**Handwritten Digit Classification Using TensorFlow & MNIST**

This project focuses on building a deep learning model using the MNIST dataset to recognize handwritten digits ranging from 0 to 9. The model leverages TensorFlow and Keras to train a neural network that classifies grayscale 28x28 images into their corresponding numeric class. After training and evaluation, the model is deployed as an interactive web application using Streamlit and ngrok, allowing users to test the model live in real time.

**Model Description**

The neural network architecture consists of an input layer that flattens the 28x28 image pixels, followed by two fully connected (dense) layers with 128 neurons each and ReLU activation functions. The output layer contains 10 neurons (one for each digit class) and uses the softmax activation function to produce a probability distribution over all possible classes.

The model is compiled using the Adam optimizer and trained with the Sparse Categorical Cross entropy loss function. The training process was conducted over three epochs using the normalized MNIST training set. Upon evaluation on the test set, the model achieved an accuracy of **97%**, surpassing the performance threshold set at the beginning of the task.

**Evaluation and Metrics**

To assess the performance of the model, a **confusion matrix** and **classification report** were generated using the test data. The confusion matrix illustrates the number of correctly and incorrectly classified instances per digit class. As seen in the matrix, the majority of digits are predicted accurately with minimal confusion between certain classes like 9 and 4 or 5 and 3.

The **classification report** includes key metrics such as precision, recall, and F1-score for each digit class. The results show strong consistency across the board, with most scores exceeding 0.96. The overall accuracy stands at **97%**, and both the macro and weighted average F1-scores are also at **0.97**, indicating balanced performance across all classes.

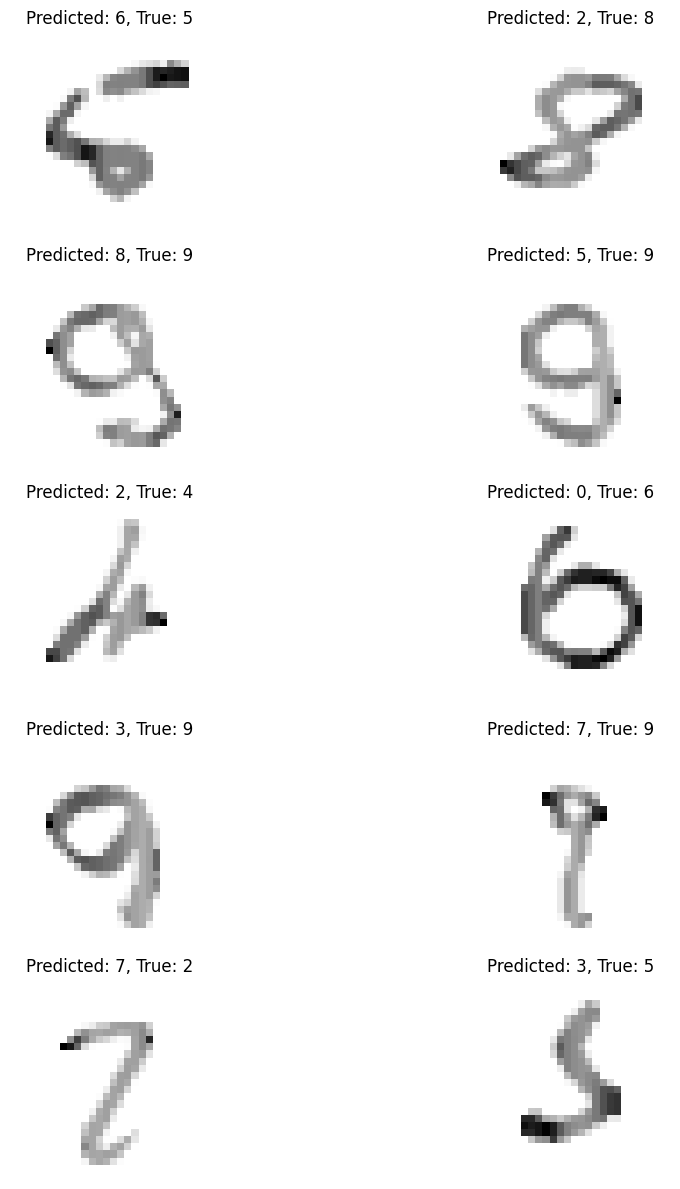


Figure 1. Test results

**Error Analysis**

To further understand the model’s weaknesses, an error analysis was conducted by visualizing some of the **misclassified examples**. The most common misclassification patterns observed were:

1. **Digit 9 misclassified as 4**
2. **Digit 5 misclassified as 3**
3. **Digit 8 misclassified as 3**

These confusions often stem from similarities in handwritten styles, where certain digits closely resemble others depending on how they are written

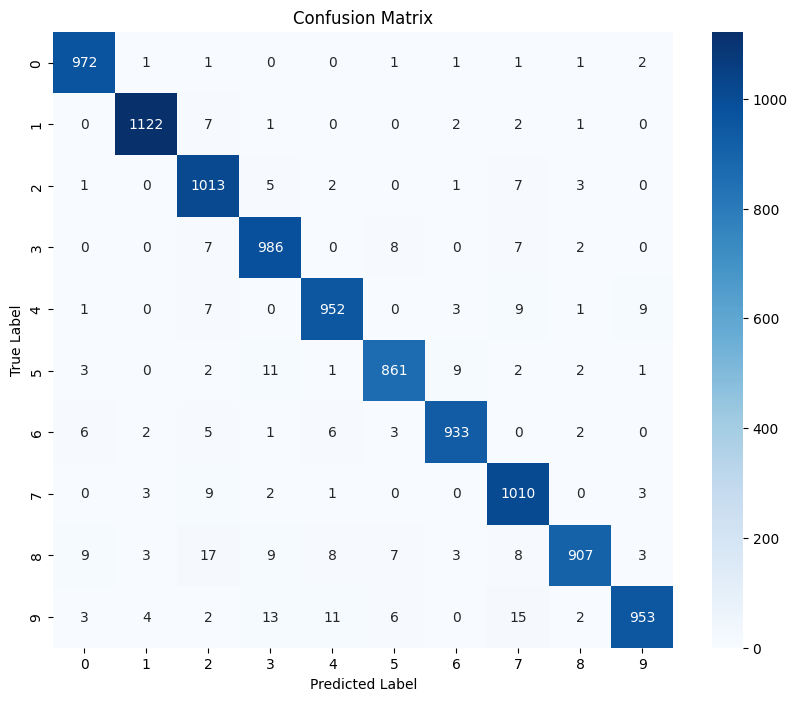


Figure 2. Confusion matrix

Classification Report:

precision recall f1-score support

0 0.98 0.99 0.98 980

1 0.99 0.99 0.99 1135

2 0.95 0.98 0.96 1032

3 0.96 0.98 0.97 1010

4 0.97 0.97 0.97 982

5 0.97 0.97 0.97 892

6 0.98 0.97 0.98 958

7 0.95 0.98 0.97 1028

8 0.98 0.93 0.96 974

9 0.98 0.94 0.96 1009

accuracy 0.97 10000

macro avg 0.97 0.97 0.97 10000

weighted avg 0.97 0.97 0.97 10000

To address these issues, the following potential solutions were considered:

* **Data augmentation**: Introducing transformations such as rotation, zooming, and shifting to make the model more robust to handwriting variations.
* **Preprocessing improvements**: Applying techniques like histogram equalization or image sharpening to enhance digit clarity.

**Implemented Improvement**

To test one of the proposed solutions, the number of neurons in the two hidden dense layers was increased from **128 to 256**. This change allowed the model to capture more complex patterns and improve generalization. After retraining the model with the new architecture for three epochs, the **test accuracy increased from 97.0% to 97.6%**, showing a measurable improvement in predictive performance.

**Deployment as Web App**

The trained model was deployed using **Streamlit**, a Python framework for building interactive web applications. The web app allows users to:

* View the model’s evaluation metrics (accuracy and loss)
* Examine the confusion matrix and classification report
* Visualize the most common misclassified digits

To make the application accessible remotely, **ngrok** was used to tunnel the local Streamlit server to a public web URL. This setup enables real-time testing and demonstration of the digit recognition system without additional hosting infrastructure.