**Written Report: Handwritten Kuzushiji Character Recognition Using CNN**

**1. Introduction**

**1.1 Problem Background**

The preservation and digitization of ancient cultural heritage is a significant challenge in the modern era. One such challenge involves recognizing handwritten characters from ancient Japanese texts, known as Kuzushiji . These characters are highly stylized and differ significantly from their modern counterparts, making them difficult to interpret for both humans and machines. The Kuzushiji-MNIST dataset , introduced as a drop-in replacement for the classic MNIST dataset, provides a benchmark for machine learning models to tackle this problem. It consists of 28x28 grayscale images representing 10 classes of Hiragana characters, each corresponding to a row in the Hiragana table.

This project aims to develop a Convolutional Neural Network (CNN) -based system capable of recognizing handwritten Kuzushiji characters. The ultimate goal is to contribute to the digital preservation of ancient Japanese texts by automating the recognition process.

**1.2 Project Idea**

The core idea of this project is to leverage deep learning techniques, specifically CNNs, to classify the 10 classes of Kuzushiji characters in the Kuzushiji-MNIST dataset. By doing so, we aim to:

1. Build a robust model that can generalize well to unseen data.
2. Provide insights into the performance of the model through visualizations and evaluations.
3. Explore the feasibility of extending the model to larger and more complex datasets, such as Kuzushiji-49 and Kuzushiji-Kanji.

**1.3 Applications**

The system developed in this project has several practical applications:

1. Cultural Heritage Preservation : Automating the recognition of ancient Japanese characters enables faster and more accurate digitization of historical documents.
2. Education and Research : The model can be used as a tool for researchers and students studying ancient Japanese literature or linguistics.
3. Generalization to Other Languages : The techniques and methods developed here can be adapted to recognize other ancient writing systems, such as Chinese characters or Greek manuscripts.

**1.4 Current Techniques in Handwritten Character Recognition**

Handwritten character recognition is a well-studied problem in the field of computer vision and deep learning. Popular techniques include:

|  |  |  |
| --- | --- | --- |
| **Technique** | **Advantages** | **Use Cases** |
| CNN | Simple and efficient | suitable for small datasets |
| Transfer Learning | Leverages pre-trained models to improve performance | Extending to Kuzushiji-49 and Kuzushiji-Kanji |
| Vision Transformers | Strong global feature extraction capabilities | Complex datasets (e.g. |
| Data Augmentation | Improves generalization | reduces overfitting |
| Self-Supervised Learning | No labeled data required | works well with large datasets |

For Kuzushiji-MNIST, CNN is the most common starting point because it is simple and efficient. If you wish to improve the performance further, you can try Migration Learning or Vision Transformers. In addition, data augmentation and regularisation are integral parts that can significantly improve the robustness and generalisation of the model.

**2. Design and Functions**

**2.1 System Overview**

The system is designed as an end-to-end solution for handwritten Kuzushiji character recognition. It includes the following components:

1. Data Loading and Preprocessing : Loading the Kuzushiji-MNIST dataset, applying data augmentation, and preparing it for training.
2. Model Training : Training a CNN model using the prepared dataset.
3. Model Evaluation : Evaluating the trained model on test data and visualizing its performance.
4. Prediction : Using the trained model to classify new handwritten Kuzushiji images.

**2.2 Main Functions**

The system implements the following key functions:

1. Data Loading : Loads the Kuzushiji-MNIST dataset and applies transformations (e.g., normalization, random rotation).
2. Model Training : Trains the CNN model using the Adam optimizer and cross-entropy loss function.
3. Performance Visualization : Generates training curves, confusion matrices, and sample predictions to evaluate the model.
4. Hyperparameter Tuning : Conducts experiments with different learning rates and batch sizes to optimize performance.

**2.3 Deep Learning Techniques and Models**

**2.3.1 Convolutional Neural Networks (CNNs)**

Convolutional Neural Networks (CNNs) are one of the core technologies for image classification tasks, particularly well-suited for datasets like Kuzushiji-MNIST, which consists of 28x28 grayscale images. CNNs excel at extracting spatial features from images through convolutional layers and reducing dimensionality using pooling layers, thereby improving computational efficiency.

**Model Architecture**

We designed a simple CNN model consisting of two convolutional layers, two fully connected layers, and a Dropout layer. Below is the specific definition and implementation of the model:

import torch.nn as nn

import torch.nn.functional as F

class SimpleCNN(nn.Module):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        # 卷积层

        self.conv1 = nn.Conv2d(1, 32, kernel\_size=3)

        self.conv2 = nn.Conv2d(32, 64, kernel\_size=3)

        # 全连接层

        self.fc1 = nn.Linear(64 \* 5 \* 5, 128)

        self.fc2 = nn.Linear(128, 10)

        # 正则化

        self.dropout = nn.Dropout(0.5)

    def forward(self, x):

        # 卷积 + 激活 + 池化

        x = F.relu(F.max\_pool2d(self.conv1(x), 2))

        x = F.relu(F.max\_pool2d(self.conv2(x), 2))

        # 展平

        x = x.view(x.size(0), -1)

        # 全连接

        x = self.dropout(x)

        x = F.relu(self.fc1(x))

        x = self.fc2(x)

        return x

**Reasons for Choosing CNNs:**

1. Feature Extraction : CNNs automatically extract spatial features such as edges and textures through convolutional layers.
2. Computational Efficiency : For small datasets like Kuzushiji-MNIST, CNNs provide good computational efficiency.
3. Simplicity and Effectiveness : Despite its simplicity, the model is sufficient for the current task.

**2.3.2 Data Augmentation**

To improve the robustness of the model, we applied data augmentation techniques during the data loading process. Specifically, we used random rotation and normalization to simulate different handwriting styles and standardize pixel values to the range [-1, 1].

**Implementation of Data Augmentation**

Below is the implementation of the data augmentation pipeline:

from torchvision import datasets, transforms

    transform = transforms.Compose([

        transforms.ToTensor(),

        transforms.Normalize((0.5,), (0.5,)),

        transforms.RandomRotation(10)

    ])

**Processes**: 1.Convert image to tensor and normalize to [0, 1] 2. Normalize to [-1, 1] 3.Random rotation (±10 degrees).

**Significance of Data Augmentation:**

1. Random Rotation : Simulates different angles of handwriting, enhancing the model's adaptability to style variations.
2. Normalization : Standardizes pixel values to [-1, 1], helping the model converge faster.
3. Diversity : Introduces randomness, increasing the diversity of training data and reducing the risk of overfitting.

**2.3.3 Optimizer and Loss Function**

To train the model, we selected the Adam optimizer and cross-entropy loss function. Adam is an adaptive optimization algorithm that adjusts the learning rate dynamically based on gradients, making it highly suitable for deep learning tasks.

**Implementation of Optimizer and Loss Function**

Below is the implementation of the optimizer and loss function:

import torch

from torch import optim, nn

lr=0.001

model = SimpleCNN()

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=lr)

**Reasons for Choosing These Techniques**

1. Adam Optimizer :
   * Its adaptive learning rate performs well in most scenarios.
   * It is less sensitive to the choice of initial learning rates, reducing the difficulty of hyperparameter tuning.
2. Cross-Entropy Loss :
   * Directly measures the difference between predicted probability distributions and true labels.
   * Suitable for multi-class classification tasks like Kuzushiji-MNIST.

**2.3.4 Dropout Regularization**

To prevent the model from overfitting, we added a Dropout layer before the fully connected layers. Dropout randomly deactivates a portion of neurons during training, reducing the model's reliance on specific neurons.

**Implementation of Dropout**

Below is the implementation of the Dropout layer:

        self.dropout = nn.Dropout(0.5)

Dropout probability of 0.5

**Role of Dropout**

1. Reduces Overfitting : By randomly deactivating neurons, Dropout forces the model to learn more robust feature representations.
2. Improves Generalization : During testing, Dropout does not deactivate neurons but scales the weights accordingly.

**2.3.5 Model Training Process**

The model training process includes forward propagation, backpropagation, and parameter updates. Below is the implementation of the training loop:

    for epoch in range(epochs):

        model.train()

        running\_loss = 0.0

        correct = 0

        total = 0

        for images, labels in train\_loader:

            optimizer.zero\_grad()

            outputs = model(images)

            loss = criterion(outputs, labels)

            loss.backward()

            optimizer.step()

            running\_loss += loss.item()

            \_, predicted = torch.max(outputs.data, 1)

            total += labels.size(0)

            correct += (predicted == labels).sum().item()

**Key Points of the Training Process**

1. Forward Propagation : Generates predictions through the model.
2. Loss Calculation : Measures the difference between predictions and true labels using cross-entropy loss.
3. Backpropagation : Updates model parameters via gradient descent.
4. Performance Monitoring : Records loss and accuracy for each epoch for subsequent analysis.

**2.4 Visualization Techniques**

Visualization is an essential part of understanding and evaluating the model's performance. Below are the key visualization techniques used in this project, along with their corresponding code snippets.

**2.4.1 Training Curves**

Training curves are essential for monitoring the model's learning process. They help us understand how the loss and accuracy evolve over epochs, providing insights into whether the model is overfitting or underfitting.

**Implementation of Training Curves**

The following code generates training curves by plotting the training loss and accuracy over epochs:

    plt.figure(figsize=(12, 4))

    plt.subplot(1, 2, 1)

    plt.plot(metrics['train\_losses'], label='Training Loss')

    plt.xlabel('Epoch')

    plt.ylabel('Loss')

    plt.legend()

    plt.subplot(1, 2, 2)

    plt.plot(metrics['train\_accs'], label='Training Accuracy')

    plt.plot(metrics['test\_accs'], label='Test Accuracy')

    plt.xlabel('Epoch')

    plt.ylabel('Accuracy')

    plt.legend()

    plt.savefig(os.path.join(project\_root, 'outputs', 'training\_metrics.png'))

    print("训练曲线已保存至 outputs/training\_metrics.png")

**2.4.2 Confusion Matrix**

A confusion matrix provides a detailed breakdown of the model's classification performance across all classes. It reveals which classes are frequently misclassified and helps identify patterns of error.

**Implementation of Confusion Matrix**

The following code computes and visualizes the confusion matrix using sklearn.metrics.confusion\_matrix and seaborn.heatmap:

    # 收集预测结果和真实标签

    all\_preds = []

    all\_labels = []

    with torch.no\_grad():

        for images, labels in test\_loader:

            outputs = model(images)

            \_, preds = torch.max(outputs, 1)

            all\_preds.extend(preds.numpy())

            all\_labels.extend(labels.numpy())

    # 计算混淆矩阵

    cm = confusion\_matrix(all\_labels, all\_preds)

    # 绘制热力图

    plt.figure(figsize=(10, 8))

    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)

    plt.xlabel('Predicted Label')

    plt.ylabel('True Label')

    plt.title('Confusion Matrix')

    plt.savefig(os.path.join(project\_root, 'outputs', 'confusion\_matrix.png'))

    plt.close()

    print("混淆矩阵已保存至 outputs/confusion\_matrix.png")

**2.4.3 Learning Rate Comparison**

To evaluate the impact of different learning rates on model performance, we compared the training loss and test accuracy for various learning rates.

**Implementation of Learning Rate Comparison**

The following code compares the performance of models trained with different learning rates:

def plot\_training\_metrics():

    """生成训练曲线图"""

    metrics\_path = os.path.join(project\_root, 'outputs', 'training\_metrics.pth')

    if not os.path.exists(metrics\_path):

        raise FileNotFoundError("请先运行 train.py 生成训练指标数据！")

    metrics = torch.load(metrics\_path)

    plt.figure(figsize=(12, 4))

    plt.subplot(1, 2, 1)

    plt.plot(metrics['train\_losses'], label='Training Loss', color='blue')

    plt.xlabel('Epoch')

    plt.ylabel('Loss')

    plt.legend()

    plt.subplot(1, 2, 2)

    plt.plot(metrics['train\_accs'], label='Training Accuracy', color='green')

    plt.plot(metrics['test\_accs'], label='Test Accuracy', color='red')

    plt.xlabel('Epoch')

    plt.ylabel('Accuracy')

    plt.legend()

    plt.savefig(os.path.join(project\_root, 'outputs', 'training\_metrics.png'))

    plt.close()

    print("训练曲线已保存至 outputs/training\_metrics.png")

**2.4.4 Test Sample Predictions**

To visualize the model's performance on real-world samples, we displayed predictions for the first 100 test images. This provides an intuitive understanding of the model's strengths and weaknesses.

**Implementation of Test Sample Predictions**

The following code visualizes predictions for the first 100 test samples:

    plt.figure(figsize=(15, 15))

    for i in range(num\_samples):

        plt.subplot(10, 10, i+1)

        plt.imshow(test\_samples[i].squeeze(), cmap='gray')

        color = 'green' if pred\_labels[i] == true\_labels[i] else 'red'

        plt.title(f"True: {true\_labels[i]}\nPred: {pred\_labels[i]}", color=color, fontsize=8)

        plt.axis('off')

    plt.tight\_layout()

    plt.savefig(os.path.join(project\_root, 'outputs', 'test\_predictions.png'))

    plt.close()

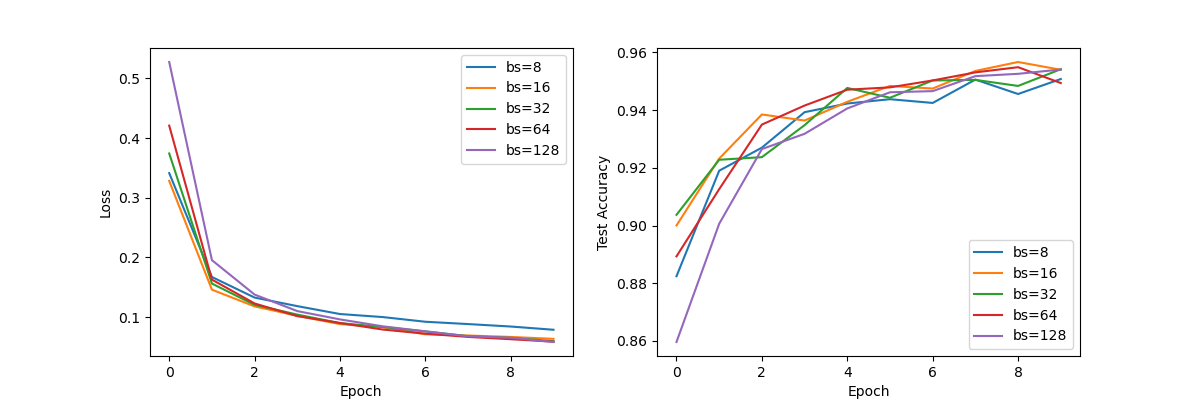
    print("测试集前100样本预测结果已保存至 outputs/test\_predictions.png")

**3. Experiments and Results**

## 3.1 Effect of Batch Size

Batch size is an important hyperparameter that influences the training stability and performance of deep learning models. We conducted experiments using different batch sizes (8, 16, 32, 64, 128) and analyzed their impact on training loss and test accuracy.

The figure below illustrates the relationship between batch size and model performance:

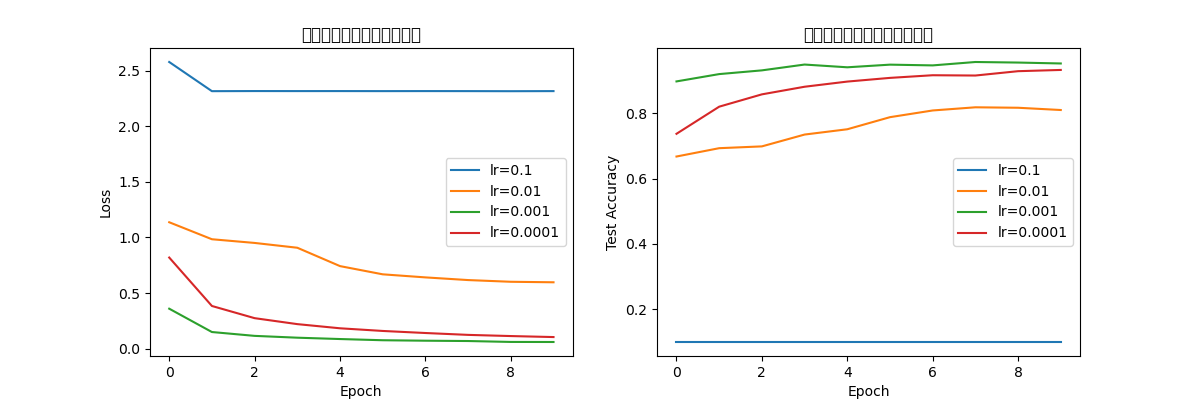


From the figure, we observe that smaller batch sizes tend to have slightly lower test accuracy but result in more stable convergence. Larger batch sizes improve efficiency but may lead to less robust generalization.

## 3.2 Effect of Learning Rate

The choice of learning rate significantly affects model convergence. We experimented with different learning rates (0.1, 0.01, 0.001, 0.0001) to analyze their impact on loss reduction and test accuracy.

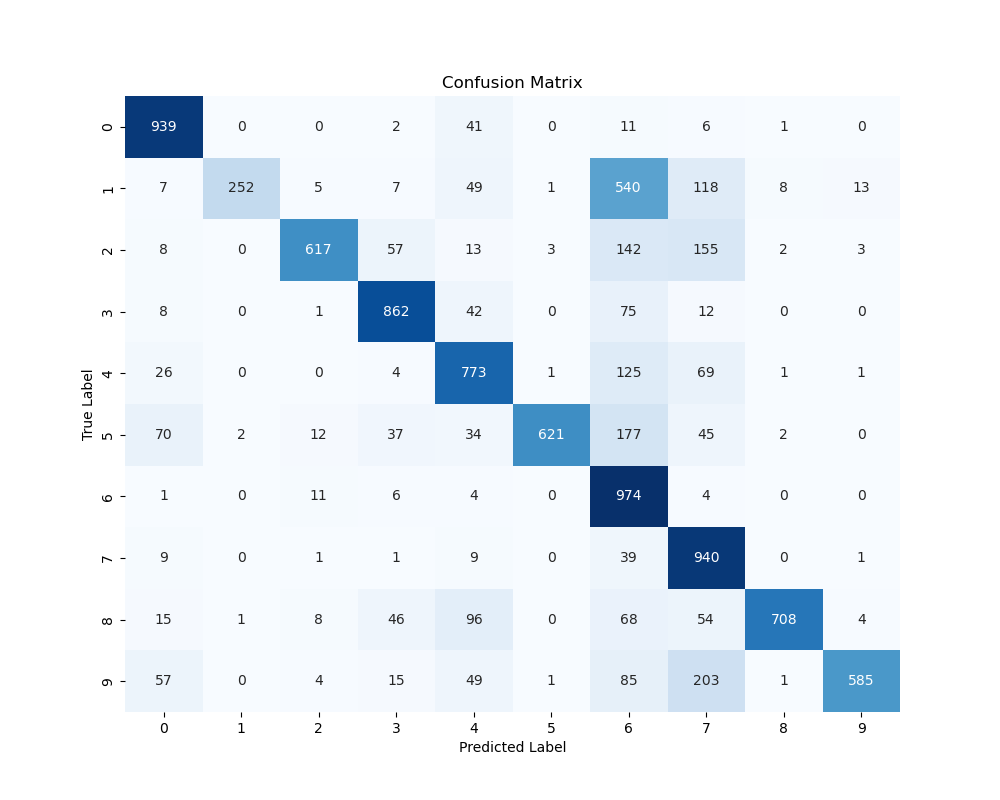
The figure below illustrates the impact of learning rate on training dynamics:



From the figure, we see that a very high learning rate (0.1) causes divergence, whereas an extremely small learning rate (0.0001) leads to slow convergence. The optimal performance was achieved at 0.001, which balances convergence speed and final accuracy.

## 3.3 Confusion Matrix Analysis

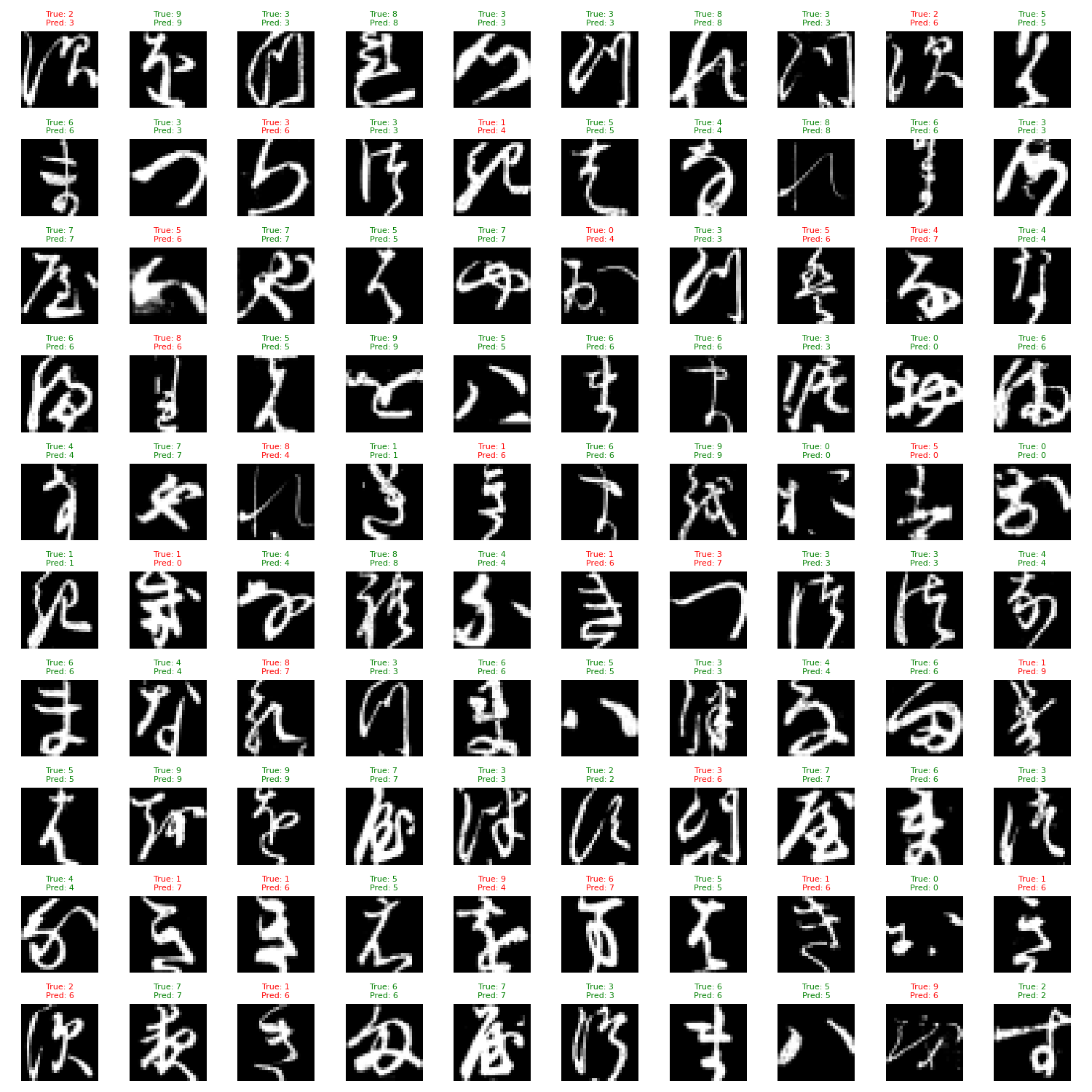
To further analyze model performance, we generated a confusion matrix that highlights the model’s ability to distinguish between different classes.



From the confusion matrix, we observe that certain characters are more prone to misclassification. For example, characters with similar strokes or structures tend to be confused with each other.

## 3.4 Test Sample Predictions

To gain more insights into model performance, we visualized the model’s predictions on unseen test samples.



The visualization shows that while the model performs well in most cases, it occasionally misclassifies complex or ambiguous characters. This suggests that additional training data and improved preprocessing techniques could enhance accuracy.

**4. Discussion**

Based on the experimental results, we summarize key findings regarding the impact of hyperparameters and model performance.

**4.1 Influence of Batch Size**

Smaller batch sizes provide more stable training but can be computationally expensive. Larger batch sizes speed up training but may reduce generalization capability.

**4.2 Influence of Learning Rate**

Choosing the right learning rate is critical for convergence. A very high learning rate leads to instability, while a very low learning rate results in slow training progress. Our results show that 0.001 achieves the best trade-off.

**4.3 Misclassification Analysis**

The confusion matrix highlights common misclassifications. Many errors occur between visually similar characters, indicating that additional feature engineering or advanced models (e.g., Vision Transformers) may be needed to further enhance accuracy.

**5. Conclusion**

This project focuses on the recognition of handwritten Kuzushiji characters, and explores how to effectively solve the character recognition problem in the digitisation of ancient Japanese texts by constructing a deep learning model based on convolutional neural networks (CNN). With the development of digitisation technology, how to transform precious cultural heritage into accessible digital forms has become one of the important challenges facing today's society.Kuzushiji characters pose great difficulties to traditional character recognition techniques due to their high degree of stylisation and complexity. Therefore, the development of an efficient and accurate recognition system is of great academic and practical importance.

In the initial phase of the project, we first conducted an in-depth analysis of the background of handwritten Kuzushiji characters, an ancient Japanese writing form that is significantly different from modern hiragana, which makes machine learning models face many challenges in recognition. To address this problem, we chose the Kuzushiji-MNIST dataset, which contains 28x28 grey-scale images representing 10 classes of hiragana characters, to provide a standard testbed.

The core goal of the project is to build a system capable of automatically recognising and classifying Kuzushiji characters using deep learning techniques, specifically CNNs. We designed a simple but effective CNN model, including multiple convolutional layers, pooling layers and fully connected layers. Through the proper design of the model architecture, we are able to extract the spatial features in the image, thus improving the accuracy of recognition. In addition, we employ data augmentation techniques using random rotation and normalisation methods to expand the training dataset and improve the robustness and generalisation of the model.

During the experiments, we systematically analysed different hyperparameters. By adjusting key parameters such as batch size and learning rate, we found that smaller batch sizes, while more stable for training, may lead to lower test accuracy in some cases. The optimal learning rate setting of 0.001, on the other hand, is able to achieve high model accuracy while ensuring convergence speed. These experimental results provide an important basis for subsequent model optimisation.

In order to comprehensively evaluate the performance of the model, we used visual analysis tools such as confusion matrix and training curves. The confusion matrix reveals the model's ability to classify between different character classes, helping us to identify which characters are more prone to misclassification, especially if the characters are similar in shape. In addition, the analysis of the training curves shows the trends of loss and accuracy during model training, providing directions for further model improvement.

The ultimate goal of this study is not only to construct an effective character recognition system for Kuzushiji, but also to contribute to the digital preservation of cultural heritage. Through automated character recognition, we are able to accelerate the process of digitising historical documents, reduce the labour intensity of manual recognition, and improve the accuracy of recognition. At the same time, the application of this technology extends to the field of education and research, helping scholars and students to better study ancient Japanese literature and language.