Handwritten Digit Recognition using TensorFlow on MNIST Dataset

Saif Ullah

September 11, 2025

1 Introduction

Handwritten digit recognition is one of the classical problems in the field of machine learning and deep learning. The MNIST dataset is widely used for benchmarking classification algorithms. It contains $60,000$ training images and $10,000$ testing images of handwritten digits (0–9), each of size 28×28 pixels.

In this work, a simple neural network using TensorFlow and Keras has been implemented to classify handwritten digits.

2 Dataset and Preprocessing

The dataset is loaded directly from the `tensorflow.keras.datasets` module. Each image is grayscale with pixel values ranging from 0 to 255. For better training efficiency, the dataset is normalized by dividing pixel values by 255, bringing all values into the range $[0, 1]$.

3 Model Architecture

The neural network used in this experiment consists of the following layers:

- **Flatten Layer:** Converts the 28×28 pixel image into a 784-dimensional vector.
- **Dense Layer (128 units, ReLU):** Fully connected hidden layer with ReLU activation to capture non-linear patterns.
- **Dropout Layer (rate=0.2):** Randomly drops 20% of neurons during training to prevent overfitting.
- **Dense Output Layer (10 units, Softmax):** Produces probability distribution over 10 classes (digits 0–9).

4 Python Implementation

The implementation in Python using TensorFlow and Keras is given below:

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models

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

# Normalize data
x_train, x_test = x_train / 255.0, x_test / 255.0

# Build model
```

```
model = models.Sequential([
    layers.Flatten(input_shape=(28, 28)),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(10, activation='softmax')
])

# Compile model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train model
model.fit(x_train, y_train, epochs=5)

# Evaluate model
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f"\nTest Accuracy: {test_acc:.4f}")
```

5 Discussion of Results

After training for 5 epochs, the model achieves a test accuracy of around **97%–98%**. This performance is remarkable considering the simplicity of the architecture. The dropout layer played a key role in reducing overfitting, ensuring the model generalizes well on unseen data.

6 Conclusion

The experiment demonstrates that even a simple feedforward neural network can achieve high accuracy on the MNIST dataset. Further improvements could be achieved by using convolutional neural networks (CNNs), which are more suitable for image recognition tasks due to their ability to capture spatial features.