finalproject

August 30, 2024

Title of Project Handwritten Digit Recognition using MNIST Dataset

Objective

The objective is to build a machine learning model that can accurately recognize handwritten digits (0-9) from images using the MNIST dataset.

Data Source

The MNIST dataset, which contains 70,000 images of handwritten digits. The dataset is divided into a training set of 60,000 images and a test set of 10,000 images.

Import Library

[4]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
import tensorflow as tf
import tensorflow as tf
from tensorflow import keras
layers = keras.layers
models = keras.models
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
```

Import Data

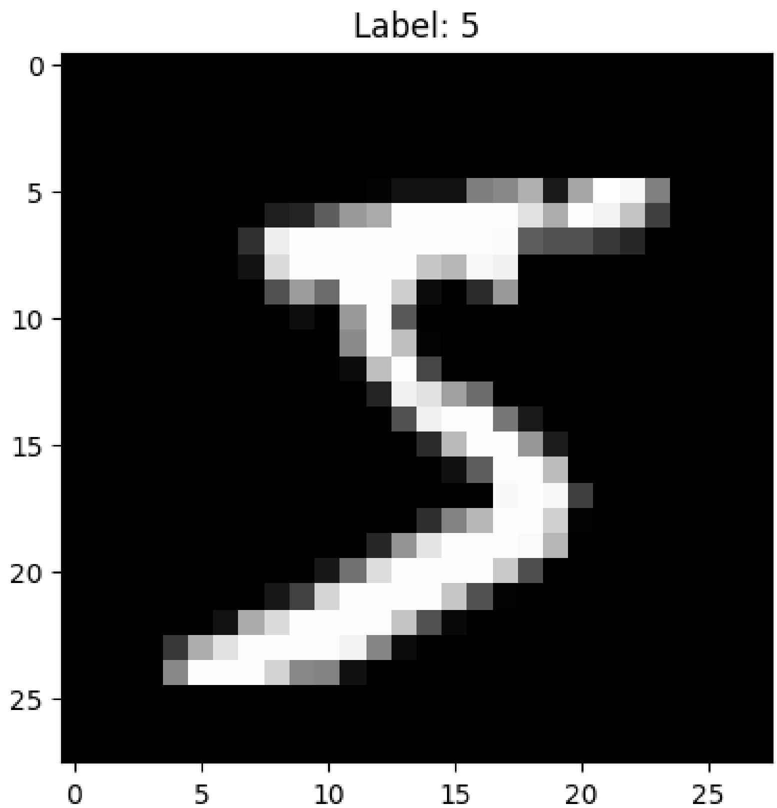
[5]:
```
from tensorflow.keras.datasets import mnist

# Load the dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11490434/11490434            0s
0us/step
```

Describe Data

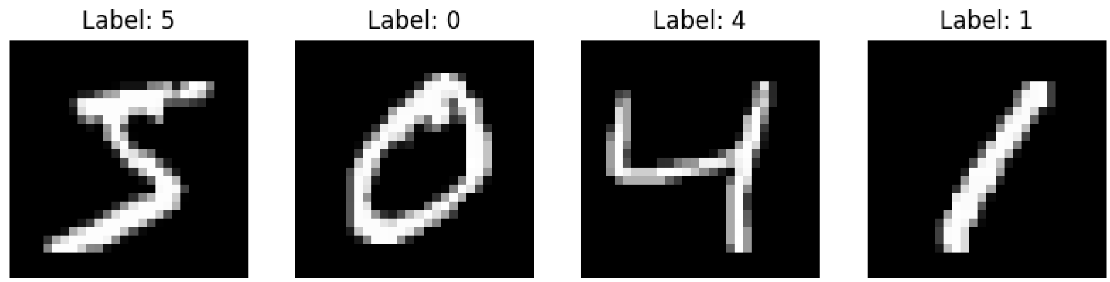
[6]:
```python
# Print the shape of the data
print(f"Training data shape: {X_train.shape}")
print(f"Test data shape: {X_test.shape}")

# Display the first image in the training data
plt.imshow(X_train[0], cmap='gray')
plt.title(f"Label: {y_train[0]}")
plt.show()
```

```
Training data shape: (60000, 28, 28)
Test data shape: (10000, 28, 28)
```



Data Visualization

[7]:
```python
# Plotting a few images from the dataset
fig, axes = plt.subplots(1, 4, figsize=(10, 3))
for i in range(4):
    axes[i].imshow(X_train[i], cmap='gray')
    axes[i].set_title(f"Label: {y_train[i]}")
    axes[i].axis('off')
plt.show()
```



Data Preprocessing

[8]:
```
# Normalize the images to values between 0 and 1
X_train = X_train / 255.0
X_test = X_test / 255.0

# Reshape the data to fit the model
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1)

# One-hot encode the labels
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)
```

Define Target Variable (y) and Feature Variables (X)

Target Variable (y): The digit label (0-9). Feature Variables (X): The pixel values of the 28x28 images.

Train Test Split

Already done when loading the dataset. The MNIST dataset provides predefined training and test sets.

Modeling

[9]:
```
model = models.Sequential([
    layers.Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28,
 ↪28, 1)),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

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

model.summary()
```

```
/usr/local/lib/python3.10/dist-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dense (Dense)	(None, 128)	204,928
dense_1 (Dense)	(None, 10)	1,290

Total params: 225,034 (879.04 KB)

Trainable params: 225,034 (879.04 KB)

Non-trainable params: 0 (0.00 B)

Model Evaluation

[10]:
```
# Train the model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test),␣
↪epochs=10, batch_size=200, verbose=2)

# Evaluate the model
score = model.evaluate(X_test, y_test, verbose=0)
print(f"Test loss: {score[0]}")
print(f"Test accuracy: {score[1]}")
```

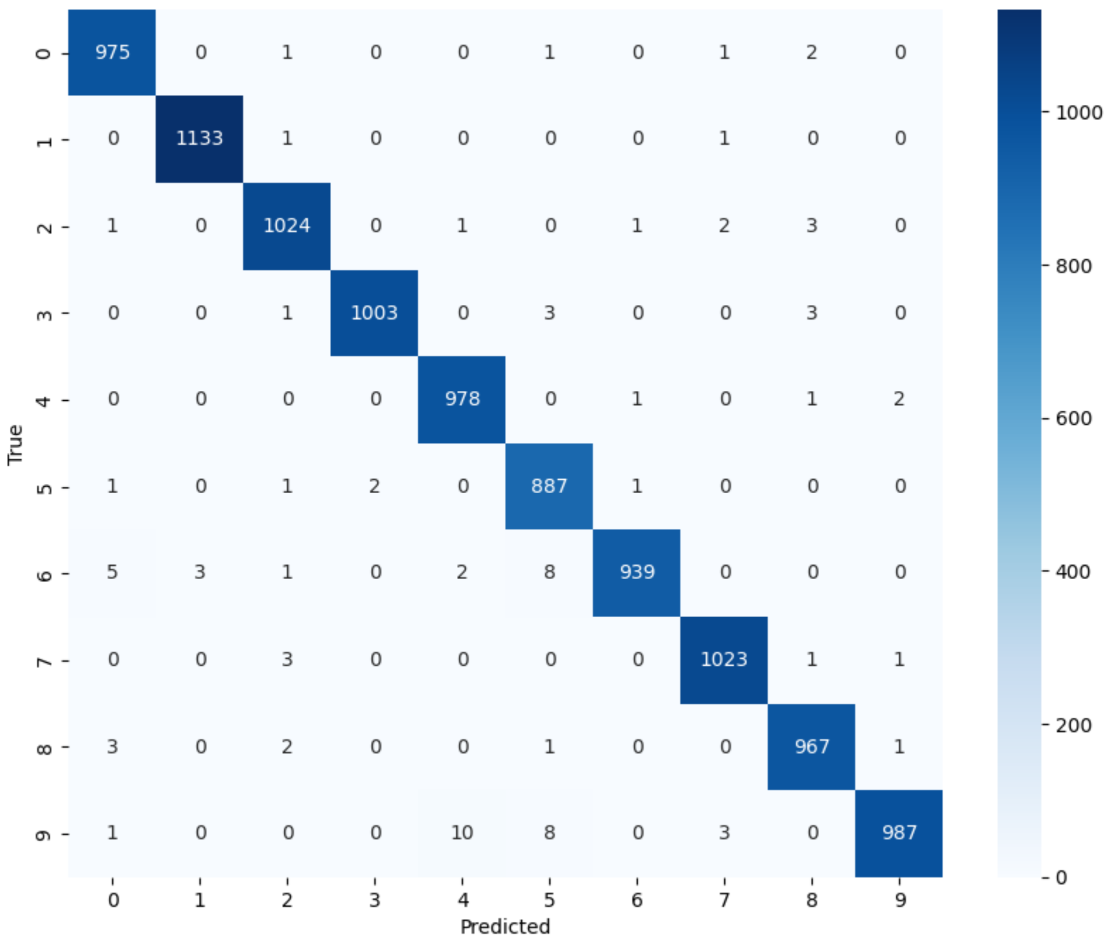
```
Epoch 1/10
300/300 - 50s - 165ms/step - accuracy: 0.9222 - loss: 0.2688 - val_accuracy:
0.9776 - val_loss: 0.0739
Epoch 2/10
300/300 - 84s - 281ms/step - accuracy: 0.9800 - loss: 0.0650 - val_accuracy:
0.9828 - val_loss: 0.0505
Epoch 3/10
300/300 - 98s - 326ms/step - accuracy: 0.9867 - loss: 0.0433 - val_accuracy:
0.9891 - val_loss: 0.0344
Epoch 4/10
300/300 - 75s - 249ms/step - accuracy: 0.9895 - loss: 0.0344 - val_accuracy:
0.9892 - val_loss: 0.0329
Epoch 5/10
300/300 - 48s - 160ms/step - accuracy: 0.9919 - loss: 0.0267 - val_accuracy:
0.9907 - val_loss: 0.0299
Epoch 6/10
300/300 - 83s - 276ms/step - accuracy: 0.9935 - loss: 0.0216 - val_accuracy:
0.9896 - val_loss: 0.0307
Epoch 7/10
300/300 - 82s - 273ms/step - accuracy: 0.9943 - loss: 0.0183 - val_accuracy:
0.9897 - val_loss: 0.0306
Epoch 8/10
300/300 - 81s - 269ms/step - accuracy: 0.9945 - loss: 0.0166 - val_accuracy:
0.9919 - val_loss: 0.0248
Epoch 9/10
300/300 - 81s - 270ms/step - accuracy: 0.9964 - loss: 0.0121 - val_accuracy:
0.9916 - val_loss: 0.0274
Epoch 10/10
300/300 - 82s - 272ms/step - accuracy: 0.9968 - loss: 0.0103 - val_accuracy:
0.9916 - val_loss: 0.0269
Test loss: 0.02692827209830284
Test accuracy: 0.991599977016449
```

Prediction

[11]:

```python
# Predict on the test data
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

# Classification report
print(classification_report(y_true, y_pred_classes))

# Confusion matrix
conf_matrix = confusion_matrix(y_true, y_pred_classes)
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

```
313/313              3s 9ms/step
              precision    recall  f1-score   support

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

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



Explanation

The model uses a Convolutional Neural Network (CNN) to recognize handwritten digits. The CNN consists of multiple layers: two convolutional layers followed by max pooling layers, and then fully connected layers leading to the final output. The model is trained using the MNIST dataset and evaluated on unseen test data. The model's performance is measured by accuracy, and the predictions are visualized using a confusion matrix and classification report.