Project Report For

Introduction to AI (AI101B)

On

Hand Written

Digit Recognition

By

Aaryan Gautam -- 2024101161000003
Akash Choudhary – 2024101161000015
Aditya Chaudhary -- 2024101161000010

Session:2024-2026 (II Semester)

Under the supervision of

MR. APOORV JAIN (Assistant Professor)

KIET Group of Institutions, Delhi-NCR, Ghaziabad

DEPARTMENT OF COMPUTER APPLICATIONS
KIET GROUP OF INSTITUTIONS, DELHI-NCR, GHAZIABAD-201206

INTRODUCTION

In recent years, artificial intelligence (AI) and machine learning (ML) have seen significant advancements, especially in the field of image processing and pattern recognition. One of the foundational problems in this domain is Handwritten Digit Recognition, which involves classifying images of handwritten digits into their corresponding numeric labels (0–9).

This task is not only fundamental for learning and understanding machine learning concepts but also has practical applications in areas such as postal mail sorting, bank cheque processing, and form digitization. The ability of a machine to read and interpret handwritten text mimics human capabilities and opens doors to developing more complex document recognition systems.

To tackle this problem, we make use of the MNIST dataset, which is a large collection of labeled images of handwritten digits. Each image is a 28x28 grayscale pixel representation of a single digit.

In this project, a Convolutional Neural Network (CNN) — a type of deep learning model — is trained on the MNIST dataset to recognize and classify handwritten digits with high accuracy. CNNs are particularly well-suited for image-based tasks as they are capable of automatically detecting spatial hierarchies and local patterns in images.

This report outlines the step-by-step development of the digit recognition model, including data preprocessing, model design, training, evaluation, and visualization of results. The objective is to demonstrate how AI can be effectively applied to solve real-world pattern recognition tasks with minimal human intervention.

Methodology

To develop an effective handwritten digit recognition system, we utilized the MNIST dataset, which consists of 70,000 grayscale images of handwritten digits ranging from 0 to 9. Each image is 28x28 pixels in size. The dataset was divided into 60,000 images for training and 10,000 for testing. Before feeding the data into the model, the images were normalized by scaling the pixel values to a range between 0 and 1, and reshaped to include a single channel for compatibility with convolutional layers. The labels were converted to one-hot encoded vectors to enable multi-class classification. A Convolutional Neural Network (CNN) was built for this task due to its superior performance in image recognition. The model architecture included two convolutional layers with ReLU activation, each followed by max pooling, then a flattening layer and two dense layers — the last using softmax activation for outputting probabilities across 10 classes. The model was compiled using the Adam optimizer and categorical crossentropy as the loss function, and trained for five epochs with a validation split of 20% to monitor overfitting. During training, the accuracy and loss were recorded for both the training and validation sets. After training, the model was evaluated on the test dataset to obtain the final accuracy. A confusion matrix and classification report were generated to analyze the model's performance in detail. Visualizations were also created, including training accuracy and loss curves, the confusion matrix heatmap, and a comparison of actual versus predicted digits to assess the effectiveness of the model visually.

1. Dataset Description

The MNIST dataset, a widely used benchmark in image classification, was used in this project. It contains 70,000 grayscale images of handwritten digits ranging from 0 to 9.

Each image is of size 28x28 pixels. The dataset is split into 60,000 training images and 10,000 testing images.

2. Data Preprocessing

To prepare the data for model training, all images were normalized by scaling the pixel values from the range [0, 255] to [0, 1]. The images were also reshaped to include a single channel to match the input shape required by convolutional neural networks. The digit labels were converted into one-hot encoded vectors to facilitate multi-class classification

3. Model Architecture

A Convolutional Neural Network (CNN) was chosen due to its proven effectiveness in image recognition tasks. The architecture consists of two convolutional layers with ReLU activation functions, each followed by a max pooling layer. This is followed by a flatten layer that converts the feature maps into a 1D vector, then a dense (fully connected) layer with 128 neurons, and finally an output layer with 10 neurons using the softmax activation function to output class probabilities.4. Evaluation Metrics.

4. Model Compilation and Training

The model was compiled using the Adam optimizer and categorical crossentropy as the loss function. It was trained for 5 epochs with a validation split of 20% to monitor the model's performance on unseen data and prevent overfitting. During training, accuracy and loss metrics were tracked for both training and validation sets.

5. Model Evaluation

After training, the model's performance was evaluated on the test dataset to determine its final accuracy. Predictions were made on the test images, and a confusion matrix was generated to visualize the model's classification performance. Additionally, a detailed classification report was produced, displaying precision, recall, and F1-score for each digit class.

Python Code:

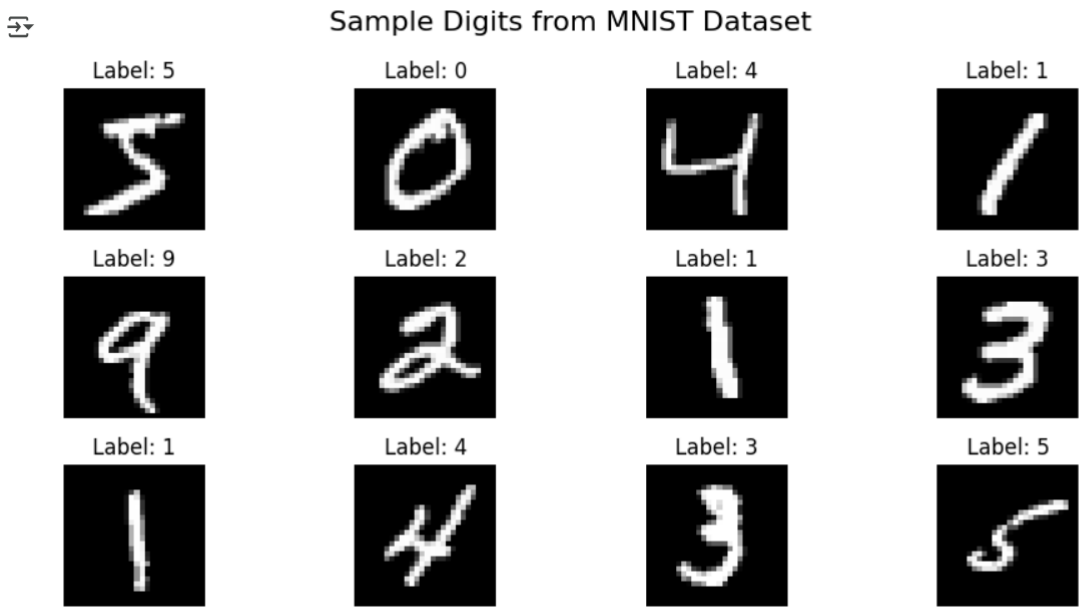
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import classification_report, confusion_matrix

# 1. Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# 2. Visualize some samples
plt.figure(figsize=(10, 5))
for i in range(12):
    plt.subplot(3, 4, i + 1)
    plt.imshow(x_train[i], cmap='gray')
    plt.title(f"Label: {y_train[i]}")
    plt.axis('off')
plt.suptitle('Sample Digits from MNIST Dataset', fontsize=16)
plt.tight_layout()
plt.show()
```

```
x_train = x_train.reshape(-1, 28, 28, 1) / 255.0
x_test = x_test.reshape(-1, 28, 28, 1) / 255.0
y_train_cat = to_categorical(y_train, 10)
y_test_cat = to_categorical(y_test, 10)

# 4. Create the model (Simple CNN)
model = Sequential([
    Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=(28,28,1)),
    MaxPooling2D(pool_size=(2,2)),
    Conv2D(64, kernel_size=(3,3), activation='relu'),
    MaxPooling2D(pool_size=(2,2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

# 5. Compile and train
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(x_train, y_train_cat, epochs=5, validation_split=0.2)

# 6. Plot training vs validation accuracy and loss
plt.figure(figsize=(12, 5))

# Accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
```

```
plt.legend()

# Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()

# 7. Evaluate on test data
test_loss, test_acc = model.evaluate(x_test, y_test_cat)
print(f"\nTest Accuracy: {test_acc:.4f}")

# 8. Confusion Matrix
y_pred = model.predict(x_test).argmax(axis=1)
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=range(10),
yticklabels=range(10))
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

```
# 9. Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# 10. Show some predictions
plt.figure(figsize=(10, 5))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.title(f"True: {y_test[i]}\nPred: {y_pred[i]}")
    plt.axis('off')
plt.tight_layout()
plt.suptitle("Predictions on Test Images", fontsize=16)
plt.show()
```

Output ScreenShot



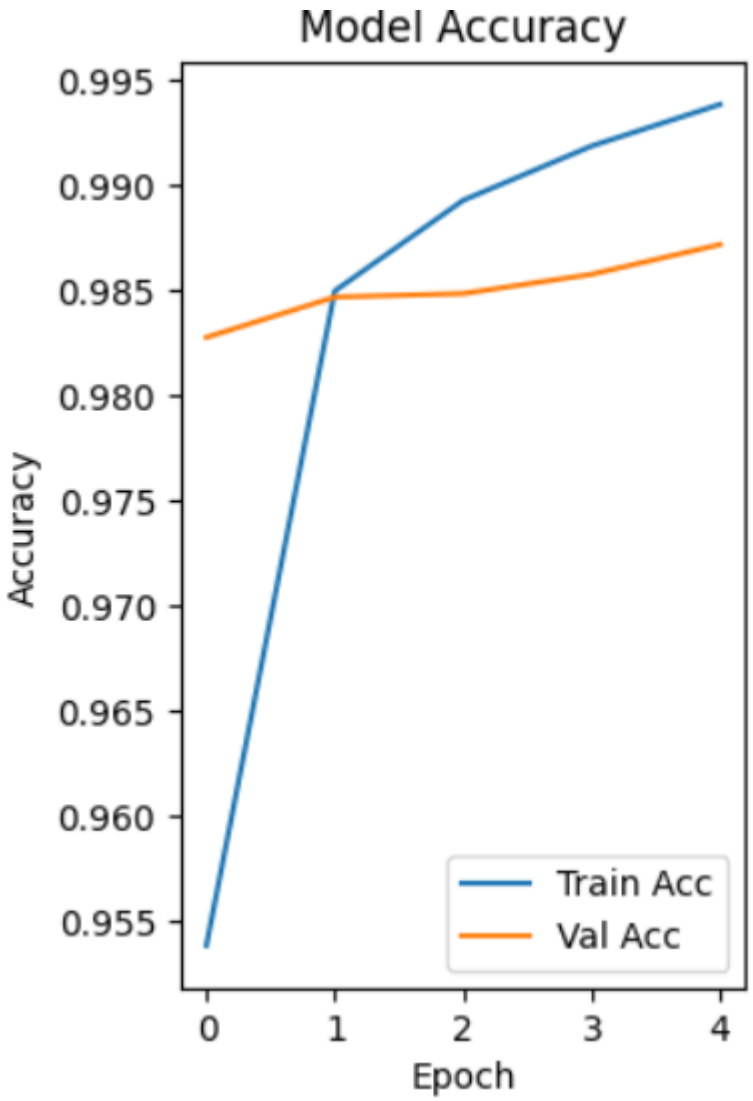
```
Epoch 1/5
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 58s 37ms/step - accuracy: 0.8968 - loss: 0.3389 - val_accuracy: 0.9827 - val_loss: 0.0576
Epoch 2/5
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 95s 46ms/step - accuracy: 0.9836 - loss: 0.0507 - val_accuracy: 0.9847 - val_loss: 0.0509
Epoch 3/5
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 66s 35ms/step - accuracy: 0.9894 - loss: 0.0317 - val_accuracy: 0.9848 - val_loss: 0.0535
Epoch 4/5
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 81s 35ms/step - accuracy: 0.9921 - loss: 0.0222 - val_accuracy: 0.9858 - val_loss: 0.0498
Epoch 5/5
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 83s 35ms/step - accuracy: 0.9946 - loss: 0.0150 - val_accuracy: 0.9872 - val_loss: 0.0451
```

```
<Figure size 1200x500 with 0 Axes>
<Figure size 1200x500 with 0 Axes>
```

Insights from Graphs

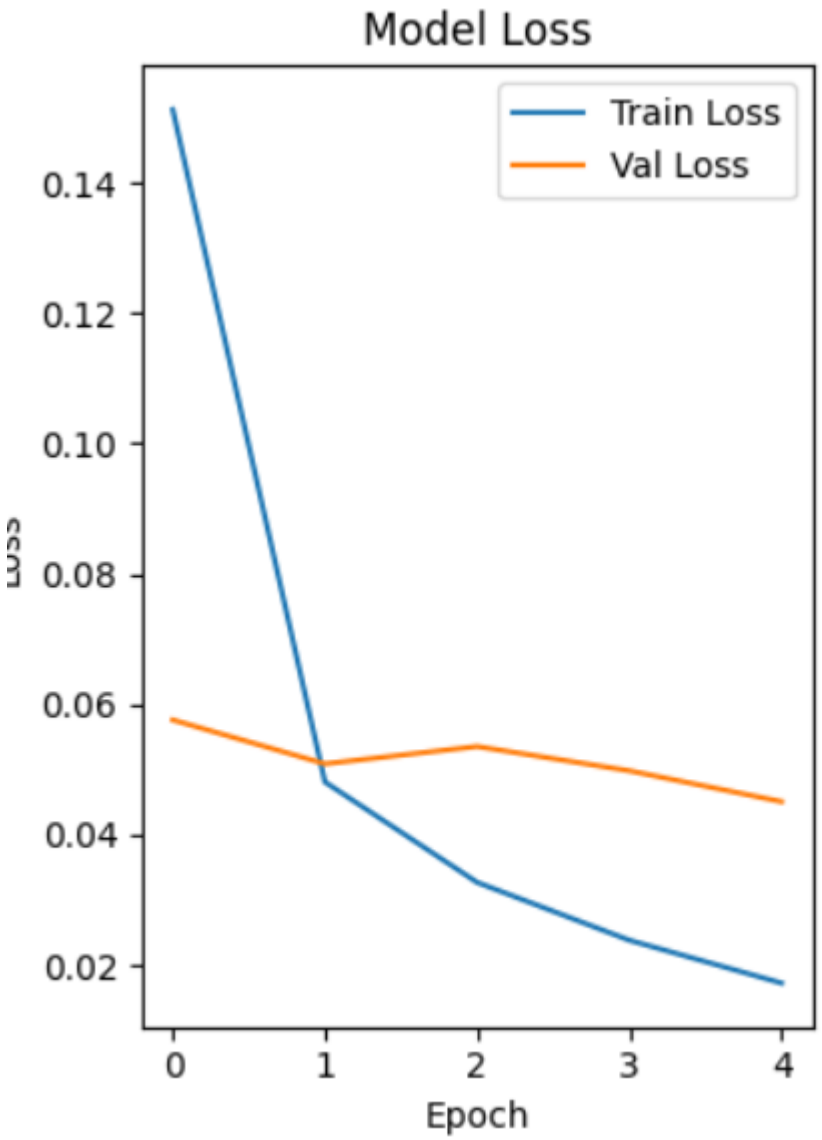
Training vs. Validation Accuracy

- The training and validation accuracy curves both show a steady upward trend across the epochs, indicating that the model is learning effectively. The validation accuracy closely follows the training accuracy, which suggests that the model is not overfitting and is generalizing well on unseen data. By the end of training, the validation accuracy reaches over 98%, reflecting the model's high performance.



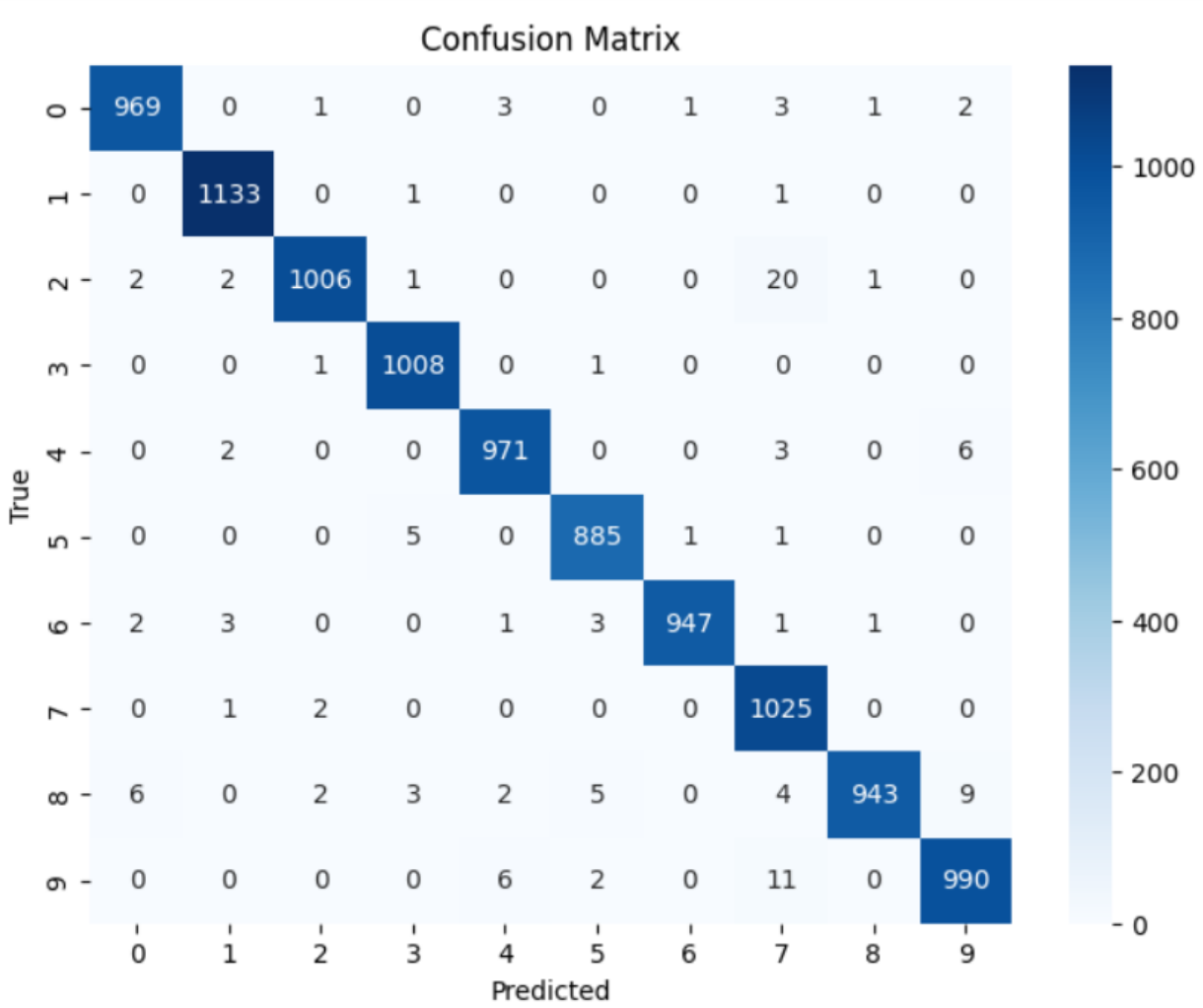
Training vs. Validation Loss

- Both the training and validation loss decrease consistently throughout the training process. This shows that the model's predictions are becoming more accurate over time. The validation loss closely tracks the training loss, further confirming the absence of significant overfitting or underfitting.



Confusion Matrix

- The confusion matrix heatmap reveals that the model is correctly predicting most of the digits with very few misclassifications. The diagonal dominance in the matrix (i.e., higher values along the diagonal) confirms high accuracy. Misclassifications are minimal and often occur between visually similar digits, such as 4 and 9, or 5 and 3.



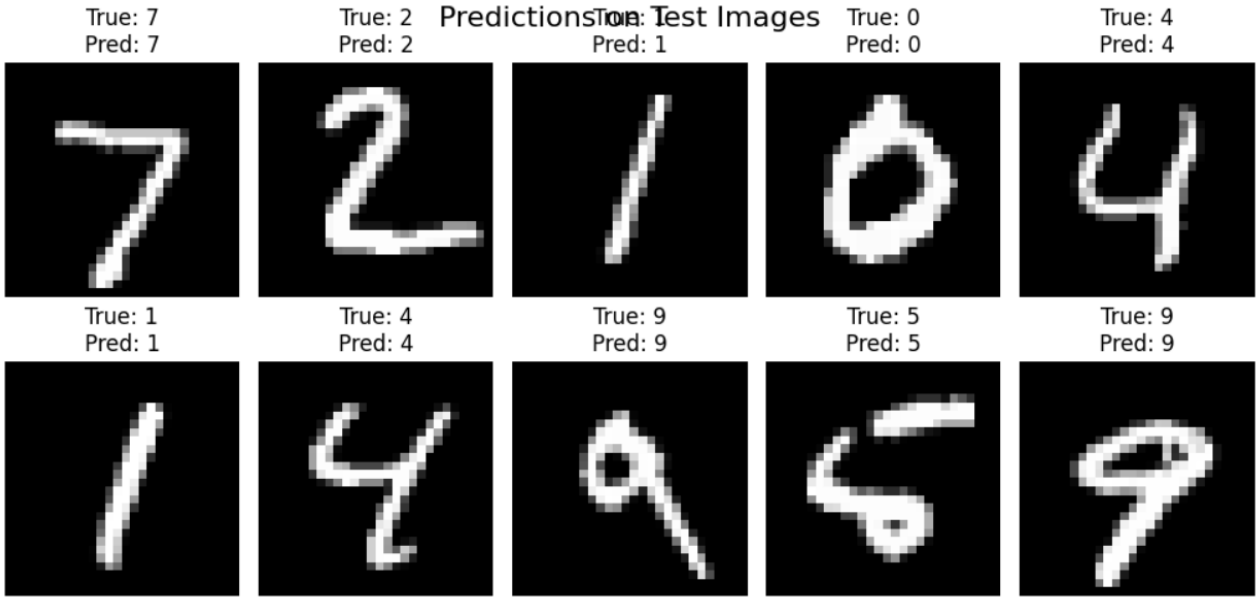
Classification Report

- The precision, recall, and F1-scores for each digit are all above 0.97, showing the model's strong performance across all classes. No particular digit is significantly underperforming, which indicates the model treats each class fairly well.

```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       980
           1       0.99      1.00      1.00      1135
           2       0.99      0.97      0.98      1032
           3       0.99      1.00      0.99      1010
           4       0.99      0.99      0.99       982
           5       0.99      0.99      0.99       892
           6       1.00      0.99      0.99       958
           7       0.96      1.00      0.98      1028
           8       1.00      0.97      0.98       974
           9       0.98      0.98      0.98      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

Sample Predictions Visualization

The visual comparison of actual and predicted labels on test images demonstrates that the model can correctly identify a wide variety of handwriting styles. Occasional errors tend to occur when the handwriting is ambiguous or poorly formed, which is expected.


Predictions on Test Images
True: 7
Pred: 7
True: 2
Pred: 2
True: 1
Pred: 1
True: 0
Pred: 0
True: 4
Pred: 4
True: 1
Pred: 1
True: 4
Pred: 4
True: 9
Pred: 9
True: 5
Pred: 5
True: 9
Pred: 9