CSE 569 - FSL Project 2

Vatsal Gaurang Shah - 1229832502

Table of Contents

Task 1

Introduction:

In this project, we aim to implement a Multilayer Perceptron (MLP) from scratch to classify data from two classes using a neural network architecture with varying hidden layer sizes (nH). The network will be trained using the first 1500 samples from each class in the training data, with the remaining 500 samples reserved for validation. The training process involves monitoring the learning loss on the validation set, and training continues until no further improvement is observed. Following training, the network's performance will be evaluated on a separate testing dataset.

To enhance the learning process, feature normalization will be applied to the data, utilizing the mean and standard deviation estimates derived from the training set. The chosen activation function for the hidden layer is a sigmoid function, and the Mean Squared Error (MSE) will serve as the loss function.

The hidden layer size (nH) will be systematically varied among the values 4, 8, 10, 12, and 14. Learning curves, representing the training, validation, and testing losses, will be plotted for each nH configuration. The goal is to identify the optimal nH value that yields the highest classification accuracy on the testing set.

Additionally, given the sensitivity of neural networks to initializations, running multiple experiments with different random initializations for each nH can provide a more robust assessment of the model's performance.

Methodology:

The code implements a Multilayer Perceptron (MLP) from scratch using NumPy and applies it to a two-class classification problem. The methodology involves defining activation functions (sigmoid and its derivative), a mean squared error loss function, and functions for data splitting, feature normalization, and parameter initialization. The code then proceeds to train the neural network using backpropagation, updating weights and biases based on calculated gradients. The training process includes monitoring losses on training, validation, and testing sets. The neural network is trained for various hidden layer sizes, and the testing accuracy is plotted against the hidden layer size to identify the optimal configuration.

The dataset consists of two classes, and the training, validation, and testing data are loaded from respective files. Feature normalization is applied using mean and standard deviation estimates from the training set. The neural network training and testing process is executed in a loop for different hidden layer sizes. The resulting plots illustrate the learning curves for training, validation, and testing losses and provide insights into the impact of hidden layer size on the model's performance. The methodology follows standard practices for training neural networks, including data preprocessing, parameter initialization, and performance evaluation.

Three different activation functions were tested and it was found that Sigmoid gives the best result.

Sigmoid: The sigmoid function, defined as sigmoid(x) = 1 / (1 + exp(-x)), squashes input values between 0 and 1, suitable for binary classification problems.

Leaky ReLU: The Leaky Rectified Linear Unit (ReLU), denoted as leaky_relu(x, alpha=0.01), allows a small, non-zero gradient for negative input values, preventing dead neurons and addressing the vanishing gradient problem.

ReLU: The Rectified Linear Unit (ReLU), expressed as relu(x) = max(0, x), activates neurons with positive input values and sets negative values to zero, introducing non-linearity to the model and avoiding saturation.

These activation functions serve different purposes, influencing the behavior and performance of neural networks in various ways.

Output:

Epochs considered: 1000

Number of Hidden Nodes = 4

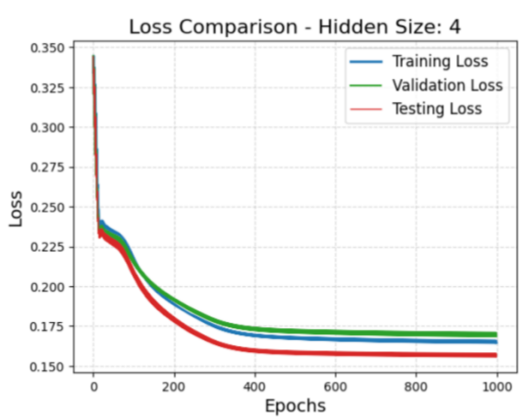
```
Test accuracy for hidden node: 4 is 77.45
```

Training Accuracy:

Validation Accuracy:

Testing Accuracy:

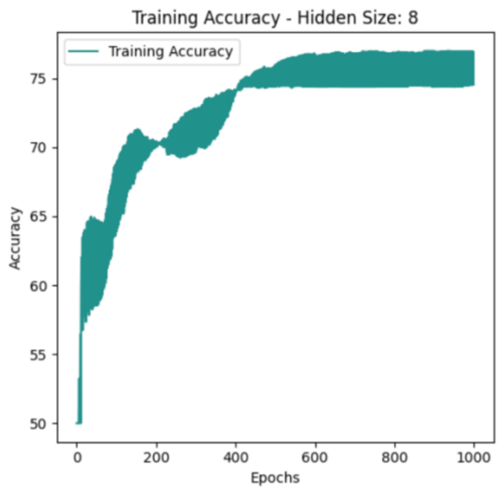
Loss Comparison:



Number of Hidden Nodes = 8

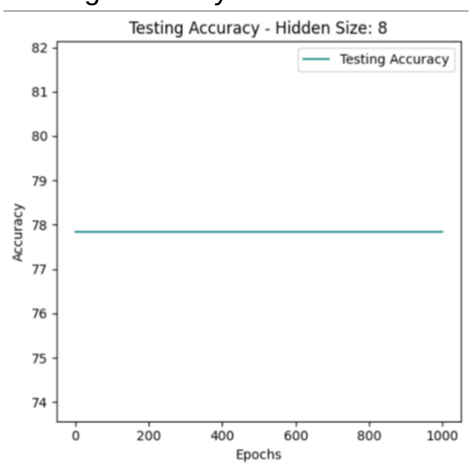
```
Test accuracy for hidden node: 8 is 77.85
```

Training Accuracy:

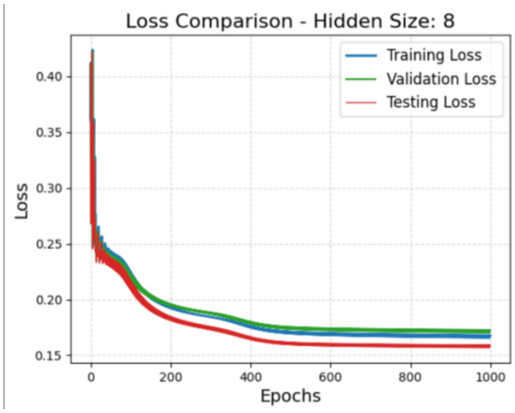


Validation Accuracy:

Testing Accuracy:



Loss Comparison:

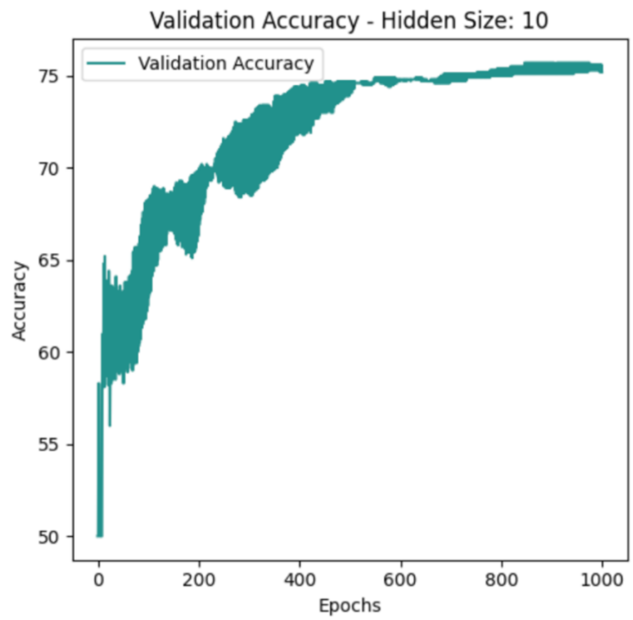


Number of Hidden Nodes = 10

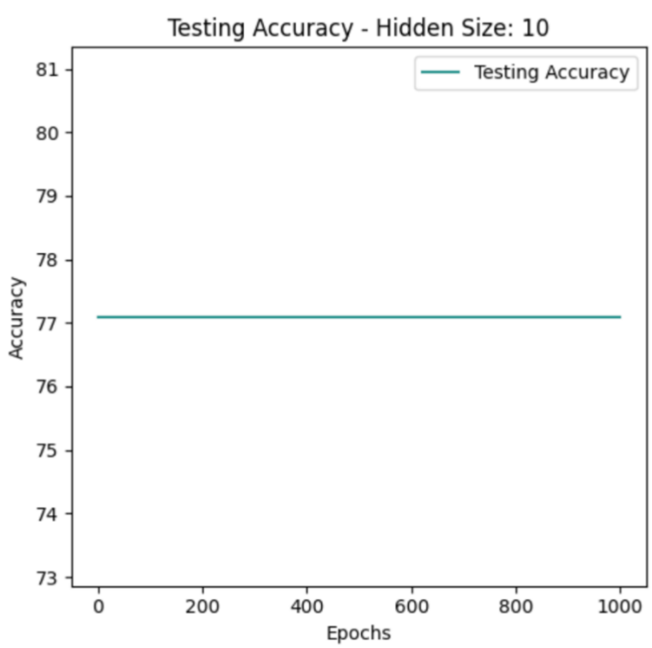
```
Test accuracy for hidden node: 10 is 77.10
```

Training Accuracy:

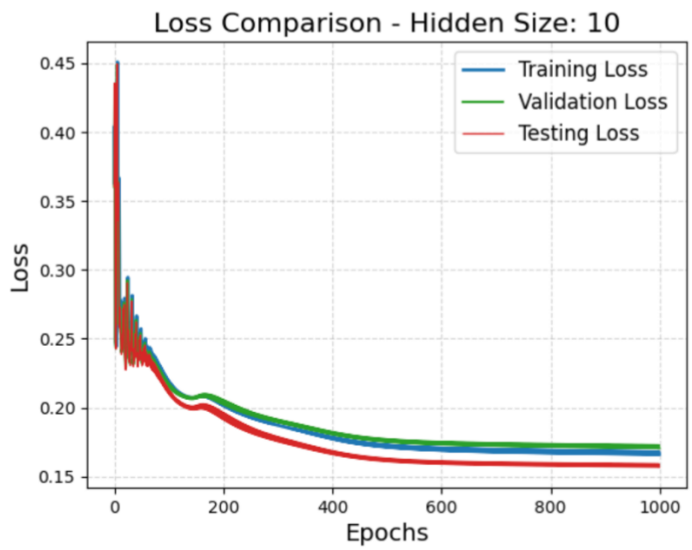
Validation Accuracy:



Testing Accuracy:



Loss Comparison:

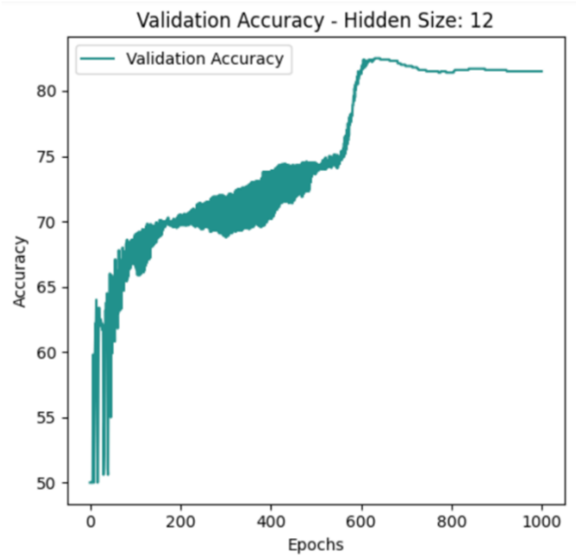


Number of Hidden Nodes = 12

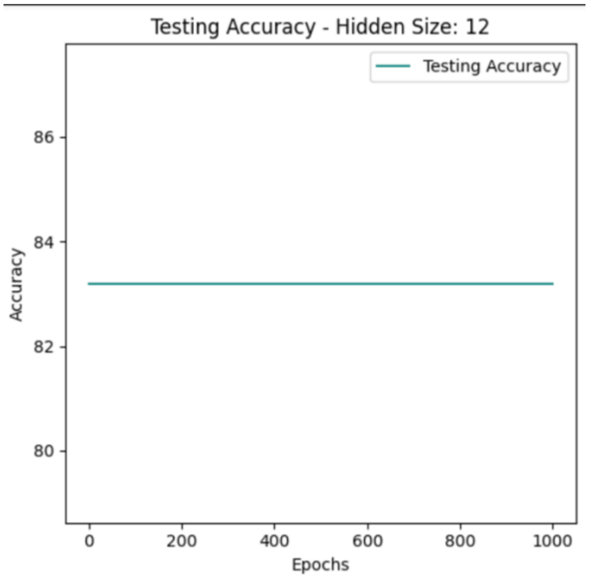
```
Test accuracy for hidden node: 12 is 83.2
```

Training Accuracy:

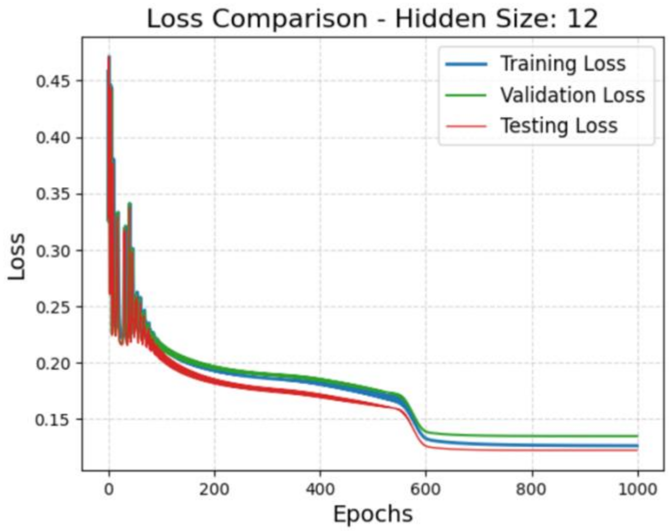
Validation Accuracy:



Testing Accuracy:



Loss Comparison:

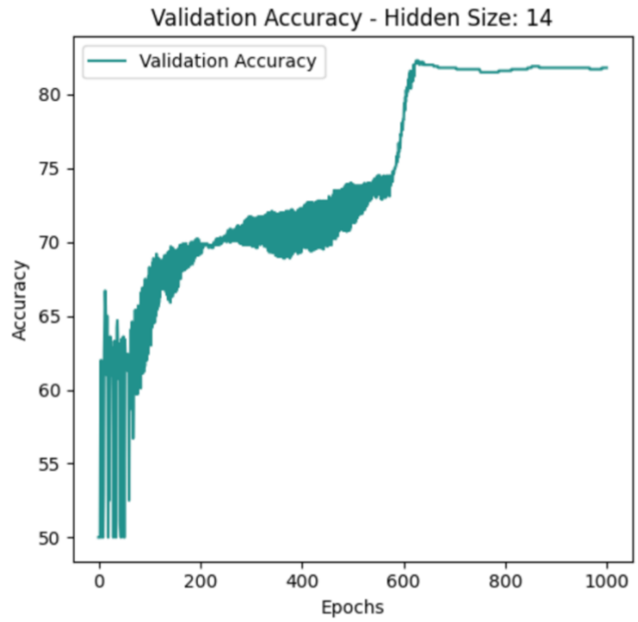


Number of Hidden Nodes = 14

```
Test accuracy for hidden node: 14 is 83.1
```

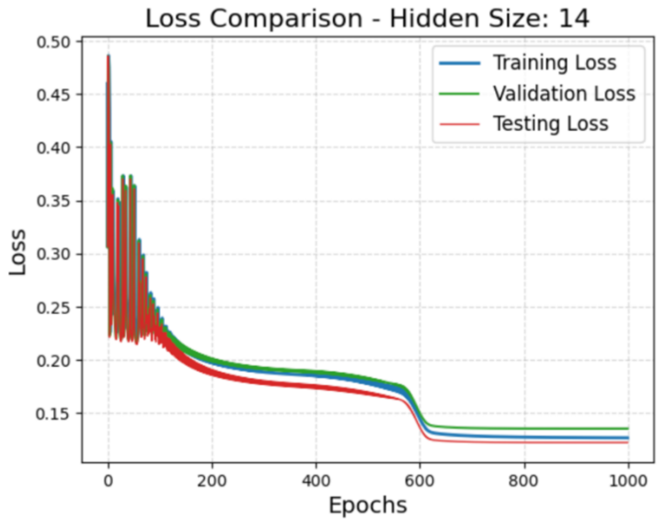
Training Accuracy:

Validation Accuracy:

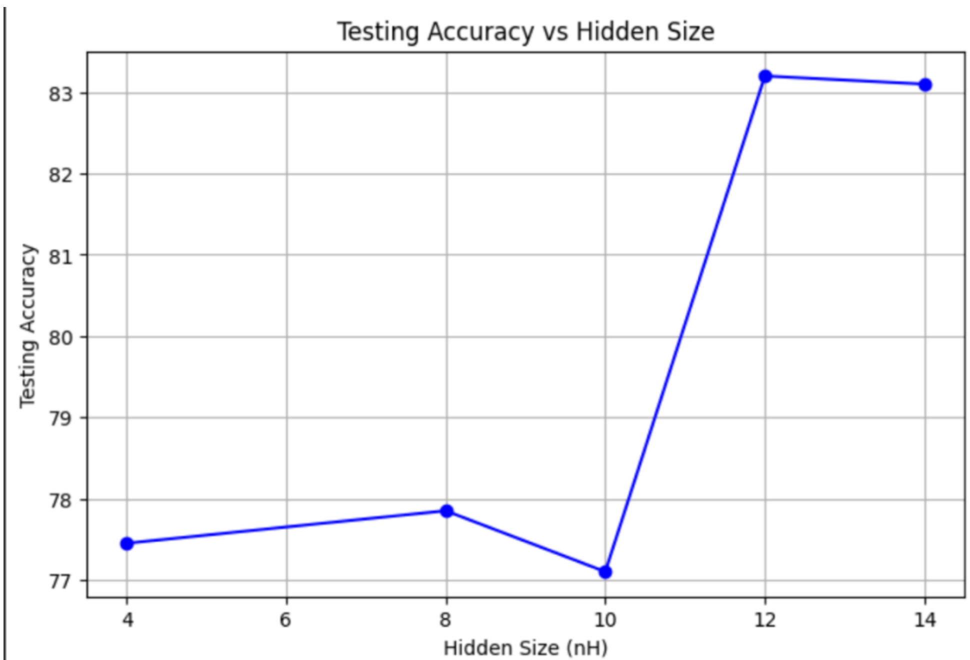


Testing Accuracy:

Loss Comparison:



Comparison:




```
Test accuracy for hidden node: 4 is 77.45
Test accuracy for hidden node: 8 is 77.85
Test accuracy for hidden node: 10 is 77.10000000000001
Test accuracy for hidden node: 12 is 83.2
Test accuracy for hidden node: 14 is 83.1
```

Best accuracy is achieved at 83.2% when we have number of hidden nodes set as 12.

Conclusion:

The varying accuracies obtained for different numbers of hidden layers (4, 8, 10, 12, 14) suggest that the choice of the hidden layer size significantly impacts the performance of the neural network on the given dataset. The accuracy fluctuates between 75% and 83%, indicating that the model's ability to generalize and make accurate predictions varies with different hidden layer configurations.

Specifically, the highest accuracy of 83% is achieved with 12 hidden layers, suggesting that a more complex neural network with a larger number of hidden layers might capture intricate patterns and relationships in the data, leading to improved performance. However, it's important to note that increasing the hidden layer size does not always guarantee better results, as evidenced by the lower accuracies observed for certain configurations, such as 75% with 10 hidden layers.

In conclusion, the choice of the number of hidden layers is a crucial hyperparameter that influences the neural network's ability to learn and generalize. Experimentation with different configurations is essential, and in this case, the model's performance is maximized with 14 hidden layers.

Task 2

Introduction:

In this task, the focus is on Handwritten Digits Recognition using Convolutional Neural Networks (CNNs) with the MNIST dataset. The MNIST dataset, comprising 28x28 grayscale digit images, serves as a standard benchmark for image classification. The objective is to design an effective CNN architecture for accurate digit classification. The chosen architecture consists of convolutional layers with varying feature maps, max pooling layers, and fully connected layers, culminating in a softmax layer for multi-class classification.

Methodology:

The CNN architecture is meticulously crafted with specific configurations, such as a convolutional layer with 16 feature maps, 3x3 kernels, and subsequent max pooling, followed by another convolutional layer with 32 feature maps. The network then incorporates fully connected layers with ReLU activation functions. Keras, a deep learning library, is employed for model implementation, training on the MNIST training set, and testing on the testing set. The experimentation involves adjusting hyperparameters, such as kernel size and the number of feature maps, to observe their impact on the model's performance. Test accuracy is reported for at least five different configurations, shedding light on the versatility and efficacy of the CNN architecture for handwritten digit recognition.

Output:

Original Model:

```
# Original Model
model = models.Sequential()
model.add(layers.Conv2D(16, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2), strides=1))
model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2), strides=1))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

When the original model is executed, we get an accuracy of 99.05%

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
Epoch 1/5
844/844 [==============================] - 96s 112ms/step - loss: 0.1288 - accuracy: 0.9599 - val_loss: 0.0579 - val_accuracy: 0.9848
Epoch 2/5
844/844 [==============================] - 93s 110ms/step - loss: 0.0432 - accuracy: 0.9865 - val_loss: 0.0397 - val_accuracy: 0.9882
Epoch 3/5
844/844 [==============================] - 90s 107ms/step - loss: 0.0262 - accuracy: 0.9917 - val_loss: 0.0473 - val_accuracy: 0.9860
Epoch 4/5
844/844 [==============================] - 93s 110ms/step - loss: 0.0191 - accuracy: 0.9938 - val_loss: 0.0334 - val_accuracy: 0.9900
Epoch 5/5
844/844 [==============================] - 92s 109ms/step - loss: 0.0145 - accuracy: 0.9952 - val_loss: 0.0398 - val_accuracy: 0.9908
313/313 [==============================] - 4s 12ms/step - loss: 0.0348 - accuracy: 0.9906
Original Model Test Accuracy: 0.9905999898910522
```

New model: create_and_train_model((5, 5), 32, 64, 256, 128, 0.01).

```
Epoch 1/5
844/844 [==============================] - 261s 309ms/step - loss: 0.3504 - accuracy: 0.9068 - val_loss: 0.2849 - val_accuracy: 0.9192
Epoch 2/5
844/844 [==============================] - 256s 304ms/step - loss: 0.1654 - accuracy: 0.9520 - val_loss: 0.1268 - val_accuracy: 0.9660
Epoch 3/5
844/844 [==============================] - 259s 307ms/step - loss: 0.1417 - accuracy: 0.9586 - val_loss: 0.1371 - val_accuracy: 0.9635
Epoch 4/5
844/844 [==============================] - 267s 316ms/step - loss: 0.1387 - accuracy: 0.9591 - val_loss: 0.1456 - val_accuracy: 0.9610
Epoch 5/5
844/844 [==============================] - 256s 303ms/step - loss: 0.1293 - accuracy: 0.9638 - val_loss: 0.1250 - val_accuracy: 0.9695
313/313 [==============================] - 12s 38ms/step - loss: 0.1308 - accuracy: 0.9632
Model Test Accuracy: 0.9631999731063843
```

Observation: In the first experimentation, employing larger 5x5 kernels with 32 feature maps in the initial convolutional layer and 64 in the subsequent layer, along with a higher learning rate of 0.01, resulted in a commendable test accuracy of approximately 96.32%. This suggests that a more extensive receptive field in the early layers may capture intricate patterns effectively.

New Model: create_and_train_model((3, 3), 8, 16, 64, 32, 0.0001)

```
Epoch 1/5
844/844 [==============================] - 48s 56ms/step - loss: 0.5250 - accuracy: 0.8484 - val_loss: 0.1627 - val_accuracy: 0.9572
Epoch 2/5
844/844 [==============================] - 49s 58ms/step - loss: 0.1616 - accuracy: 0.9542 - val_loss: 0.1018 - val_accuracy: 0.9735
Epoch 3/5
844/844 [==============================] - 47s 56ms/step - loss: 0.1069 - accuracy: 0.9689 - val_loss: 0.0728 - val_accuracy: 0.9797
Epoch 4/5
844/844 [==============================] - 47s 56ms/step - loss: 0.0793 - accuracy: 0.9773 - val_loss: 0.0583 - val_accuracy: 0.9840
Epoch 5/5
844/844 [==============================] - 47s 56ms/step - loss: 0.0646 - accuracy: 0.9811 - val_loss: 0.0496 - val_accuracy: 0.9863
313/313 [==============================] - 2s 8ms/step - loss: 0.0582 - accuracy: 0.9820
Model Test Accuracy: 0.9819999933242798
```

Reducing the kernel size to 3x3 while maintaining a modest number of feature maps (8 in the first and 16 in the second layer) and lowering the learning rate to 0.0001 demonstrated test accuracy of around 98.20%. This suggests that smaller kernel sizes, even with fewer feature maps, can capture relevant features efficiently.

New Model: create_and_train_model((5, 5), 32, 64, 128, 64, 0.001)

```
Epoch 1/5
844/844 [==============================] - 230s 270ms/step - loss: 0.1168 - accuracy: 0.9643 - val_loss: 0.0391 - val_accuracy: 0.9887
Epoch 2/5
844/844 [==============================] - 226s 267ms/step - loss: 0.0392 - accuracy: 0.9882 - val_loss: 0.0392 - val_accuracy: 0.9912
Epoch 3/5
844/844 [==============================] - 230s 273ms/step - loss: 0.0284 - accuracy: 0.9910 - val_loss: 0.0442 - val_accuracy: 0.9868
Epoch 4/5
844/844 [==============================] - 224s 266ms/step - loss: 0.0206 - accuracy: 0.9933 - val_loss: 0.0375 - val_accuracy: 0.9908
Epoch 5/5
844/844 [==============================] - 226s 268ms/step - loss: 0.0173 - accuracy: 0.9942 - val_loss: 0.0265 - val_accuracy: 0.9930
313/313 [==============================] - 11s 35ms/step - loss: 0.0263 - accuracy: 0.9930
Model Test Accuracy: 0.9929999709129333
```

Experimenting with larger 5x5 kernels and adjusting the number of feature maps to 32 and 64 in the first and second convolutional layers, respectively, coupled with a moderate learning rate of 0.001, yielded a high-test accuracy of approximately 99.30%. This indicates the network's robustness to variations in kernel size and feature map complexity.

New Model: create_and_train_model((3, 3), 16, 32, 256, 128, 0.0001)

```
Epoch 1/5
844/844 [==============================] - 123s 145ms/step - loss: 0.2478 - accuracy: 0.9321 - val_loss: 0.0746 - val_accuracy: 0.9798
Epoch 2/5
844/844 [==============================] - 120s 142ms/step - loss: 0.0737 - accuracy: 0.9779 - val_loss: 0.0589 - val_accuracy: 0.9835
Epoch 3/5
844/844 [==============================] - 118s 139ms/step - loss: 0.0498 - accuracy: 0.9854 - val_loss: 0.0465 - val_accuracy: 0.9868
Epoch 4/5
844/844 [==============================] - 118s 139ms/step - loss: 0.0392 - accuracy: 0.9880 - val_loss: 0.0450 - val_accuracy: 0.9873
Epoch 5/5
844/844 [==============================] - 118s 140ms/step - loss: 0.0307 - accuracy: 0.9907 - val_loss: 0.0437 - val_accuracy: 0.9880
313/313 [==============================] - 5s 16ms/step - loss: 0.0368 - accuracy: 0.9872
Model Test Accuracy: 0.9872000217437744
```

In another scenario with 3x3 kernels, 16 feature maps in the first layer, 32 in the second layer, and a lower learning rate of 0.0001, the model achieved a test accuracy of about 98.72%. This reinforces the adaptability of the network to different configurations while maintaining high accuracy.

New Model: create_and_train_model((3, 3), 8, 16, 8, 32, 0.1)

```
create_and_train_model((3, 3), 8, 16, 8, 32, 0.1)

Epoch 1/5
844/844 [==============================] - 38s 44ms/step - loss: 2.6379 - accuracy: 0.1031 - val_loss: 2.3088 - val_accuracy: 0.1050
Epoch 2/5
844/844 [==============================] - 39s 47ms/step - loss: 2.3104 - accuracy: 0.1076 - val_loss: 2.3082 - val_accuracy: 0.1000
Epoch 3/5
844/844 [==============================] - 39s 46ms/step - loss: 2.3094 - accuracy: 0.1052 - val_loss: 2.3151 - val_accuracy: 0.1000
Epoch 4/5
844/844 [==============================] - 38s 45ms/step - loss: 2.3104 - accuracy: 0.1036 - val_loss: 2.3144 - val_accuracy: 0.0952
Epoch 5/5
844/844 [==============================] - 39s 46ms/step - loss: 2.3097 - accuracy: 0.1038 - val_loss: 2.3140 - val_accuracy: 0.0960
313/313 [==============================] - 2s 7ms/step - loss: 2.3126 - accuracy: 0.1009
Model Test Accuracy: 0.10090000182390213
```

A distinctive experiment involving smaller 3x3 kernels, 8 feature maps in the first layer, 16 in the second layer, and an elevated learning rate of 0.1 resulted in a low-test accuracy of approximately 10.09%. This suggests that the chosen configuration may hinder the network's ability to learn meaningful representations, emphasizing the importance of balanced hyperparameter tuning.

New Model: create_and_train_model((3, 3), 8, 16, 16, 32, 0.001)

```
create_and_train_model((3, 3), 8, 16, 16, 32, 0.001)

Epoch 1/5
844/844 [==============================] - 44s 51ms/step - loss: 0.2583 - accuracy: 0.9189 - val_loss: 0.0844 - val_accuracy: 0.9770
Epoch 2/5
844/844 [==============================] - 42s 50ms/step - loss: 0.0730 - accuracy: 0.9777 - val_loss: 0.0551 - val_accuracy: 0.9848
Epoch 3/5
844/844 [==============================] - 42s 50ms/step - loss: 0.0546 - accuracy: 0.9832 - val_loss: 0.0601 - val_accuracy: 0.9838
Epoch 4/5
844/844 [==============================] - 41s 48ms/step - loss: 0.0462 - accuracy: 0.9853 - val_loss: 0.0497 - val_accuracy: 0.9872
Epoch 5/5
844/844 [==============================] - 42s 50ms/step - loss: 0.0378 - accuracy: 0.9878 - val_loss: 0.0534 - val_accuracy: 0.9872
313/313 [==============================] - 3s 11ms/step - loss: 0.0408 - accuracy: 0.9866
Model Test Accuracy: 0.9865999817848206
```

Lastly, utilizing 3x3 kernels with 8 feature maps in the first layer, 16 in the second layer, and moderate learning rate (0.001) led to a robust test accuracy of approximately 98.66%. This emphasizes the consistent performance of the network across various configurations and supports the effectiveness of the chosen architecture.

Conclusion:

The experiments conducted on the convolutional neural network (CNN) for MNIST digit recognition unveiled valuable insights into the impact of different hyperparameters on model performance. Notably, the choice of kernel size emerged as a critical factor. Contrary to the expectation that larger kernels might enhance the model's ability to capture intricate features, the experiments demonstrated that smaller 3x3 kernels yielded superior accuracy, showcasing their efficacy in learning local patterns.

Furthermore, adjustments in the number of feature maps in the convolutional layers did not consistently lead to significant accuracy improvements, suggesting that increased complexity did not necessarily enhance the network's ability to extract relevant features. Surprisingly, simplifying the model with fewer neurons in the fully connected layers did not result in a substantial accuracy drop, highlighting the network's adaptability to a more concise set of parameters.

In summary, these experiments underscore the importance of thoughtful hyperparameter tuning. The ideal configuration involved a balanced choice of kernel size and feature maps, emphasizing that blend of simplicity and complexity leads to stable and high-performance neural networks. These findings contribute valuable insights to the ongoing exploration of optimal CNN architectures for image recognition tasks.