9. Build Convolution Neural Network for image classification

```
import numpy as np
import struct
import tensorflow as tf
import matplotlib.pyplot as plt

# Paths to the actual MNIST files (update with full path to files)
TRAIN_IMAGES = r"C:\Users\Sahyadri\Desktop\mnist\train-images-idx3-ubyte\train-images-idx3-ubyte"
TRAIN_LABELS = r"C:\Users\Sahyadri\Desktop\mnist\train-labels-idx1-ubyte\train-labels-idx1-ubyte"
TEST_IMAGES = r"C:\Users\Sahyadri\Desktop\mnist\t10k-images-idx3-ubyte\t10k-images-idx3-ubyte"
TEST_LABELS = r"C:\Users\Sahyadri\Desktop\mnist\t10k-labels-idx1-ubyte\t10k-labels-idx1-ubyte"

# Function to load images
def load_images(file_path):
    with open(file_path, 'rb') as f:
        magic, num, rows, cols = struct.unpack(">IIII", f.read(16))
        image_data = np.frombuffer(f.read(), dtype=np.uint8)
        images = image_data.reshape(num, rows, cols, 1) / 255.0  # Normalize
    return images

# Function to load labels
def load_labels(file_path):
    with open(file_path, 'rb') as f:
        magic, num = struct.unpack(">II", f.read(8))
        labels = np.frombuffer(f.read(), dtype=np.uint8)
    return labels

# Load data
x_train = load_images(TRAIN_IMAGES)
```

```
y_train = load_labels(TRAIN_LABELS)
x_test = load_images(TEST_IMAGES)
y_test = load_labels(TEST_LABELS)

print(f"Training set: {x_train.shape}, {y_train.shape}")
print(f"Test set: {x_test.shape}, {y_test.shape}")

# Build the model
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')  # Output for 10 digits
])

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

# Train and store history
history = model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))

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

# ---     Plot Accuracy and Loss ---
plt.figure(figsize=(12, 5))

# Accuracy plot
plt.subplot(1, 2, 1)
```

```
plt.plot(history.history['accuracy'], label="Train Accuracy")
plt.plot(history.history['val_accuracy'], label="Validation Accuracy")
plt.title("Model Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()

# Loss plot
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label="Train Loss")
plt.plot(history.history['val_loss'], label="Validation Loss")
plt.title("Model Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()

plt.tight_layout()
plt.show()

# ---     Show one test image per digit class (0-9) with prediction ---
import random

shown_digits = set()
plt.figure(figsize=(15, 5))

for i in range(len(x_test)):
    label = y_test[i]
    if label not in shown_digits:
        shown_digits.add(label)
        img = x_test[i]
        prediction = np.argmax(model.predict(img.reshape(1, 28, 28, 1), verbose=0))
        plt.subplot(2, 5, label + 1)
        plt.imshow(img.reshape(28, 28), cmap='gray')
        plt.title(f"Label: {label}, Pred: {prediction}")
```

```
        plt.axis('off')
        if len(shown_digits) == 10:
            break

plt.suptitle("Sample Predictions for Each Digit", fontsize=16)
plt.tight_layout()
plt.show()
```