# UNIVERSITY OF MUMBAI

## DEPARTMENT OF COMPUTER SCIENCE

मुंबई विद्यापीठ
University of Mumbai
Re-accredited with A++ Grade
(CGPA 3.65) by NAAC (3rd Cycle 2021)

M.Sc. Computer Science – Semester IV

Advanced Deep Learning

JOURNAL

2023-2024

Seat No.__________

मुंबई विद्यापीठ
University of Mumbai
Re-accredited with A++ Grade
(CGPA 3.65) by NAAC (3rd Cycle 2021)

# UNIVERSITY OF MUMBAI

## DEPARTMENT OF COMPUTER SCIENCE

## CERTIFICATE

This is to certify that the work entered in this journal was done in the University Department of Computer Science laboratory by Mr/Ms. **Anagha Joshi** Seat No. __________for the course of M.Sc. Computer Science - Semester IV (CBCS) (Revised) during the academic year 2023- 2024 in a satisfactory manner.

__________________ __________________

**Subject In-charge** **Head of Department**

__________________

**External Examiner**

# INDEX

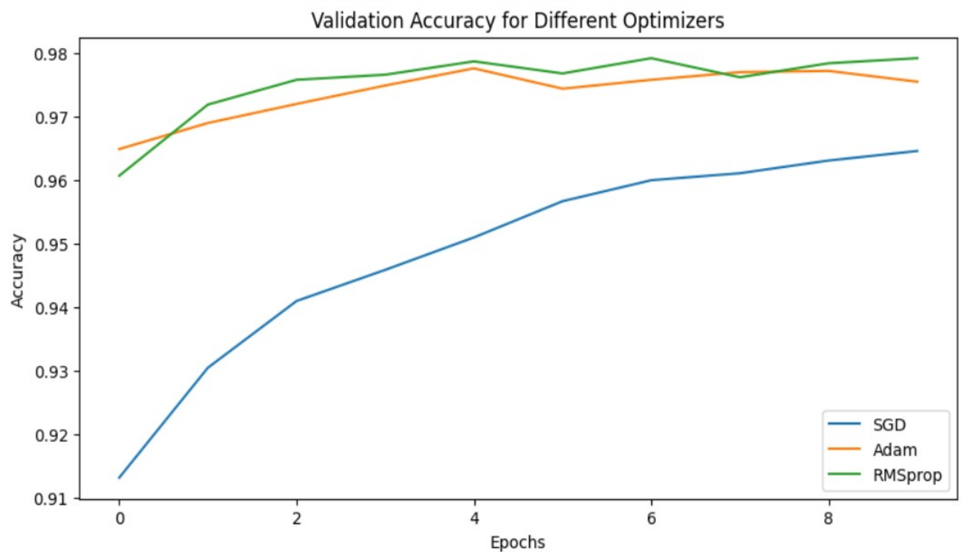
# Practical 1

**Aim : Implement Feed-forward Neural Network and train the network with different optimizers and compare the results**

```
# !pip install tensorflow matplotlib
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD, Adam, RMSprop
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

# Load and preprocess MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train.reshape(-1, 784) / 255.0, x_test.reshape(-1, 784) / 255.0
y_train, y_test = tf.keras.utils.to_categorical(y_train), tf.keras.utils.to_categorical(y_test)

# Define a function to build the model
def build_model():
    return Sequential([
        Dense(128, activation='relu', input_shape=(784,)),
        Dense(64, activation='relu'),
        Dense(10, activation='softmax')
    ])

# Optimizers to test
optimizers = {'SGD': SGD(), 'Adam': Adam(), 'RMSprop': RMSprop()}
results = {}

# Train and evaluate model for each optimizer
for name, optimizer in optimizers.items():
    model = build_model()
    model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
    history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test), verbose=0)
    results[name] = history.history['val_accuracy']

# Plot validation accuracy
plt.figure(figsize=(10, 5))
for name, val_acc in results.items():
    plt.plot(val_acc, label=name)
plt.title('Validation Accuracy for Different Optimizers')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
```

```
plt.legend()

plt.show()
```

**Output :**

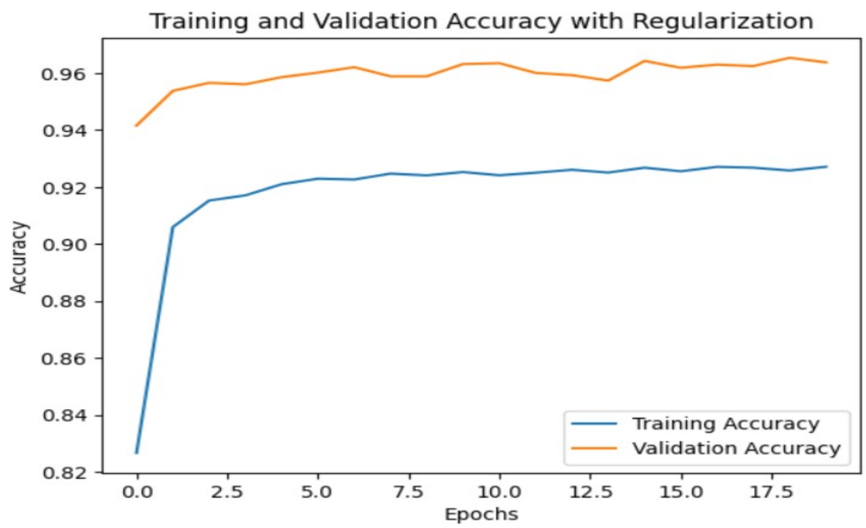


# Practical 2

**Aim : Write a program to implement regularization to prevent the model from overfitting**

```
# !pip install tensorflow matplotlib
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

# Load and preprocess MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train.reshape(-1, 784) / 255.0, x_test.reshape(-1, 784) / 255.0
y_train, y_test = tf.keras.utils.to_categorical(y_train), tf.keras.utils.to_categorical(y_test)

# Build the model with regularization and dropout
model = Sequential([
    Dense(128, activation='relu', input_shape=(784,),
kernel_regularizer=tf.keras.regularizers.l2(0.001)),
    Dropout(0.5),
    Dense(64, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.001)),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

# Compile and train the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(x_train, y_train, epochs=20, validation_data=(x_test, y_test), verbose=2)

# Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy with Regularization')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

**Output :**


Training and Validation Accuracy with Regularization
Accuracy
0.96
0.94
0.92
0.90
0.88
0.86
0.84
0.82
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epochs
Training Accuracy
Validation Accuracy

# Practical 3

**Aim : Implement deep learning for recognizing classes for datasets like CIFAR-10images for previously unseen images and assign them to one of the 10 classes**

```
# !pip install tensorflow matplotlib
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.datasets import cifar10
import matplotlib.pyplot as plt
import numpy as np

# Load and preprocess the CIFAR-10 dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0
classes = ["airplane", "automobile", "bird", "cat", "deer", "dog", "frog", "horse", "ship", "truck"]

# Build the CNN model
model = Sequential([
   Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
   MaxPooling2D((2, 2)),
   Conv2D(32, (3, 3), activation='relu'),
   MaxPooling2D((2, 2)),
   Flatten(),
   Dense(10, activation='softmax') ])

# Compile and train the model
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), verbose=2)

# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f'Test accuracy: {test_acc:.2f}')

# Predict and visualize some test images
predictions = model.predict(X_test[:10])
plt.figure(figsize=(20, 4))
for i in range(10):
   plt.subplot(2, 10, i + 1)
   plt.imshow(X_test[i])
   plt.xticks([])
   plt.yticks([])
   plt.title(classes[np.argmax(predictions[i])])
```

```
plt.show()
```

**Output**

```
Epoch 8/10
1563/1563 - 16s - 10ms/step - accuracy: 0.6931 - loss: 0.8915 - val_accuracy: 0.6685 - val_loss: 0.9581
Epoch 9/10
1563/1563 - 16s - 10ms/step - accuracy: 0.7012 - loss: 0.8652 - val_accuracy: 0.6789 - val_loss: 0.9485
Epoch 10/10
1563/1563 - 16s - 10ms/step - accuracy: 0.7068 - loss: 0.8501 - val_accuracy: 0.6862 - val_loss: 0.9316
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.6899 - loss: 0.9267
Test accuracy: 0.69
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step
```

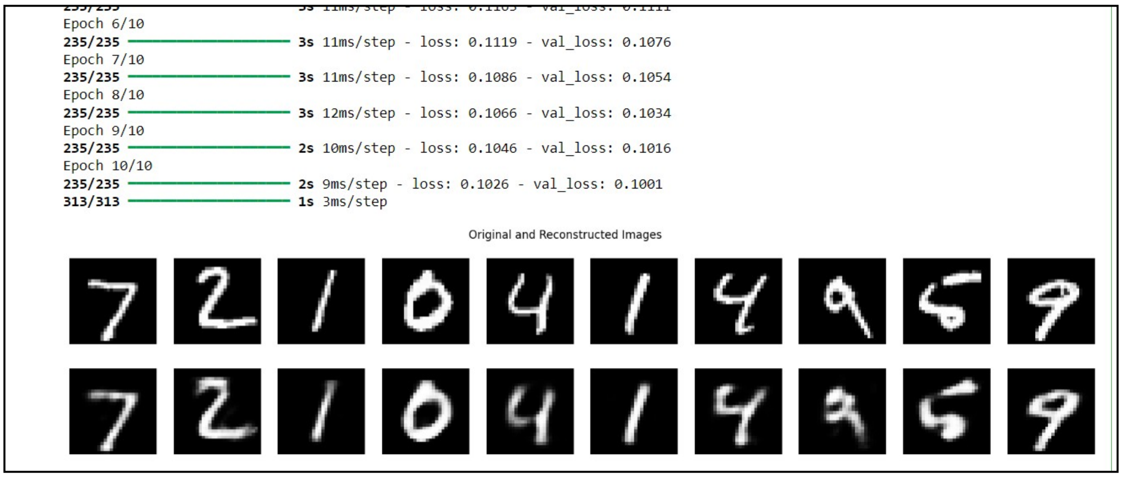
# Practical 4

**Aim : Implement deep learning for the prediction of the autoencoder from the testdata(e.g. MNIST data set)**

```
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

# Load and Prepare Data
(x_train, _), (x_test, _) = mnist.load_data()
x_train, x_test = x_train.reshape(-1, 784) / 255.0, x_test.reshape(-1, 784) / 255.0

# Define the Autoencoder Model
input_img = Input(shape=(784,))
encoded = Dense(64, activation='relu')(input_img)
encoded = Dense(32, activation='relu')(encoded)
decoded = Dense(64, activation='relu')(encoded)
decoded = Dense(784, activation='sigmoid')(decoded)

autoencoder = Model(input_img, decoded)

# Compile and Train the Autoencoder
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.fit(x_train, x_train, epochs=10, batch_size=256, validation_data=(x_test, x_test))

# Predict and Display Results
decoded_imgs = autoencoder.predict(x_test)

# Plotting original and reconstructed images
plt.figure(figsize=(20, 4))
for i in range(10):
    # Original images
    ax = plt.subplot(2, 10, i + 1)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    ax.axis('off')

    # Reconstructed images
    ax = plt.subplot(2, 10, i + 11)
    plt.imshow(decoded_imgs[i].reshape(28, 28), cmap='gray')
    ax.axis('off')
```

plt.suptitle('Original and Reconstructed Images')
plt.show()

**Output :**



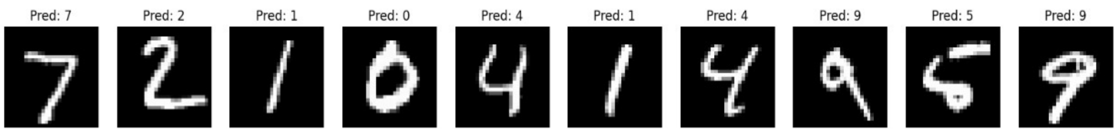
# Practical 5

**Aim : Implement Convolutional Neural Network for Digit recognition on the MNISTdataset**

```
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

# Load and Prepare Data
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Build the CNN 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(64, activation='relu'),
    Dense(10, activation='softmax')
])

# Compile and Train the Model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model.fit(x_train.reshape(-1, 28, 28, 1), y_train, epochs=5, batch_size=64,
validation_data=(x_test.reshape(-1, 28, 28, 1), y_test))

# Evaluate the Model
test_loss, test_acc = model.evaluate(x_test.reshape(-1, 28, 28, 1), y_test)
print(f'Test accuracy: {test_acc:.4f}')

# Predictions and Visualization
predictions = model.predict(x_test.reshape(-1, 28, 28, 1))
plt.figure(figsize=(20, 4))
for i in range(10):
    ax = plt.subplot(2, 10, i + 1)
    plt.imshow(x_test[i], cmap='gray')
    ax.axis('off')
    ax.set_title(f'Pred: {predictions[i].argmax()}')
plt.show()
```

## Output :

```
Epoch 1/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 22s 22ms/step - accuracy: 0.8835 - loss: 0.4038 - val_accuracy: 0.9814 - val_loss: 0.0592
Epoch 2/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 22s 24ms/step - accuracy: 0.9826 - loss: 0.0571 - val_accuracy: 0.9875 - val_loss: 0.0376
Epoch 3/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 21s 22ms/step - accuracy: 0.9886 - loss: 0.0388 - val_accuracy: 0.9893 - val_loss: 0.0338
Epoch 4/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 21s 22ms/step - accuracy: 0.9899 - loss: 0.0296 - val_accuracy: 0.9888 - val_loss: 0.0353
Epoch 5/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 21s 22ms/step - accuracy: 0.9932 - loss: 0.0220 - val_accuracy: 0.9909 - val_loss: 0.0287
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9877 - loss: 0.0358
Test accuracy: 0.9909
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step
```

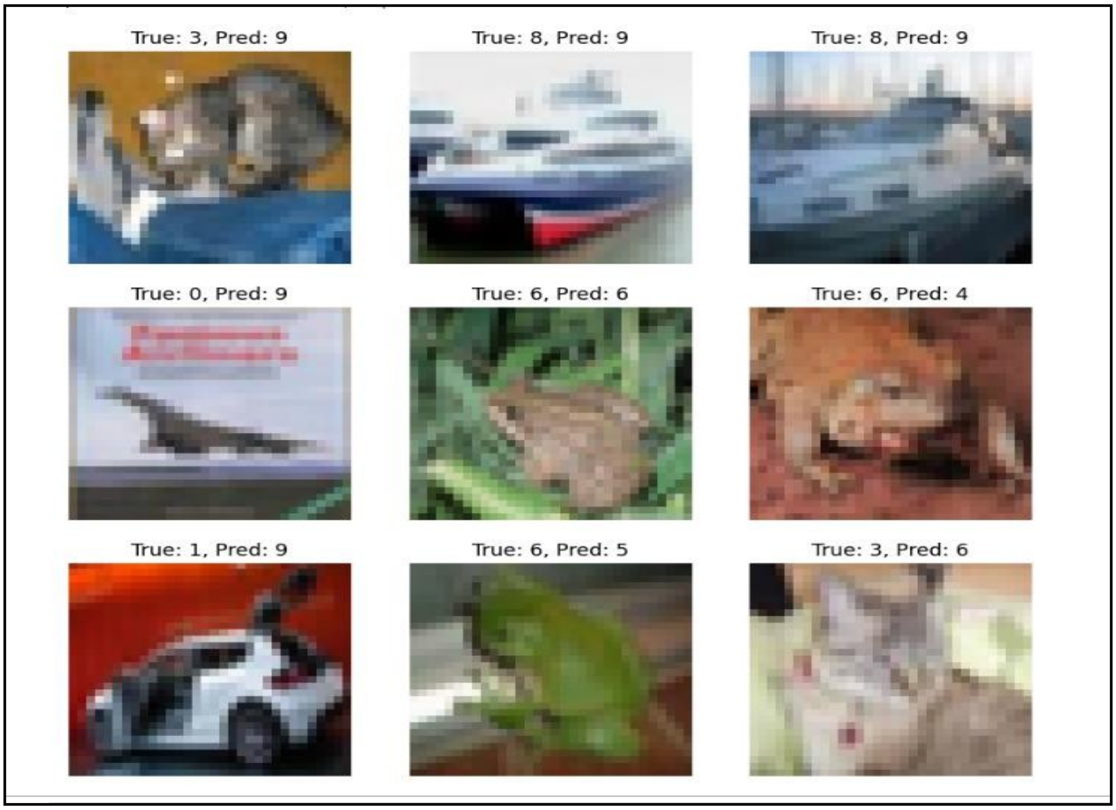


# Practical 6

**Aim : Write a program to implement Transfer Learning on the suitable dataset**

```
import tensorflow as tf
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense
from tensorflow.keras.models import Model
import matplotlib.pyplot as plt

# Load and preprocess CIFAR-10 dataset
(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

# Load pre-trained MobileNetV2 model
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(32, 32, 3))
x = GlobalAveragePooling2D()(base_model.output)
predictions = Dense(10, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)

# Freeze base model layers
base_model.trainable = False

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

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

# Predict and visualize some test images
predictions = model.predict(x_test[:9])
plt.figure(figsize=(10, 10))
for i in range(9):
    plt.subplot(3, 3, i+1)
    plt.imshow(x_test[i])
    plt.title(f"True: {y_test[i][0]}, Pred: {predictions[i].argmax()}")
    plt.axis('off')
plt.show()
```

**Output :**


True: 3, Pred: 9
True: 8, Pred: 9
True: 8, Pred: 9
True: 0, Pred: 9
True: 6, Pred: 6
True: 6, Pred: 4
True: 1, Pred: 9
True: 6, Pred: 5
True: 3, Pred: 6

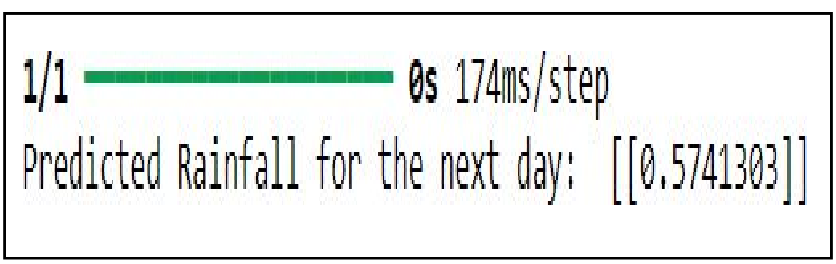
# Practical 7

**Aim : Write a program to implement a simple form of a recurrent neural network.**

**a. E.g. (4-to-1 RNN) to show that the quantity of rain on a certain day also depends on the values of the previous day**

```
import numpy as np
import tensorflow as tf

# Generate synthetic data
data = np.random.rand(100, 1)
X, y = [], []
for i in range(len(data) - 4):
    X.append(data[i:i+4])
    y.append(data[i+4])
X, y = np.array(X), np.array(y)

# Build and train the RNN model
model = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(50, input_shape=(4, 1)),
    tf.keras.layers.Dense(1)
])
model.compile(optimizer='adam', loss='mse')
model.fit(X, y, epochs=200, verbose=0)

# Predict the next value
print('Predicted Rainfall for the next day: ',model.predict(data[-4:].reshape(1, 4, 1)))
```

**Output :**



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 174ms/step
Predicted Rainfall for the next day:  [[0.5741303]]
```

**b. LSTM for sentiment analysis on datasets like UMICH SI650 for similar.**

```
import tensorflow as tf
from sklearn.model_selection import train_test_split
import nltk
from nltk.corpus import movie_reviews
nltk.download('movie_reviews')

# Load and preprocess data
sentences = [" ".join(movie_reviews.words(fileid)) for fileid in movie_reviews.fileids()]
labels = [1 if fileid.startswith('pos') else 0 for fileid in movie_reviews.fileids()]

tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=5000)
X = tokenizer.texts_to_sequences(sentences)
X = tf.keras.preprocessing.sequence.pad_sequences(X, maxlen=100)
y = np.array(labels)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build and train the LSTM model
model = tf.keras.Sequential([
   tf.keras.layers.Embedding(5000, 128, input_length=100),
   tf.keras.layers.LSTM(128),
   tf.keras.layers.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_data=(X_test, y_test))

# Evaluate the model
print('Test Accuracy: ',model.evaluate(X_test, y_test)[1])
```

**Output:**

```
[nltk_data] Downloading package movie_reviews to
[nltk_data]     C:\Users\Admin\AppData\Roaming\nltk_data...
[nltk_data]   Package movie_reviews is already up-to-date!
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\layers\core\embedding.py:90: UserWarning: Ar
gument `input_length` is deprecated. Just remove it.
  warnings.warn(

Epoch 1/5
25/25 ━━━━━━━━━━━━━━━━━━━━ 7s 161ms/step - accuracy: 0.4857 - loss: 0.6946 - val_accuracy: 0.5025 - val_loss: 0.6931
Epoch 2/5
25/25 ━━━━━━━━━━━━━━━━━━━━ 4s 150ms/step - accuracy: 0.5049 - loss: 0.6938 - val_accuracy: 0.4975 - val_loss: 0.6932
Epoch 3/5
25/25 ━━━━━━━━━━━━━━━━━━━━ 4s 146ms/step - accuracy: 0.4961 - loss: 0.6933 - val_accuracy: 0.5025 - val_loss: 0.6931
Epoch 4/5
25/25 ━━━━━━━━━━━━━━━━━━━━ 4s 147ms/step - accuracy: 0.4934 - loss: 0.6933 - val_accuracy: 0.4975 - val_loss: 0.6932
Epoch 5/5
25/25 ━━━━━━━━━━━━━━━━━━━━ 4s 146ms/step - accuracy: 0.4986 - loss: 0.6932 - val_accuracy: 0.5025 - val_loss: 0.6931
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.4988 - loss: 0.6932
Test Accuracy:  0.5024999976158142
```

# Practical 8

**Aim : Write a program for object detection using pre-trained models to use objectDetection**

```
#importing libraries
import cv2
import matplotlib.pyplot as plt
from matplotlib import ft2font
#importing and using necessary files
config_file=r'C:\Users\HARDIK PATIL\ssd_mobilenet_v3_large_coco_2020_01_14(1).pbtxt'
frozen_model=r'C:\Users\HARDIK PATIL\frozen_inference_graph.pb'
#Tenserflow object detection model
model = cv2.dnn_DetectionModel(frozen_model,config_file)

#Reading Coco dataset
classLabels=[]
filename=r'C:\Users\HARDIK PATIL\coco.names'
with open(filename,'rt') as fpt:
  classLabels = fpt.read().rstrip('\n').split('\n')

print("Number of Classes")
print(len(classLabels))
print("Class labels")
print(classLabels)
```

```
Number of Classes
80
Class labels
['person', 'bicycle', 'car', 'motorbike', 'aeroplane', 'bus', 'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'stop
sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow', 'elephant', 'bear', 'zebra', 'giraffe', 'backpa
ck', 'umbrella', 'handbag', 'tie', 'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat', 'baseball
glove', 'skateboard', 'surfboard', 'tennis racket', 'bottle', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana',
'apple', 'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'chair', 'sofa', 'pottedplant', 'be
d', 'diningtable', 'toilet', 'tvmonitor', 'laptop', 'mouse', 'remote', 'keyboard', 'cell phone', 'microwave', 'oven', 'toaste
r', 'sink', 'refrigerator', 'book', 'clock', 'vase', 'scissors', 'teddy bear', 'hair drier', 'toothbrush']
```

```
#Model training
model.setInputSize(320,320)
model.setInputScale(1.0/127.5)
model.setInputMean((127.5,127.5,127.5))
model.setInputSwapRB(True)
#reading image
img = cv2.imread(r'C:\Users\HARDIK PATIL\test.jpg')
plt.imshow(img)
```

```
Out[5]: <matplotlib.image.AxesImage at 0x21466c52fd0>
```

```
#object detection
ClassIndex, confidence, bbox = model.detect(img, confThreshold=0.5)

for class_id, conf, box in zip(ClassIndex.flatten(), confidence.flatten(), bbox):
        if class_id in desired_classes:
            class_counts[class_id] += 1  # Increment counter for the detected class
            label = f'{classLabels[class_id-1]}: {conf:.2f}'
            cv2.rectangle(img, box, color=(0, 255, 0), thickness=2)
           cv2.putText(img, label, (box[0], box[1] - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,255, 0),2)

    # Display the image with detected objects
print(class_counts)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.show()
```

**Output :**

```
{1: 22, 3: 0}
```