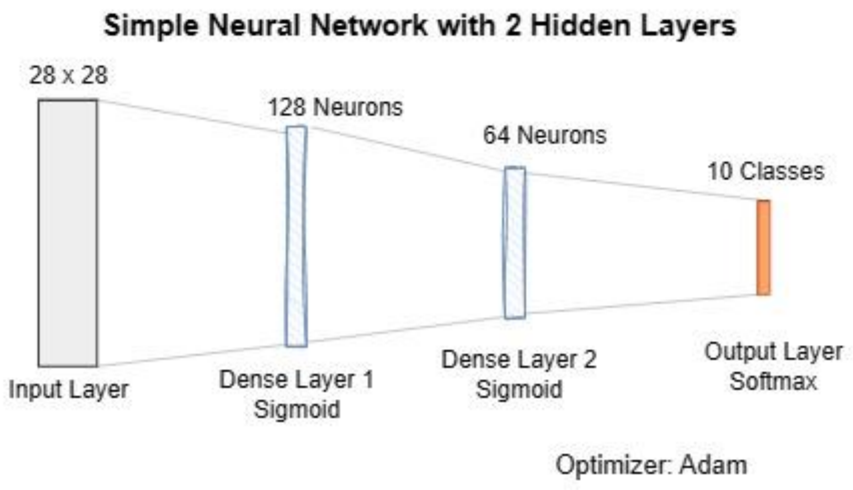
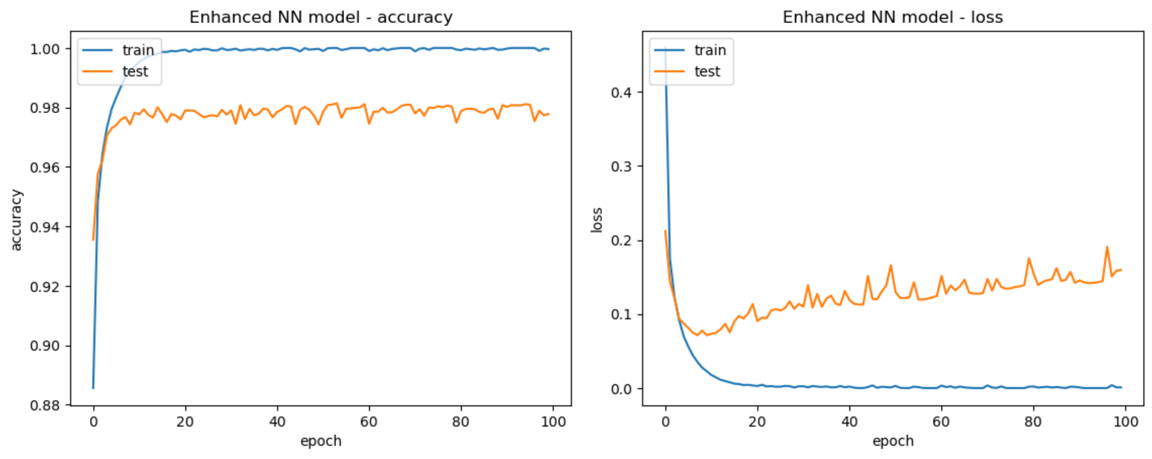


Chart 2: Accuracy Metric & Loss of the Enhanced simple NN model over Epochs



Comparing Chart 1 and Chart 2 that the enhance model with Adam optimizer converges more quickly to a ‘relatively’ optimal point compared to the simple neural network model. Furthermore, the addition of dense layer and increased neurons both contributes to higher accuracy in the result. Its accuracy rate in validation data is only 97.78%, which is higher than the accuracy rate in simple neural network model.

However, the increasing gap between the loss curves of the training data and validation data suggests a tendency towards overfitting in the enhanced model. Although the model achieves nearly perfect accuracy rate in the training data (99.97%), it fails to generalize well in the new data. In addition, the model exhibits noticeable variability in prediction when presented with unseen dataset.

Table 1: Accuracy Comparison between Simple NN and Enhanced Simple NN

Model	Training Data	Validating Data
Simple Neural Network	98.64%	96.98%
Enhanced Simple Neural Network	99.97%	97.78%

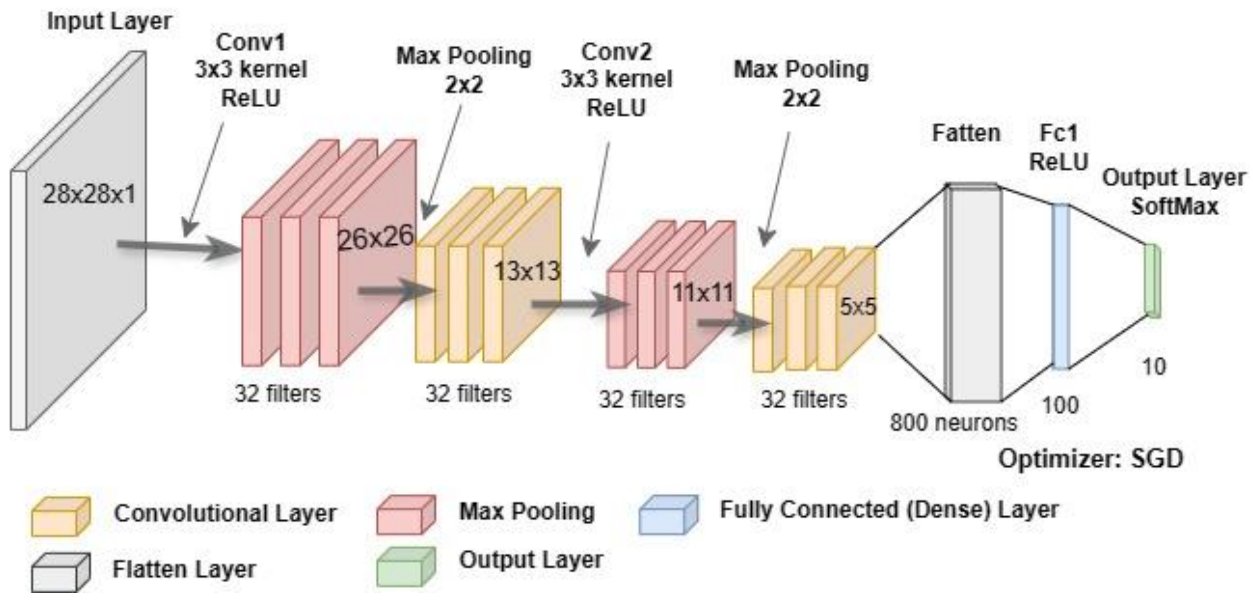
3.2 Convolutional Neural Network (CNN)

Convolutional Neural Network

Next, we construct a simple convolutional neural network (CNN) to explore whether CNNs can achieve better classification results for our task. CNNs are particularly effective for image classification tasks as they excel at capturing the spatial relationships between pixels in an image. They consist of three main types of layers: Convolutional Layers, Pooling Layers, and Fully Connected Layers.

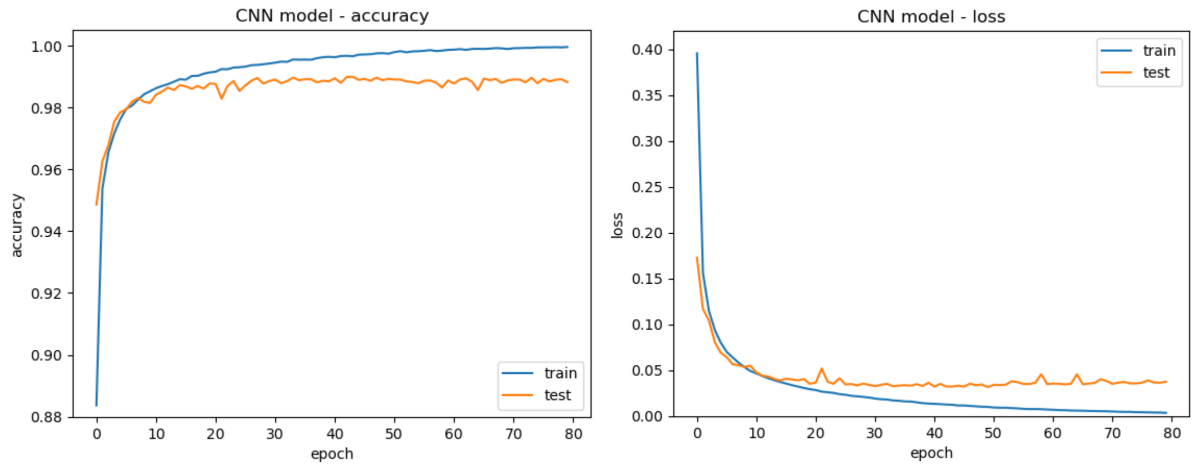
Convolutional layers and pooling layers enable the model to extract features at different level through a hierarchical extraction process. This enables the CNN to accurately perform classification tasks by learning and leveraging meaningful patterns in the data.

Figure 4: Convolutional Neural Network Structure (Basic)



We construct a simple CNN model with 2 convolutional layers, 2 pooling layers and 1 fully connected layer. The layer structure, number of layers, number of nodes for each layer, activation function used for each layer and optimization algorithm are all outline in the structure chart above.

Chart 3: Accuracy Metric & Loss of the simple CNN model over Epochs



Transitioning from a simple neural network (NN) to a convolutional neural network (CNN) resulted in an improvement in the accuracy rate of the trained model from 97.78% to 98.82% on the validation data. This suggests that convolutional layers are more efficient in classifying image data.

Enhanced Convolutional Neural Network

An enhanced CNN model was further constructed to improve accuracy. The enhanced model consists of 4 convolutional layers, 2 pooling layers (one with Max pooling and the other with Average pooling), and 2 fully connected layers. Padding is applied to all convolutional layers to preserve spatial information. To mitigate overfitting, dropout techniques are implemented in the last fully connected layers. The purpose of dropout is to enhance the model's generalization ability by forcing it to make decision based on a subset of features or information. The detail model structure is illustrated below.

Figure 5: Convolutional Neural Network with Enhanced Structure

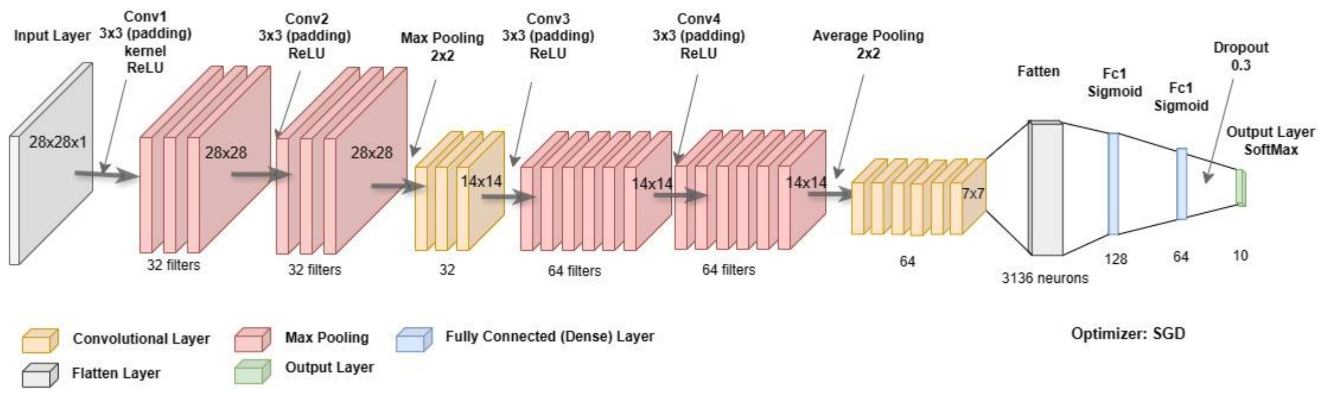
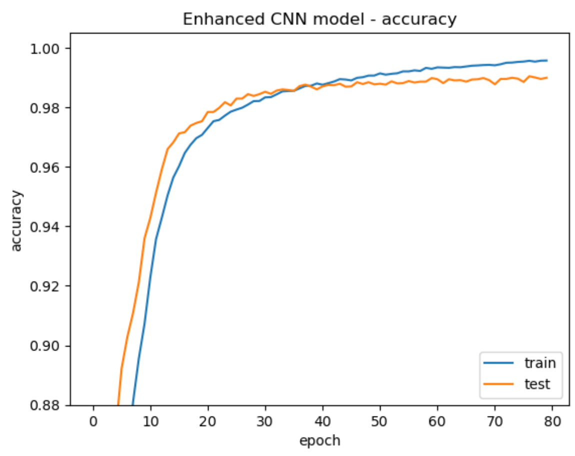


Chart 4: Accuracy Metric & Loss of the Enhanced CNN model over Epochs



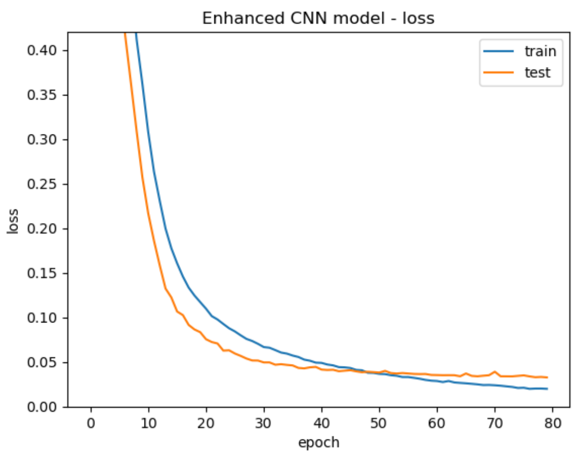


Table 2: Accuracy Comparison between Simple CNN and Enhanced CNN

Model	Training Data	Validating Data
Simple CNN	99.95%	98.82%
Enhanced CNN	99.57%	98.99%

The addition of multiple layers in the enhanced model only resulted in a slight improvement in the accuracy of the validation data from 98.82% to 98.99%. This is likely due to the fact that the MNIST dataset consists of simple grayscale images, for which a simple CNN is already sufficient to achieve good classification accuracy.

However, it's noteworthy that the discrepancy between the accuracy curves of the two models is smaller in the enhanced model, indicating that it has better generalization capabilities. Therefore, enhanced CNN models are preferable over simple CNN models, as they exhibit improved generalization performance.

The optimizer employed in this final model was SGD (Stochastic Gradient Descent). The default learning rate in Keras is set to 0.01 and the momentum default to 0.

3.3 Optimizer Optimization for CNN

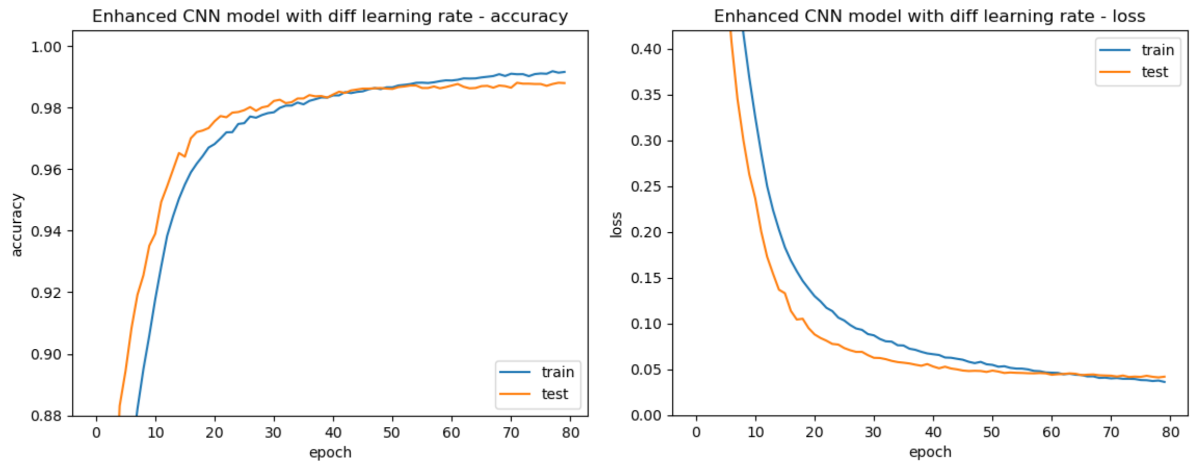
After structure fine-tuning, we then look at optimizer fine tuning for the enhance CNN model. Optimizer refers to the algorithm used to adjust the weight of the network during training in order to minimize the loss function and find the optimal model. In this process, I'm trying to find the better optimizer or learning rate for the enhanced CNN model we developed in previous stage.

Enhanced CNN with Modified SGD (Different Learning Rate)

Choosing appropriate values for the learning rate and momentum is crucial for training neural networks effectively. The learning rate controls the size of the step that the optimizer takes while updating the parameters, while momentum enhances the convergence speed and robustness of the optimization process.

In below test, we modified the optimizer to utilize SGD with momentum, adjusting the learning rate to 0.005 and momentum setting to 0.5.

Chart 5: Accuracy Metric & Loss of the Enhanced CNN model with different learning rate over Epochs

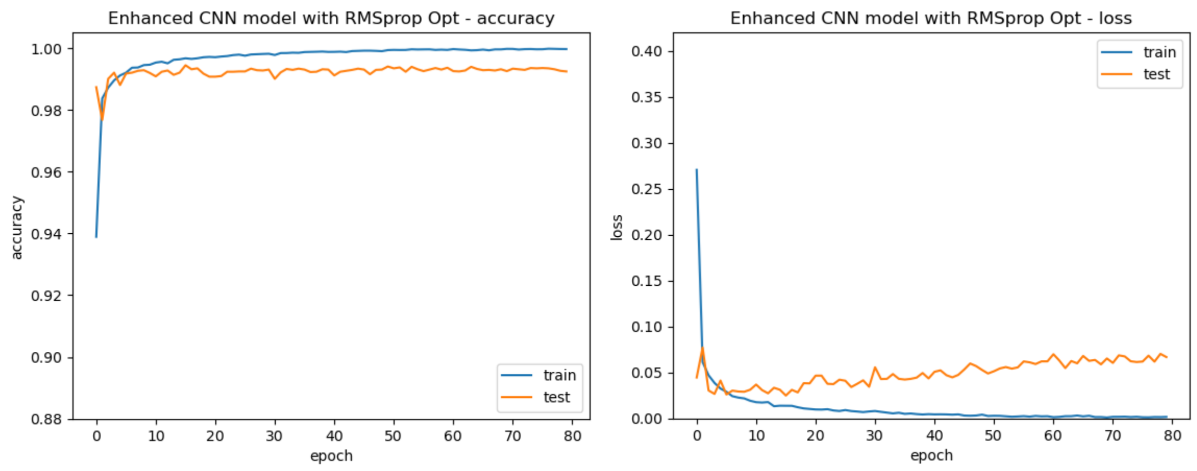


With adjustments made to the learning rate and momentum while maintaining the same number of epochs, the final trained model does not achieve higher accuracy compared to using the default hyperparameters.

Enhanced CNN with RMSprop Optimizer

We then try to fine-tune the optimizer type. I first try the model with RMSprop (Root Mean Square Propagation) optimizer. It’s an extension of the stochastic gradient descent (SGD) algorithm with momentum, it adapts the learning rate based on the magnitude of recent gradients.

Chart 6: Accuracy Metric & Loss of the Enhanced CNN model with RMSprop Optimizer over Epochs

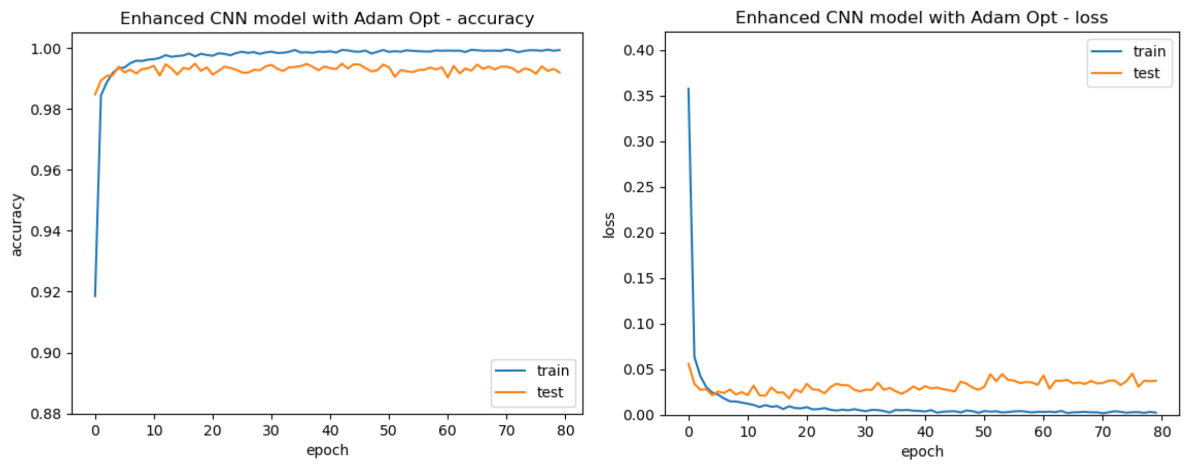


As expected, the RMSprop optimizer demonstrates swift convergence to a ‘relatively’ optimal point compared to other optimizers previously tested. However, a notable inclination towards overfitting is observed starting from about 20 epochs. We see a significantly increase in loss from 20 to 80 epochs from the chart above. This tendency may be due to the increased complexity of the model resulting from the change in optimizer. Additionally, the adaptive learning rate combined with the powerful CNN structure may render 80 epochs excessive for relatively simple image data like MNIST.

Enhanced CNN with Adam Optimizer

The last optimizer tested is Adam (Adaptive Moment Estimation), which combines elements of both Momentum optimization and RMSprop to achieve efficient and effective optimization. It dynamically adjusts learning rates and incorporates a momentum term and bias correction in its update rules, thereby accelerating convergence and eliminating the need for a fixed learning rate.

Chart 7: Accuracy Metric & Loss of the Enhanced CNN model with Adam Optimizer over Epochs



Similar to RMSprop, the Adam optimizer converges quickly to a ‘relatively’ optimal point with the dynamic learning rates. The final trained model generates better accuracy than other optimizers. However, its performance tends to fluctuate somewhat across epochs and there is a slight indication that the model may be overfitting, as evidenced by a slight degradation in performance on the validation dataset observed after the 50th epochs.

With 50 epochs, the final trained model utilizing the Adam optimizer achieves more outstanding accuracy compared to the other optimizers.

Chart 8: Accuracy Metric & Loss of the Enhanced CNN model with Adam Optimizer over 50 Epochs

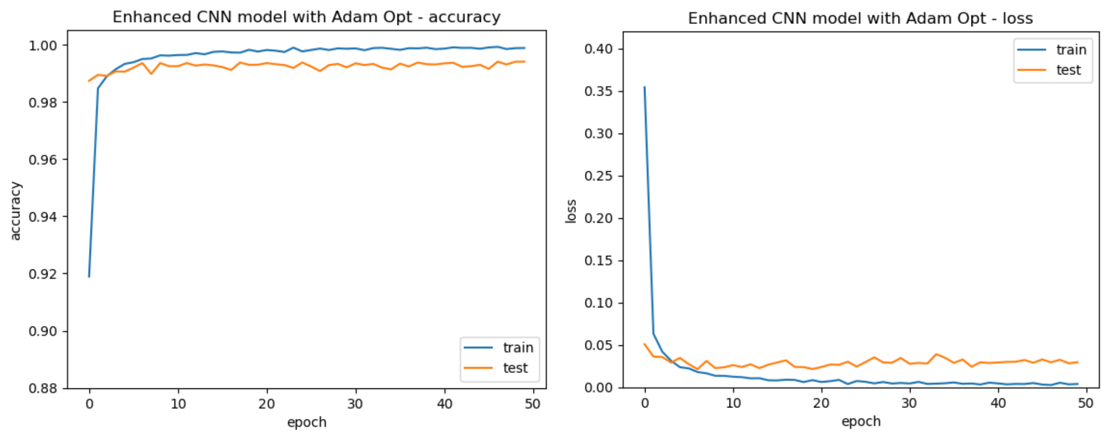
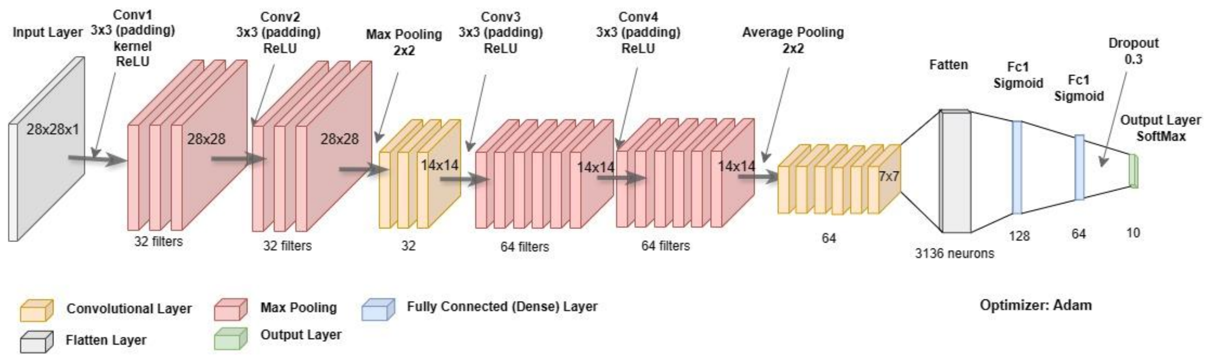


Table 3: Accuracy Comparison between Enhanced CNN with different optimizers

Model	Training Data	Validating Data
Enhanced CNN with Default SGD	99.57%	98.99%
Enhanced CNN with Modified SGD	98.64%	98.41%
Enhanced CNN with RMSprop	99.96%	99.24%
Enhanced CNN with Adam	99.88%	99.40%

From the performance comparison, we will choose the model trained using the enhanced CNN with Adam optimizer for 50 epochs as our final model. Below figure illustrate the structure and hyperparameters of the CNN model we employed.

Figure 6: Convolutional Neural Network with Enhanced Structure



3.4 Models Evaluation & Performance Ranking

Before this stage, all previous tests were conducted using training data and validation data. The final step of this study is to examine the generalization ability of the optimized models and provide an unbiased assessment of their performance on new unseen data. All models with their final weights are then evaluated on the testing dataset. The performance of each model is detailed below along with their respective ranking based on performance metric.

Table 4: Ranking of Performance -Accuracy Rate from Test Data

Model	Ranking	Performance
Enhanced CNN with Adam	1	99.40%
Enhanced CNN with RMSprop	2	99.24%
Enhanced CNN with Default SGD	3	98.99%
Simple CNN	4	98.82%
Enhanced CNN with Modified L Rate	5	98.41%
Enhanced Simple NN	6	97.78%
Simple NN	7	96.98%

* Performance = prediction accuracy in classifying images from the test set

Key findings from the above result comparison:

1. The top-performing model across all tests is the Enhanced CNN model with the Adam optimizer. This model generates almost perfect accuracy rate (99.40%) on test data. This success is attributed to the combination of its complex convolutional model structure and the effectiveness of the Adam optimizer.
2. CNN models generally outperform simple neural network models due to their ability to extract more detailed feature information from images.

3. When dealing with relatively simple image data, overly complex model structures can lead to overfitting. To mitigate this, techniques such as Data Augmentation, Dropout, Pretraining, or Ensemble methods can be employed.
4. Adam is typically superior to SGD, SGD with momentum, and RMSprop as an optimizer. However, Adam and RMSprop may exhibit a higher tendency to overfit compared to SGD in certain scenarios.
5. From above ranking, a Simple CNN model overperform the Enhanced CNN model with modified SGD suggests the critical importance of selecting appropriate learning rates and momentum during model training.
6. For MNIST data, a relatively simple CNN is generally sufficient for classification tasks. While Enhanced CNNs may yield slightly better performance, they are also more prone to overfitting.

Appendices

Appendix 1. Structure and Parameter Summary of Simple Neural Network

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 32)                25120     
_________________________________________________________________
dense_2 (Dense)              (None, 10)                330       
=================================================================
Total params: 25,450
Trainable params: 25,450
Non-trainable params: 0
_________________________________________________________________
```

Appendix 2. Structure and Parameter Summary of Enhanced Simple Neural Network

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_3 (Dense)              (None, 128)               100480    
_________________________________________________________________
dense_4 (Dense)              (None, 64)                8256      
_________________________________________________________________
dense_5 (Dense)              (None, 10)                650       
=================================================================
Total params: 109,386
Trainable params: 109,386
Non-trainable params: 0
_________________________________________________________________
```

Appendix 3. Structure and Parameter Summary of Basic Convolutional Neural Network

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 32)        9248      
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 32)          0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 800)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 100)               80100     
_________________________________________________________________
dense_7 (Dense)              (None, 10)                1010      
=================================================================
Total params: 90,678
Trainable params: 90,678
Non-trainable params: 0
_________________________________________________________________
```

Appendix 4. Structure and Parameter Summary of Enhanced Convolutional Neural Network

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_3 (Conv2D)            (None, 28, 28, 32)        320       
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 28, 28, 32)        9248      
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 32)        0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 14, 14, 64)        18496     
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 14, 14, 64)        36928     
_________________________________________________________________
average_pooling2d_1 (Average (None, 7, 7, 64)          0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 3136)              0         
_________________________________________________________________
dense_8 (Dense)              (None, 128)               401536    
_________________________________________________________________
dense_9 (Dense)              (None, 64)                8256      
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_10 (Dense)             (None, 10)                650       
=================================================================
Total params: 475,434
Trainable params: 475,434
Non-trainable params: 0
_________________________________________________________________
```

Bibliography

Wikipedia. (2024, April 02). MNIST database. Wikipedia. https://en.wikipedia.org/wiki/MNIST_database

Keras. (n.d). Accuracy metrics. Keras 3 API documentation. https://keras.io/api/metrics/accuracy_metrics/ - categoricalaccuracy-class

Zvornicanin, E., & Piwowarek, G. (2024, March 18). Convolutional Neural Network vs. Regular Neural Network. Baeldung on Computer Science. https://www.baeldung.com/cs/convolutional-vs-regular-nn - :~:text=This article explained the main,of parameters in the network.

Uniqtech. (2018, December 22). Multilayer Perceptron (MLP) vs Convolutional Neural Network in Deep Learning. Data Science Bootcamp. https://medium.com/data-science-bootcamp/multilayer-perceptron-mlp-vs-convolutional-neural-network-in-deep-learning-c890f487a8f1

Keras. (n.d). Adam. Keras 3 API documentation. https://keras.io/api/optimizers/adam/

Keras. (n.d). SGD. Keras 3 API documentation. https://keras.io/api/optimizers/sgd/

Srivastav,A. (2024, March 8). Understanding the Role of Optimizers and Initializers in Keras. Medium. https://medium.com/@srivastavayushmaan1347/understanding-the-role-of-optimizers-and-initializers-in-keras-1003ea5b462c

Franco,F. (2023,October 26). What is RMSprop? (with code!). Adaptive Optimization in Neural Networks. Medium. https://medium.com/thedeephub/what-is-rmsprop-0f54effc47e4