**Lab Experiment - 1**

1. **Write a program to demonstrate the working of different activation functions like sigmoid, tanh, ReLU & softmax to train a neural network.**

**Program:**

import numpy as np

from sklearn.datasets import make\_moons

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

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

import matplotlib.pyplot as plt

# Dataset

X, y = make\_moons(n\_samples=500)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Compare activation functions

activations = ['sigmoid', 'tanh', 'relu']

histories = {}

for act in activations:

model = Sequential([

Dense(16, activation=act, input\_dim=2),

Dense(8, activation=act),

Dense(1, activation='sigmoid')

])

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

histories[act] = model.fit(X\_train,

y\_train,

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

epochs=20,

verbose=0).history

# Plot validation accuracy

for act in activations:

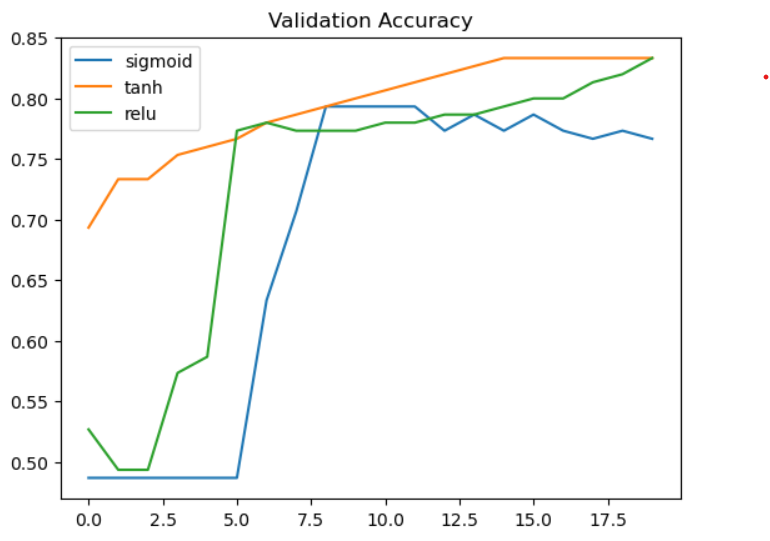
plt.plot(histories[act]['val\_accuracy'], label=act)

plt.legend()

plt.title('Validation Accuracy')

plt.show()

**Output:**



**Lab Experiment - 2**

* 1. **Design a single unit perceptron for classification og linearly separable binary dataset without using pre-defined models. Use the perceptron from sklearn.**
  2. **Identify the problems in single unit perceptron using AND, OR, XOR data and analyze the results.**

**Program:**

**A.**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make\_classification

from sklearn.linear\_model import Perceptron

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

from sklearn.metrics import accuracy\_score

# Generate a linearly separable binary dataset

X, y = make\_classification(n\_samples=100, n\_features=2, n\_classes=2,

n\_informative=2, n\_redundant=0, random\_state=42)

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

# Train the Perceptron

model = Perceptron(max\_iter=1000, tol=1e-3)

model.fit(X\_train, y\_train)

# Evaluate and print accuracy

accuracy = model.score(X\_test, y\_test) \* 100

print(f"Accuracy: {accuracy:.2f}%")

# Plot decision boundary

x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, 0.01), np.arange(y\_min, y\_max, 0.01))

Z = model.predict(np.c\_[xx.ravel(), yy.ravel()]).reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.8, cmap=plt.cm.coolwarm)

plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', cmap=plt.cm.coolwarm)

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.title('Perceptron Decision Boundary')

plt.show()

**B.**

from sklearn.linear\_model import Perceptron

import numpy as np

import matplotlib.pyplot as plt

# Define the datasets

datasets = {

"AND": (np.array([[0, 0], [0, 1], [1, 0], [1, 1]]), np.array([0, 0, 0, 1])),

"OR": (np.array([[0, 0], [0, 1], [1, 0], [1, 1]]), np.array([0, 1, 1, 1])),

"XOR": (np.array([[0, 0], [0, 1], [1, 0], [1, 1]]), np.array([0, 1, 1, 0]))

}

# Function to train and evaluate the perceptron

def train\_and\_evaluate(X, y, title):

perceptron = Perceptron(max\_iter=1000, eta0=1, random\_state=0).fit(X, y)

predictions = perceptron.predict(X)

# Plot data and predictions

plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm', edgecolor='k', s=100)

for i, pred in enumerate(predictions):

plt.text(X[i, 0], X[i, 1], str(pred), color='white', ha='center', va='center')

plt.title(f'{title} Dataset')

plt.show()

print(f'{title} Accuracy: {np.mean(predictions == y) \* 100:.2f}%\n')

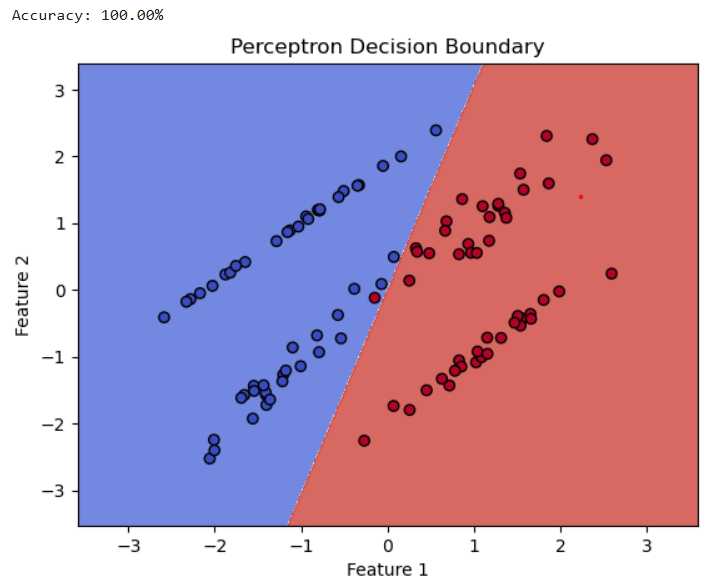
# Evaluate datasets

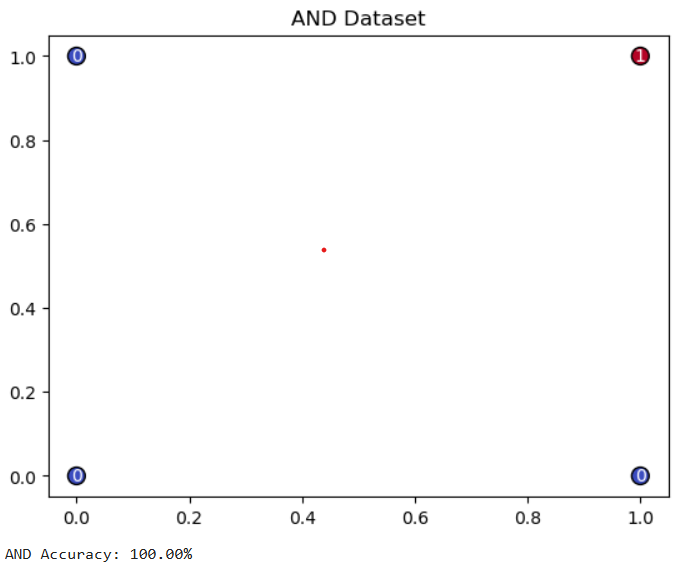
for name, (X, y) in datasets.items():

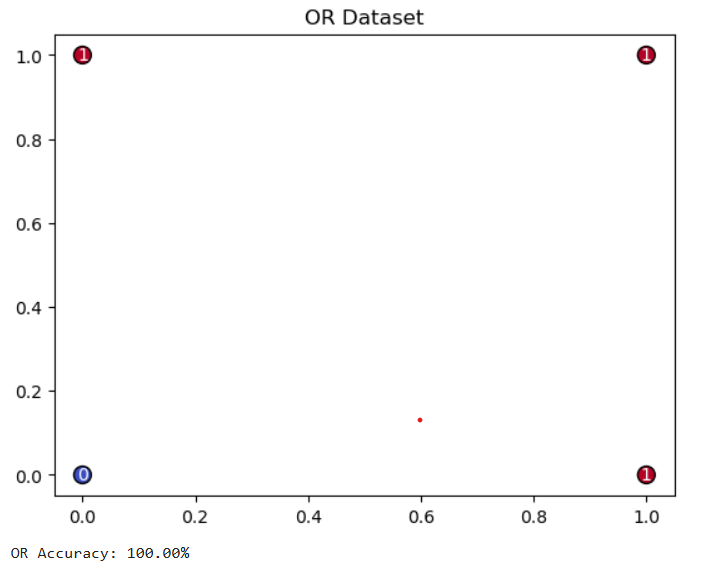
train\_and\_evaluate(X, y, name)

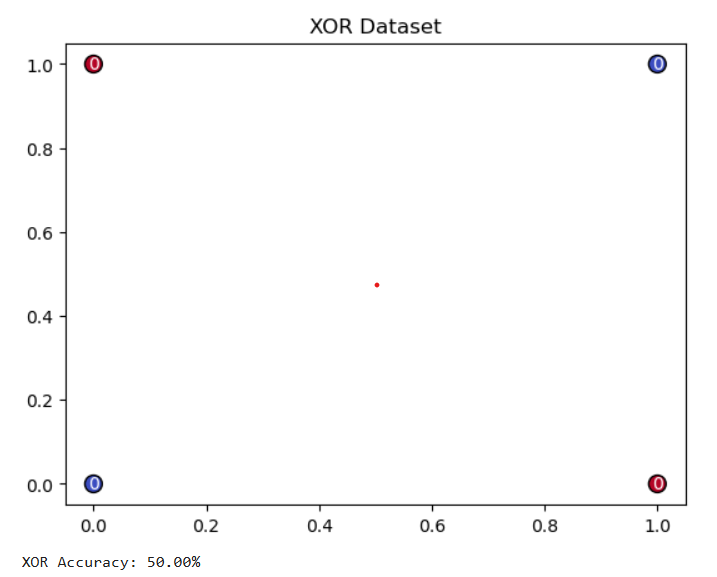
**Output:**

**A.**



**B.**

****

****

**Lab Experiment - 3**

1. **Build a deep feed-forward Artificial Neural Network by implementing the back propagation algorithm and test the same using appropriate datasets. Use the number of hidden layers greater than or equal to 4.**

**Program:**

import tensorflow as tf

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.datasets import mnist

# Load and preprocess MNIST dataset

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

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0 # Normalize inputs to [0, 1]

# Define the model

model = Sequential([

Input(shape=(28, 28)), # Define the input shape explicitly using Input layer

Flatten(), # Flatten 28x28 images into a vector

Dense(128, activation='relu'), # First hidden layer

Dense(128, activation='relu'), # Second hidden layer

Dense(64, activation='relu'), # Third hidden layer

Dense(64, activation='relu'), # Fourth hidden layer

Dense(10, activation='softmax') # Output layer with 10 classes

])

# Compile and train the model

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

model.fit(x\_train, y\_train, epochs=10, batch\_size=32, validation\_split=0.2)

print("\n Training complete! \n")

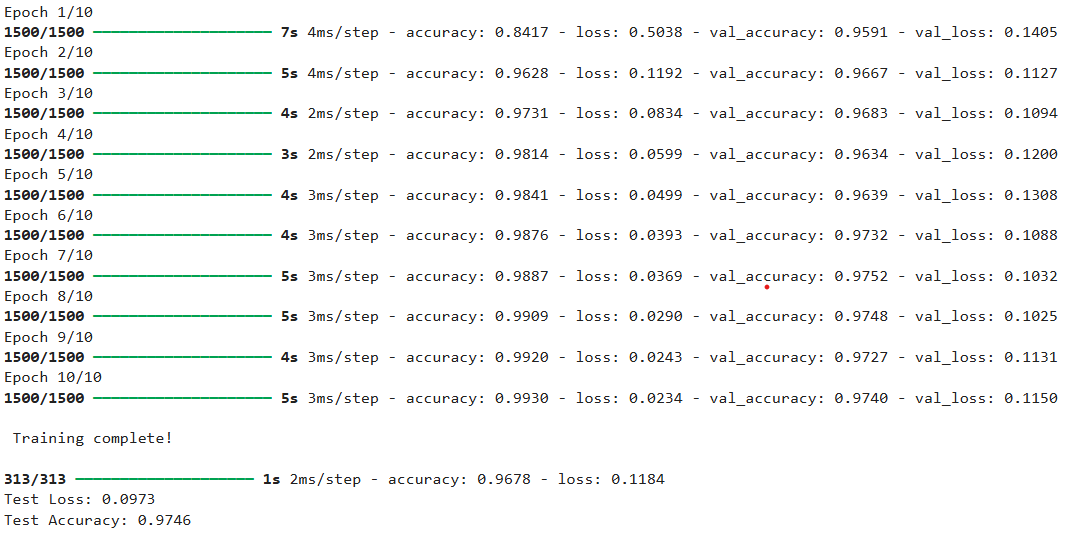
# Evaluate the model

loss, accuracy = model.evaluate(x\_test, y\_test)

print(f"Test Loss: {loss:.4f}")

print(f"Test Accuracy: {accuracy:.4f}")

**Output:**



**Lab Experiment - 4**

1. **Design and implement a CNN model with 4+ layers of convolution to classify multicategory image datasets. Use the concept of regularization and dropout while designing CNN model. Use the fashion MNIST datset. Record the training accuracy corresponding to the following architecture:**
   1. **Base model**
   2. **Model with L1 Regularization**
   3. **Model with L2 Regularization**
   4. **Model with Dropout**

**Program:**

import tensorflow as tf

from tensorflow.keras import layers, models, regularizers

from tensorflow.keras.datasets import fashion\_mnist

# Load and preprocess the dataset

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

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

x\_train = x\_train[..., None] # Add channel dimension

x\_test = x\_test[..., None]

y\_train, y\_test = tf.keras.utils.to\_categorical(y\_train, 10), tf.keras.utils.to\_categorical(y\_test, 10)

# Function to create the model with optional regularization or dropout

def create\_model(regularizer=None, dropout\_rate=None):

model = models.Sequential([

layers.Input(shape=(28, 28, 1)),

layers.Conv2D(32, (3, 3), activation='relu'),

layers.MaxPooling2D(),

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

layers.MaxPooling2D(),

layers.Flatten(),

layers.Dense(128, activation='relu', kernel\_regularizer=regularizer),

layers.Dropout(dropout\_rate) if dropout\_rate else layers.Dense(128, activation='relu'),

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

])

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

return model

# List of configurations for model creation

configurations = [

("Base Model", None, None),

("Model with L1 Regularization", regularizers.l1(1e-4), None),

("Model with L2 Regularization", regularizers.l2(1e-4), None),

("Model with Dropout", None, 0.5)

]

# Train and evaluate each model configuration

for name, regularizer, dropout\_rate in configurations:

print(f"\nTraining {name}...")

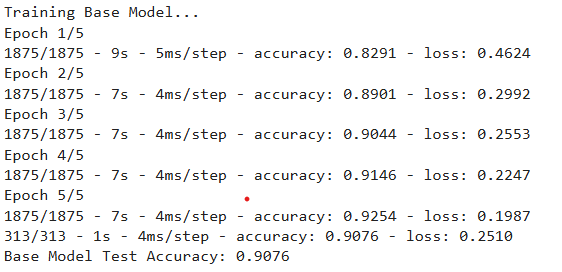
model = create\_model(regularizer, dropout\_rate)

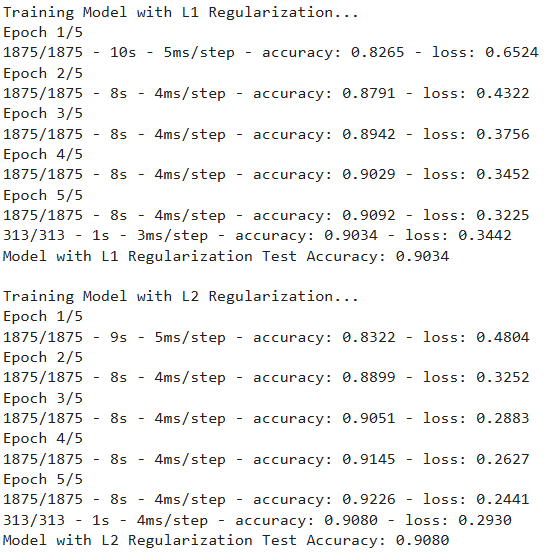
model.fit(x\_train, y\_train, epochs=5, batch\_size=32, verbose=2)

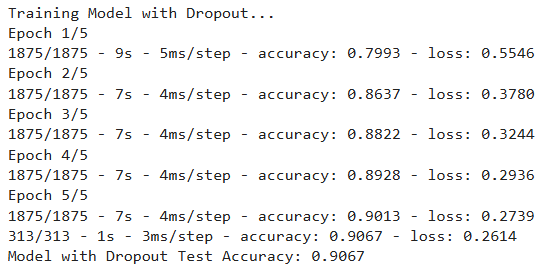
test\_loss, test\_acc = model.evaluate(x\_test, y\_test, verbose=2)

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

**Output:**







**Lab Experiment - 5**

1. **Design and implement an image classification model to classify a dataset of images using deep feed-forward neural network. Record the accuracy corresponding to the number of epochs. Use MNIST dataset.**

**Program:**

import tensorflow as tf

from tensorflow.keras import Sequential

from tensorflow.keras.layers import Dense

import matplotlib.pyplot as plt

# Load and preprocess the MNIST dataset

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

x\_train, x\_test = x\_train.reshape(-1, 784) / 255.0, x\_test.reshape(-1, 784) / 255.0

# Define and compile the model

model = Sequential([

Dense(128, activation='relu', input\_shape=(784,)), # First hidden layer

Dense(64, activation='relu'), # Second hidden layer

Dense(10, activation='softmax') # Output layer with 10 neurons (for 10 classes)

])

# Compile the model

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

# Train the model

history = model.fit(x\_train, y\_train, validation\_data=(x\_test, y\_test), epochs=10, batch\_size=32)

# Plot accuracy

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

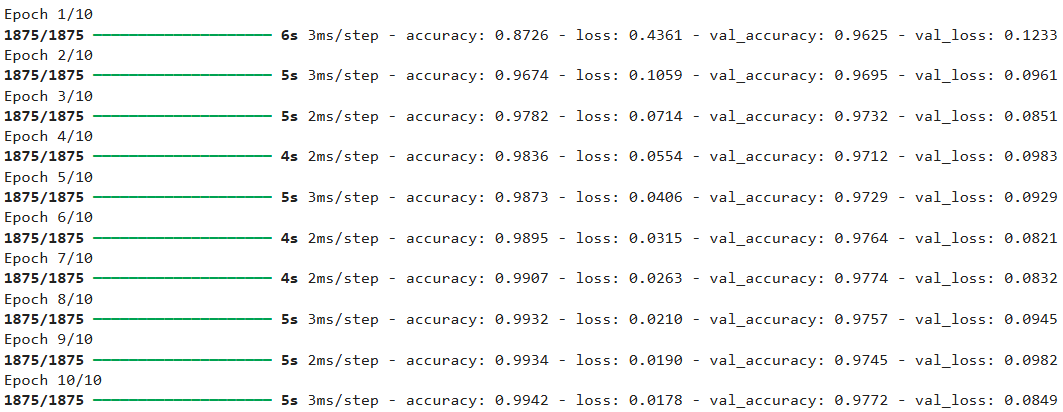
plt.show()

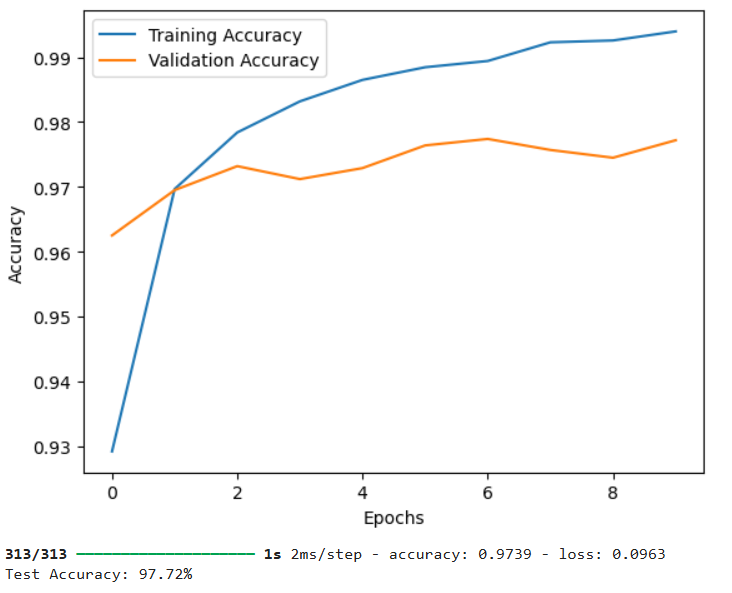
# Evaluate the model

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

print(f"Test Accuracy: {test\_accuracy \* 100:.2f}%")

**Output:**

****

****

**Lab Experiment - 6**

1. **Implement Bi-directional LSTM for Sentiment analysis on movie reviews.**

**Program:**

import tensorflow as tf

from tensorflow.keras.datasets import imdb

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, Bidirectional, LSTM, Dense

# Parameters

max\_features = 10000 # Top 10000 most common words

max\_len = 200 # Max length of each review

# Load and preprocess data

(x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data(num\_words=max\_features)

x\_train, x\_test = map(lambda x: pad\_sequences(x, maxlen=max\_len), (x\_train, x\_test))

# Build, compile, and train the model

model = Sequential([

Embedding(input\_dim=max\_features, output\_dim=64, input\_length=max\_len),

Bidirectional(LSTM(32)),

Dense(1, activation='sigmoid')

])

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

model.fit(x\_train, y\_train, epochs=5, batch\_size=64, validation\_split=0.2)

# Evaluate the model

print(f"Test Accuracy: {model.evaluate(x\_test, y\_test)[1]:.2f}")

# Test on a custom review

example\_review = "The movie was absolutely amazing, I loved it!"

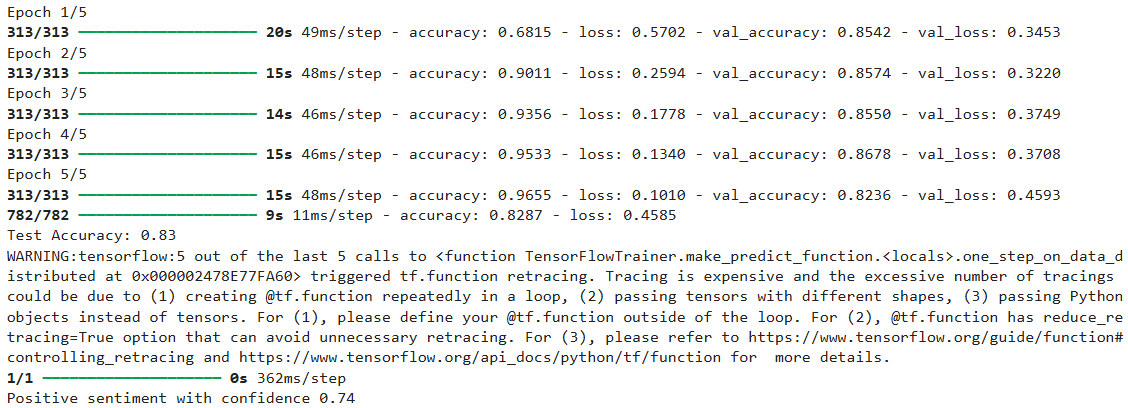
encoded\_review = [imdb.get\_word\_index().get(word, 2) for word in example\_review.lower().split()]

padded\_review = pad\_sequences([encoded\_review], maxlen=max\_len)

prediction = model.predict(padded\_review)[0][0]

print(f"{'Positive' if prediction < 0.5 else 'Negative'} sentiment with confidence {1 - prediction if prediction < 0.5 else prediction:.2f}")

**Output:**

****

**Lab Experiment - 7**

1. **Implement the standard VGG16 and 19 CNN architecture model to classify multicategory image dataset and check the accuracy.**

**Program:**

**A.**

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

from tensorflow.keras.utils import to\_categorical

# Load and preprocess Fashion MNIST dataset

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

x\_train, x\_test = x\_train[..., None] / 255.0, x\_test[..., None] / 255.0 # Normalize and add channel

y\_train, y\_test = to\_categorical(y\_train, 10), to\_categorical(y\_test, 10)

# Define a simpler VGG-like model

model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

MaxPooling2D((2, 2)),

Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Flatten(),

Dense(128, activation='relu'),

Dropout(0.5),

Dense(10, activation='softmax')

])

# Compile and train the model

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

model.fit(x\_train, y\_train, validation\_data=(x\_test, y\_test), epochs=5, batch\_size=128)

# Evaluate the model

loss, accuracy = model.evaluate(x\_test, y\_test)

print(f"Test Accuracy: {accuracy:.2f}")

**B.**

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

from tensorflow.keras.utils import to\_categorical

# Load and preprocess Fashion MNIST dataset

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

x\_train, x\_test = x\_train[..., None] / 255.0, x\_test[..., None] / 255.0 # Normalize and add channel

y\_train, y\_test = to\_categorical(y\_train, 10), to\_categorical(y\_test, 10)

# Define a VGG19-inspired model

model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

Conv2D(32, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Conv2D(64, (3, 3), activation='relu'),

Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Flatten(),

Dense(128, activation='relu'),

Dropout(0.5),

Dense(10, activation='softmax')

])

# Compile and train the model

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

model.fit(x\_train, y\_train, validation\_data=(x\_test, y\_test), epochs=5, batch\_size=128)

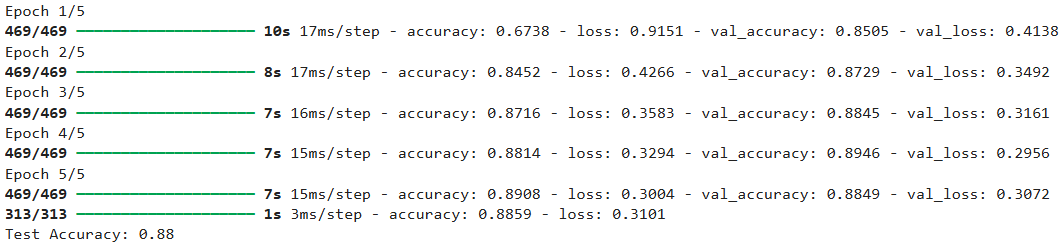
# Evaluate the model

loss, accuracy = model.evaluate(x\_test, y\_test)

print(f"Test Accuracy: {accuracy:.2f}")

**Output:**

**A.**



**B.**

