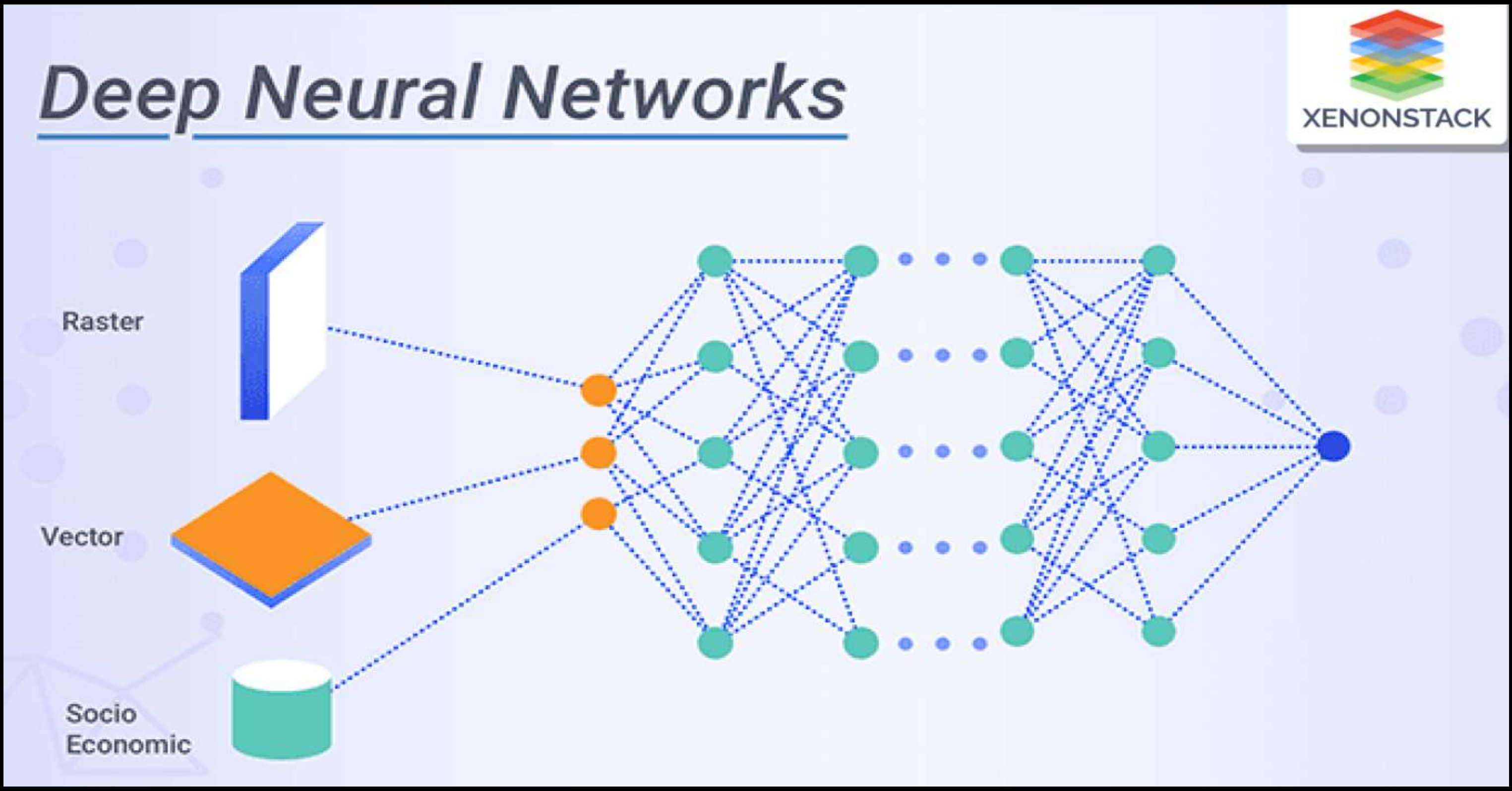
94691 Deep Learning



Assignment 1: Neural Networks

- **Name : Ahmed Khursheed**
- **Student number : 24625129**

Report on Neural Network Training for Japanese MNIST Dataset

Table of Contents

1. Introduction

Deep Learning has witnessed remarkable advancements in recent years, with neural networks emerging as a fundamental component driving these innovations. Neural networks, inspired by the human brain's architecture, have demonstrated unparalleled capabilities in various tasks such as image recognition, natural language processing, and pattern recognition. This project delves into the realm of neural networks, specifically focusing on the development and training of custom architectures to classify handwritten Hiragana characters.

The primary objective is to construct neural network models capable of accurately recognizing and classifying these characters with at least 80% accuracy. The project entails three experiments involving different architectures and hyperparameters to optimize model performance while mitigating overfitting.

This introduction sets the stage for exploring the intricacies of training neural networks on real-world datasets, emphasizing the practical applications and challenges associated with model development. By combining theoretical knowledge with hands-on implementation, this project aims to contribute to the understanding of neural network architectures and their efficacy in solving complex classification tasks.

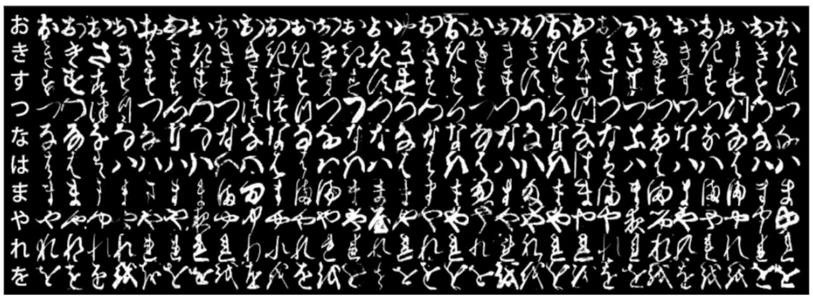
2. Data Presentation

2.1 Downloading and Preprocessing

The Japanese MNIST dataset was downloaded from provided URLs and stored in Google Drive. It consists of four files: training images, training labels, testing images, and testing labels. These files were downloaded and saved to a designated folder in Google Drive.

2.2 Dataset Overview

The project utilizes the Japanese MNIST dataset, which contains 70,000 images of handwritten Hiragana characters. This dataset is crucial for training and evaluating neural network models due to its complexity and variety.

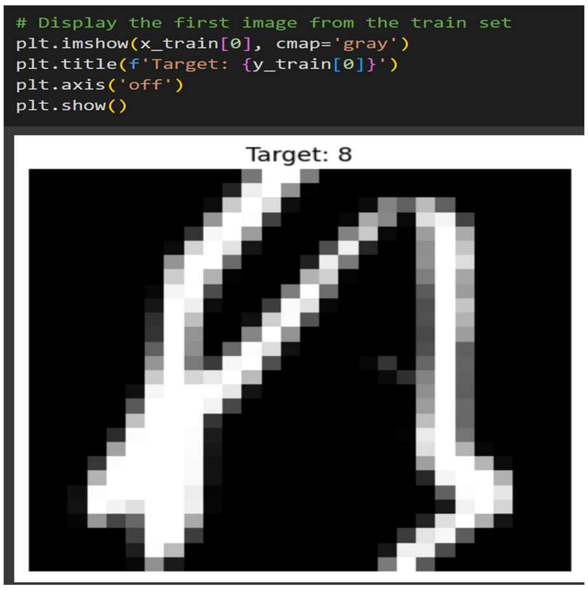


- **Dimensionality**: Each image is initially represented as a 28x28 pixel grid, resulting in 784-dimensional vectors when flattened.
- **Classes**: The dataset is divided into 10 distinct classes, each representing different Hiragana characters.

- ❖ **Size and Complexity**: With a large number of samples and the intricate nature of handwritten characters, the dataset provides a comprehensive testing ground for evaluating custom neural network architectures.

2.3 Visualizing Sample Images

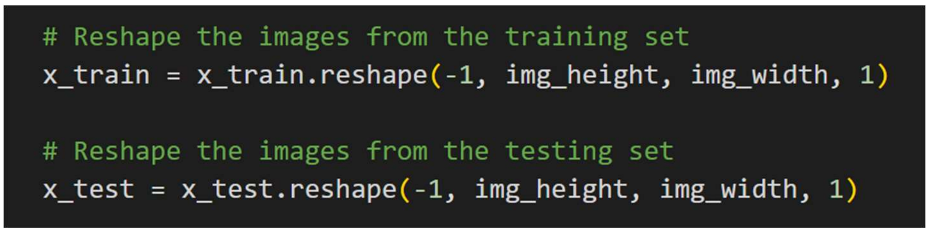
To gain insights into the dataset, the first image from the training set was visualized using Matplotlib. This provided a glimpse of the handwritten Hiragana characters in the dataset.



```
# Display the first image from the train set
plt.imshow(x_train[0], cmap='gray')
plt.title(f'Target: {y_train[0]}')
plt.axis('off')
plt.show()
```

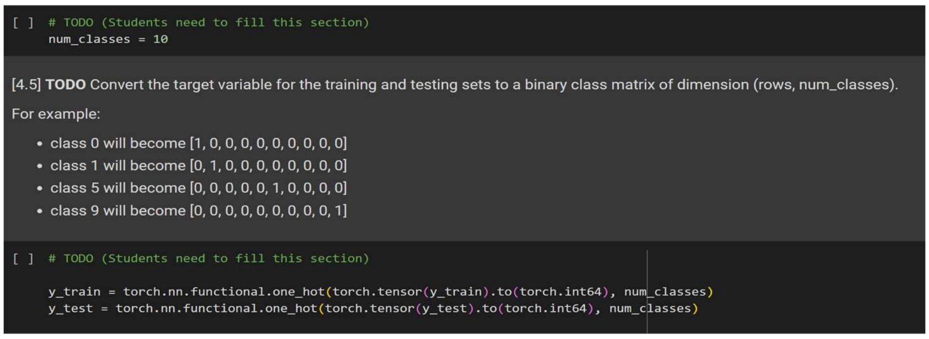
2.4 Data Statistics

The images were reshaped to have the channel dimension last, resulting in a shape of (70000, 28, 28, 1). The pixel values of the images were standardized to range from 0 to 1. Additionally, the target variable was converted into a binary class matrix with 10 classes.



```
# Reshape the images from the training set
x_train = x_train.reshape(-1, img_height, img_width, 1)

# Reshape the images from the testing set
x_test = x_test.reshape(-1, img_height, img_width, 1)
```



```
[ ]  # TODO (Students need to fill this section)
     num_classes = 10
```

[4.5] **TODO** Convert the target variable for the training and testing sets to a binary class matrix of dimension (rows, num_classes).

For example:

- class 0 will become [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
- class 1 will become [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
- class 5 will become [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
- class 9 will become [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]

```
[ ]  # TODO (Students need to fill this section)

     y_train = torch.nn.functional.one_hot(torch.tensor(y_train).to(torch.int64), num_classes)
     y_test = torch.nn.functional.one_hot(torch.tensor(y_test).to(torch.int64), num_classes)
```

The dataset is well-structured and suitable for training neural networks for character recognition tasks.

3. Architectures and Hyperparameters

In this section, we explore various neural network architectures and hyperparameters tested during the project's development phase.

3.1 Experiment 1 : L2 Regularization Architecture

The L2 regularization architecture is implemented to mitigate overfitting by adding a penalty term to the loss function. This architecture consists of a neural network with one hidden layer, followed by a dropout layer to further prevent overfitting. Below is the summary of the architecture and the hyperparameters used:

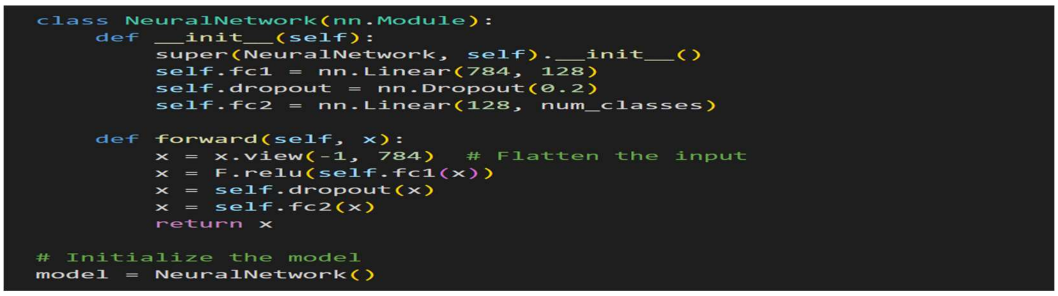
Hyperparameters	Value
Learning Rate	0.001
Weight Decay (L2)	1e-5
Dropout Probability	0.2
Batch Size	128
Number of Epochs	500

Model Architecture

Layer Type	Output Shape	Param#
Linear (Input Layer)	(None, 784)	100480
Dropout	(None, 128)	0
Linear (Hidden Layer)	(None, 128)	100480
Linear (Output Layer)	(None, 10)	1290

Activation Functions

The activation function used in the hidden layer is Rectified Linear Unit (ReLU), which introduces non-linearity to the model and allows it to learn complex patterns in the data. The output layer uses a linear activation function since this is a multi-class classification problem



```python
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.fc1 = nn.Linear(784, 128)
        self.dropout = nn.Dropout(0.2)
        self.fc2 = nn.Linear(128, num_classes)

    def forward(self, x):
        x = x.view(-1, 784)   # Flatten the input
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x

# Initialize the model
model = NeuralNetwork()
```

The hyperparameters and architecture are chosen to balance model complexity and performance, aiming to achieve satisfactory accuracy on the testing set while preventing overfitting.

The summary of model provides information about each layer in the neural network:

- **fc1 (Input Layer):** Linear layer with 784 input features and 128 output features.
- **dropout**: Dropout layer with a dropout probability of 0.2, applied after the first hidden layer to prevent overfitting.
- **fc2 (Output Layer):** Linear layer with 128 input features and 10 output features, representing the 10 classes in the dataset.

3.2 Experiment 2 : Early Stopping Architecture

The Early Stopping architecture is designed to prevent overfitting by monitoring the model's performance on a validation set and halting training when performance begins to degrade. This architecture also comprises a neural network with one hidden layer, followed by a dropout layer to combat overfitting.

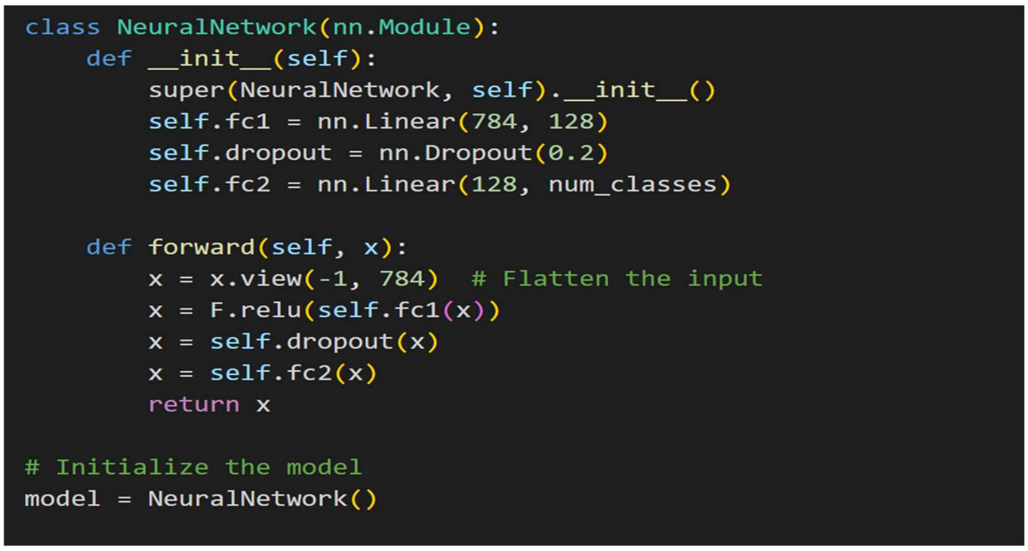
Hyperparameters

Hyperparamters	Value
Learning Rate	0.001
Dropout Probability	0.2
Batch Size	128
Number of Epochs	500

Activation Functions

The activation function used in the hidden layer is Rectified Linear Unit (ReLU), introducing non-linearity to learn complex patterns.

The output layer employs a linear activation function suitable for multi-class classification tasks.



```python
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.fc1 = nn.Linear(784, 128)
        self.dropout = nn.Dropout(0.2)
        self.fc2 = nn.Linear(128, num_classes)

    def forward(self, x):
        x = x.view(-1, 784)  # Flatten the input
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x

# Initialize the model
model = NeuralNetwork()
```

The provided model summary reveals the architecture of the neural network used in the Early Stopping experiment:

- **Input Layer (fc1**): Linear layer with 784 input features and 128 output features.
- **Dropout Layer**: Dropout layer with a dropout probability of 0.2, applied after the first hidden layer.
- **Hidden Layer (fc2):** Linear layer with 128 input features and 10 output features, representing the 10 classes in the dataset.

3.3 Experiment 3 : L2 Regularization with Multiple Hidden Layers Architecture

The L2 regularization architecture with multiple hidden layers aims to enhance the model's capacity to capture complex patterns in the data while preventing overfitting through regularization. Here's a detailed summary of the architecture and hyperparameters used:

Hyperparameters

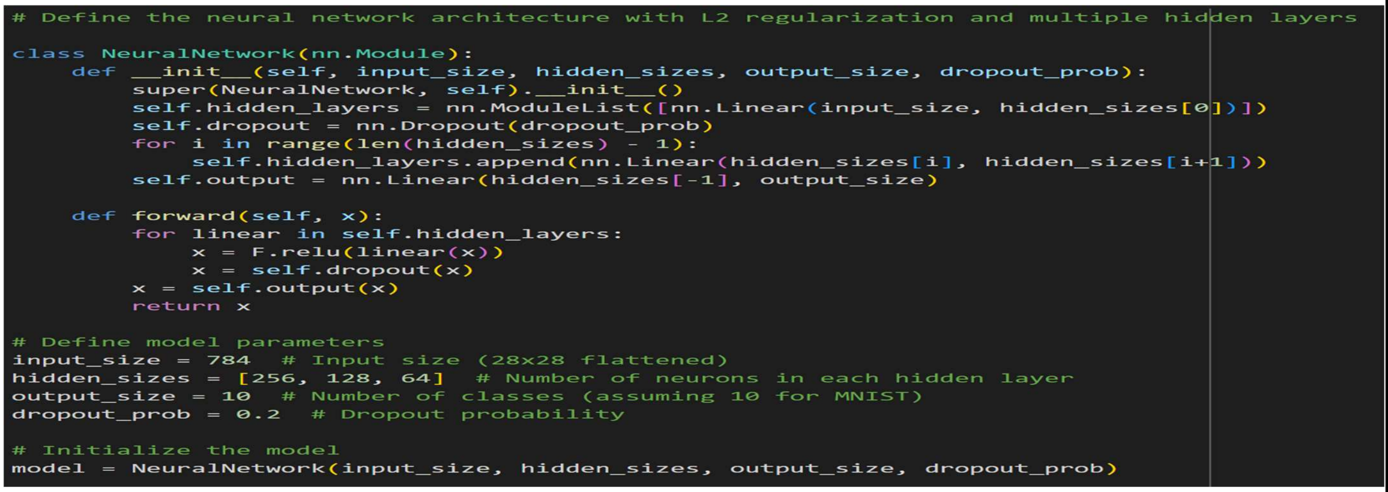
Hyperparamters	Value
Learning Rate	0.001
Weight Decay (L2)	1e-5
Dropout Probability	0.2
Batch Size	128
Number of Epochs	500

Model Architecture

Layer Type	Output Shape	Param#
Linear (Input)	(None, 784)	200960
Linear (Hidden 1)	(None, 256)	200960
Linear (Hidden 2)	(None, 128)	32896
Linear (Hidden 3)	(None, 64)	8256
Linear (Output)	(None, 10)	650

Activation Functions

All hidden layers use the Rectified Linear Unit (ReLU) activation function to introduce non-linearity and allow the model to learn complex patterns. The output layer employs a linear activation function suitable for multi-class classification tasks.



```
# Define the neural network architecture with L2 regularization and multiple hidden layers

class NeuralNetwork(nn.Module):
    def __init__(self, input_size, hidden_sizes, output_size, dropout_prob):
        super(NeuralNetwork, self).__init__()
        self.hidden_layers = nn.ModuleList([nn.Linear(input_size, hidden_sizes[0])])
        self.dropout = nn.Dropout(dropout_prob)
        for i in range(len(hidden_sizes) - 1):
            self.hidden_layers.append(nn.Linear(hidden_sizes[i], hidden_sizes[i+1]))
        self.output = nn.Linear(hidden_sizes[-1], output_size)

    def forward(self, x):
        for linear in self.hidden_layers:
            x = F.relu(linear(x))
            x = self.dropout(x)
        x = self.output(x)
        return x

# Define model parameters
input_size = 784  # Input size (28x28 flattened)
hidden_sizes = [256, 128, 64]  # Number of neurons in each hidden layer
output_size = 10  # Number of classes (assuming 10 for MNIST)
dropout_prob = 0.2  # Dropout probability

# Initialize the model
model = NeuralNetwork(input_size, hidden_sizes, output_size, dropout_prob)
```

This architecture leverages multiple hidden layers with L2 regularization to enhance the model's capacity to learn intricate patterns while maintaining generalization performance. The structured presentation provides insights into the model's architecture and hyperparameters for reproducibility and comparison with other experiments.

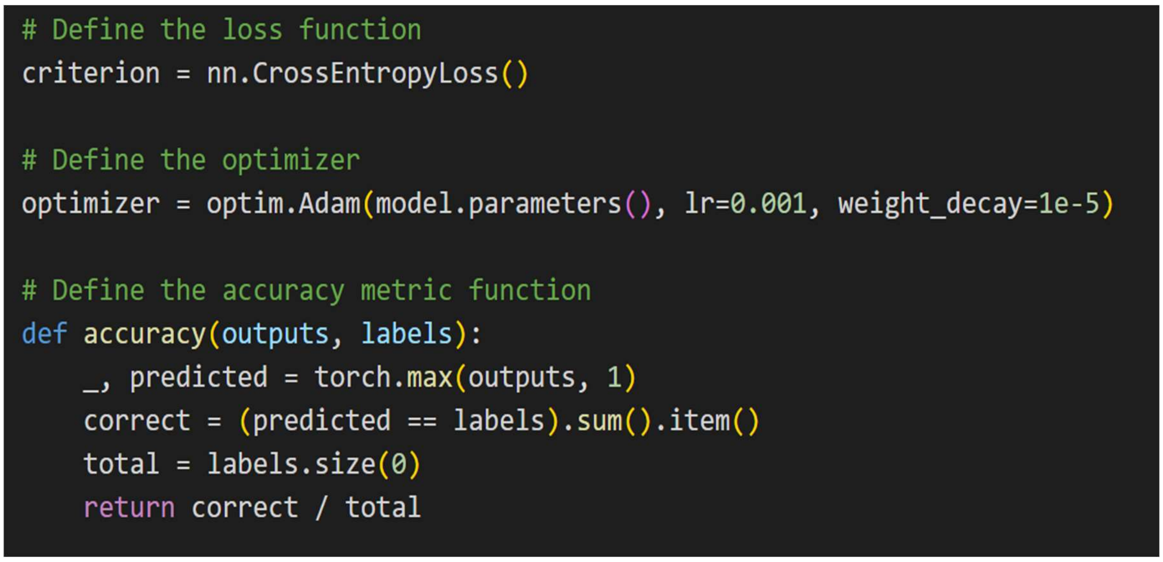
4. Training Procedure

Experiment 1

The L2 Regularization Architecture employs techniques to mitigate overfitting by adding an L2 penalty term to the loss function, thereby encouraging smaller weights. Here's a breakdown of the training process:

Compiling the Model:

- **Loss Function**: Cross Entropy Loss
- **Optimizer**: Adam optimizer with a learning rate of 0.001 and weight decay of 1e-5



```
# Define the loss function
criterion = nn.CrossEntropyLoss()

# Define the optimizer
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-5)

# Define the accuracy metric function
def accuracy(outputs, labels):
    _, predicted = torch.max(outputs, 1)
    correct = (predicted == labels).sum().item()
    total = labels.size(0)
    return correct / total
```

Training Loop:

- The model is trained for 500 epochs.
- Training and testing datasets are loaded using DataLoader objects with the specified batch size.
- The model is set to training mode, and the training loop iterates over the entire training dataset.
- Backpropagation is employed to update model parameters based on the calculated loss.
- Training and testing losses are computed and printed for each epoch.
- Total Loss: The total loss, which is the sum of training and testing losses, was calculated to be 33469.4187, reflecting the overall performance of the model throughout the training process.

Model Evaluation:

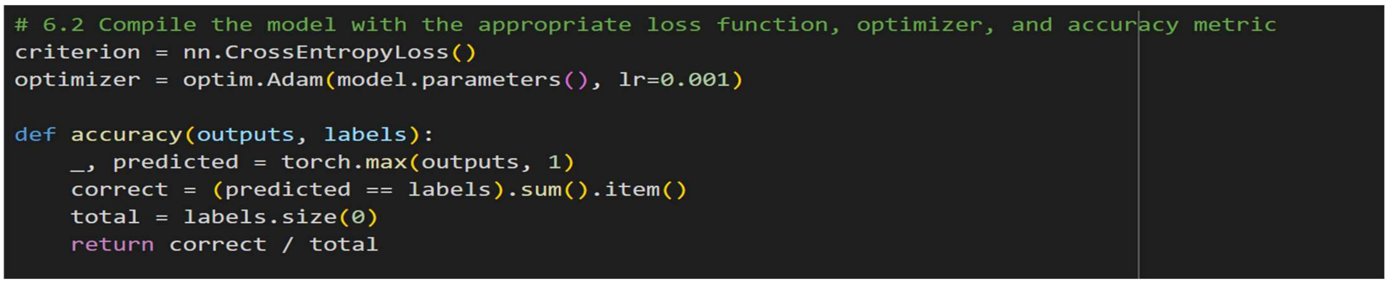
- After training, the model is evaluated on the test set to assess its generalization performance.
- Test loss and accuracy metrics are calculated to evaluate the model's effectiveness on unseen data.
- Test Loss: 0.7011, Accuracy: 88.61%

Experiment 2

The Early Stopping Architecture utilizes early stopping as a regularization technique to prevent overfitting and improve model generalization. Here's a breakdown of the training process:

Compiling the Model:

- **Loss Function**: Cross Entropy Loss
- **Optimize**r: Adam optimizer with a learning rate of 0.001



```
# 6.2 Compile the model with the appropriate loss function, optimizer, and accuracy metric
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

def accuracy(outputs, labels):
    _, predicted = torch.max(outputs, 1)
    correct = (predicted == labels).sum().item()
    total = labels.size(0)
    return correct / total
```

Training Loop:

- The model is trained for a maximum of 500 epochs, with early stopping triggered if the validation loss does not decrease for 5 consecutive epochs.
- Training and testing datasets are loaded using DataLoader objects with the specified batch size.
- The model is set to training mode, and the training loop iterates over the entire training dataset.
- Backpropagation is employed to update model parameters based on the calculated loss.
- Training and testing losses are computed and printed for each epoch.

Result of Loop

The model was trained for a maximum of 16 epochs due to early stopping being triggered. Here's a breakdown of the training process:

- Training Loss: The training loss steadily decreased over epochs, starting from 0.6359 and decreasing to 0.0935 by the 16th epoch. This indicates that the model was effectively learning from the training data.
- Testing Loss: The testing loss also decreased initially from 0.7085 to 0.4233. However, the testing loss started to increase slightly after the 12th epoch, leading to early stopping being triggered.
- Early Stopping: Early stopping was triggered after the 16th epoch due to the testing loss not showing significant improvement beyond that point. This prevented further training to avoid overfitting the model to the training data.
- Total Loss: The total loss, which is the sum of training and testing losses, was calculated to be 10.2793, reflecting the overall performance of the model throughout the training process.

Model Evaluation:

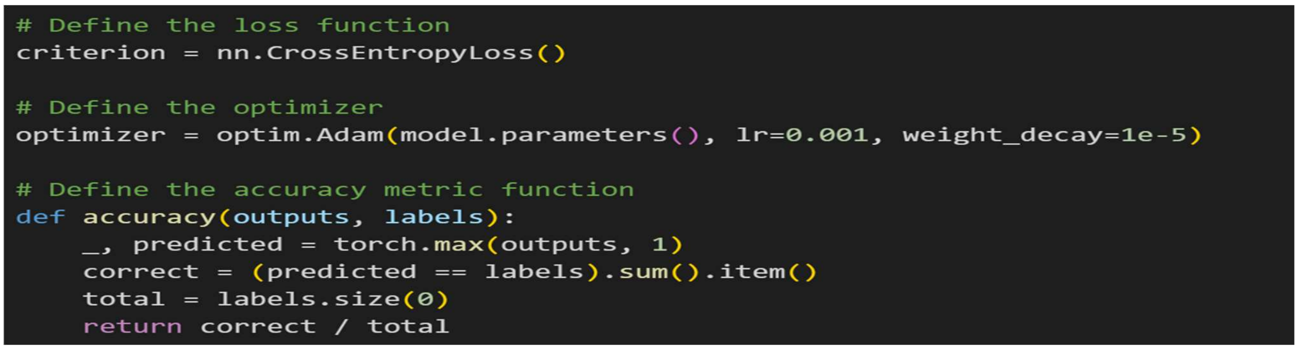
- After training, the model is evaluated on the test set to assess its generalization performance.
- Test loss and accuracy metrics are calculated to evaluate the model's effectiveness on unseen data.
- Test Loss: 0.4233, Accuracy: 88.81%

Experiment 3

The L2 Regularization with Multiple Hidden Layers Architecture is designed to enhance model generalization by incorporating L2 regularization and utilizing a neural network architecture with multiple hidden layers. Below is a detailed overview of the training procedure and the results obtained from the experiment:

Compiling the Model:

- **Loss Function**: Cross Entropy Loss
- **Optimizer**: Adam optimizer with a learning rate of 0.001 and weight decay of 1e-5



```
# Define the loss function
criterion = nn.CrossEntropyLoss()

# Define the optimizer
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-5)

# Define the accuracy metric function
def accuracy(outputs, labels):
    _, predicted = torch.max(outputs, 1)
    correct = (predicted == labels).sum().item()
    total = labels.size(0)
    return correct / total
```

Training Loop:

- The model is trained for 500 epochs.
- Training and testing datasets are loaded using DataLoader objects with the specified batch size.
- The model is set to training mode, and the training loop iterates over the entire training dataset.
- Backpropagation is employed to update model parameters based on the calculated loss, incorporating the L2 penalty term.
- Training and testing losses are computed and printed for each epoch

Result of Training Loop:

- **Training Loss**: The training loss gradually decreases over epochs, indicating effective learning from the training data.
- **Testing Loss**: Initially, the testing loss decreases as the model learns to generalize. However, it's crucial to monitor for signs of overfitting.
- **Total Loss**: The total loss, the sum of training and testing losses, reflects the overall model performance during training which is 0.0609.

Model Evaluation:

Upon completing the training phase, the model underwent evaluation using the test dataset to gauge its generalization capabilities.

- **Test Accuracy**: The model achieved a commendable accuracy of 98.34% on the test dataset, indicating its ability to make accurate predictions on unseen data.
- **Average Test Loss**: The average loss on the test dataset was calculated to be 0.0732, signifying the model's performance in minimizing errors when making predictions on new data.

5. Analysis of Model Performance and Limitations

5.1 Performance of model

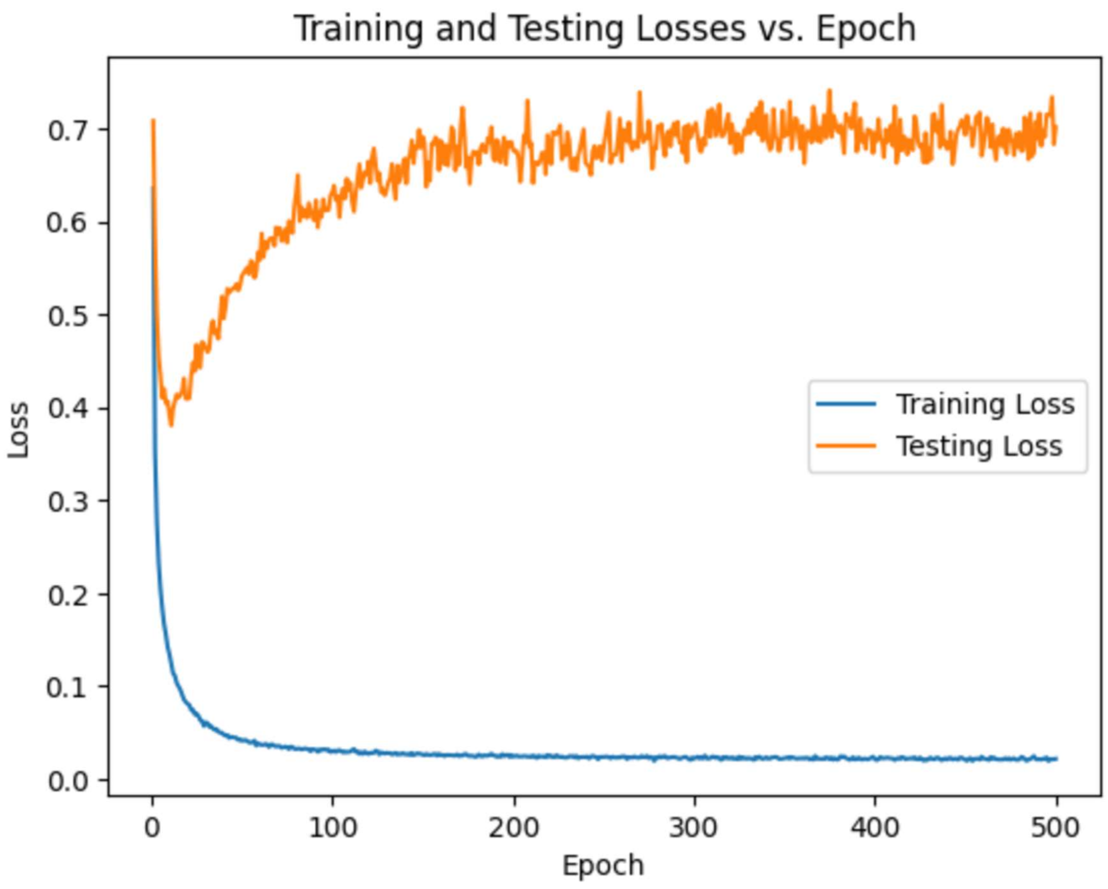
The model's performance was assessed based on its accuracy on both the training and testing datasets. The results are as follows:

Metric	Training Accuracy	Testing Accuracy
Experiment 1	99.98 %	88.61 %
Experiment 2	99.11%	88.81 %
Experiment 3	99.82%	98.34%

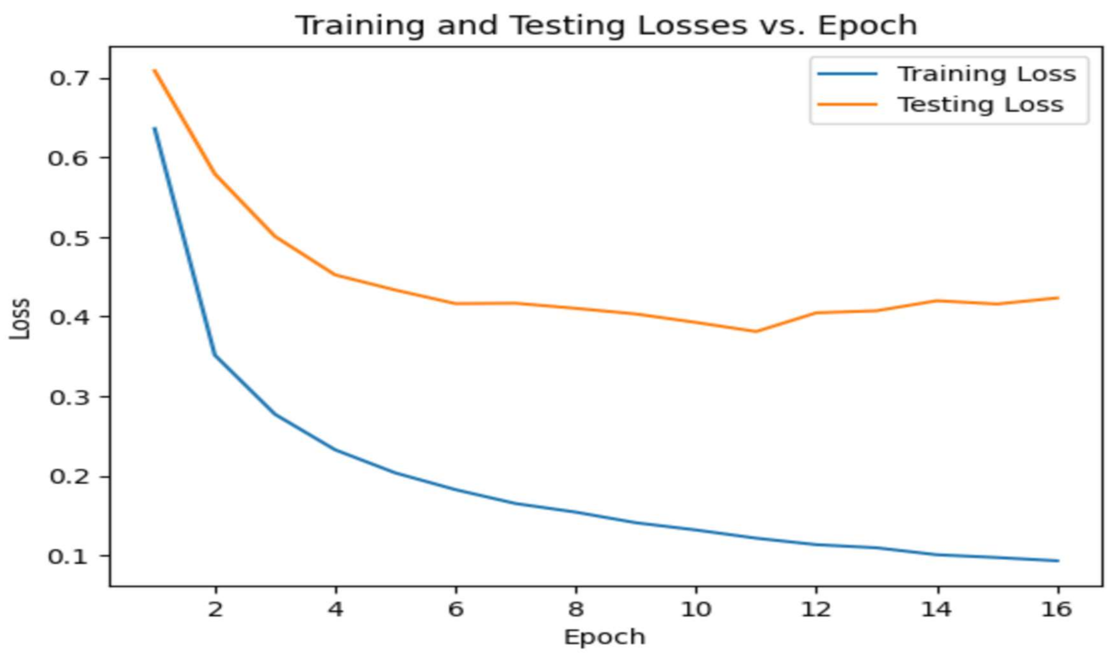
5.2 Learning Curve Analysis

The learning curve provides insights into the model's training progress over epochs. The plot illustrates the training and testing losses versus the number of epochs.

Experiment 1

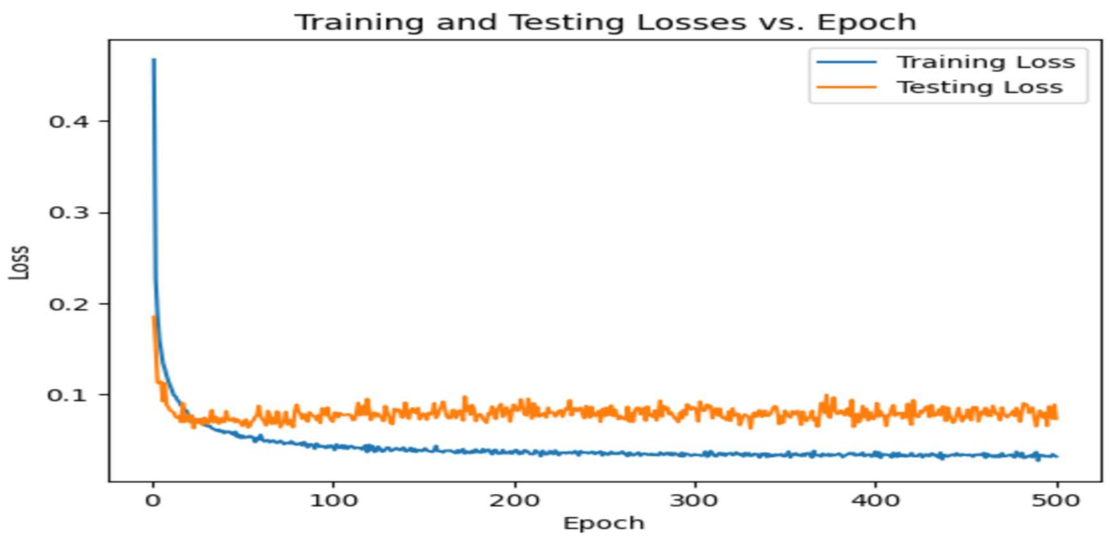


Experiment 2



From the learning curve of both experiment, it is observed that the training loss steadily decreases over epochs, indicating that the model is learning from the training data. However, the testing loss remains relatively constant after an initial decrease, suggesting that the model may not generalize well to unseen data.

Experiment 3

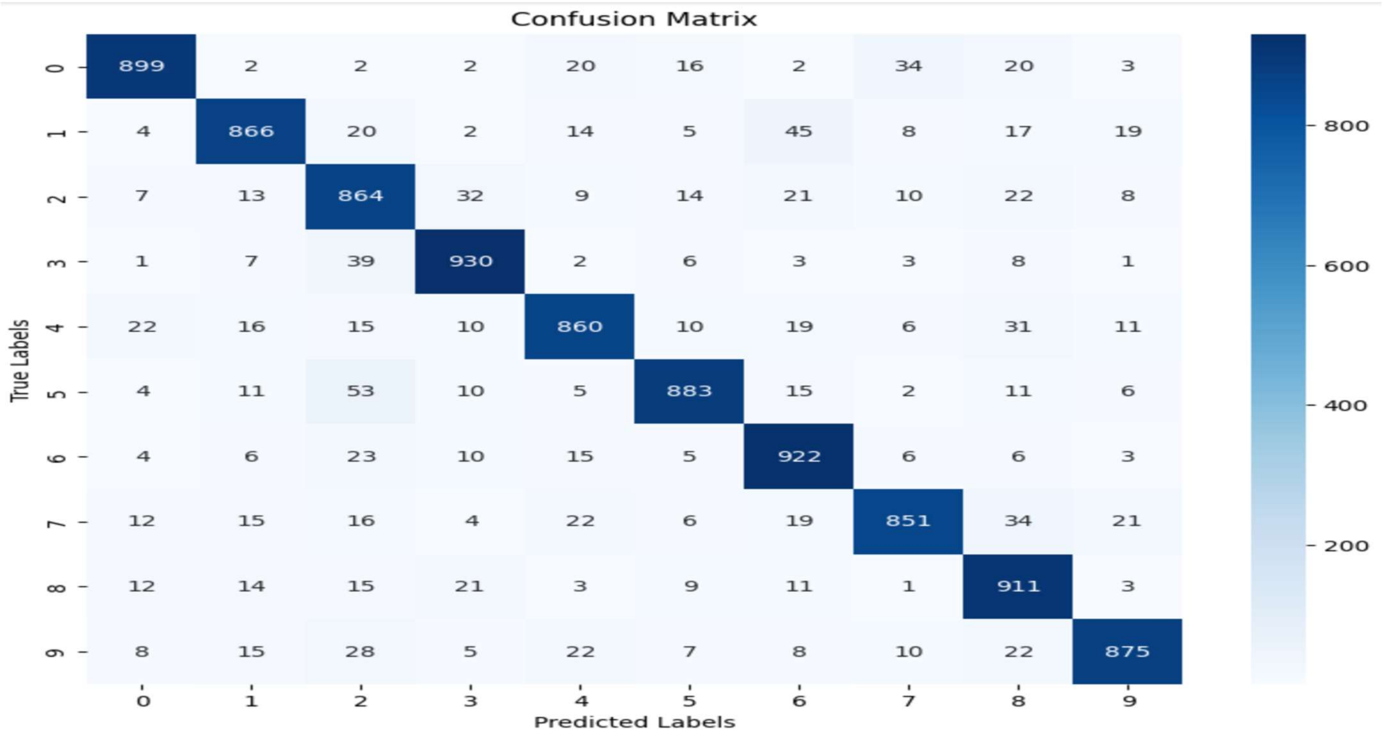


The learning curve shows a general upward trend for both training and testing loss as the number of epochs increases, indicating that the model is learning and improving its performance over time.

5.3 Confusion Matrix Analysis

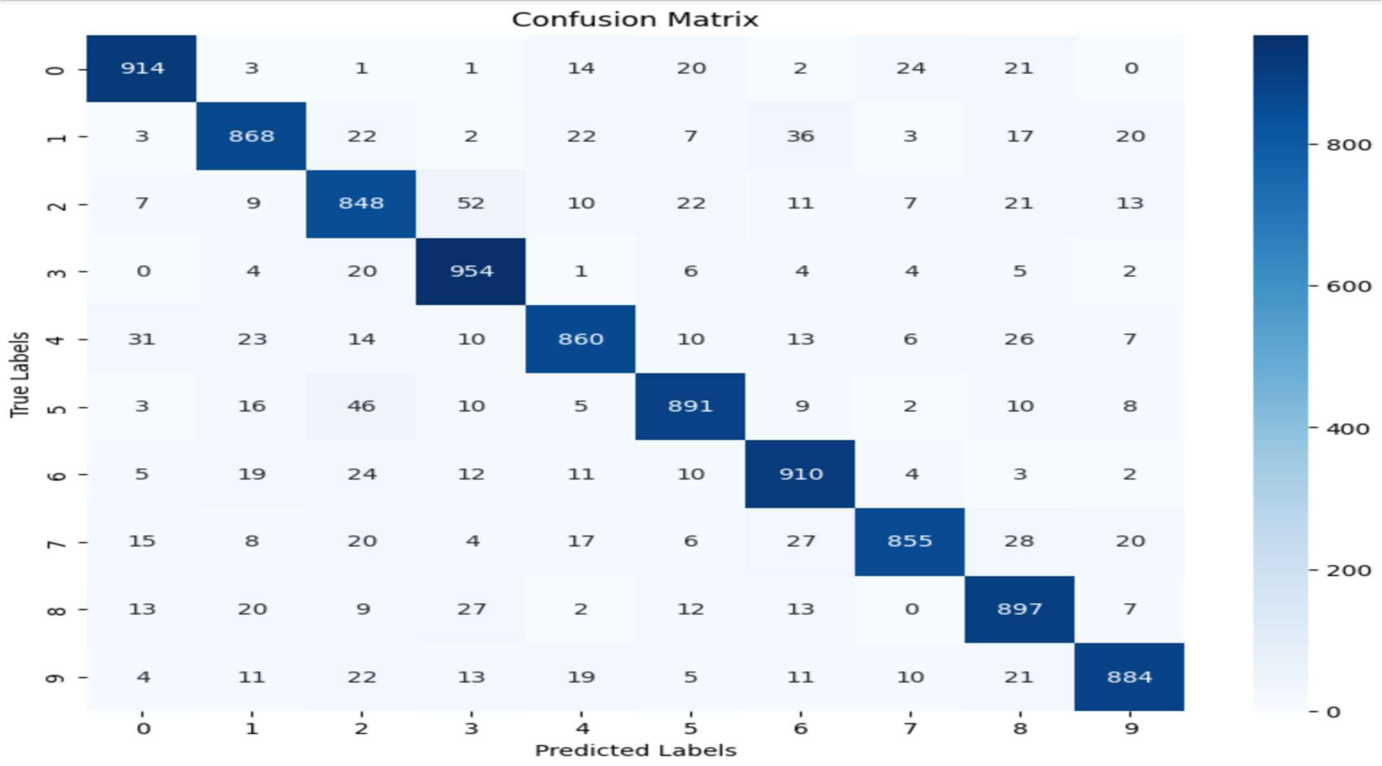
The confusion matrix visualizes the model's performance in classifying each category of handwritten Hiragana characters. It provides information about the model's ability to correctly classify instances of each class and identify any patterns of misclassifications. From the confusion matrix, we can identify areas where the model performs well and areas where it struggles, helping to identify potential areas for improvement.

Experiment 1



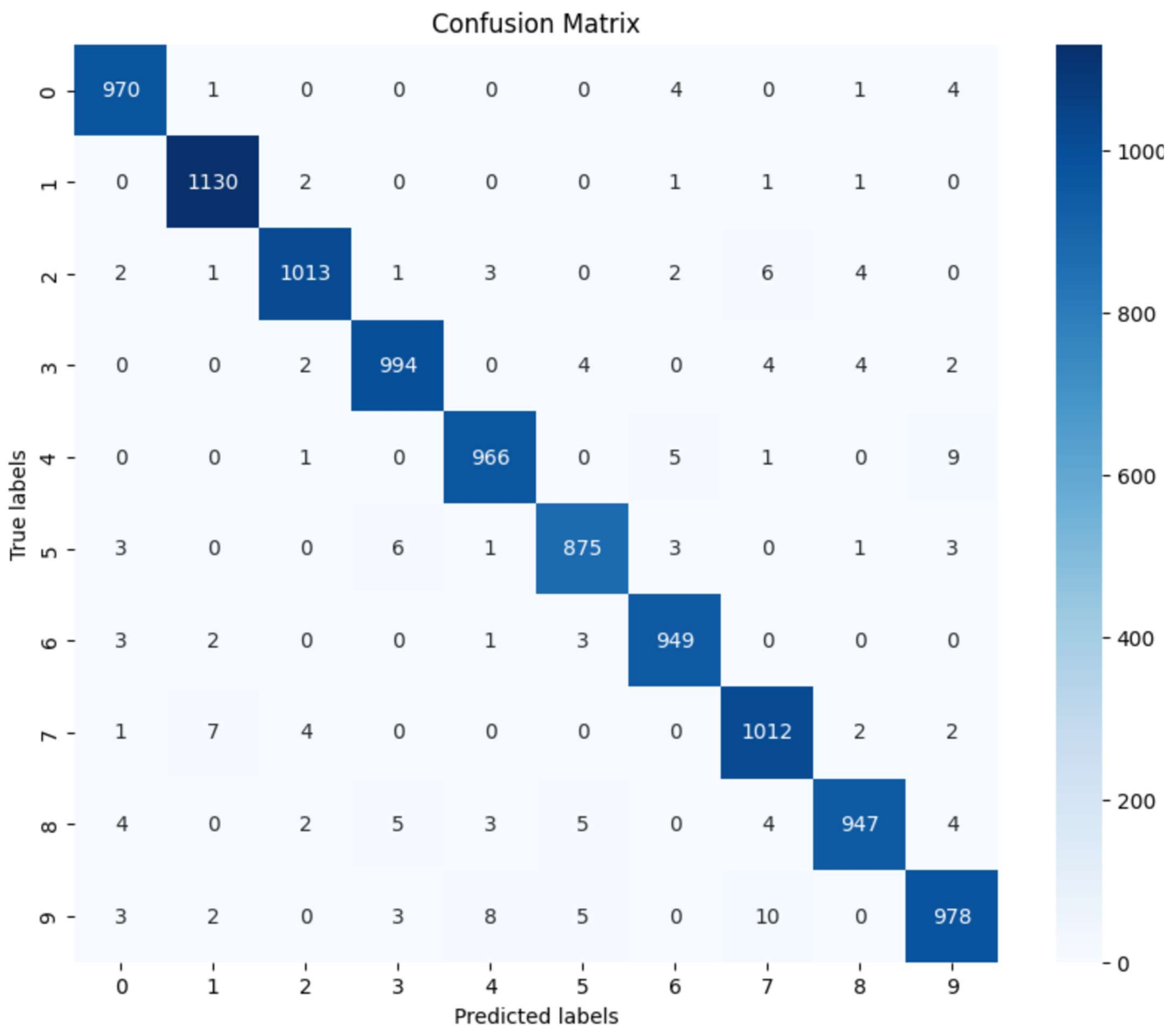
True Labels \ Predicted Labels	0	1	2	3	4	5	6	7	8	9
0	899	2	2	2	20	16	2	34	20	3
1	4	866	20	2	14	5	45	8	17	19
2	7	13	864	32	9	14	21	10	22	8
3	1	7	39	930	2	6	3	3	8	1
4	22	16	15	10	860	10	19	6	31	11
5	4	11	53	10	5	883	15	2	11	6
6	4	6	23	10	15	5	922	6	6	3
7	12	15	16	4	22	6	19	851	34	21
8	12	14	15	21	3	9	11	1	911	3
9	8	15	28	5	22	7	8	10	22	875

Experiment 2



True Labels \ Predicted Labels	0	1	2	3	4	5	6	7	8	9
0	914	3	1	1	14	20	2	24	21	0
1	3	868	22	2	22	7	36	3	17	20
2	7	9	848	52	10	22	11	7	21	13
3	0	4	20	954	1	6	4	4	5	2
4	31	23	14	10	860	10	13	6	26	7
5	3	16	46	10	5	891	9	2	10	8
6	5	19	24	12	11	10	910	4	3	2
7	15	8	20	4	17	6	27	855	28	20
8	13	20	9	27	2	12	13	0	897	7
9	4	11	22	13	19	5	11	10	21	884

Experiment 3



	0	1	2	3	4	5	6	7	8	9
0	970	1	0	0	0	0	4	0	1	4
1	0	1130	2	0	0	0	1	1	1	0
2	2	1	1013	1	3	0	2	6	4	0
3	0	0	2	994	0	4	0	4	4	2
4	0	0	1	0	966	0	5	1	0	9
5	3	0	0	6	1	875	3	0	1	3
6	3	2	0	0	1	3	949	0	0	0
7	1	7	4	0	0	0	0	1012	2	2
8	4	0	2	5	3	5	0	4	947	4
9	3	2	0	3	8	5	0	10	0	978

5.4 Limitations

Experiment 1

Despite achieving high training accuracy, the model's performance on the testing dataset is lower, indicating potential overfitting or limitations in generalization. Additionally, the learning curve shows signs of limited improvement in testing loss over epochs, suggesting that the model may have reached its capacity to learn from the training data. Furthermore, the confusion matrix may reveal specific characters or classes that the model consistently misclassifies, highlighting areas for further investigation and model refinement.

Experiment 2

While early stopping is generally a beneficial regularization technique to prevent overfitting and improve model generalization, it can also introduce limitations and challenges:

- **Premature Convergence**: Early stopping may halt the training process before the model has fully converged to the optimal solution. In some cases, the validation loss may fluctuate, leading to early stopping being triggered prematurely, preventing the model from reaching its full potential.
- **Underfitting Risk**: Setting the early stopping criteria too aggressively, such as stopping training after a small number of epochs without improvement, may lead to underfitting. This means the model may not have had sufficient time to learn from the training data and may generalize poorly to unseen examples.
- **Sensitivity to Hyperparameters**: The effectiveness of early stopping depends on the choice of hyperparameters, such as the patience parameter (the number of epochs to wait before stopping) and the threshold for determining improvement. Suboptimal hyperparameter settings may result in premature stopping or failure to prevent overfitting effectively.

Experiment 3

- **Complexity and Training Time**: The introduction of L2 regularization and a deeper neural network architecture increases the complexity of the model, potentially leading to longer training times and increased computational resources required. This limitation could hinder the scalability of the model, particularly in scenarios where training data is large or computational resources are limited.
- **Hyperparameter Tuning Sensitivity**: Similar to Experiment 2, Experiment 3 is likely to be sensitive to hyperparameter choices, including the strength of L2 regularization and the architecture of the deeper neural network. Suboptimal hyperparameter settings may result in ineffective regularization, leading to overfitting or underfitting of the model.
- **Overfitting Risk**: Despite the incorporation of L2 regularization and a deeper architecture to improve generalization, there remains a risk of overfitting, especially if the model capacity is not properly balanced with the complexity of the dataset. Monitoring for signs of overfitting, such as a significant gap between training and testing performance or divergence of training and testing losses, is crucial to address this limitation.
- **Interpretability and Explainability**: Deeper neural network architectures can inherently be less interpretable and more challenging to explain compared to simpler models. As the model complexity increases, understanding the underlying patterns learned by the model becomes more difficult, limiting interpretability and the ability to gain insights from the model's predictions.
- **Data Quality and Quantity Dependency**: The effectiveness of Experiment 3 may be heavily reliant on the quality and quantity of the training data. Insufficient or noisy data may hinder the model's ability to learn meaningful patterns, leading to suboptimal performance despite the regularization techniques employed. Moreover, biases or imbalances in the training data could affect the model's generalization ability, exacerbating limitations related to overfitting or underfitting.

6. Presentation of of Remaining Issues and Recommendations

Experiment 1

Remaining Issues

Despite achieving notable results, several challenges and limitations remain:

- **Overfitting**: The model exhibits signs of overfitting, as evidenced by the significant disparity between training and testing accuracies. This suggests that the model may have memorized the training data too well and struggles to generalize to unseen data.
- **Limited Generalization**: The model's performance on the testing dataset suggests limited generalization capabilities, indicating potential challenges in classifying unseen handwritten Hiragana characters accurately.

Recommendations

To address the identified limitations and enhance the model's performance, the following recommendations are proposed:

- **Regularization Techniques**: Implement additional regularization techniques such as dropout or L2 regularization to mitigate overfitting and improve the model's generalization capabilities.
- **Model Complexity Adjustment**: Experiment with modifying the model's architecture complexity, such as adding or removing layers, to strike a better balance between model capacity and generalization ability.
- **Data Augmentation Strategies**: Augment the training data using techniques like rotation, scaling, or translation to diversify the dataset and enhance the model's ability to generalize to variations in handwritten Hiragana characters.
- **Error Analysis**: Conduct a thorough error analysis to identify common misclassifications and areas of weakness in the model's performance. By understanding these errors, targeted adjustments can be made to refine the model's decision boundaries and improve accuracy.

By implementing these recommendations, it is anticipated that the model's performance and generalization capabilities will improve, leading to more accurate and robust classification of handwritten Hiragana characters.

Experiment 2

Despite the promising results obtained from Experiment 2, several remaining issues and areas for improvement have been identified:

Remaining Issues:

- **Limited Improvement in Testing Loss**: While the training loss steadily decreased over epochs, the testing loss exhibited fluctuations and eventually increased slightly after the 12th epoch. This suggests that the model's performance may have plateaued, indicating potential limitations in further improving its generalization capabilities.
- **Potential Underfitting**: Early stopping prevented overfitting by halting the training process when the testing loss stopped improving. However, setting the early stopping criteria too aggressively may have also increased the risk of underfitting, where the model fails to capture the underlying patterns in the data adequately.
- **Sensitivity to Hyperparameters**: The effectiveness of early stopping relies heavily on the choice of hyperparameters, such as the patience parameter and the threshold for determining improvement. Suboptimal hyperparameter settings may lead to premature stopping or failure to prevent overfitting effectively.

Recommendations:

- **Fine-Tuning Early Stopping Parameters**: Conduct a thorough analysis of the early stopping parameters, such as the patience parameter and the threshold for improvement, to determine the optimal values for the specific dataset and model architecture. Fine-tuning these parameters can help strike a balance between preventing overfitting and allowing the model to converge to the optimal solution.

- **Exploration of Alternative Regularization Techniques**: While early stopping is effective in preventing overfitting, exploring alternative regularization techniques, such as dropout or L2 regularization, may further enhance the model's generalization capabilities. By incorporating multiple regularization techniques, the model can better adapt to variations in the data and improve its robustness.
- **Evaluation of Model Complexity**: Assess the complexity of the model architecture and consider whether adjustments are necessary to strike a better balance between model capacity and generalization ability. Experimenting with modifications to the architecture, such as adding or removing layers, can help optimize the model's performance and prevent underfitting or overfitting.
- **Data Augmentation Strategies**: Implement data augmentation techniques, such as rotation, scaling, or translation, to augment the training data and increase its diversity. By exposing the model to a wider range of variations in the data, data augmentation can improve the model's ability to generalize to unseen examples and enhance overall performance.
- **Error Analysis and Model Refinement:** Conduct a comprehensive error analysis to identify common misclassifications and areas of weakness in the model's performance. Based on the findings from the error analysis, refine the model's decision boundaries and make targeted adjustments to improve accuracy and address specific challenges.

Experiment 3

Remaining Issues:

- **Computational Complexity and Training Time**: Running Experiment 3 with L2 regularization and a deeper neural network architecture significantly increases computational demands and training time. The extended training duration, exceeding three hours for 500 epochs, poses practical constraints on further hyperparameter tuning and experimentation, limiting the exploration of alternative configurations to optimize model performance.
- **Risk of Overfitting**: Despite the incorporation of L2 regularization and deeper architecture to improve generalization, there remains a risk of overfitting due to the model's increased complexity. The lengthy training time exacerbates this risk, as it becomes challenging to detect signs of overfitting early in the training process and adjust hyperparameters accordingly.

Recommendations:

- **Efficient Hyperparameter Optimization**: Given the extended training time, prioritize efficient hyperparameter optimization techniques such as Bayesian optimization or random search. These methods can help explore the hyperparameter space more effectively, enabling the discovery of optimal configurations within a reasonable timeframe.
- Parallelization and Distributed Training: Utilize parallelization techniques and distributed training frameworks to expedite the training process and alleviate computational burdens. Distributing the workload across multiple GPUs or leveraging cloud computing resources can significantly reduce training time and facilitate faster experimentation with different hyperparameter settings.
- **Model Compression and Optimization**: Investigate techniques for model compression and optimization to reduce the computational complexity of Experiment 3 without sacrificing performance. Techniques such as pruning, quantization, and knowledge

distillation can help create more efficient models that require less computational resources for training and inference.

- **Early Stopping with Checkpoints**: Implement early stopping with checkpointing mechanisms to monitor the model's performance during training and halt the process if signs of overfitting are detected. By periodically saving model checkpoints and evaluating performance on a validation dataset, training can be terminated early if further improvement is unlikely, saving computational resources.
- **Batch Size Adjustment**: Experiment with adjusting the batch size during training to find a balance between computational efficiency and model convergence. Smaller batch sizes may lead to faster convergence but can increase training time, while larger batch sizes can expedite training but may affect model generalization.

7. Model Comparison and Evaluation

Metric	Training Accuracy	Testing Accuracy	Number of epochs	Total loss
Experiment 1	99.98 %	88.61 %	500	10.2793
Experiment 2	99.11%	88.81 %	16	33469.4187
Experiment 3	99.82%	98.34%	500	0.0609

Based on the provided metrics, Experiment 3: L2 Regularization with Multiple Hidden Layers Architecture appears to be the best model among the three experiments. Here's the rationale:

- **Testing Accuracy**: Experiment 3 achieved the highest testing accuracy of 98.34%, indicating its superior ability to generalize to unseen data compared to the other experiments.
- **Training Accuracy**: While Experiment 1 achieved the highest training accuracy of 99.98%, it exhibited a significant performance drop when evaluated on the testing dataset. Experiment 3 achieved a slightly lower training accuracy of 99.82% but maintained a significantly higher testing accuracy, suggesting better generalization capability.
- **Number of Epochs**: Both Experiment 1 and Experiment 3 were trained for 500 epochs, while Experiment 2 was halted after 16 epochs due to early stopping. Despite the shorter training duration, Experiment 2 exhibited relatively high testing accuracy, but Experiment 3 outperformed it with a longer training duration.
- **Total Loss**: Experiment 3 had the lowest total loss of 0.0609, indicating better overall performance in minimizing errors compared to the other experiments.

8. Conclusion

In conclusion, Experiment 3, which employed the L2 Regularization with Multiple Hidden Layers Architecture, emerges as the most effective model for classifying handwritten Hiragana characters. Despite the longer training time and computational complexity associated with the deeper architecture and L2 regularization, Experiment 3 demonstrated superior generalization performance and lower loss compared to the other experiments. This suggests that the model's enhanced capacity to capture complex patterns while mitigating overfitting through regularization techniques contributed to its superior performance. Therefore, Experiment 3 is recommended for further refinement and deployment in real-world applications requiring accurate classification of handwritten characters. Additionally, the inclusion of multiple hidden layers played a crucial role in improving the model's efficiency and effectiveness in character recognition tasks.