Handwritten Digit Recognition

Problem Statement:

Handwritten digit recognition is a fundamental task in computer vision and deep learning, widely used in applications such as postal mail sorting, bank check processing, and automated form reading. However, real-world handwritten digits vary significantly in size, shape, orientation, and clarity, making accurate recognition a challenging task.

Objective

Develop a web-based handwritten digit recognition system that can accurately classify digits (0-9) from user-uploaded images. The system should:

1. Leverage Convolutional Neural Networks (CNNs) trained on the MNIST dataset.
2. Handle blurred or noisy images using preprocessing techniques like OpenCV.
3. Provide an intuitive web interface for users to upload images and get real-time predictions.

Importing 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, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Loading the Dataset

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

Preprocessing the images

```
# Normalize pixel values to [0,1]
x_train, x_test = x_train / 255.0, x_test / 255.0


# Reshape data to fit CNN input (28x28x1)
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)


# Convert labels to categorical
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)


# Image Augmentation
datagen = ImageDataGenerator(
    rotation_range=10,   # Rotate images by 10 degrees
    zoom_range=0.1,      # Random zoom
    width_shift_range=0.1, # Shift width
    height_shift_range=0.1 # Shift height
)
datagen.fit(x_train)
```

Building the model

```
# Build CNN model
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    MaxPooling2D((2,2)),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D((2,2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.3),
```

```
    Dense(10, activation='softmax')  # 10 classes (digits 0-9)
])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

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

Model Training

```
# Train model using augmented images
history=model.fit(datagen.flow(x_train, y_train, batch_size=64), validation_data=(x_test, y_test), epochs=10)
```

```
Epoch 1/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 72s 74ms/step - accuracy: 0.7694 - loss: 0.7011 - val_accuracy: 0.9809 - val_loss: 0.0579
Epoch 2/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 67s 71ms/step - accuracy: 0.9478 - loss: 0.1683 - val_accuracy: 0.9855 - val_loss: 0.0375
Epoch 3/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 67s 71ms/step - accuracy: 0.9624 - loss: 0.1213 - val_accuracy: 0.9893 - val_loss: 0.0298
Epoch 4/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 67s 72ms/step - accuracy: 0.9698 - loss: 0.0989 - val_accuracy: 0.9902 - val_loss: 0.0266
Epoch 5/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 67s 71ms/step - accuracy: 0.9748 - loss: 0.0842 - val_accuracy: 0.9916 - val_loss: 0.0223
Epoch 6/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 66s 71ms/step - accuracy: 0.9764 - loss: 0.0772 - val_accuracy: 0.9917 - val_loss: 0.0244
Epoch 7/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 67s 71ms/step - accuracy: 0.9801 - loss: 0.0665 - val_accuracy: 0.9926 - val_loss: 0.0229
Epoch 8/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 82s 71ms/step - accuracy: 0.9803 - loss: 0.0665 - val_accuracy: 0.9927 - val_loss: 0.0200
Epoch 9/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 66s 71ms/step - accuracy: 0.9823 - loss: 0.0591 - val_accuracy: 0.9940 - val_loss: 0.0187
Epoch 10/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 66s 71ms/step - accuracy: 0.9837 - loss: 0.0515 - val_accuracy: 0.9924 - val_loss: 0.0222
```

Validation

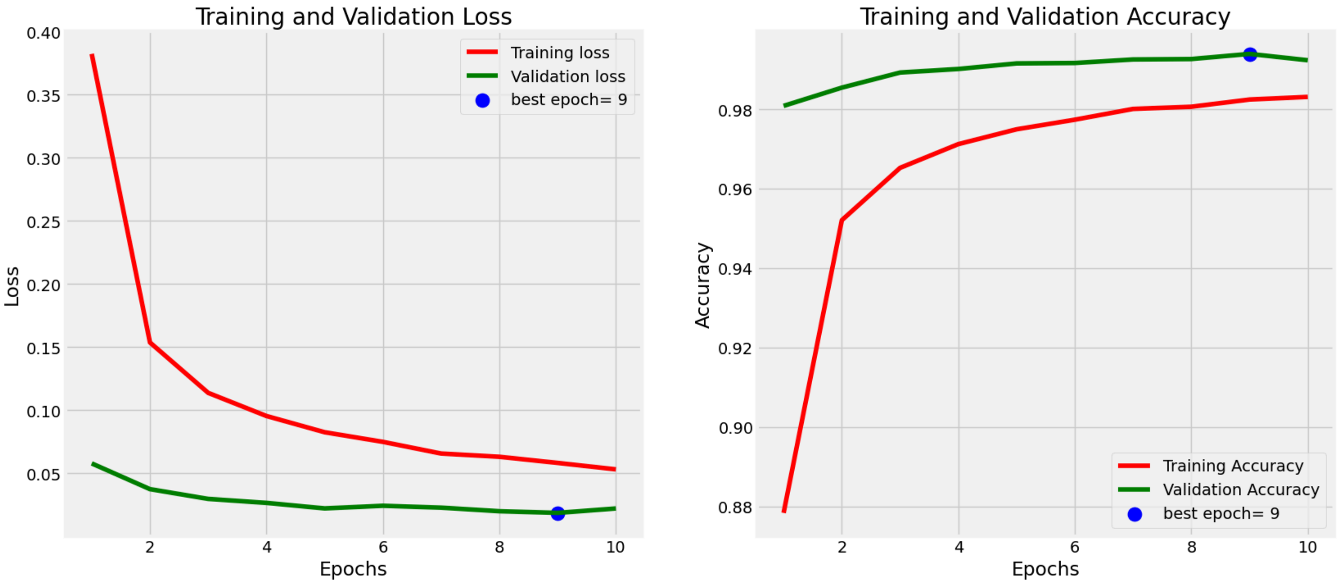
```
#print training and validation accuracy
print("Training Accuracy:", history.history['accuracy'][-1]*100)
print("Validation Accuracy:", history.history['val_accuracy'][-1]*100)
```

```
Training Accuracy: 98.31666946411133
Validation Accuracy: 99.23999905586243
```

Visualization

```
# Define needed variables
tr_acc = history.history['accuracy']
tr_loss = history.history['loss']
val_acc = history.history['val_accuracy']
val_loss = history.history['val_loss']
index_loss = np.argmin(val_loss)
val_lowest = val_loss[index_loss]
index_acc = np.argmax(val_acc)
acc_highest = val_acc[index_acc]

Epochs = [i+1 for i in range(len(tr_acc))]
loss_label = f'best epoch= {str(index_loss + 1)}'
acc_label = f'best epoch= {str(index_acc + 1)}'

# Plot training history
plt.figure(figsize= (20, 8))
plt.style.use('fivethirtyeight')

plt.subplot(1, 2, 1)
plt.plot(Epochs, tr_loss, 'r', label= 'Training loss')
plt.plot(Epochs, val_loss, 'g', label= 'Validation loss')
plt.scatter(index_loss + 1, val_lowest, s= 150, c= 'blue', label= loss_label)
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(Epochs, tr_acc, 'r', label= 'Training Accuracy')
plt.plot(Epochs, val_acc, 'g', label= 'Validation Accuracy')
plt.scatter(index_acc + 1 , acc_highest, s= 150, c= 'blue', label= acc_label)
plt.title('Training and Validation Accuracy')
```

```
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout
plt.show()
```




```
# Save the trained model
model.save("digit_recognition_model.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
```

Observations:

1. The model performs exceptionally well on clean MNIST images.
2. High validation accuracy (99.24%) indicates excellent generalization on test data.
3. Low validation loss (~0.02) means the model is not overfitting significantly.