Batch: B - 1 **Roll No.: 16014022050**

Experiment No. 8

Title: Implementation of CNN algorithm

Course Outcome:

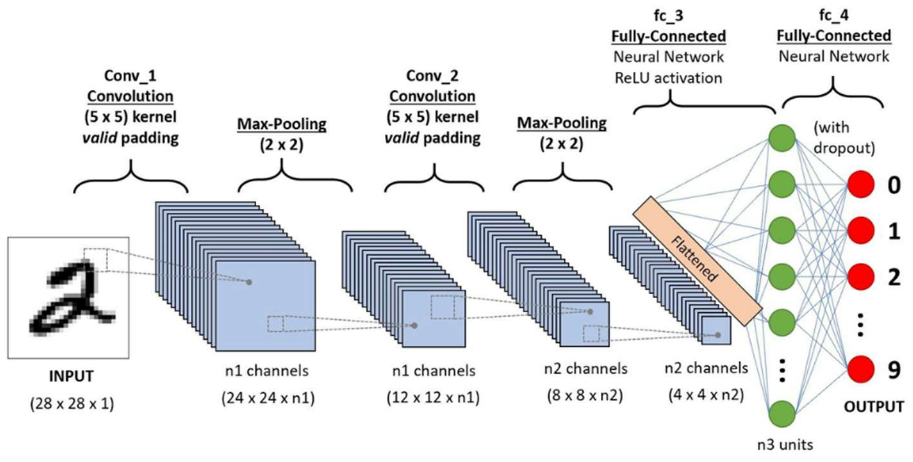
CO4: Implement Deep learning and evaluate the performance using various model evaluation techniques.

Books/ Journals/ Websites referred:

1. **"Artificial Intelligence: a Modern Approach" by Russell and Norving, Pearson education Publications**
2. **"Artificial Intelligence" By Rich and knight, Tata McGraw Hill Publications**

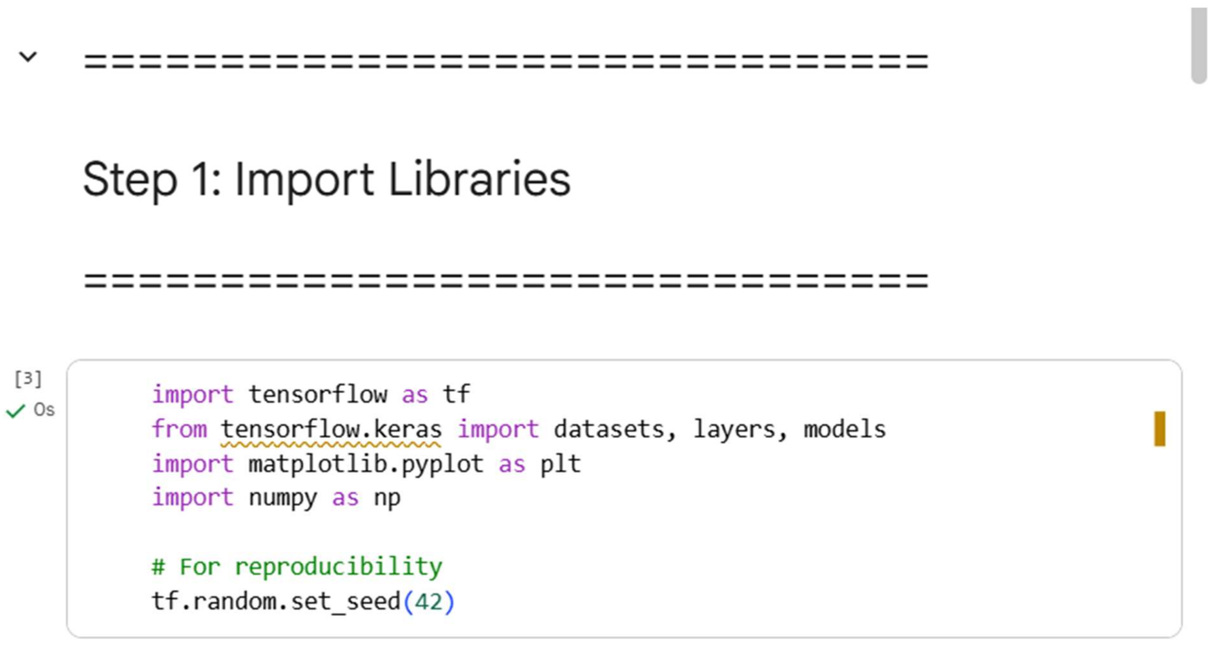
Introduction:

Convolutional Neural Networks (CNNs) are specialized deep learning architectures designed to process data with a grid-like topology, such as images. They have revolutionized computer vision by automatically learning hierarchical feature representations directly from raw pixel data.
Instead of manually designing features (as in traditional machine learning), CNNs use **convolutional layers** to detect spatial patterns like edges and textures, **pooling layers** to reduce dimensionality while preserving essential information, and **fully connected layers** for classification.
By leveraging shared weights and local connections, CNNs efficiently capture spatial dependencies, reduce parameters, and achieve remarkable accuracy in tasks such as image classification, object detection, and facial recognition.



Implementation:

Step 1: Select data
Step 2: Apply CNN algorithm
Step 3: Find performance measures.
Step 4: Visualize the result



====================================

Step 1: Import Libraries

====================================

[3]
✓ 0s

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np

# For reproducibility
tf.random.set_seed(42)
```

==================================

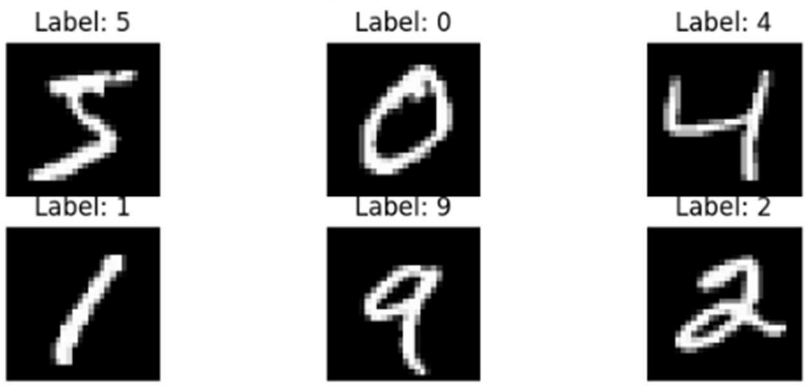
Step 2: Load and Preprocess Dataset

==================================

[4]
2s

```python
# Load MNIST dataset (handwritten digits 0-9)
(x_train, y_train), (x_test, y_test) = datasets.mnist.load_data()

# Reshape (add channel dimension) and normalize pixel values
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

print("Training data shape:", x_train.shape)
print("Testing data shape:", x_test.shape)

# Display few sample images
plt.figure(figsize=(8, 3))
for i in range(6):
    plt.subplot(2, 3, i + 1)
    plt.imshow(x_train[i].reshape(28,28), cmap='gray')
    plt.title(f"Label: {y_train[i]}")
    plt.axis('off')
plt.show()
```

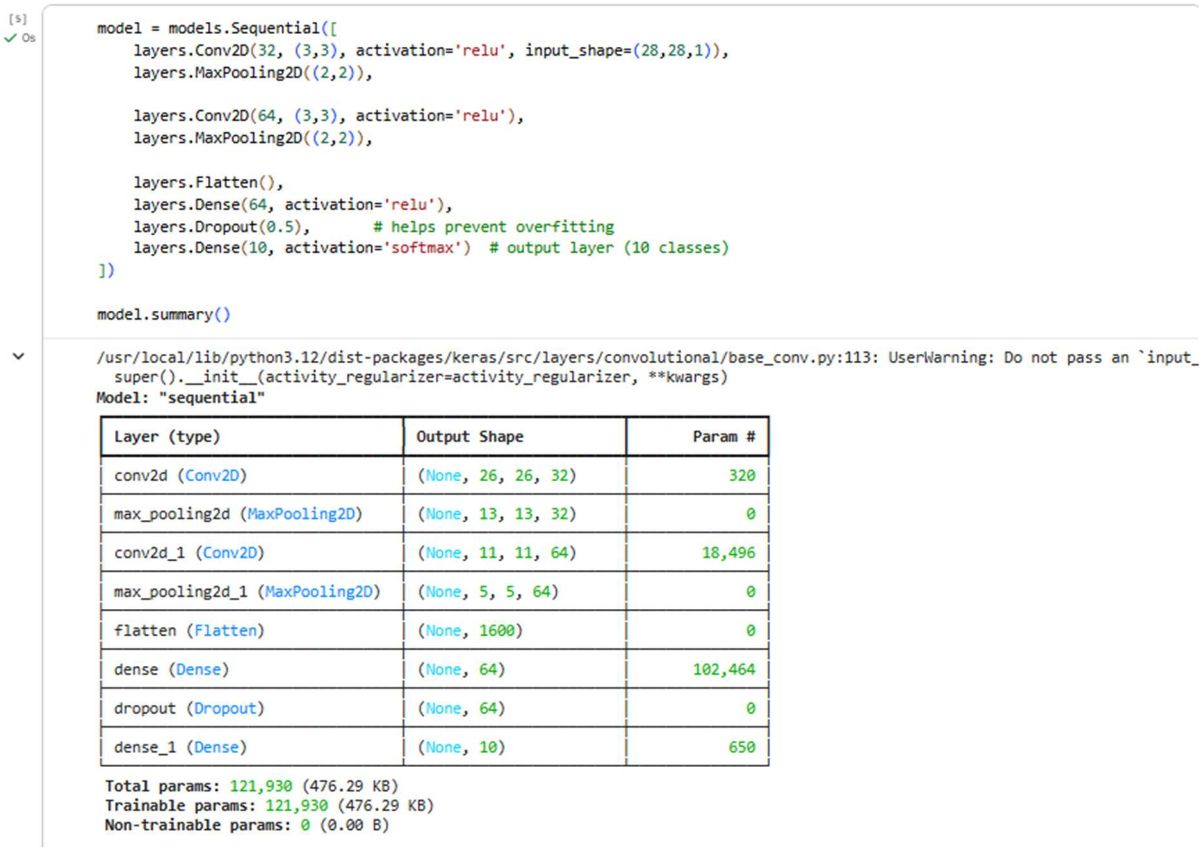
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Training data shape: (60000, 28, 28, 1)
Testing data shape: (10000, 28, 28, 1)
```



================================

Step 3: Define CNN Model

================================



```
model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    layers.MaxPooling2D((2,2)),

    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.5),       # helps prevent overfitting
    layers.Dense(10, activation='softmax')  # output layer (10 classes)
])

model.summary()
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input_
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
```

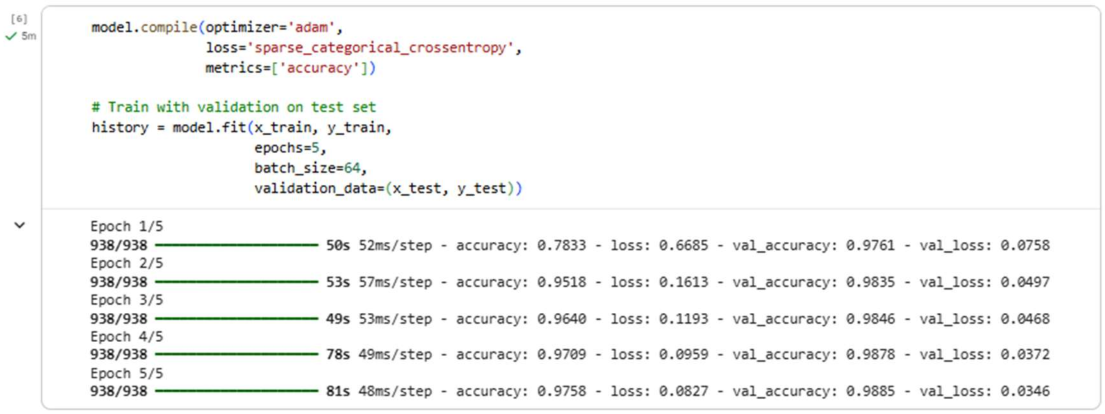
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dense (Dense)	(None, 64)	102,464
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 10)	650

```
 Total params: 121,930 (476.29 KB)
 Trainable params: 121,930 (476.29 KB)
 Non-trainable params: 0 (0.00 B)
```

================================

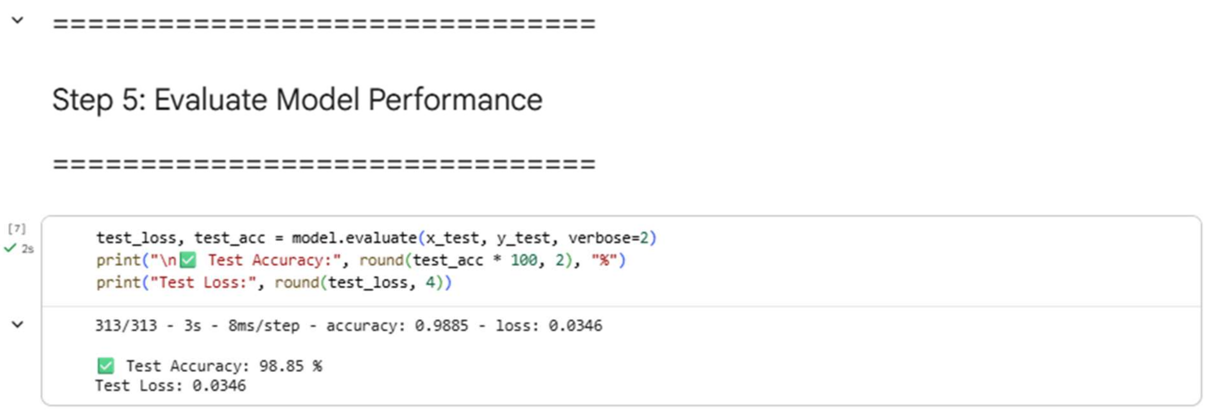
Step 4: Compile and Train the Model

================================



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

# Train with validation on test set
history = model.fit(x_train, y_train,
                    epochs=5,
                    batch_size=64,
                    validation_data=(x_test, y_test))
```

```
Epoch 1/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 50s 52ms/step - accuracy: 0.7833 - loss: 0.6685 - val_accuracy: 0.9761 - val_loss: 0.0758
Epoch 2/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 53s 57ms/step - accuracy: 0.9518 - loss: 0.1613 - val_accuracy: 0.9835 - val_loss: 0.0497
Epoch 3/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 49s 53ms/step - accuracy: 0.9640 - loss: 0.1193 - val_accuracy: 0.9846 - val_loss: 0.0468
Epoch 4/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 78s 49ms/step - accuracy: 0.9709 - loss: 0.0959 - val_accuracy: 0.9878 - val_loss: 0.0372
Epoch 5/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 81s 48ms/step - accuracy: 0.9758 - loss: 0.0827 - val_accuracy: 0.9885 - val_loss: 0.0346
```



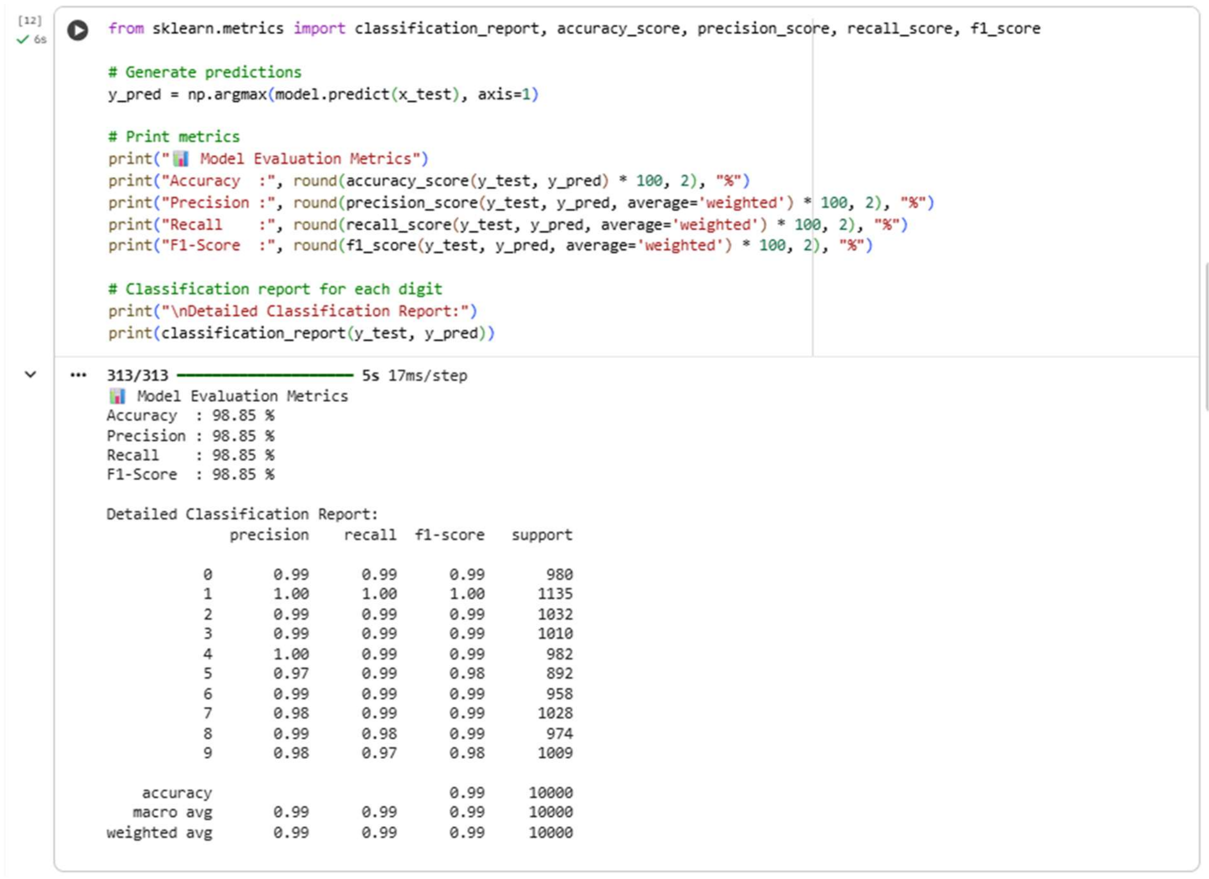
================================

Step 5: Evaluate Model Performance

================================

```
[7]
✓ 2s
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print("\n✅ Test Accuracy:", round(test_acc * 100, 2), "%")
print("Test Loss:", round(test_loss, 4))
```

```
313/313 - 3s - 8ms/step - accuracy: 0.9885 - loss: 0.0346

✅ Test Accuracy: 98.85 %
Test Loss: 0.0346
```



```
[12]
✓ 6s
from sklearn.metrics import classification_report, accuracy_score, precision_score, recall_score, f1_score

# Generate predictions
y_pred = np.argmax(model.predict(x_test), axis=1)

# Print metrics
print("📊 Model Evaluation Metrics")
print("Accuracy  :", round(accuracy_score(y_test, y_pred) * 100, 2), "%")
print("Precision :", round(precision_score(y_test, y_pred, average='weighted') * 100, 2), "%")
print("Recall    :", round(recall_score(y_test, y_pred, average='weighted') * 100, 2), "%")
print("F1-Score  :", round(f1_score(y_test, y_pred, average='weighted') * 100, 2), "%")

# Classification report for each digit
print("\nDetailed Classification Report:")
print(classification_report(y_test, y_pred))
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 5s 17ms/step
📊 Model Evaluation Metrics
Accuracy  : 98.85 %
Precision : 98.85 %
Recall    : 98.85 %
F1-Score  : 98.85 %

Detailed Classification Report:
              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       0.99      0.99      0.99      1010
           4       1.00      0.99      0.99       982
           5       0.97      0.99      0.98       892
           6       0.99      0.99      0.99       958
           7       0.98      0.99      0.99      1028
           8       0.99      0.98      0.99       974
           9       0.98      0.97      0.98      1009

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

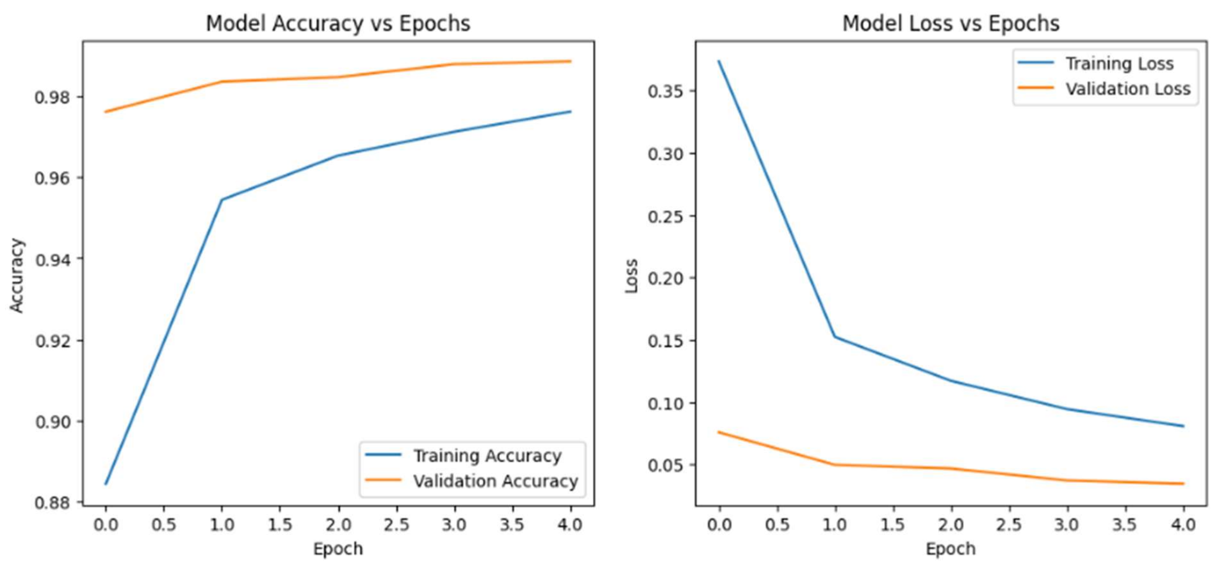
================================

Step 6: Visualize Training Results

================================

[8]
0s

```
plt.figure(figsize=(12,5))

# Accuracy Plot
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy vs Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Loss Plot
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss vs Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legendZ()

plt.show()
```



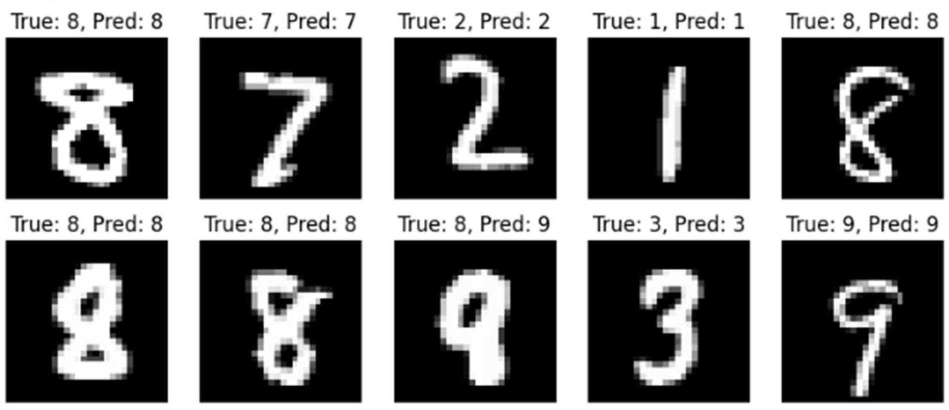
================================

Step 7: Visualize Predictions

================================

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

# Show 10 random predictions
plt.figure(figsize=(10,4))
for i in range(10):
    idx = np.random.randint(0, len(x_test))
    plt.subplot(2,5,i+1)
    plt.imshow(x_test[idx].reshape(28,28), cmap='gray')
    plt.title(f"True: {y_test[idx]}, Pred: {pred_labels[idx]}")
    plt.axis('off')
plt.show()
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step
```



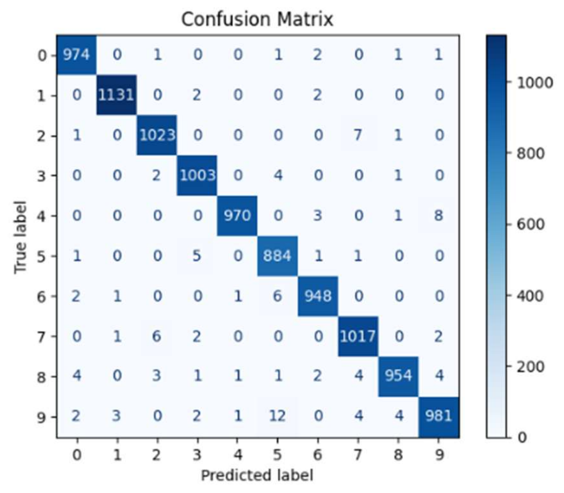
================================

Step 8: Confusion Matrix

================================

[11] ✓ 0s

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_test, pred_labels)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Blues', values_format='d')
plt.title("Confusion Matrix")
plt.show()
```



================================

Step 9: Testing with User Input Image

================================

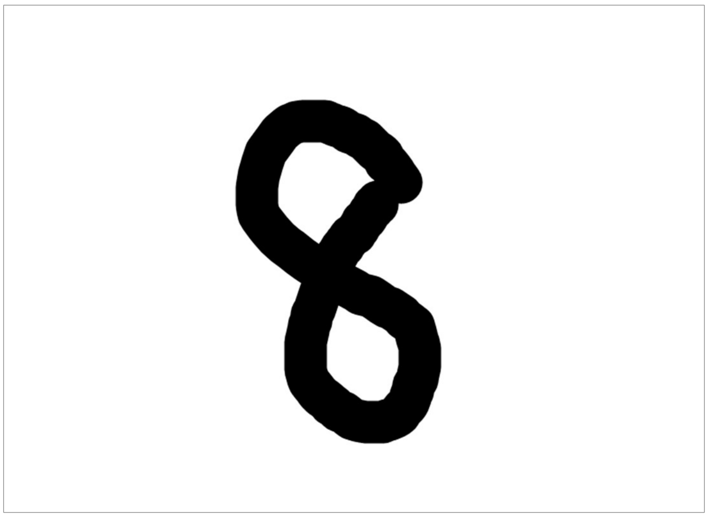
[16] ✓ 0s

```
import cv2
```

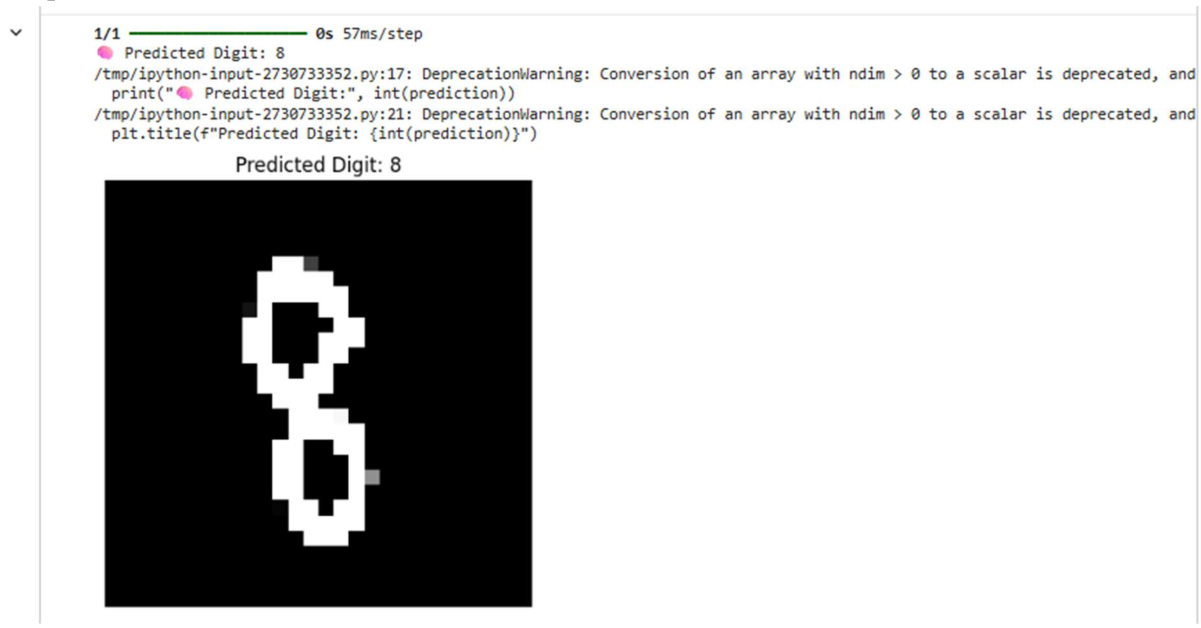
[15] ✓ 0s

```
# Load your image
img_path = '/content/Screenshot 2025-11-07 122049.png'
img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)

# Invert colors (white background, black digit)
img = 255 - img

# Resize to 28x28 pixels (MNIST format)
img_resized = cv2.resize(img, (28,28))

# Normalize and reshape for model input
img_resized = img_resized.astype('float32') / 255.0
img_input = np.expand_dims(img_resized, axis=(0,-1))

# Predict
prediction = np.argmax(model.predict(img_input), axis=1)
print("🧠 Predicted Digit:", int(prediction))

# Show image
plt.imshow(img_resized, cmap='gray')
plt.title(f"Predicted Digit: {int(prediction)}")
plt.axis('off')
plt.show()
```

Input Image:



Output:



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
🧠 Predicted Digit: 8
/tmp/ipython-input-2730733352.py:17: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and
  print("🧠 Predicted Digit:", int(prediction))
/tmp/ipython-input-2730733352.py:21: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and
  plt.title(f"Predicted Digit: {int(prediction)}")
```

<u>Post Lab Descriptive Questions:</u>

1. **Explain how convolutional layers in a CNN help in automatically extracting features from images. How does this differ from traditional machine learning methods that use manual feature extraction?**
   Convolutional layers apply multiple learnable filters across the image to capture spatial features such as edges, curves, and textures. These filters automatically adapt during training to identify important visual patterns. Unlike traditional machine learning methods (e.g., SVMs, Random Forests) that depend on manually crafted features, CNNs perform automatic feature extraction, learning both low-level and high-level representations directly from data, reducing human bias and improving accuracy.

2. **What is the role of filters (kernels) in a CNN, and how does changing the number or size of filters affect the model's learning capability and computational complexity?**
   Filters (kernels) are small matrices that slide over the input image to detect local patterns.
   - Number of filters: More filters enable the model to learn diverse features but increase computation and memory usage.
   - Filter size: Larger filters capture broader contextual information, while smaller filters focus on finer details. Hence, filter design involves a trade-off between accuracy and computational efficiency.

3. **Why is pooling used in CNNs? Compare max pooling and average pooling in terms of their effect on feature retention and model performance.**
   Pooling layers reduce spatial dimensions, lower computational cost, and help control overfitting by summarizing feature map regions.
   - Max Pooling: Selects the maximum value, preserving the most prominent features and improving robustness to noise.
   - Average Pooling: Computes the mean of values, maintaining smoothness but possibly losing strong feature signals.

   In practice, max pooling is more common as it retains sharper, more discriminative features.

4. **If your CNN model achieves high accuracy on training data but poor accuracy on test data, what could be the reasons? Suggest at least two methods to reduce overfitting in CNNs.**
   This situation indicates overfitting, where the model memorizes training examples instead of generalizing.
   Reasons:
   1. Model too deep or complex for the dataset
   2. Lack of sufficient or diverse training data
      Solutions:
   3. Data Augmentation: Introduce variations (rotation, flipping, scaling) to improve generalization.
   4. Regularization: Apply Dropout or L2 weight regularization to prevent over-dependence on specific neurons.
   5. Early Stopping: Halt training when validation loss starts to increase to avoid overfitting.