**CNN IMAGE CLASSIFICATION**

# Import necessary libraries  
from tensorflow.keras.datasets import mnist  
from tensorflow.keras.utils import to\_categorical  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense  
import matplotlib.pyplot as plt  
import numpy as np  
  
# Step 1: Load and preprocess the dataset  
# Load the MNIST dataset (handwritten digits 0-9)  
(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()  
  
# Normalize the pixel values to range 0-1 for faster training and better performance  
X\_train = X\_train / 255.0  
X\_test = X\_test / 255.0  
  
# Reshape the data to include a channel dimension (28x28 images with 1 channel)  
X\_train = X\_train.reshape(-1, 28, 28, 1) # Training data shape: (60000, 28, 28, 1)  
X\_test = X\_test.reshape(-1, 28, 28, 1) # Testing data shape: (10000, 28, 28, 1)  
  
# One-hot encode the labels for multi-class classification  
y\_train = to\_categorical(y\_train, 10) # 10 classes (0-9)  
y\_test = to\_categorical(y\_test, 10)  
  
# Step 2: Visualize some sample images from the training dataset  
# Display the first 9 images with their labels  
for i in range(9):  
 plt.subplot(3, 3, i + 1)  
 plt.imshow(X\_train[i].reshape(28, 28), cmap='gray') # Reshape to 2D for visualization  
 plt.title(f"Label: {np.argmax(y\_train[i])}") # Get the actual label from one-hot encoding  
 plt.axis('off')  
plt.show()  
  
# Step 3: Define the CNN model  
# Create a Sequential model and add layers step-by-step  
model = Sequential([  
 # First convolutional layer with 32 filters and a 3x3 kernel, followed by ReLU activation  
 Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),  
 # First max pooling layer to reduce spatial dimensions by 2x2  
 MaxPooling2D((2, 2)),  
  
 # Second convolutional layer with 64 filters and a 3x3 kernel, followed by ReLU activation  
 Conv2D(64, (3, 3), activation='relu'),  
 # Second max pooling layer to further reduce spatial dimensions by 2x2  
 MaxPooling2D((2, 2)),  
  
 # Flatten the feature map into a 1D vector for input into dense layers  
 Flatten(),  
  
 # Fully connected dense layer with 128 units and ReLU activation  
 Dense(128, activation='relu'),  
  
 # Output layer with 10 units (for 10 classes) and softmax activation for probabilities  
 Dense(10, activation='softmax')  
])  
  
# Step 4: Compile the model  
# Use Adam optimizer, categorical cross-entropy loss, and accuracy metric  
model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])  
  
# Print the model's architecture  
print(model.summary())  
  
# Step 5: Train the model  
# Train the CNN for 5 epochs with a batch size of 32, and validate on the test set  
history = model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=5, batch\_size=32)  
  
# Step 6: Evaluate the model's performance  
# Evaluate the trained model on the test set and print the accuracy  
test\_loss, test\_acc = model.evaluate(X\_test, y\_test, verbose=2)  
print(f"\nTest Accuracy: {test\_acc:.2f}")  
  
# Step 7: Make predictions  
# Predict the classes of the test set images  
predictions = model.predict(X\_test)  
  
# Visualize the first 9 predictions alongside the actual images  
for i in range(9):  
 plt.subplot(3, 3, i + 1)  
 plt.imshow(X\_test[i].reshape(28, 28), cmap='gray') # Reshape to 2D for visualization  
 plt.axis('off')  
 plt.title(f"Predicted: {np.argmax(predictions[i])}")  
plt.show()