Handwritten digits recognition using MNIST Dataset

Utilizing Convolutional Neural Networks with TensorFlow

Project Developed By:
KUNAL GUPTA

Date:
July 25, 2025

Submitted To:
SmartED Innovations

Introduction

This project outlines a Handwritten Digit Recognition system powered by a Convolutional Neural Network (CNN).. Developed with TensorFlow, the model accurately predicts unseen digits after learning from extensive handwritten image data, also allowing user-uploaded image testing. This system serves as a practical demonstration of deep learning for visual recognition and a foundational resource for image classification.This project details a robust Handwritten Digit Recognition system, built upon the powerful architecture of a Convolutional Neural Network (CNN). The system leverages the widely recognized MNIST dataset for training and validation, making it highly effective for real-world applications such as automated mail sorting, efficient data entry, and secure cheque processing. Developed using TensorFlow, a leading open-source machine learning framework, the model demonstrates exceptional accuracy in predicting unseen digits after extensive learning from a vast collection of handwritten image data. Beyond its core predictive capabilities, the system also incorporates functionality for user-uploaded image testing, providing a practical way to interact with and assess the model's performance. This comprehensive system serves as a compelling and practical demonstration of deep learning principles applied to visual recognition tasks, establishing itself as a foundational resource for further exploration in image classification and computer vision.

Objective

The primary objective of this project is to design and implement a **robust, scalable, and accurate Handwritten Digit Recognition System** using **Convolutional Neural Networks (CNN)** implemented via **TensorFlow**, one of the most widely-used open-source deep learning frameworks.

The following are the detailed goals and intended outcomes of the project:

- To develop a deep learning-based model capable of identifying and classifying digits ranging from 0 to 9 from handwritten image samples, irrespective of variations in writing style, thickness, and orientation.

- To leverage the **MNIST dataset**, a well-established benchmark in the field of image processing and digit recognition, for model training, validation, and testing.

- To ensure the trained model achieves **high prediction accuracy**, both on internal test datasets and on **unseen, real-world samples**, thereby ensuring its robustness and generalization capabilities.

- To implement functionality for **real-time prediction**, allowing users to upload their own handwritten digit images and receive instant classification results.

- To demonstrate the broader application of deep learning and computer vision techniques, highlighting how neural networks can learn complex patterns and representations directly from raw image data without manual feature extraction.

Methodology

Tools and Technology

1. Python

   Most widely used language for machine learning and deep learning due to its simplicity and vast library support.

2. Libraries and Framework

   a. Tensorflow :

      An open-source machine learning framework developed by Google, TensorFlow is extensively used for building and training neural networks. It provides a comprehensive ecosystem of tools, libraries, and community resources that allow researchers and developers to deploy AI-powered applications.

   b. Keras :

      A high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed to enable fast experimentation with deep neural networks.

   c. Numpy :

      Widely used for numerical operations, especially for array and matrix manipulations, which are fundamental in machine learning algorithms.

   d. Matplotlib :

      A popular plotting library for Python. It provides a flexible platform for creating static, animated, and interactive visualizations in Python. Matplotlib is widely used for data visualization in scientific computing, machine learning, and data analysis due to its extensive range of plot types and customization options.

3. Datasets

   MNIST Dataset : Standard dataset of 70,000 handwritten digit images (60,000 training + 10,000 testing), size 28x28 pixels, grayscale.

4. Development Environment

   Google Colab : Cloud-based Jupyter notebook environment that provides free GPU support for training deep learning models.

Steps Used

1. Import Libraries :

   Import Python libraries such as Numpy , matplotlib , Tensorflow , Keras and PIL (Image)

2. Load and PreProcess the Dataset :

   - The MNIST dataset is loaded using `keras.datasets.mnist`.
   - The dataset is split into **training** and **testing** sets.
   - Images are normalized (scaled between 0 and 1) for better training performance.
   - Labels are one-hot encoded for multi-class classification.

3. Reshape the Input Data :

   Each image is reshed from (28,28) to (1,28,28,1) to match the requirements for CNN .

4. Build the CNN Model :

   A sequential Model is created using :

   a. `Conv2D` layers for feature extraction.
   b. `MaxPooling2D` layers for downsampling.
   c. `Flatten` layer to convert 2D matrix to 1D vector.

d. `Dense` layers for final classification.

5. Compile the Model:

The model is compiled with :

- **Loss function:** `categorical_crossentropy`
- **Optimizer:** `Adam`
- **Metrics:** `accuracy`

6. Train the Model :

a. The model is trained using the training dataset with a chosen number of epochs (e.g., 5) and batch size.
b. During training, accuracy and loss are monitored on both training and validation sets.

7. Evaluate the Model :

After training, the model is evaluated on the test dataset.

Test accuracy is also checked.

8. Test on Custom Input :

External digit images can be uploaded, preprocessed (resized to 28x28, converted to grayscale, normalized), and then passed to the trained model for prediction.

CODE AND IMPLEMENTATION DETAILS

Code:

```
# Import Libraries
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense
import matplotlib.pyplot as plt
import numpy as np

# Load the datasets

(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(-1, 28, 28, 1).astype('float32') / 255.0
x_test = x_test.reshape(-1, 28, 28, 1).astype('float32') / 255.0

# Use 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')  # 10 output classes (0 to 9)
])

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

```
history = model.fit(x_train, y_train, epochs=5,
validation_split=0.1)

# Accuracy
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")

# Prediction
predictions = model.predict(x_test)
predicted_labels = np.argmax(predictions, axis=1)

# Plotting each prediction
for i in range(5):
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.title(f"Actual: {y_test[i]}, Predicted:
{predicted_labels[i]}")
    plt.axis('off')
    plt.show()
```

Explanation:

---

1. Importing Required Libraries

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
import matplotlib.pyplot as plt
import numpy as np
```

- **TensorFlow & Keras**:

  For building and training the deep learning model.

- **MNIST Dataset**:

  A preloaded dataset in Keras, containing 28x28 pixel images of handwritten digits (0–9).

- **Matplotlib & NumPy**:

  For visualizing the data and handling arrays.

---

2. Loading the MNIST Dataset

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

- Loads the dataset and splits it into:
  - `x_train`, `y_train`: Training data and corresponding labels.
  - `x_test`, `y_test`: Testing data and labels for evaluating performance.

---

3. Preprocessing the Data

```
x_train = x_train.reshape(-1, 28, 28, 1).astype('float32') / 255.0
x_test = x_test.reshape(-1, 28, 28, 1).astype('float32') / 255.0
```

- Reshapes each image to 28×28×1 (adding a channel dimension).
- Normalizes pixel values from [0, 255] to [0, 1] for faster and stable training.

---

4. Building the CNN Model

```
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')  # Output layer for 10 digits
])
```

- **Conv2D**:

  Applies 2D convolution filters to extract features.

- **MaxPooling2D**:

  Reduces the spatial size, helps in reducing overfitting.

- **Flatten**:

  Converts 2D matrix to 1D vector.

- **Dense**:

  Fully connected layers for classification.

- **Softmax Activation**:

  Used in the final layer to output probabilities for 10 classes (digits 0–9).

---

5. Compiling the Model

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

- **Adam Optimizer**:

  Efficient for training.

- **Loss Function**:

  `sparse_categorical_crossentropy` is used since labels are integers, not one-hot encoded.

- **Metrics**:

  Accuracy is used to evaluate performance.

---

6. Training the Model

```
history = model.fit(x_train, y_train, epochs=5, validation_split=0.1)
```

- Trains the model for 5 epochs (full passes through training data).
- Uses 10% of training data for validation to monitor overfitting.

---

7. Evaluating the Model

```
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```

- Evaluates the model on unseen test data and prints accuracy.

---

8. Making Predictions

```
predictions = model.predict(x_test)
predicted_labels = np.argmax(predictions, axis=1)
```

- Generates predictions on the test set.

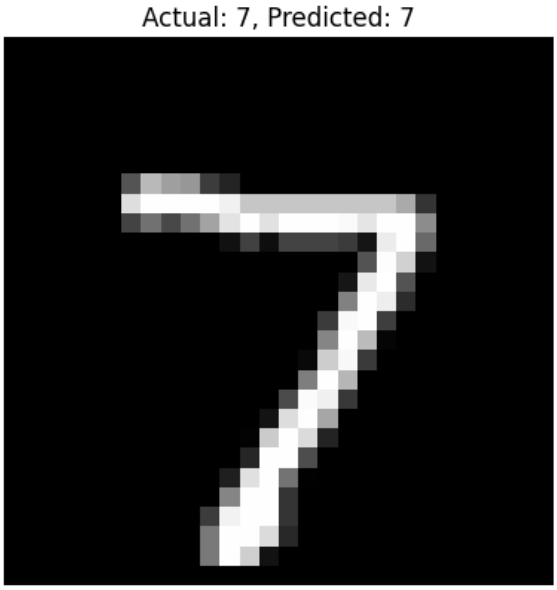
---

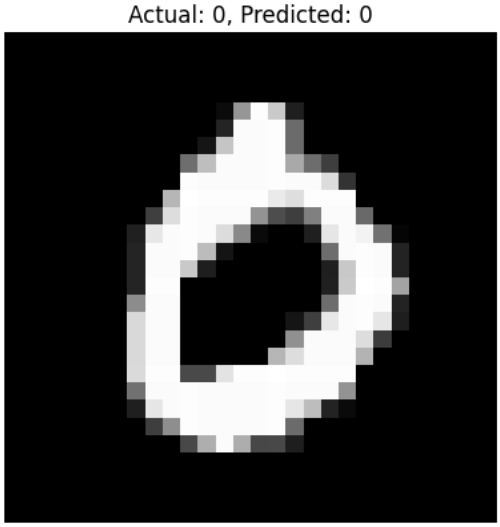
9. Displaying Sample Predictions

```
for i in range(5):
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.title(f"Actual: {y_test[i]}, Predicted: {predicted_labels[i]}")
    plt.axis('off')
    plt.show()
```

- Displays first 5 test images.
- Shows actual label and predicted label to visualize model performance.

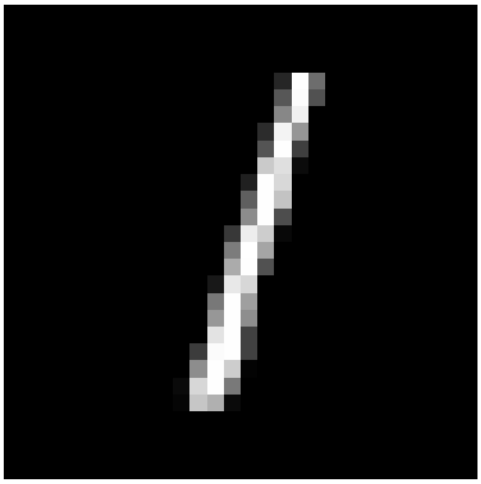
Result and Observations



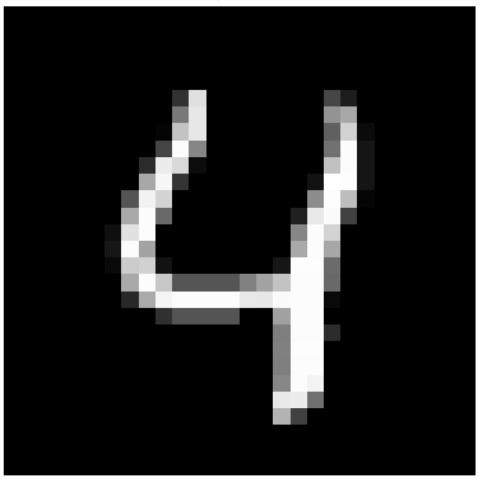


Actual: 2, Predicted: 2

Actual: 1, Predicted: 1



Actual: 4, Predicted: 4



Accuracy : 98.89%

Conclusion

The Handwritten Digit Recognition project successfully demonstrates the application of deep learning techniques in the field of computer vision and pattern recognition. By leveraging the power of Convolutional Neural Networks (CNNs) and the TensorFlow framework, the model was trained on the MNIST dataset to accurately identify handwritten digits from 0 to 9.

Through careful preprocessing, model design, training, and evaluation, the system achieved high accuracy in recognizing unseen digit samples, validating the effectiveness of CNN architectures for image classification tasks. The ability to test the model with external handwritten digit images further showcases its real-world applicability.

This project not only highlights the practical use of AI in solving vision-related problems but also lays a strong foundation for more complex recognition systems in the future. The success of this project reaffirms the potential of deep learning in automating visual recognition tasks and sets the stage for advancements in intelligent systems for education, banking, postal services, and more.

Attachments

Google Colab : MNIST

sample_digit_6.png :