EXPERIMENT 8B:

import numpy as np

import matplotlib.pyplot as plt

from skimage import data, color, io, img\_as\_ubyte

from skimage.filters import sobel

from skimage.segmentation import watershed, mark\_boundaries

from skimage.measure import label

from skimage.color import label2rgb

from skimage.morphology import dilation, erosion, square

from skimage.feature import peak\_local\_max

from scipy import ndimage as ndi

# Load a sample image from skimage

image = data.coins() # Replace with your own image if needed

# Convert the image to grayscale

gray\_image = color.rgb2gray(image) if len(image.shape) == 3 else image

# Compute the gradient magnitude using the Sobel filter

elevation\_map = sobel(gray\_image)

# Markers for the background and the objects

markers = np.zeros\_like(gray\_image)

markers[gray\_image < 30] = 1 # Background

markers[gray\_image > 150] = 2 # Objects

# Perform the Watershed algorithm

segmentation = watershed(elevation\_map, markers)

# Label the segmented objects

labeled\_segments = label(segmentation)

# Visualize the results

fig, axes = plt.subplots(1, 3, figsize=(15, 5))

ax = axes.ravel()

ax[0].imshow(gray\_image, cmap='gray')

ax[0].set\_title('Original Image')

ax[1].imshow(elevation\_map, cmap='gray')

ax[1].set\_title('Elevation Map')

ax[2].imshow(mark\_boundaries(gray\_image, labeled\_segments))

ax[2].set\_title('Segmentation')

for a in ax:

a.axis('off')

plt.tight\_layout()

plt.show()

# Optional: Further evaluation using metrics (if ground truth is available)

# For demonstration, we assume a ground truth mask is available

# ground\_truth = ...

# Compute evaluation metrics

# jaccard\_index = jaccard\_score(ground\_truth.flatten(), labeled\_segments.flatten(), average='macro')

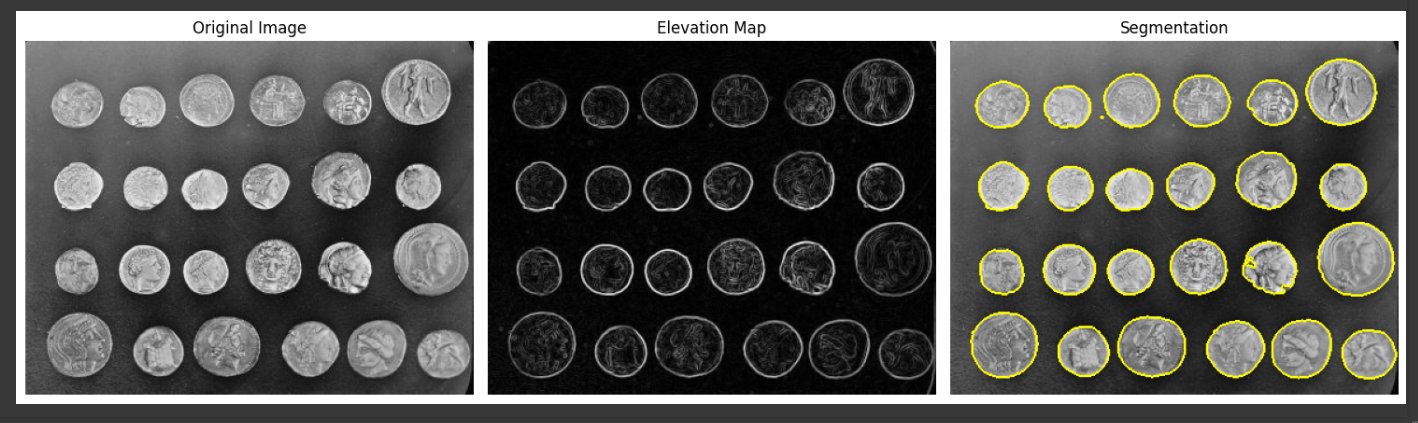
# dice\_coefficient = f1\_score(ground\_truth.flatten(), labeled\_segments.flatten(), average='macro')

# Print evaluation metrics

# print(f'Jaccard Index: {jaccard\_index}')

# print(f'Dice Coefficient: {dice\_coefficient}')

OUTPUT:



EXPERIMENT 9A:

import tensorflow as tf

from tensorflow.keras import datasets, layers, models

import matplotlib.pyplot as plt

# Load and preprocess the MNIST dataset

(X\_train, y\_train), (X\_test, y\_test) = datasets.mnist.load\_data()

X\_train, X\_test = X\_train / 255.0, X\_test / 255.0 # Normalize the images to [0, 1]

# Flatten the images for the fully connected layers

X\_train = X\_train.reshape(-1, 28 \* 28)

X\_test = X\_test.reshape(-1, 28 \* 28)

# Build the neural network model

model = models.Sequential([

layers.Dense(128, activation='tanh', input\_shape=(28 \* 28,)),

layers.Dense(64, activation='tanh'),

layers.Dense(10, activation='softmax')

])

# Compile the model

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

# Train the model

history = model.fit(X\_train, y\_train, epochs=10,

validation\_data=(X\_test, y\_test))

# Evaluate the model

test\_loss, test\_acc = model.evaluate(X\_test, y\_test, verbose=2)

print(f'\nTest accuracy: {test\_acc}')

# Plot training & validation accuracy values

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

# Plot training & validation loss values

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Model loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

plt.tight\_layout()

plt.show()

# Visualize some predictions

predictions = model.predict(X\_test)

plt.figure(figsize=(10, 10))

for i in range(25):

plt.subplot(5, 5, i + 1)

plt.xticks([])

plt.yticks([])

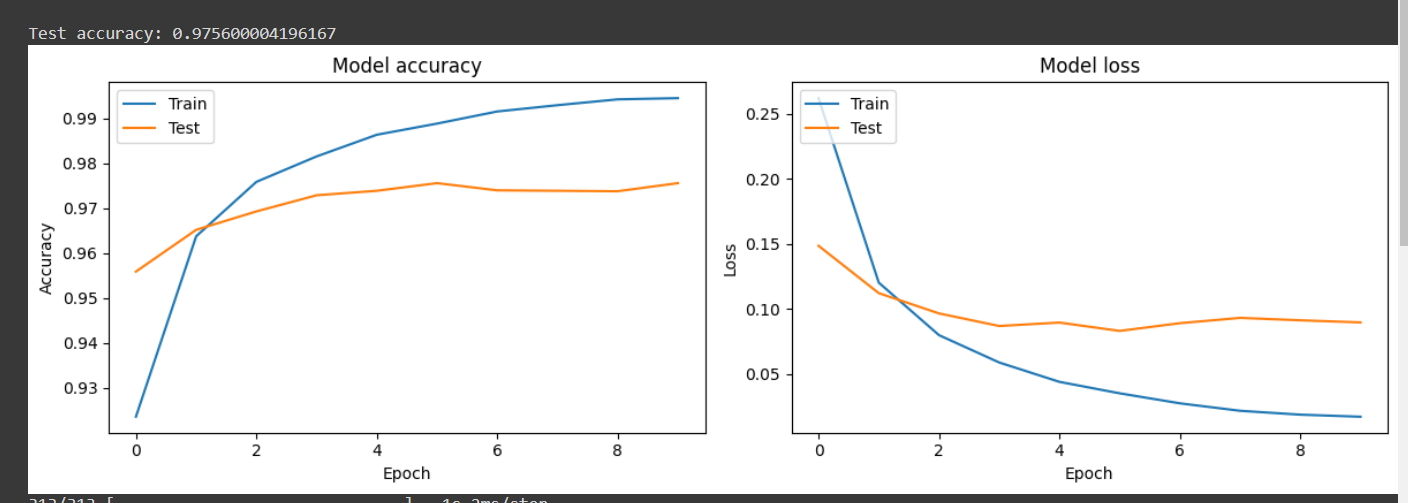
plt.grid(False)

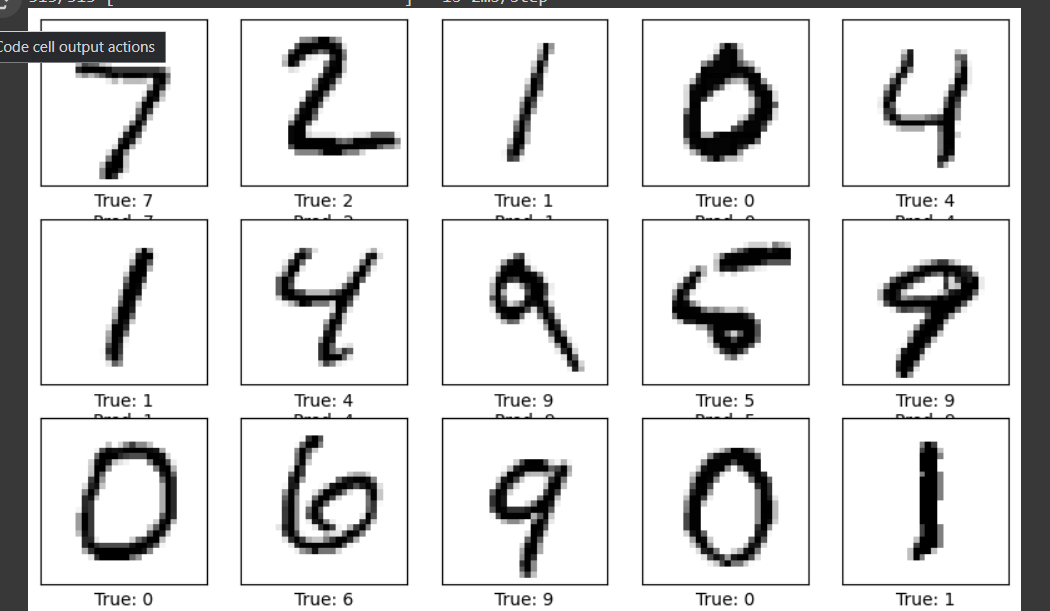
plt.imshow(X\_test[i].reshape(28, 28), cmap=plt.cm.binary)

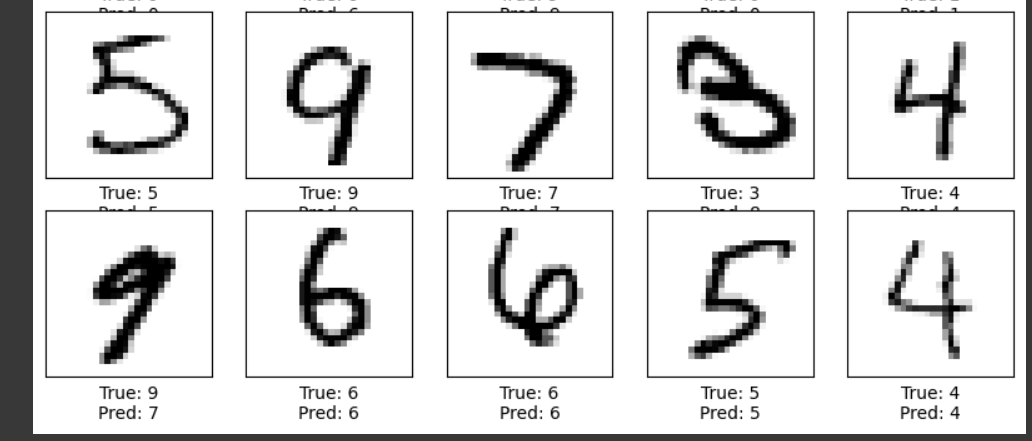
plt.xlabel(f"True: {y\_test[i]}\nPred: {predictions[i].argmax()}")

plt.show()

OUTPUT:







EXPERIMENT 9B::

import tensorflow as tf

from tensorflow.keras import datasets, layers, models

import matplotlib.pyplot as plt

# Load and preprocess the MNIST dataset

(X\_train, y\_train), (X\_test, y\_test) = datasets.mnist.load\_data()

X\_train, X\_test = X\_train / 255.0, X\_test / 255.0 # Normalize the images to [0, 1]

# Flatten the images for the fully connected layers

X\_train = X\_train.reshape(-1, 28 \* 28)

X\_test = X\_test.reshape(-1, 28 \* 28)

# Build the neural network model

model = models.Sequential([

layers.Dense(128, activation='sigmoid', input\_shape=(28 \* 28,)),

layers.Dense(64, activation='sigmoid'),

layers.Dense(10, activation='softmax')

])

# Compile the model

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

# Train the model

history = model.fit(X\_train, y\_train, epochs=10,

validation\_data=(X\_test, y\_test))

# Evaluate the model

test\_loss, test\_acc = model.evaluate(X\_test, y\_test, verbose=2)

print(f'\nTest accuracy: {test\_acc}')

# Plot training & validation accuracy values

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

# Plot training & validation loss values

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Model loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

plt.tight\_layout()

plt.show()

# Visualize some predictions

predictions = model.predict(X\_test)

plt.figure(figsize=(10, 10))

for i in range(25):

plt.subplot(5, 5, i + 1)

plt.xticks([])

plt.yticks([])

plt.grid(False)

plt.imshow(X\_test[i].reshape(28, 28), cmap=plt.cm.binary)

plt.xlabel(f"True: {y\_test[i]}\nPred: {predictions[i].argmax()}")

plt.show()

OUTPUT:

