

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

**PRACTIAL RECORD BOOK**

**Subject: Artificial Neural Networks and Deep Learning (22ADG64)**

NAME USN YEAR/SEM SECTION

BRANCH

: NANDISH KUMAR T R

: 1NT22AD037

: 3/ 6th SEM

: A

:ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

A blue and white logo

Description automatically generated

**CERTIFICATE**

This is to certify that Mr. Nandish Kumar T R , studying in semester 6 Branch Artificial Intelligence and Data Science has satisfactorily completed the course of experiments in Laboratory for the academic session 2024-2025 as prescribed by the Visvesvaraya Technological University.

Register Number of the Candidate: 1NT22AD037

Signature of the Internal Examiner Signature of HOD

Date: Date:

| **Sl. No.** | **Program / Exercise** |
| --- | --- |
|  | 1. Create a Python program to implement a basic perceptron (single layer) from scratch using TensorFlow. |
| 1. Develop a Python program to demonstrate a simple perceptron model to represent AND, OR logic and visualize its decision boundary |
| 1. Write a Python program to train the perceptron using a straightforward dataset and visualize the decision boundary of the trained perceptron |
|  | 1. Write a Python program to implement a feedforward neural network and backpropagation algorithm for realizing the XOR logic function |
| 1. Write a Python program to train the network on a dataset using gradient descent. |
|  | 1. Write a Python program to demonstrate the use of various activation functions including Linear, Sigmoid, tanh, ReLU, Leaky ReLU, PReLU, ELU, SELU, Softmax, GELU, and Swish. Illustrate the behavior of these activation functions by plotting graphs |
| 1. i) Write a Python program to implement a simple multilayer artificial neural network (ANN) architecture using TensorFlow for image classification on a dataset such as MNIST or CIFAR-10.   ii) In the proposed architecture from 2d) i), experiment with different activation functions and observe their impact on training. |
|  | 1. Build a simple CNN architecture for image classification. Train the CNN on a dataset like MNIST or CIFAR-10. Visualize the learned filters and feature maps. 2. Given Data: Consider the RGB Action Recognition (Video) dataset given. Dataset Link: <https://drive.google.com/drive/folders/1XZJxsDsPVje0GdCIyAVwltpfWTFu0GjO?usp=drive_link> 3. Dataset Creation: Create an RGB Image dataset of the four classes given: (Crop Images from Video)   (i) Punching – 800 Images  (ii) Kicking – 800 Images  (iii) Taking Selfie – 800 Images  (iv) Pushing – 800 Images   1. Data Preprocessing:   (i) Normalization  (ii) Data Augmentation (Optional)   1. Algorithm/Model: Create a Convolutional Neural Network 2. Model Evaluation:    Train a CNN model on the dataset created.   Evaluate the model’s performance using   1. Accuracy and Confusion Matrix. |
|  | 1. Write a Python program to implement a basic RNN for sequence prediction. 2. Write a Python program to implement an LSTM and compare its performance on a sequential task |
|  | 1. Write a Python program to implement a Autoencoder for image reconstruction. 2. Write a Python program to implement a Variational Autoencoder for image reconstruction. 3. Write a Python program to implement a GAN to generate synthetic images. |

**INDEX SHEET**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SL.NO** | **Programs** | **Page no** | **Date of Execution** | **Internal Examiner Signature** |
|  | 1. Create a Python program to implement a basic perceptron (single layer) from scratch using TensorFlow. | 1-2 | 19/3/25 |  |
| 1. Develop a Python program to demonstrate a simple perceptron model to represent AND, OR logic and visualize its decision boundary | 3 | 19/3/25 |  |
| 1. Write a Python program to train the perceptron using a straightforward dataset and visualize the decision boundary of the trained perceptron | 4-5 | 19/3/25 |  |
|  | 1. Write a Python program to implement a feedforward neural network and backpropagation algorithm for realizing the XOR logic function | 6-7 | 19/3/25 |  |
| 1. Write a Python program to train the network on a dataset using gradient descent. | 8-9 | 19/3/25 |  |
|  | 1. Write a Python program to demonstrate the use of various activation functions including Linear, Sigmoid, tanh, ReLU, Leaky ReLU, PReLU, ELU, SELU, Softmax, GELU, and Swish. Illustrate the behavior of these activation functions by plotting graphs | 10-11 | 25/3/25 |  |
| 1. i) Write a Python program to implement a simple multilayer artificial neural network (ANN) architecture using TensorFlow for image classification on a dataset such as MNIST or CIFAR-10.   ii) In the proposed architecture from 3c) i), experiment with different activation functions and observe their impact on training. | 12-13 | 25/3/25 |  |
|  | 1. Build a simple CNN architecture for image classification. Train the CNN on a dataset like MNIST or CIFAR-10. Visualize the learned filters and feature maps. 2. Given Data: Consider the RGB Action Recognition (Video) dataset given. Dataset Link: <https://drive.google.com/drive/folders/1XZJxsDsPVje0GdCIyAVwltpfWTFu0GjO?usp=drive_link> 3. Dataset Creation: Create an RGB Image dataset of the four classes given: (Crop Images from Video)   (i) Punching – 800 Images  (ii) Kicking – 800 Images  (iii) Taking Selfie – 800 Images  (iv) Pushing – 800 Images   1. Data Preprocessing:   (i) Normalization  (ii) Data Augmentation (Optional)   1. Algorithm/Model: Create a Convolutional Neural Network 2. Model Evaluation:    Train a CNN model on the dataset created.   Evaluate the model’s performance using   1. Accuracy and Confusion Matrix. | 14-16  17-21 | 3/5/25 |  |
|  | 1. Write a Python program to implement a basic RNN for sequence prediction. 2. Write a Python program to implement an LSTM and compare its performance on a sequential task | 22-23  24 | 20/5/25 |  |
|  | 1. Write a Python program to implement a Autoencoder for image reconstruction. 2. Write a Python program to implement a Variational Autoencoder for image reconstruction. 3. Write a Python program to implement a GAN to generate synthetic images. | 25-26  27-32  33-36 | 20/5/25 |  |

## Program 1. a)

## Problem Definition:

## Create a Python program to implement a basic perceptron (single layer) from scratch using TensorFlow.

## Code:

## import numpy as np

## class Perceptron:

## def \_\_init\_\_(self, learning\_rate=0.01, n\_iters=1000):

## self.lr = learning\_rate

## self.n\_iters = n\_iters

## self.activation\_func = self.\_unit\_step\_function

## self.weights = None

## self.bias = None

## def fit(self, X, y):

## n\_samples, n\_features = X.shape

## self.weights = np.zeros(n\_features)

## self.bias = 0

## for \_ in range(self.n\_iters):

## for idx, x\_i in enumerate(X):

## linear\_output = np.dot(x\_i, self.weights) + self.bias

## y\_predicted = self.activation\_func(linear\_output)

## update = self.lr \* (y[idx] - y\_predicted)

## self.weights += update \* x\_i

## self.bias += update

## def predict(self, X):

## linear\_output = np.dot(X, self.weights) + self.bias

## y\_predicted = self.activation\_func(linear\_output)

## return y\_predicted

## def \_unit\_step\_function(self, x):

## return np.where(x >= 0, 1, 0)

## if \_\_name\_\_ == "\_\_main\_\_":

## # Sample training data (AND Logic Gate)

## X = np.array([

## [0, 0],

## [0, 1],

## [1, 0],

## [1, 1]

## ])

## y = np.array([0, 0, 0, 1])

## perceptron = Perceptron(learning\_rate=0.1, n\_iters=10)

## perceptron.fit(X, y)

## print("Predictions on training data:")

## predictions = perceptron.predict(X)

## print(predictions)

## OUTPUT:

## Screenshot 2025-05-20 at 6.45.31 PM

## Program 1. b)

## Problem Definition:

## Develop a Python program to demonstrate a simple perceptron model to represent AND, OR logic and visualize its decision boundary

## Code:

## import numpy as np

## # Input data for AND gate

## X = np.array([

## [0, 0],

## [0, 1],

## [1, 0],

## [1, 1]

## ])

## Y = np.array([0, 0, 0, 1])

## weights = np.zeros(X.shape[1])

## bias = 0

## learning\_rate = 0.1

## for i in range(len(X)):

## linear\_output = np.dot(X[i], weights) + bias

## y\_pred = 1 if linear\_output >= 0 else 0

## error = Y[i] - y\_pred

## weights += learning\_rate \* error \* X[i]

## bias += learning\_rate \* error

## for i in range(len(X)):

## linear\_output = np.dot(X[i], weights) + bias

## y\_pred = 1 if linear\_output >= 0 else 0

## print(f"Input: {X[i]}, Predicted Output: {y\_pred}, Actual Output: {Y[i]}")

## OUTPUT:

## Screenshot 2025-05-20 at 6.52.29 PM

## Program 1. c)

## Problem Definition:

## Write a Python program to train the perceptron using a straightforward dataset and visualize the decision boundary of the trained perceptron

## Code:

## import numpy as np

## import matplotlib.pyplot as plt

## class Perceptron:

## def \_\_init\_\_(self, learning\_rate=0.01, n\_iters=1000):

## self.lr = learning\_rate

## self.n\_iters = n\_iters

## self.activation\_func = self.\_unit\_step\_func

## self.weights = None

## self.bias = None

## def fit(self, X, y):

## n\_samples, n\_features = X.shape

## self.weights = np.zeros(n\_features)

## self.bias = 0

## for \_ in range(self.n\_iters):

## for idx, x\_i in enumerate(X):

## linear\_output = np.dot(x\_i, self.weights) + self.bias

## y\_predicted = self.activation\_func(linear\_output)

## update = self.lr \* (y[idx] - y\_predicted)

## self.weights += update \* x\_i

## self.bias += update

## def predict(self, X):

## linear\_output = np.dot(X, self.weights) + self.bias

## return self.activation\_func(linear\_output)

## def \_unit\_step\_func(self, x):

## return np.where(x >= 0, 1, 0)

## X = np.array([

## [1, 1],

## [2, 1],

## [1, 2],

## [2, 2],

## [3, 3],

## [4, 4]

## ])

## y = np.array([0, 0, 0, 1, 1, 1])

## perceptron = Perceptron(learning\_rate=0.1, n\_iters=100)

## perceptron.fit(X, y)

## fig = plt.figure(figsize=(8,6))

## plt.scatter(X[:, 0], X[:, 1], c=y, cmap='bwr', s=50)

## x1\_values = np.linspace(0, 5, 10)

## x2\_values = -(perceptron.weights[0] / perceptron.weights[1]) \* x1\_values - (perceptron.bias / perceptron.weights[1])

## plt.plot(x1\_values, x2\_values, 'k--', label="Decision Boundary")

## plt.xlabel('Feature 1')

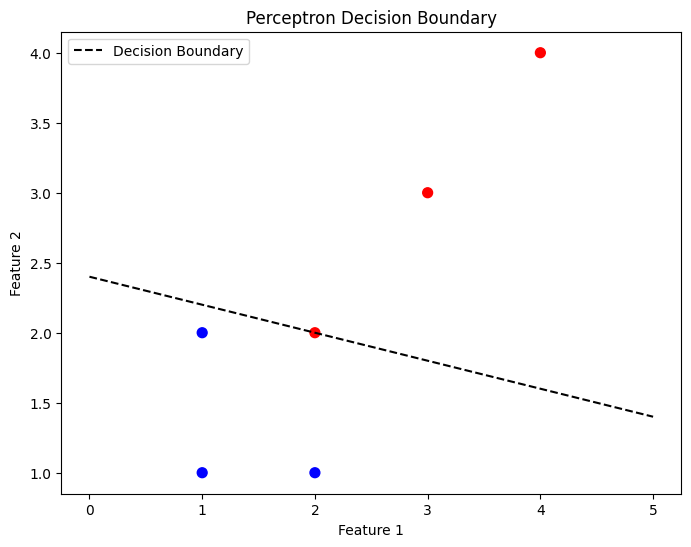
## plt.ylabel('Feature 2')

## plt.title('Perceptron Decision Boundary')

## plt.legend()

## plt.show()

## OUTPUT :



## Program 2. a)

## Problem Definition:

## Write a Python program to implement a feedforward neural network and backpropagation algorithm for realizing the XOR logic function

## 

Code:

import numpy as np

def sigmoid(x):

return 1 / (1 + np.exp(-x))

def sigmoid\_derivative(x):

return x \* (1 - x)

X = np.array([[0, 0],

[0, 1],

[1, 0],

[1, 1]])

y = np.array([[0],

[1],

[1],

[0]])

input\_layer\_neurons = X.shape[1]

hidden\_layer\_neurons = 4

output\_neurons = 1

np.random.seed(42)

weights\_input\_hidden = np.random.uniform(size=(input\_layer\_neurons, hidden\_layer\_neurons))

weights\_hidden\_output = np.random.uniform(size=(hidden\_layer\_neurons, output\_neurons))

bias\_hidden = np.random.uniform(size=(1, hidden\_layer\_neurons))

bias\_output = np.random.uniform(size=(1, output\_neurons))

learning\_rate = 0.1

epochs = 10000

for epoch in range(epochs):

hidden\_layer\_input = np.dot(X, weights\_input\_hidden) + bias\_hidden

hidden\_layer\_output = sigmoid(hidden\_layer\_input)

final\_input = np.dot(hidden\_layer\_output, weights\_hidden\_output) + bias\_output

predicted\_output = sigmoid(final\_input)

error = y - predicted\_output

d\_predicted\_output = error \* sigmoid\_derivative(predicted\_output)

error\_hidden\_layer = d\_predicted\_output.dot(weights\_hidden\_output.T)

d\_hidden\_layer = error\_hidden\_layer \* sigmoid\_derivative(hidden\_layer\_output)

weights\_hidden\_output += hidden\_layer\_output.T.dot(d\_predicted\_output) \* learning\_rate

weights\_input\_hidden += X.T.dot(d\_hidden\_layer) \* learning\_rate

bias\_output += np.sum(d\_predicted\_output, axis=0, keepdims=True) \* learning\_rate

bias\_hidden += np.sum(d\_hidden\_layer, axis=0, keepdims=True) \* learning\_rate

if epoch % 1000 == 0:

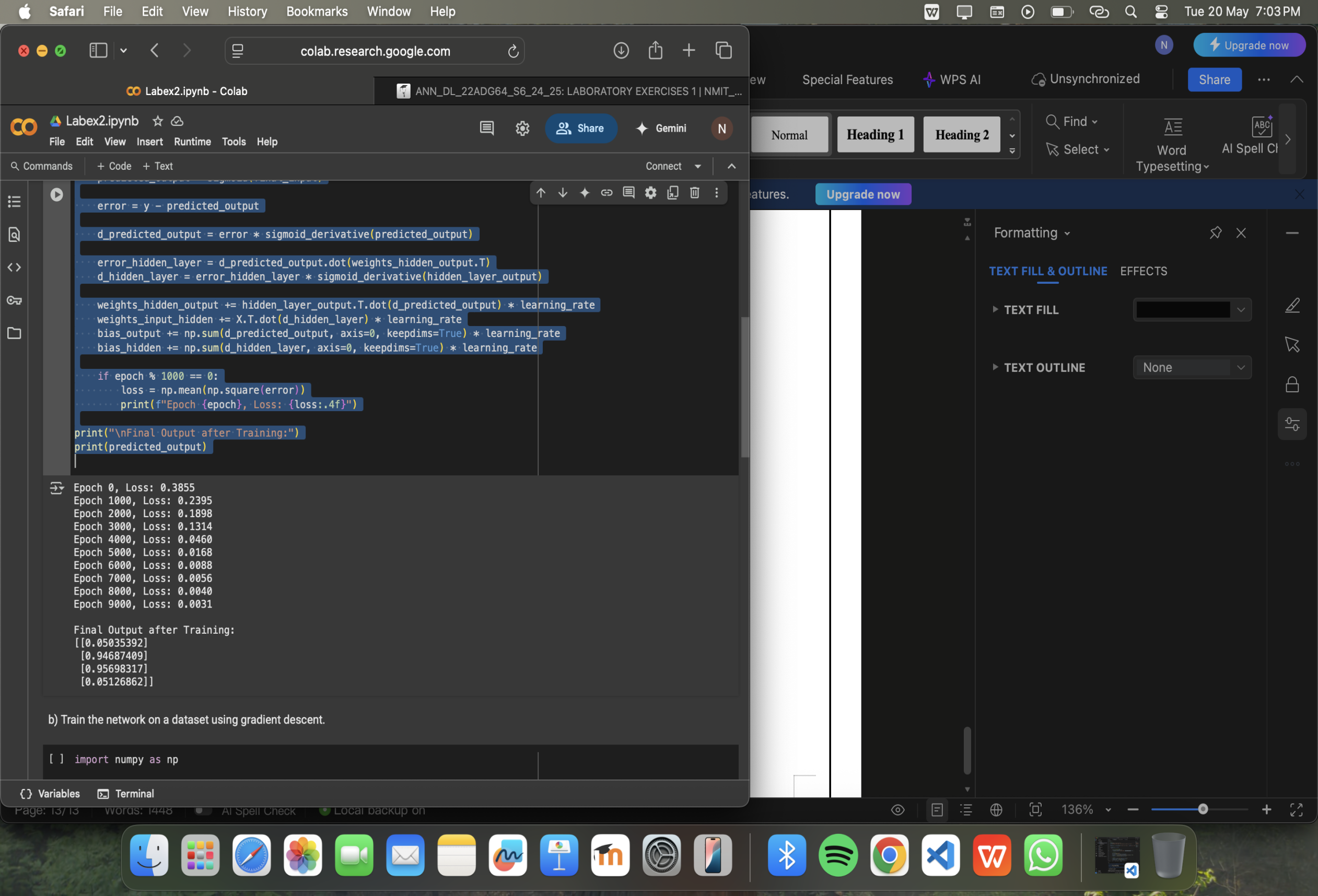
loss = np.mean(np.square(error))

print(f"Epoch {epoch}, Loss: {loss:.4f}")

print("\nFinal Output after Training:")

print(predicted\_output)

OUTPUT:



## Program 2. b)

## Problem Definition:

1. Write a Python program to train the network on a dataset using gradient descent.

Code:

import numpy as np

def sigmoid(x):

return 1 / (1 + np.exp(-x))

def sigmoid\_derivative(x):

return x \* (1 - x)

X = np.array([[0, 0],

[0, 1],

[1, 0],

[1, 1]])

y = np.array([[0],

[1],

[1],

[0]])

input\_neurons = X.shape[1]

hidden\_neurons = 4

output\_neurons = 1

np.random.seed(42)

weights\_input\_hidden = np.random.uniform(size=(input\_neurons, hidden\_neurons))

weights\_hidden\_output = np.random.uniform(size=(hidden\_neurons, output\_neurons))

bias\_hidden = np.random.uniform(size=(1, hidden\_neurons))

bias\_output = np.random.uniform(size=(1, output\_neurons))

learning\_rate = 0.1

epochs = 10000

for epoch in range(epochs):

# Forward Pass

hidden\_input = np.dot(X, weights\_input\_hidden) + bias\_hidden

hidden\_output = sigmoid(hidden\_input)

final\_input = np.dot(hidden\_output, weights\_hidden\_output) + bias\_output

predicted\_output = sigmoid(final\_input)

# Error Calculation

error = y - predicted\_output

# Backward Pass (Gradient Descent)

d\_predicted\_output = error \* sigmoid\_derivative(predicted\_output)

d\_hidden\_layer = d\_predicted\_output.dot(weights\_hidden\_output.T) \* sigmoid\_derivative(hidden\_output)

# Gradient Descent - Update weights and biases

weights\_hidden\_output += hidden\_output.T.dot(d\_predicted\_output) \* learning\_rate

weights\_input\_hidden += X.T.dot(d\_hidden\_layer) \* learning\_rate

bias\_output += np.sum(d\_predicted\_output, axis=0, keepdims=True) \* learning\_rate

bias\_hidden += np.sum(d\_hidden\_layer, axis=0, keepdims=True) \* learning\_rate

if epoch % 1000 == 0:

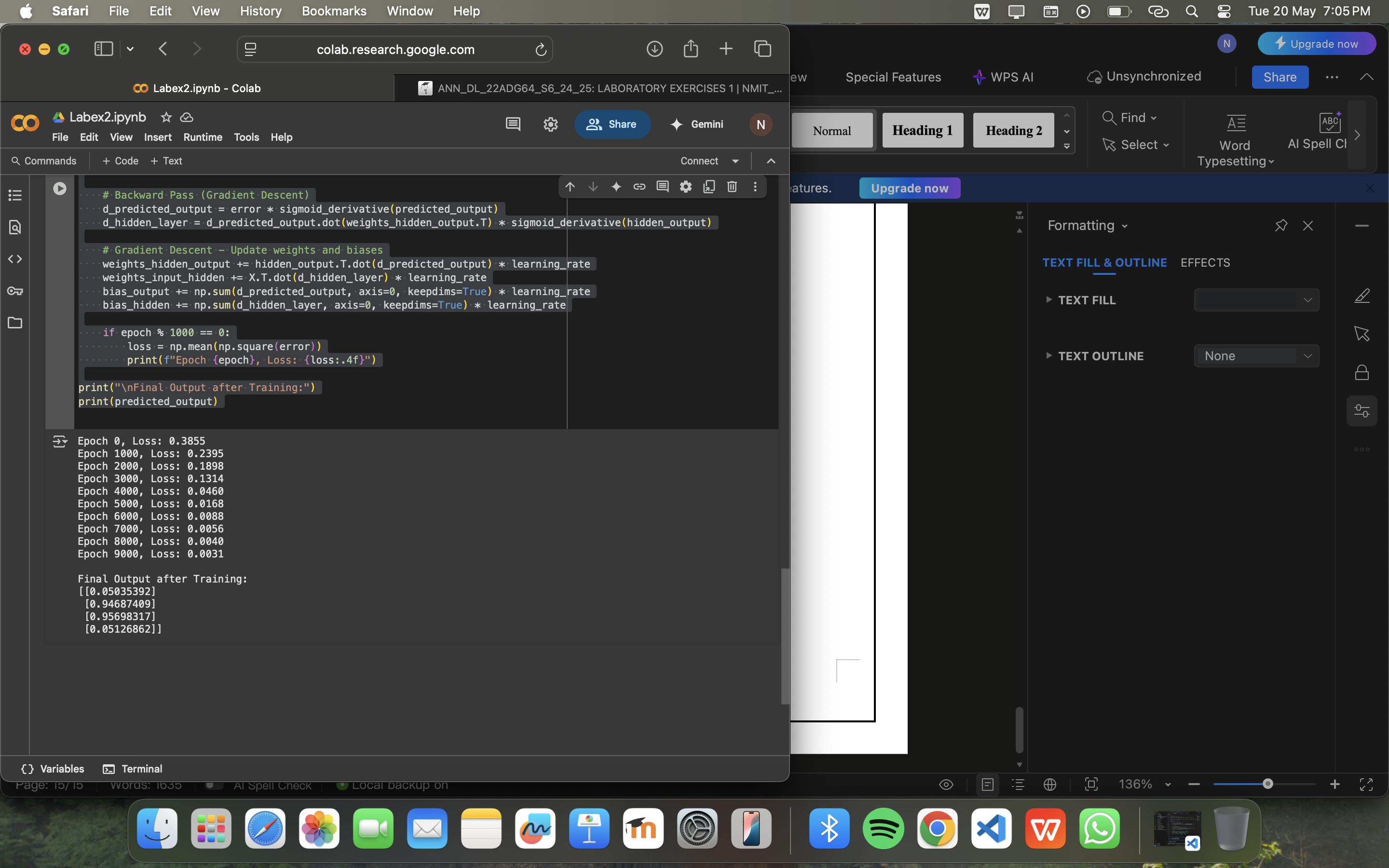
loss = np.mean(np.square(error))

print(f"Epoch {epoch}, Loss: {loss:.4f}")

print("\nFinal Output after Training:")

print(predicted\_output)

OUTPUT:



## Program 3. a)

## Problem Definition:

1. Write a Python program to demonstrate the use of various activation functions including Linear, Sigmoid, tanh, ReLU, Leaky ReLU, PReLU, ELU, SELU, Softmax, GELU, and Swish. Illustrate the behavior of these activation functions by plotting graphs

Code:

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

import torch

import torch.nn.functional as F

# Input range

x = np.linspace(-10, 10, 500)

# Activation Functions

activations = {

"Linear": x,

"Sigmoid": 1 / (1 + np.exp(-x)),

"Tanh": np.tanh(x),

"ReLU": np.maximum(0, x),

"Leaky ReLU": np.where(x > 0, x, 0.1 \* x),

"PReLU (α=0.2)": np.where(x > 0, x, 0.2 \* x),

"ELU (α=1.0)": np.where(x > 0, x, 1.0 \* (np.exp(x) - 1)),

"SELU": 1.0507 \* np.where(x > 0, x, 1.67326 \* (np.exp(x) - 1)),

"Softmax": np.exp(x) / np.sum(np.exp(x)),

"GELU (approx)": 0.5 \* x \* (1 + np.tanh(np.sqrt(2 / np.pi) \* (x + 0.044715 \* np.power(x, 3)))),

"Swish (β=1)": x / (1 + np.exp(-x))

}

plt.figure(figsize=(16, 18))

for i, (name, y) in enumerate(activations.items(), 1):

plt.subplot(4, 3, i)

plt.plot(x, y)

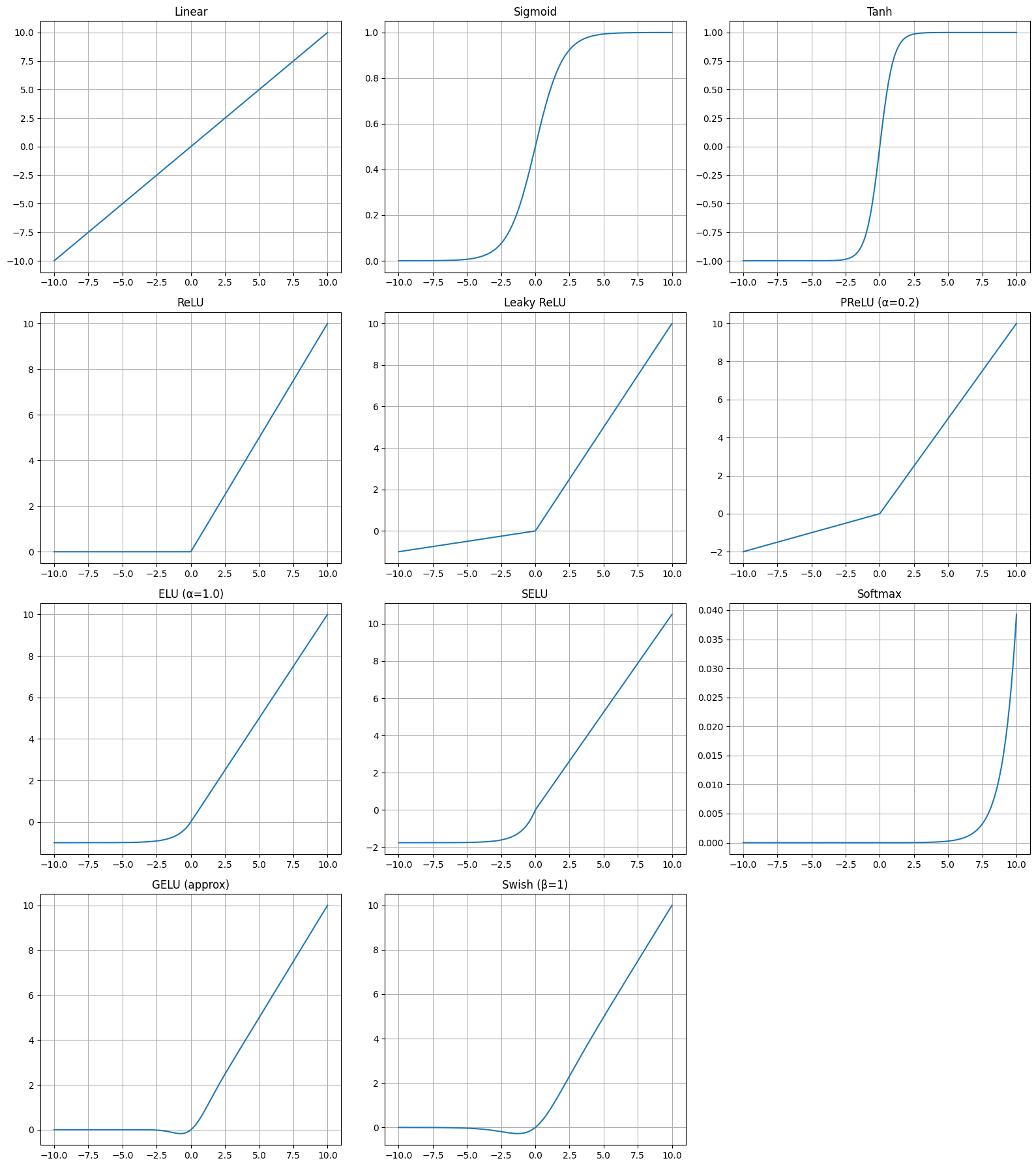
plt.title(name)

plt.grid(True)

plt.tight\_layout()

plt.show()

OUTPUT:



## Program 3. b)

## Problem Definition:

1. i) Write a Python program to implement a simple multilayer artificial neural network (ANN) architecture using TensorFlow for image classification on a dataset such as MNIST or CIFAR-10.

ii) In the proposed architecture from 3c) i), experiment with different activation functions and observe their impact on training.

Code:

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import mnist

from tensorflow.keras.utils import to\_categorical

# Load MNIST Dataset

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

x\_train = x\_train.reshape((x\_train.shape[0], 28 \* 28)) / 255.0

x\_test = x\_test.reshape((x\_test.shape[0], 28 \* 28)) / 255.0

y\_train = to\_categorical(y\_train)

y\_test = to\_categorical(y\_test)

# ANN Model

def build\_model(activation='relu'):

model = models.Sequential()

model.add(layers.Dense(128, activation=activation, input\_shape=(784,)))

model.add(layers.Dense(64, activation=activation))

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

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

return model

# Train Model with ReLU Activation

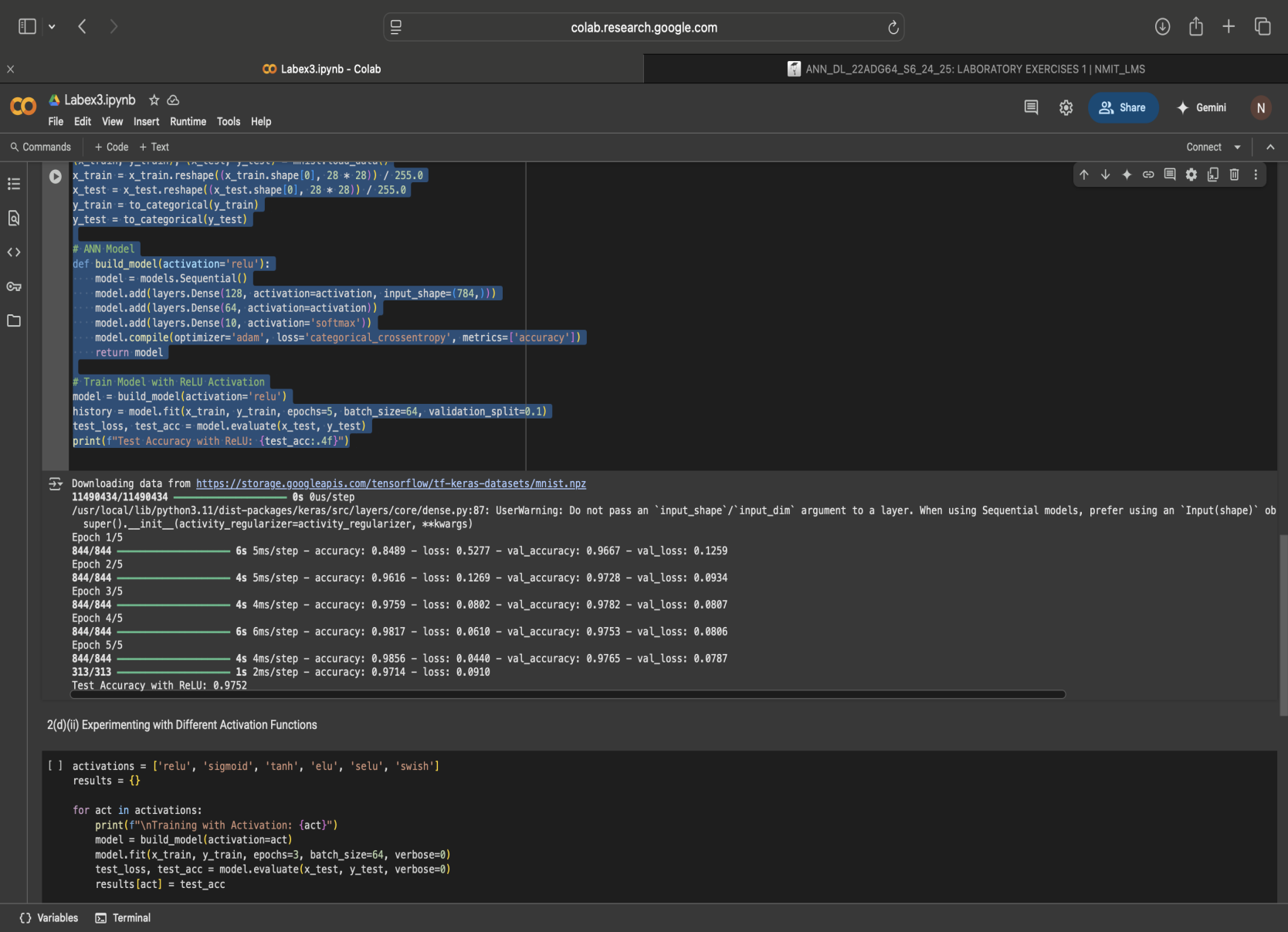
model = build\_model(activation='relu')

history = model.fit(x\_train, y\_train, epochs=5, batch\_size=64, validation\_split=0.1)

test\_loss, test\_acc = model.evaluate(x\_test, y\_test)

print(f"Test Accuracy with ReLU: {test\_acc:.4f}")

OUTPUT:



Code:

activations = ['relu', 'sigmoid', 'tanh', 'elu', 'selu', 'swish']

results = {}

for act in activations:

print(f"\nTraining with Activation: {act}")

model = build\_model(activation=act)

model.fit(x\_train, y\_train, epochs=3, batch\_size=64, verbose=0)

test\_loss, test\_acc = model.evaluate(x\_test, y\_test, verbose=0)

results[act] = test\_acc

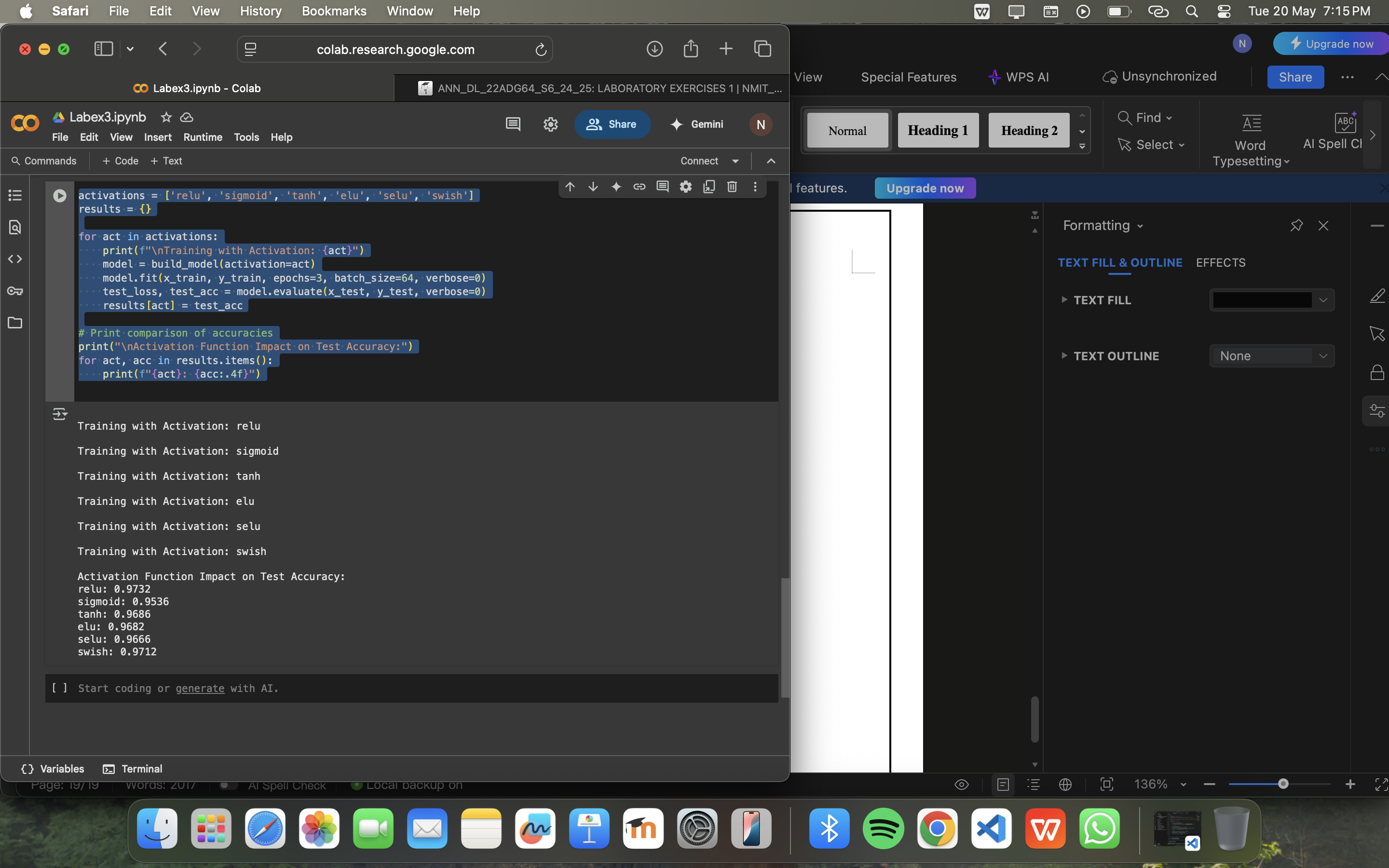
# Print comparison of accuracies

print("\nActivation Function Impact on Test Accuracy:")

for act, acc in results.items():

print(f"{act}: {acc:.4f}")

OUTPUT:



## Program 4. a)

## Problem Definition:

a.Build a simple CNN architecture for image classification. Train the CNN on a dataset like MNIST or CIFAR-10. Visualize the learned filters and feature maps.

Code:

!pip install tensorflow

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras import layers,Model,Input

# Load and normalize CIFAR-10

(x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.cifar10.load\_data()

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

inputs = Input(shape=(32, 32, 3), name='input\_layer')

# Convolutional layers

x = layers.Conv2D(32, (3, 3), activation='relu', name='conv1')(inputs)

x = layers.MaxPooling2D((2, 2), name='pool1')(x)

x = layers.Conv2D(64, (3, 3), activation='relu', name='conv2')(x)

x = layers.MaxPooling2D((2, 2), name='pool2')(x)

x = layers.Conv2D(64, (3, 3), activation='relu', name='conv3')(x)

# Fully connected layers

x = layers.Flatten()(x)

x = layers.Dense(64, activation='relu')(x)

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

# Build model

model = Model(inputs=inputs, outputs=outputs, name='cnn\_model')

model.summary()

model.compile(optimizer="adam",loss="sparse\_categorical\_crossentropy",metrics=["accuracy"])

history = model.fit(x\_train, y\_train, epochs=10, validation\_data=(x\_test, y\_test))

filters,\_= model.get\_layer("conv1").get\_weights() plt.showrange(6):

plt.showubplot(1, 6, i+1)

plt.imshow(filters[:, :, 0, i], cmap='gray')

plt.axis('off')

plt.suptitle('Filters of Conv1')

plt.show()

layers= ['conv1','conv2','conv3']

activation\_model = Model(inputs=model.input,outputs=[model.get\_layer(name).output for name in layers])

img= np.expand\_dims(x\_test[0],axis=0)

activations= activation\_model.predict(img)

for i,act in enumerate(activations):

plt.figure(figsize=(12,3))

for j in range(6):

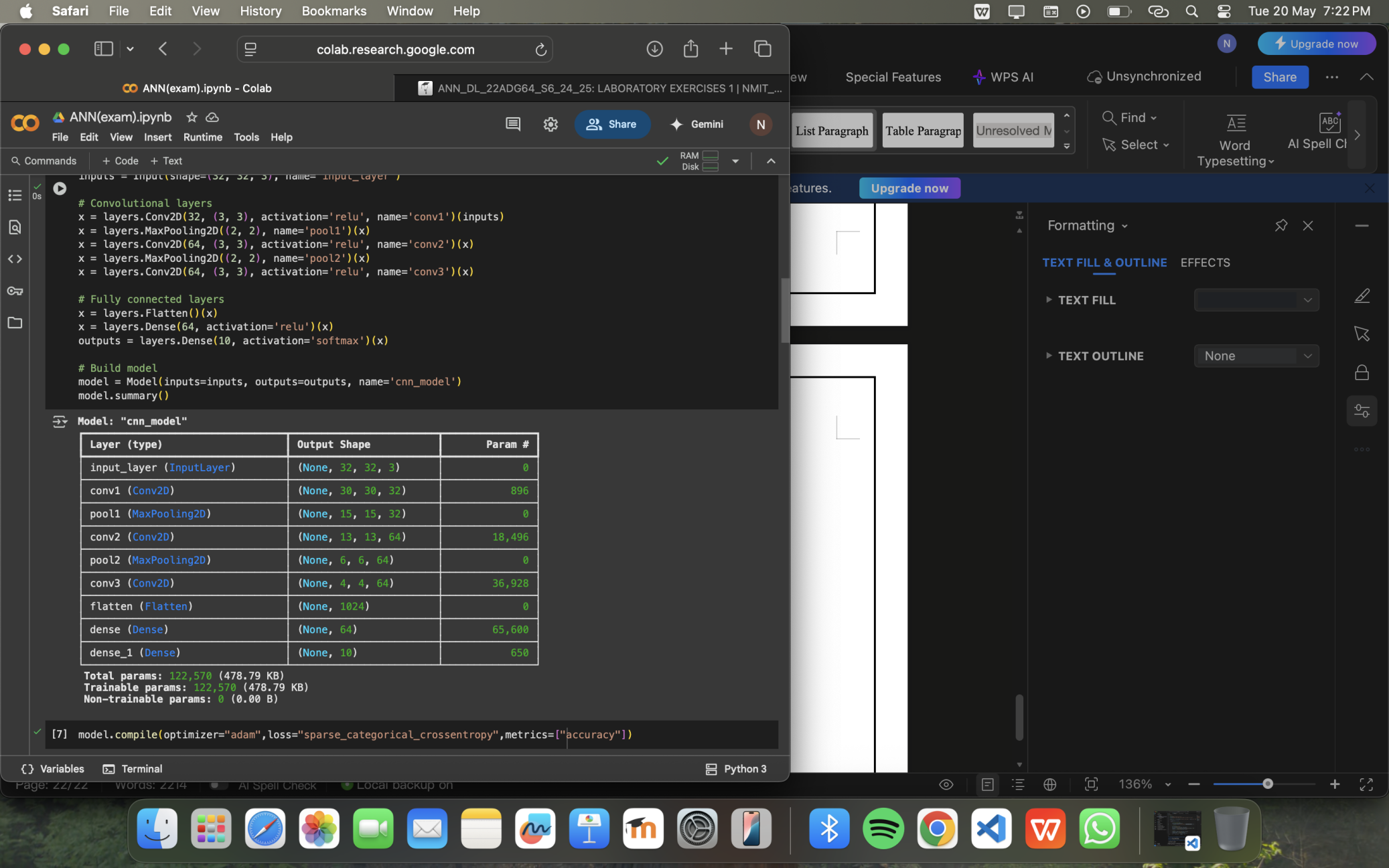
plt.subplot(1, 6, j+1)

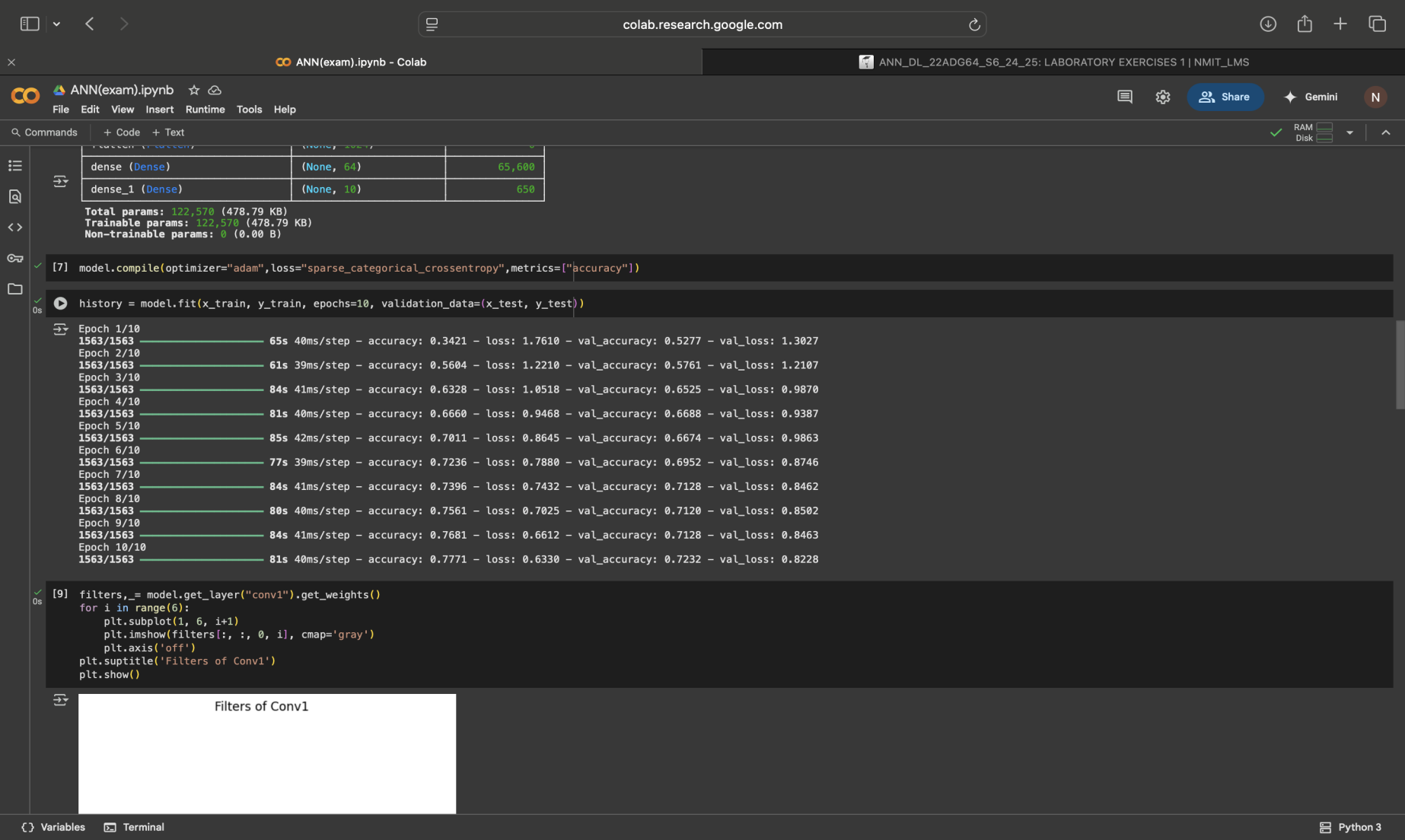
plt.imshow(act[0, :, :, j], cmap='viridis')

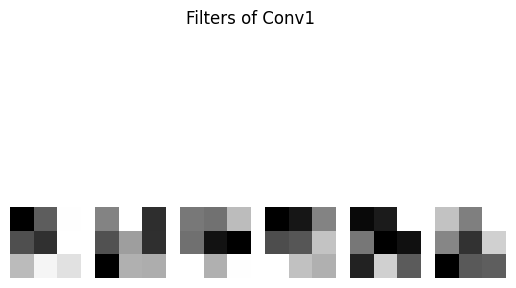
plt.axis('off')

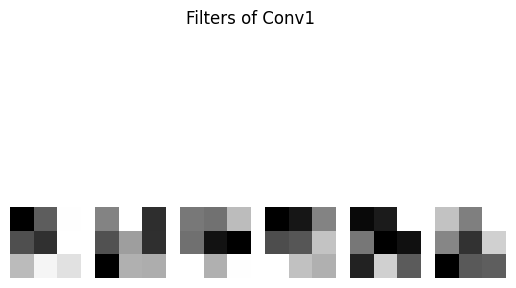
plt.suptitle(f'Feature maps from layer {i+1}')

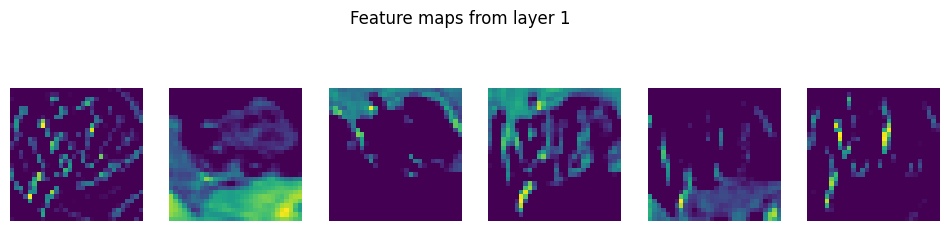
OUTPUT :

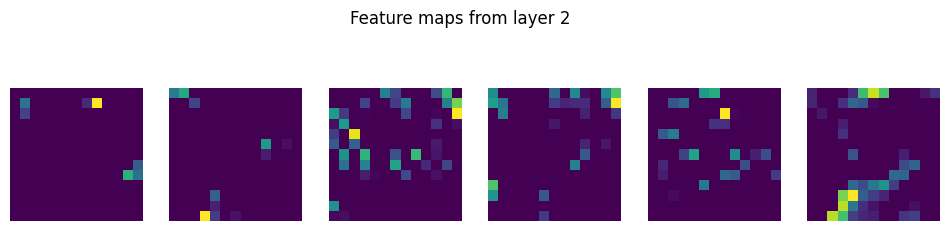


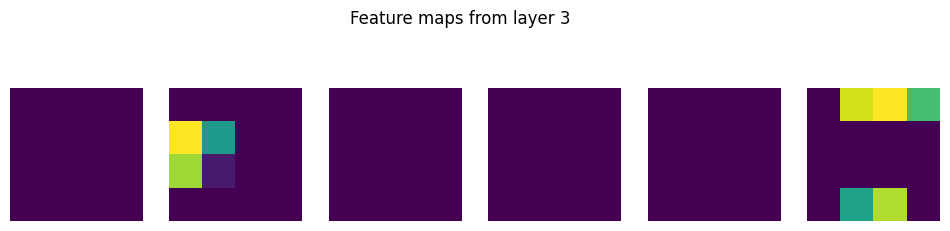












## Program 4. b)

## Problem Definition:

b .

1. Given Data: Consider the RGB Action Recognition (Video) dataset given. Dataset Link: <https://drive.google.com/drive/folders/1XZJxsDsPVje0GdCIyAVwltpfWTFu0GjO?usp=drive_link>
2. Dataset Creation: Create an RGB Image dataset of the four classes given: (Crop Images from Video)

(i) Punching – 800 Images

(ii) Kicking – 800 Images

(iii) Taking Selfie – 800 Images

(iv) Pushing – 800 Images

3. Data Preprocessing:

(i) Normalization

(ii) Data Augmentation (Optional)

4.Algorithm/Model: Create a Convolutional Neural Network

5. Model Evaluation:

Train a CNN model on the dataset created.

Evaluate the model’s performance using

## 6.Accuracy and Confusion Matrix.

## Code:

## import os

## import cv2

## import numpy as np

## import tensorflow as tf

## from sklearn.model\_selection import train\_test\_split

## from sklearn.metrics import accuracy\_score, confusion\_matrix

## import matplotlib.pyplot as plt

## import seaborn as sns

## # Step 1: Extract frames from videos

## video\_root = 'Videos'

## output\_root = 'dataset'

## classes = ['Punch', 'Kicking', 'Taking\_Selfie', 'Pushing']

## frames\_per\_class = 400

## frame\_interval = 3

## os.makedirs(output\_root, exist\_ok=True)

## for cls in classes:

## video\_dir = os.path.join(video\_root, cls)

## output\_dir = os.path.join(output\_root, cls)

## os.makedirs(output\_dir, exist\_ok=True)

## total\_saved = 0

## for video\_file in os.listdir(video\_dir):

## if not video\_file.endswith('.avi') or total\_saved >= frames\_per\_class:

## continue

## cap = cv2.VideoCapture(os.path.join(video\_dir, video\_file))

## frame\_count = 0

## while cap.isOpened() and total\_saved < frames\_per\_class:

## ret, frame = cap.read()

## if not ret:

## break

## if frame\_count % frame\_interval == 0:

## resized = cv2.resize(frame, (64, 64))

## filename = os.path.join(output\_dir, f'{cls}\_{total\_saved}.jpg')

## cv2.imwrite(filename, resized)

## total\_saved += 1

## frame\_count += 1

## cap.release()

## print("Frame extraction complete.")

## # Step 2: Load images and labels

## X, y = [], []

## for idx, label in enumerate(classes):

## folder = os.path.join(output\_root, label)

## for img\_name in os.listdir(folder):

## img\_path = os.path.join(folder, img\_name)

## img = cv2.imread(img\_path)

## if img is not None:

## img = img / 255.0

## X.append(img)

## y.append(idx)

## X = np.array(X)

## y = np.array(y)

## # Step 3: Train/Test Split

## X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, stratify=y)

## # Step 4: CNN Model

## model = tf.keras.Sequential([

## tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(64, 64, 3)),

## tf.keras.layers.MaxPooling2D(2, 2),

## tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),

## tf.keras.layers.MaxPooling2D(2, 2),

## tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),

## tf.keras.layers.Flatten(),

## tf.keras.layers.Dense(128, activation='relu'),

## tf.keras.layers.Dense(4, activation='softmax')

## ])

## model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

## # Step 5: Training

## model.fit(X\_train, y\_train, epochs=10, validation\_split=0.1)

## # Step 6: Evaluation

## y\_pred = np.argmax(model.predict(X\_test), axis=1)

## acc = accuracy\_score(y\_test, y\_pred)

## print("Accuracy:", acc)

## cm = confusion\_matrix(y\_test, y\_pred)

## sns.heatmap(cm, annot=True, fmt='d', xticklabels=classes, yticklabels=classes)

## plt.xlabel("Predicted")

## plt.ylabel("Actual")

## plt.show()

## import os

## import matplotlib.pyplot as plt

## import cv2

## import random

## dataset\_path = 'dataset'

## classes = ['Punch', 'Kicking', 'Taking\_Selfie', 'Pushing']

## for cls in classes:

## class\_folder = os.path.join(dataset\_path, cls)

## image\_files = os.listdir(class\_folder)

## selected\_images = random.sample(image\_files, min(30, len(image\_files)))

## 

## plt.figure(figsize=(15, 8))

## plt.suptitle(cls.replace("\_", " ").title(), fontsize=16)

## 

## for i, img\_name in enumerate(selected\_images):

## img\_path = os.path.join(class\_folder, img\_name)

## img = cv2.imread(img\_path)

## img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

## 

## plt.subplot(5, 6, i + 1)

## plt.imshow(img)

## plt.axis('off')

## 

## plt.tight\_layout(rect=[0, 0, 1, 0.95])

## plt.show()

## OUTPUT:

## Screenshot 2025-05-20 at 8.14.00 PM

## Screenshot 2025-05-20 at 8.14.08 PM

## Program 5. a)

## Problem Definition:

a.Write a Python program to implement a basic RNN for sequence prediction.

## Code:

## import numpy as np

## import matplotlib.pyplot as plt

## from tensorflow.keras.models import Sequential

## from tensorflow.keras.layers import SimpleRNN, Dense

## from sklearn.model\_selection import train\_test\_split

## # Prepare dataset

## def create\_dataset(seq\_length=4):

## X, y = [], []

## for i in range(100):

## seq = np.arange(i, i + seq\_length)

## X.append(seq)

## y.append(i + seq\_length)

## return np.array(X), np.array(y)

## # Create and reshape data

## X, y = create\_dataset()

## X = X.reshape((X.shape[0], X.shape[1], 1)) # (samples, time\_steps, features)

## # Split for validation

## X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

## # Build RNN model

## model = Sequential([

## SimpleRNN(50, activation='relu', input\_shape=(X.shape[1], 1)),

## Dense(1)

## ])

## # Compile and train

## model.compile(optimizer='adam', loss='mse')

## history = model.fit(X\_train, y\_train, validation\_data=(X\_val, y\_val), epochs=200, verbose=0)

## # Test prediction

## test\_input = np.array([97, 98, 99, 100]).reshape((1, 4, 1))

## predicted = model.predict(test\_input, verbose=0)

## print("RNN Prediction for [97, 98, 99, 100]:", predicted.flatten()[0])

## print("Rounded:", round(predicted.flatten()[0]))

## OUTPUT:

## Screenshot 2025-05-20 at 7.36.07 PM

## Program 5. b)

## Problem Definition:

## b.Write a Python program to implement an LSTM and compare its performance on a sequential task

## Code:

## import numpy as np

## from tensorflow.keras.models import Sequential

## from tensorflow.keras.layers import LSTM, Dense

## # Prepare sequence data

## def create\_dataset(seq\_length=4):

## X, y = [], []

## for i in range(100):

## seq = np.arange(i, i + seq\_length)

## X.append(seq)

## y.append(i + seq\_length)

## return np.array(X), np.array(y)

## X, y = create\_dataset()

## X = X.reshape((X.shape[0], X.shape[1], 1)) # reshape for LSTM input

## # Define LSTM model

## model = Sequential([

## LSTM(50, activation='relu', input\_shape=(X.shape[1], 1)),

## Dense(1)

## ])

## model.compile(optimizer='adam', loss='mse')

## model.fit(X, y, epochs=200, verbose=0)

## # Prediction

## test\_input = np.array([97, 98, 99, 100]).reshape((1, 4, 1))

## predicted = model.predict(test\_input, verbose=0)

## print("LSTM Prediction for [97, 98, 99, 100]:", predicted.flatten()[0])

## print("Rounded:", round(predicted.flatten()[0]))

## OUTPUT:

## Screenshot 2025-05-20 at 7.38.26 PM

## Program 6. a)

## Problem Definition:

## a.Write a Python program to implement a Autoencoder for image reconstruction.

## Code:

## import numpy as np plt.showtplotlib.pyplot as plt plt.showorflow.keras.datasets import mnist

## from tensorflow.keras.models import Model

## from tensorflow.keras.layers import Input, Dense, Flatten, Reshape

## # Load data

## (x\_train, \_), (x\_test, \_) = mnist.load\_data()

## x\_train = x\_train.astype('float32') / 255.0

## x\_test = x\_test.astype('float32') / 255.0

## # Flatten images

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

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

## # Encoder

## input\_img = Input(shape=(784,))

## encoded = Dense(64, activation='relu')(input\_img)

## # Decoder

## decoded = Dense(784, activation='sigmoid')(encoded)

## # Autoencoder model

## autoencoder = Model(input\_img, decoded)

## autoencoder.compile(optimizer='adam', loss='binary\_crossentropy')

## autoencoder.fit(x\_train, x\_train, epochs=10, batch\_size=256, shuffle=True, validation\_data=(x\_test, x\_test))

## # Reconstruct

## reconstructed = autoencoder.predict(x\_test)

## # Show original and reconstructed

## n = 5

## plt.figure(figsize=(10, 4))

## for i in range(n):

## # Original

## plt.subplot(2, n, i + 1)

## plt.imshow(x\_test[i].reshape(28, 28), cmap='gray')

## plt.axis('off')

## # Reconstructed

## plt.subplot(2, n, i + 1 + n)

## plt.imshow(reconstructed[i].reshape(28, 28), cmap='gray')

## plt.axis('off')

## plt.show()

## OUTPUT:

## Screenshot 2025-05-20 at 7.42.34 PM

## Program 6. b)

## Problem Definition:

## b)Write a Python program to implement a Variational Autoencoder for image reconstruction.

## Code:

## import tensorflow as tf

## from tensorflow import keras

## from tensorflow.keras import layers

## import numpy as np

## import matplotlib.pyplot as plt

## # --- 1. Load and Preprocess Data ---

## # We'll use the Fashion MNIST dataset for this example

## (x\_train, \_), (x\_test, \_) = keras.datasets.fashion\_mnist.load\_data()

## # Normalize pixel values to [0, 1] and reshape for convolutional layers

## image\_size = x\_train.shape[1]

## x\_train = np.reshape(x\_train, [-1, image\_size, image\_size, 1]).astype("float32") / 255.0

## x\_test = np.reshape(x\_test, [-1, image\_size, image\_size, 1]).astype("float32") / 255.0

## # --- 2. Define the VAE Model ---

## class Sampling(layers.Layer):

## """

## Uses (z\_mean, z\_log\_var) to sample z, the latent vector.

## Reparameterization trick allows backpropagation through sampling.

## """

## def call(self, inputs):

## z\_mean, z\_log\_var = inputs

## batch = tf.shape(z\_mean)[0]

## dim = tf.shape(z\_mean)[1]

## epsilon = tf.keras.backend.random\_normal(shape=(batch, dim))

## return z\_mean + tf.exp(0.5 \* z\_log\_var) \* epsilon

## class VAE(keras.Model):

## def \_\_init\_\_(self, encoder, decoder, \*\*kwargs):

## super(VAE, self).\_\_init\_\_(\*\*kwargs)

## self.encoder = encoder

## self.decoder = decoder

## self.total\_loss\_tracker = keras.metrics.Mean(name="total\_loss")

## self.reconstruction\_loss\_tracker = keras.metrics.Mean(name="reconstruction\_loss")

## self.kl\_loss\_tracker = keras.metrics.Mean(name="kl\_loss")

## @property

## def metrics(self):

## return [

## self.total\_loss\_tracker,

## self.reconstruction\_loss\_tracker,

## self.kl\_loss\_tracker,

## ]

## def train\_step(self, data):

## with tf.GradientTape() as tape:

## z\_mean, z\_log\_var, z = self.encoder(data)

## reconstruction = self.decoder(z)

## reconstruction\_loss = tf.reduce\_mean(

## tf.reduce\_sum(

## keras.losses.binary\_crossentropy(data, reconstruction), axis=(1, 2)

## )

## )

## kl\_loss = -0.5 \* (1 + z\_log\_var - tf.square(z\_mean) - tf.exp(z\_log\_var))

## kl\_loss = tf.reduce\_mean(tf.reduce\_sum(kl\_loss, axis=1))

## total\_loss = reconstruction\_loss + kl\_loss

## grads = tape.gradient(total\_loss, self.trainable\_weights)

## self.optimizer.apply\_gradients(zip(grads, self.trainable\_weights))

## self.total\_loss\_tracker.update\_state(total\_loss)

## self.reconstruction\_loss\_tracker.update\_state(reconstruction\_loss)

## self.kl\_loss\_tracker.update\_state(kl\_loss)

## return {

## "loss": self.total\_loss\_tracker.result(),

## "reconstruction\_loss": self.reconstruction\_loss\_tracker.result(),

## "kl\_loss": self.kl\_loss\_tracker.result(),

## }

## # --- 3. Build the Encoder ---

## latent\_dim = 2 # We'll use a 2D latent space for easy visualization

## encoder\_inputs = keras.Input(shape=(image\_size, image\_size, 1))

## x = layers.Conv2D(32, 3, activation="relu", strides=2, padding="same")(encoder\_inputs)

## x = layers.Conv2D(64, 3, activation="relu", strides=2, padding="same")(x)

## x = layers.Flatten()(x)

## x = layers.Dense(128, activation="relu")(x)

## z\_mean = layers.Dense(latent\_dim, name="z\_mean")(x)

## z\_log\_var = layers.Dense(latent\_dim, name="z\_log\_var")(x)

## z = Sampling()([z\_mean, z\_log\_var])

## encoder = keras.Model(encoder\_inputs, [z\_mean, z\_log\_var, z], name="encoder")

## encoder.summary()

## # --- 4. Build the Decoder ---

## latent\_inputs = keras.Input(shape=(latent\_dim,))

## x = layers.Dense(7 \* 7 \* 64, activation="relu")(latent\_inputs) # Adjust based on image\_size and strides in encoder

## x = layers.Reshape((7, 7, 64))(x)

## x = layers.Conv2DTranspose(64, 3, activation="relu", strides=2, padding="same")(x)

## x = layers.Conv2DTranspose(32, 3, activation="relu", strides=2, padding="same")(x)

## decoder\_outputs = layers.Conv2DTranspose(1, 3, activation="sigmoid", padding="same")(x) # Sigmoid for [0,1] output

## decoder = keras.Model(latent\_inputs, decoder\_outputs, name="decoder")

## decoder.summary()

## # --- 5. Compile and Train the VAE ---

## vae = VAE(encoder, decoder)

## vae.compile(optimizer=keras.optimizers.Adam())

## vae.fit(x\_train, epochs=5, batch\_size=128)

## # --- 6. Image Reconstruction and Generation ---

## # a) Reconstruct images from the test set

## def plot\_reconstructed\_images(data, model, num\_images=10):

## \_, \_, z\_test = model.encoder.predict(data)

## reconstructions = model.decoder.predict(z\_test)

## plt.figure(figsize=(20, 4))

## for i in range(num\_images):

## # Original Image

## ax = plt.subplot(2, num\_images, i + 1)

## plt.imshow(data[i].reshape(image\_size, image\_size), cmap="gray")

## plt.title("Original")

## plt.axis("off")

## # Reconstructed Image

## ax = plt.subplot(2, num\_images, i + 1 + num\_images)

## plt.imshow(reconstructions[i].reshape(image\_size, image\_size), cmap="gray")

## plt.title("Reconstructed")

## plt.axis("off")

## plt.show()

## print("\n--- Reconstructing Test Images ---")

## plot\_reconstructed\_images(x\_test, vae)

## # b) Generate new images by sampling from the latent space

## def plot\_generated\_images(model, num\_images=10, latent\_dim=2):

## # Sample random points from the latent space (standard normal distribution)

## random\_latent\_vectors = tf.random.normal(shape=(num\_images, latent\_dim))

## generated\_images = model.decoder.predict(random\_latent\_vectors)

## plt.figure(figsize=(10, 2))

## for i in range(num\_images):

## ax = plt.subplot(1, num\_images, i + 1)

## plt.imshow(generated\_images[i].reshape(image\_size, image\_size), cmap="gray")

## plt.axis("off")

## plt.suptitle("Generated Images from Sampled Latent Vectors")

## plt.show()

## print("\n--- Generating New Images ---")

## plot\_generated\_images(vae, num\_images=10, latent\_dim=latent\_dim)

## # c) Visualize the latent space (for 2D latent space)

## def plot\_latent\_space(encoder, data, labels, latent\_dim=2):

## if latent\_dim != 2:

## print("Latent space visualization is only for 2D latent space.")

## return

## z\_mean, \_, \_ = encoder.predict(data)

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

## plt.scatter(z\_mean[:, 0], z\_mean[:, 1], c=labels, cmap="viridis")

## plt.colorbar()

## plt.xlabel("z[0]")

## plt.ylabel("z[1]")

## plt.title("2D Latent Space Visualization")

## plt.show()

## # To visualize latent space, we need the original labels (y\_test)

## # which we discarded earlier for VAE training. Let's reload them.

## (\_, \_), (\_, y\_test) = keras.datasets.fashion\_mnist.load\_data()

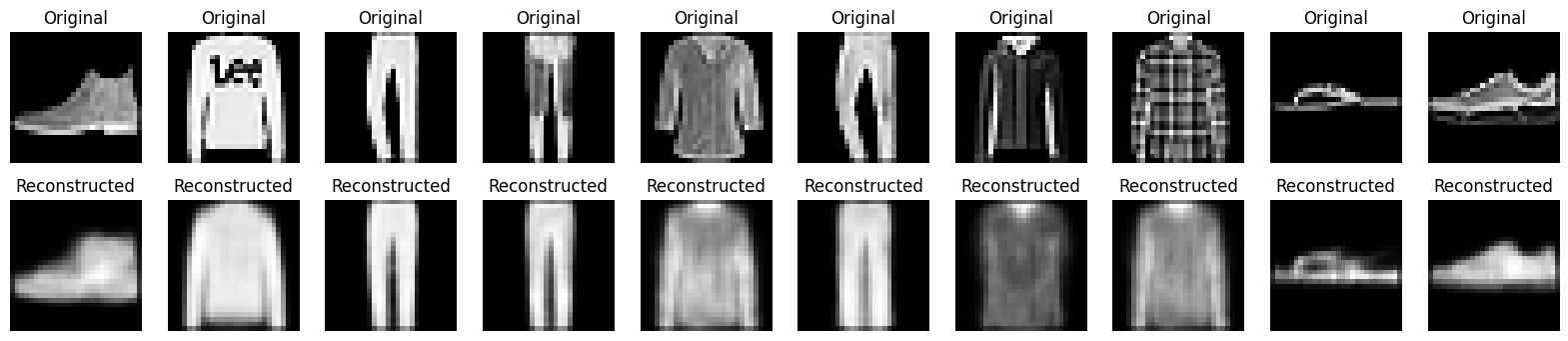
## print("\n--- Visualizing 2D Latent Space ---")

## plot\_latent\_space(vae.encoder, x\_test, y\_test, latent\_dim=latent\_dim)

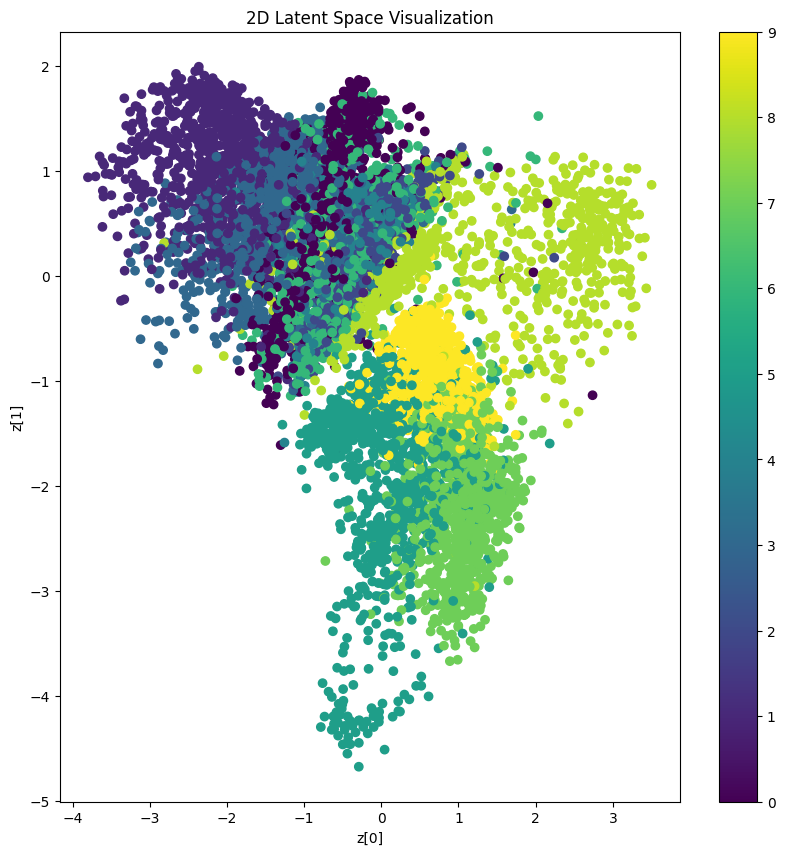
## OUTPUT:

## Screenshot 2025-05-20 at 7.46.04 PM

## Screenshot 2025-05-20 at 7.46.55 PM







## Program 6. c)

## Problem Definition:

## c.Write a Python program to implement a GAN to generate synthetic images.

## Code:

## import tensorflow as tf

## from tensorflow.keras import layers

## import numpy as np

## import matplotlib.pyplot as plt

## import os

## # 1. Load and prepare the MNIST dataset

## (x\_train, \_), \_ = tf.keras.datasets.mnist.load\_data()

## x\_train = (x\_train.astype("float32") - 127.5) / 127.5 # Normalize to [-1, 1]

## x\_train = np.expand\_dims(x\_train, axis=-1) # Add channel dimension

## BATCH\_SIZE = 256

## train\_dataset = tf.data.Dataset.from\_tensor\_slices(x\_train).shuffle(60000).batch(BATCH\_SIZE)

## # 2. Build the Generator model

## def build\_generator(latent\_dim):

## model = tf.keras.Sequential([

## layers.Dense(7\*7\*256, use\_bias=False, input\_shape=(latent\_dim,)),

## layers.BatchNormalization(),

## layers.LeakyReLU(),

## layers.Reshape((7, 7, 256)),

## layers.Conv2DTranspose(128, 5, strides=1, padding='same', use\_bias=False),

## layers.BatchNormalization(),

## layers.LeakyReLU(),

## layers.Conv2DTranspose(64, 5, strides=2, padding='same', use\_bias=False),

## layers.BatchNormalization(),

## layers.LeakyReLU(),

## layers.Conv2DTranspose(1, 5, strides=2, padding='same', use\_bias=False, activation='tanh')

## ])

## return model

## # 3. Build the Discriminator model

## def build\_discriminator():

## model = tf.keras.Sequential([

## layers.Conv2D(64, 5, strides=2, padding='same', input\_shape=(28, 28, 1)),

## layers.LeakyReLU(),

## layers.Dropout(0.3),

## layers.Conv2D(128, 5, strides=2, padding='same'),

## layers.LeakyReLU(),

## layers.Dropout(0.3),

## layers.Flatten(),

## layers.Dense(1)

## ])

## return model

## # 4. Define loss functions

## cross\_entropy = tf.keras.losses.BinaryCrossentropy(from\_logits=True)

## def discriminator\_loss(real, fake):

## real\_loss = cross\_entropy(tf.ones\_like(real), real)

## fake\_loss = cross\_entropy(tf.zeros\_like(fake), fake)

## return real\_loss + fake\_loss

## def generator\_loss(fake):

## return cross\_entropy(tf.ones\_like(fake), fake)

## # 5. Create optimizers

## generator\_optimizer = tf.keras.optimizers.Adam(1e-4)

## discriminator\_optimizer = tf.keras.optimizers.Adam(1e-4)

## # 6. Training step

## @tf.function

## def train\_step(generator, discriminator, images, latent\_dim):

## noise = tf.random.normal([tf.shape(images)[0], latent\_dim])

## with tf.GradientTape() as gen\_tape, tf.GradientTape() as disc\_tape:

## generated\_images = generator(noise, training=True)

## real\_output = discriminator(images, training=True)

## fake\_output = discriminator(generated\_images, training=True)

## gen\_loss = generator\_loss(fake\_output)

## disc\_loss = discriminator\_loss(real\_output, fake\_output)

## gradients\_gen = gen\_tape.gradient(gen\_loss, generator.trainable\_variables)

## gradients\_disc = disc\_tape.gradient(disc\_loss, discriminator.trainable\_variables)

## generator\_optimizer.apply\_gradients(zip(gradients\_gen, generator.trainable\_variables))

## discriminator\_optimizer.apply\_gradients(zip(gradients\_disc, discriminator.trainable\_variables))

## # 7. Generate and save images

## def generate\_and\_save\_images(model, epoch, latent\_dim, num\_images=16):

## noise = tf.random.normal([num\_images, latent\_dim])

## generated\_images = model(noise, training=False)

## generated\_images = (generated\_images + 1) / 2 # Rescale to [0,1]

## fig, axs = plt.subplots(4, 4, figsize=(4, 4))

## for i in range(num\_images):

## axs[i//4, i%4].imshow(generated\_images[i, :, :, 0], cmap='gray')

## axs[i//4, i%4].axis('off')

## plt.suptitle(f"Epoch {epoch}")

## plt.tight\_layout()

## if not os.path.exists('generated\_images'):

## os.makedirs('generated\_images')

## plt.savefig(f'generated\_images/image\_at\_epoch\_{epoch}.png')

## plt.show()

## # 8. Training loop

## def train(generator, discriminator, dataset, latent\_dim, epochs):

## for epoch in range(1, epochs + 1):

## for image\_batch in dataset:

## train\_step(generator, discriminator, image\_batch, latent\_dim)

## if epoch % 5 == 0 or epoch == epochs:

## generate\_and\_save\_images(generator, epoch, latent\_dim)

## print(f"Epoch {epoch} completed")

## if \_\_name\_\_ == "\_\_main\_\_":

## latent\_dim = 100

## epochs = 20

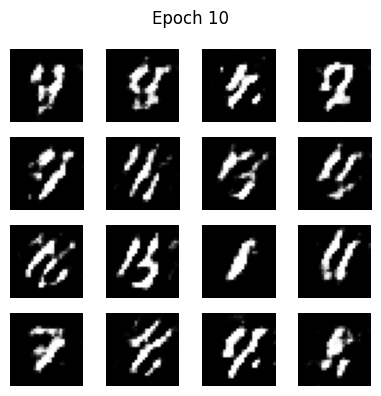
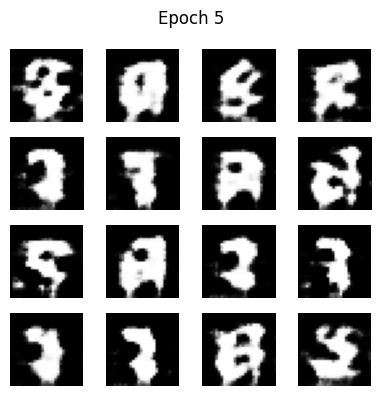
## generator = build\_generator(latent\_dim)

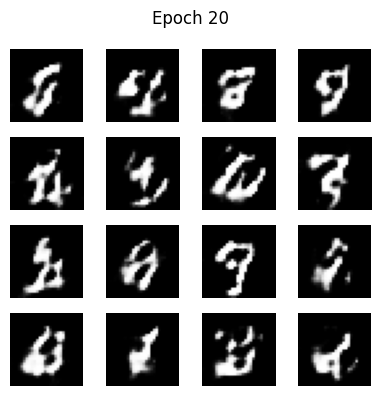
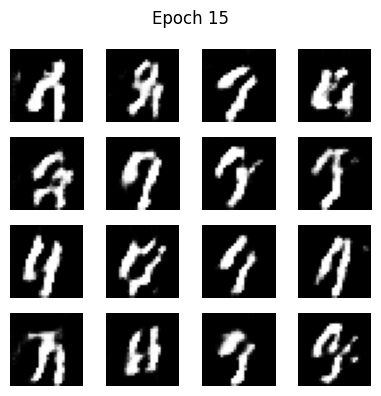
## discriminator = build\_discriminator()

## train(generator, discriminator, train\_dataset, latent\_dim, epochs)

## print("Training finished!")

## OUTPUT:





**Internal Evaluation for Record**

Date:

Name of the program obtained:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sl.no** | **Particulars** | **Max Marks in Observation** | **Obtained Marks in Observation** | **Internal Examiner Signature** |
| 1. | Program 1 a) | 10 |  |  |
| Program 1 b) | 10 |  |  |
| Program 1 c) | 10 |  |  |
| 2. | Program 2 a) | 10 |  |  |
| Program 2 b) | 10 |  |  |
| 3. | Program 3 a) | 10 |  |  |
| Program 3 b) | 10 |  |  |
| 4. | Program 4 a) | 15 |  |  |
| Program 4 b) | 15 |  |  |
| 5. | Program 5 a) | 10 |  |  |
| Program 5 b) | 10 |  |  |
| 6. | Program 6 a) | 10 |  |  |
| Program 6 b) | 10 |  |  |
| Program 6 c) | 10 |  |  |
|  | Total | 150 |  |  |
| **Total (Out of 15)** | 15 |  |  |

Total marks secured: /15

Signature of Faculty with date

**Internal Evaluation**

Date:

Name of the program obtained:

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.no** | **Particulars** | **Max Marks** | **Obtained Marks** |
| 1. | Observation Book | 15 |  |
| 2. | Record Book | 15 |  |
| 3. | Program Writeup | 10 |  |
| 4. | Program Execution | 05 |  |
| 5. | Viva | 05 |  |
|  | **Total** | 50 |  |

Total marks secured: **/50**

Signature of Faculty with date