**1 ) Solving XOR problem using DNN**

**PROGRAM :**

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

X = np.array([[0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 1], [1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1]])

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

model = Sequential()

model.add(Dense(4, activation='relu', input\_shape=(3,)))

model.add(Dense(16, activation='relu'))

model.add(Dense(32, activation='relu'))

model.add(Dense(64, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

model.summary()

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

model.fit(X, y, epochs=100, verbose=1)

**loss,accuracy = model.evaluate(X,y)**

**print(f"loss:{loss}")**

**print(f"Accuracy:{accuracy}")**

**predictions = model.predict(X)**

**result = 0**

**for i in range(8):**

**if predictions[i] < 0.5:**

**result = 0**

**else:**

**result = 1**

**print(X[i], result)**

**2 A ) BUILDING A CNN MODEL FOR MNIST DATA**

**PROGRAM :**

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers, models

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

import seaborn as sns

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

X\_train = X\_train.reshape((X\_train.shape[0], 28, 28, 1)).astype('float32') / 255.0

X\_test = X\_test.reshape((X\_test.shape[0], 28, 28, 1)).astype('float32') / 255.0

print(X\_train.shape, y\_train.shape, X\_test.shape, y\_test.shape)

plt.imshow(X\_train[1000].reshape(28, 28), cmap='gray')

plt.title(f'Label: {y\_train[1000]}')

plt.axis('off')

plt.show()

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

    layers.Flatten(),

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

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

])

model.compile(optimizer='adam',

              loss='sparse\_categorical\_crossentropy',

              metrics=['accuracy'])

model.fit(X\_train, y\_train, epochs=15, batch\_size=64, validation\_split=0.1)

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

print(f'Test accuracy: {test\_acc:.4f}')

loss, accuracy = model.evaluate(X\_train, y\_train)

print(f"Loss: {loss}")

print(f"Accuracy: {accuracy}")  result = model.predict(X\_test)

y\_pred = np.argmax(result, axis=1)

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

**2 B ) BUILDING A CNN MODEL FOR CIFAR DATA**

**PROGRAM :**

import numpy as np

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from sklearn.metrics import confusion\_matrix, classification\_report

import matplotlib.pyplot as plt

import seaborn as sns

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

X\_train = X\_train.astype('float32') / 255.0

X\_test = X\_test.astype('float32') / 255.0

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

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

model = keras.Sequential([

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

    layers.MaxPooling2D((2, 2)),

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

    layers.MaxPooling2D((2, 2)),

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

    layers.Flatten(),

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

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

])

model.compile(optimizer='adam',

              loss='categorical\_crossentropy',   metrics=['accuracy'])

model.fit(X\_train, y\_train, epochs=50, batch\_size=64, validation\_data=(X\_test, y\_test))

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

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

y\_pred\_prob = model.predict(X\_test)

y\_pred = np.argmax(y\_pred\_prob, axis=1)  # Get the predicted class indices

y\_true = np.argmax(y\_test, axis=1)  # Get the true class indices for the ground truth

cm = confusion\_matrix(y\_true, y\_pred)

**3(a) FACE RECOGNITION USING CNN Date:**

from tensorflow.keras.preprocessing.image import ImageDataGenerator import os

data\_dir='C:/Users/VarshaK/Downloads/lfw\_facedata/lfw-deepfunneled/lfw-deepfunneled' datagen=ImageDataGenerator(rescale=1./255, validation\_split=0.2)

train\_data=datagen.flow\_from\_directory(

directory=data\_dir,

target\_size=(128,128),

batch\_size=32,

class\_mode='categorical',

subset='training'

)

val\_data=datagen.flow\_from\_directory(

directory=data\_dir,

target\_size=(128,128),

batch\_size=32,

class\_mode='categorical',

subset='validation'

)

pip install pillow

pip install matplotlib

import matplotlib.pyplot as plt

images,labels=next(train\_data)

plt.imshow(images[3])

class\_indices=train\_data.class\_indices

index\_to\_class={v: k for k,v in class\_indices.items()}

predicted\_label=3

person\_name=index\_to\_class[predicted\_label]

print('predicted person:', person\_name)

from tensorflow.keras import layers, models

model = models.Sequential()

model.add(layers.Conv2D(32,(3,3), activation ='relu', input\_shape=(128,128,3))) model.add(layers.MaxPooling2D(2,2))

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

model.add(layers.MaxPooling2D(2,2))

model.add(layers.Conv2D(128, (3,3), activation ='relu'))

model.add(layers.MaxPooling2D((2,2)))

model.add(layers.Flatten())

model.add(layers.Dense(444, activation = 'relu'))

model.add(layers.Dropout(0,2))

model.add(layers.Dense(5749, activation='softmax'))

model.summary()

model.compile(optimizer = 'adam', loss = 'categorical\_crossentropy', metrics =['accuracy']) model.fit(train\_data, validation\_data = val\_data, epochs = 20)

import cv2

import numpy as np

import matplotlib.pyplot as plt

img\_path = "C:/Users/Varsha

K/Downloads/lfw\_facedata/lfw-deepfunneled/lfw-deepfunneled/Aaron\_Guiel/Aaron\_Guie l\_0001.jpg"

img = cv2.imread(img\_path, cv2.IMREAD\_COLOR)

if img is not None:

img\_resized = cv2.resize(img, (128, 128))

img\_resized\_rgb = cv2.cvtColor(img\_resized, cv2.COLOR\_BGR2RGB) img\_normalized = img\_resized / 255.0

plt.imshow(img\_normalized)

plt.axis('off')

plt.show()

img\_expanded = np.expand\_dims(img\_normalized, axis=0) result = model.predict(img\_expanded)

print("Prediction result:", result)

else:

print("Error: Image not found or could not be loaded.")

predicted\_label = np.argmax(result)

predicted\_label

index\_to\_class[predicted\_label]

import gradio as gr

def preprocess\_image(image):

image = cv2.cvtColor(image, cv2.IMREAD\_COLOR)

#image = cv2.imread(image, cv2.IMREAD\_COLOR)

image = cv2.resize(image, (128,128))

image = image/255.

image =np.expand\_dims(image, axis=0)

return image

def predict(image):

image = np.array(image)

image = preprocess\_image(image)

prediction = model.predict(image)

predicted\_label = np.argmax(result)

return index\_to\_class[predicted\_label]

interface = gr.Interface(

fn=predict,

inputs=gr.Image(type="numpy"),

outputs=gr.Textbox(),

live=True, # Set to True if you want live updates

)

interface.launch(share=True)

**3 B A CNN MODEL FOR FACE DETECTION**

**PROGRAM :**

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import os

from tensorflow.keras import layers,models

import Augmentor

data\_dir = r"C:\Users\Harish\Downloads\New\_class\_data"

for class\_name in os.listdir(data\_dir):

    class\_path = os.path.join(data\_dir, class\_name)

    if os.path.isdir(class\_path):

        p = Augmentor.Pipeline(class\_path)

        p.rotate(probability=0.7, max\_left\_rotation=10, max\_right\_rotation=10)

        p.zoom\_random(probability=0.5, percentage\_area=0.8)

        p.flip\_left\_right(probability=0.5)

        p.random\_contrast(probability=0.3, min\_factor=0.7, max\_factor=1.3)

        p.random\_brightness(probability=0.3, min\_factor=0.7, max\_factor=1.3)

        num\_original\_images = len(os.listdir(class\_path))

        images\_needed = 50 - num\_original\_images

        if images\_needed > 0:

            p.sample(images\_needed)

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import os

data\_dir = '/content/drive/MyDrive/New\_class\_data'

datagen = ImageDataGenerator(rescale=1./255, validation\_split=0.2)

train\_data = datagen.flow\_from\_directory(

    directory=data\_dir,

    target\_size=(128, 128),

    batch\_size=32,

    class\_mode='categorical',

    subset='training'

)

val\_data = datagen.flow\_from\_directory(

    directory=data\_dir,

    target\_size=(128, 128),

    batch\_size=32,

    class\_mode='categorical',

    subset='validation'

)

print(train\_data.class\_indices)

class\_indices = train\_data.class\_indices

index\_to\_class = {v: k for k, v in class\_indices.items()}

predicted\_label = 3

person\_name = index\_to\_class[predicted\_label]

print("Predicted person:", person\_name)

from tensorflow.keras import layers, models

model = models.Sequential()

model.add(layers.Conv2D(32,(3,3), activation = 'relu', input\_shape = (128,128,3)))

model.add(layers.MaxPooling2D(2,2))

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

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(128, (3, 3), activation='relu'))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())

model.add(layers.Dense(444, activation='relu'))

model.add(layers.Dropout(0.2))

model.add(layers.Dense(6, activation ='softmax'))

model.summary()

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

model.fit(train\_data,validation\_data = val\_data, epochs = 25)

model.save('/content/drive/MyDrive/band\_model.h5')

import gradio as gr

import cv2

import numpy as np

from tensorflow.keras.models import load\_model

model = load\_model('/content/drive/MyDrive/band\_model.h5')

def predict\_image(image):

    image = np.array(image)

    img = cv2.resize(image, (128, 128))

    img = img / 255.0

    img\_batch = np.expand\_dims(img, axis=0)

    predictions = model.predict(img\_batch)

    predicted\_class = np.argmax(predictions, axis=1)[0]

    a=index\_to\_class[predicted\_class]

    return f"Predicted Class: {a}"

interface = gr.Interface(

    fn=predict\_image,

    inputs=gr.Image(),

    outputs="text",

    title="Image Classification",

    description="Upload an image to classify it using a pre-trained model."

)

interface.launch()

**4 A MODELLING USING RNN**

**PROGRAM :**

import pandas as pd

import numpy as np

import tensorflow as tf

from tensorflow.keras.preprocessing.text import Tokenizer

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

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.utils import to\_categorical

corpus = [

    "Life is a mystery.",

    "Go with the flow.",

    "Just get over it.",

    "Everything will work out in the end.",

    "I need a break from everything."]

tokenizer = Tokenizer()

tokenizer.fit\_on\_texts(corpus)

total\_words = len(tokenizer.word\_index) + 1

input\_sequences=[]

for line in corpus:

    token\_list = tokenizer.texts\_to\_sequences([line])[0]

    for i in range(1, len(token\_list)):

        n\_gram\_sequence = token\_list[:i+1]

        input\_sequences.append(n\_gram\_sequence)

max\_sequence\_len = max([len(seq) for seq in input\_sequences])

input\_sequences = pad\_sequences(input\_sequences, maxlen = max\_sequence\_len, padding = 'pre')

X,y = input\_sequences[:, :-1], input\_sequences[:, -1]

y = to\_categorical(y, num\_classes = total\_words)

model = Sequential()

model.add(Embedding(total\_words, 100, input\_length = max\_sequence\_len-1))

model.add(SimpleRNN(150, return\_sequences = False))

model.add(Dense(total\_words, activation = 'softmax'))

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

history = model.fit(X, y, epochs = 100, verbose = 1)

def predict\_next\_word(model, tokenizer, text, max\_sequence\_len):

    token\_list = tokenizer.texts\_to\_sequences([text])[0]

    token\_list = pad\_sequences([token\_list], maxlen = max\_sequence\_len - 1, padding = 'pre')

    predicted = model.predict(token\_list, verbose = 0)

    predicted\_word\_index = np.argmax(predicted, axis = -1)[0]

    for word, index in tokenizer.word\_index.items():

        if index == predicted\_word\_index:

            return word

seed\_text = "I need a"

next\_word = predict\_next\_word(model,tokenizer, seed\_text, max\_sequence\_len)

print(f"Next word after '{seed\_text}': {next\_word}")

seed\_text = "Life is"

next\_word = predict\_next\_word(model, tokenizer, seed\_text, max\_sequence\_len)

print(f"Next word after '{seed\_text}' : {next\_word}

**4 B MODELLING USING RNN**

**PROGRAM AND OUTPUT :**

import pandas as pd

import os

import numpy as np

import tensorflow as tf

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

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

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.models import Sequential

medium\_data = pd.read\_csv("Downloads/medium\_data.csv/medium\_data.csv")

medium\_data.head()

print("Number of records:", medium\_data.shape[0])

print("Number of fields:", medium\_data.shape[1])

medium\_data["title"]

medium\_data['title'] = medium\_data['title'].apply(lambda x: x.replace(u'\xa0', u' '))

medium\_data['title'] = medium\_data['title'].apply(lambda x: x.replace('\u200a', ' '))

tokenizer = Tokenizer(oov\_token = '<oov>')

tokenizer.fit\_on\_texts(medium\_data['title'])

total\_words = len(tokenizer.word\_index) + 1

print("Total number of words : ", total\_words)

input\_sequences = []

for line in medium\_data['title']:

    token\_list = tokenizer.texts\_to\_sequences([line])[0]

    for i in range(1, len(token\_list)):

        n\_gram\_sequences = token\_list[:i+1]

        input\_sequences.append(n\_gram\_sequences)

print("Total input sequences: " , len(input\_sequences))

max\_sequence\_len = max([len(x) for x in input\_sequences])

input\_sequences = np.array(pad\_sequences(input\_sequences, maxlen = max\_sequence\_len, padding = 'pre'))

input\_sequences[1]

xs, labels = input\_sequences[:,:-1], input\_sequences[:,-1]

ys = tf.keras.utils.to\_categorical(labels, num\_classes = total\_words)

print(xs[5])

print(labels[5])

print(ys[5][14])

model = Sequential()

model.add(Embedding(total\_words, 100, input\_length = max\_sequence\_len - 1))

model.add(SimpleRNN(150))

model.add(Dense(total\_words, activation = 'softmax'))

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

history = model.fit(xs, ys, epochs = 1, verbose = 1)

print(model)

seed\_text = "A Beginner's Guide"

next\_words = 2

for \_ in range(next\_words):

    token\_list = tokenizer.texts\_to\_sequences([seed\_text])[0]

    token\_list = pad\_sequences([token\_list], maxlen = max\_sequence\_len - 1, padding = 'pre')

    predicted = model.predict(token\_list, verbose = 0)

    predicted\_word\_index = np.argmax(predicted, axis = -1)

    output\_word = ""

    for word, index in tokenizer.word\_index.items():

        if index == predicted\_word\_index:

            output\_word = word

            break

    seed\_text += " " + output\_word

print(seed\_text)

**5 A SENTIMENT ANALYSIS USING LSTM**

**PROGRAM :**

import numpy as np

import pandas as pd

from tensorflow.keras.preprocessing.text import Tokenizer

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

from tensorflow.keras.models import Sequential

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

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

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import classification\_report, accuracy\_score

data = pd.read\_csv('/content/drive/MyDrive/IMDB Dataset.csv')

texts = data['review'].values

labels = data['sentiment'].values

label\_encoder = LabelEncoder()

labels = label\_encoder.fit\_transform(labels)

max\_vocab\_size = 5000

max\_sequence\_length = 100

tokenizer = Tokenizer(num\_words=max\_vocab\_size, oov\_token="<OOV>")

tokenizer.fit\_on\_texts(texts)

word\_index = tokenizer.word\_index

sequences = tokenizer.texts\_to\_sequences(texts)

padded\_sequences = pad\_sequences(sequences, maxlen=max\_sequence\_length, padding='post')

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

model = Sequential()

model.add(Embedding(input\_dim=max\_vocab\_size, output\_dim=64, input\_length=max\_sequence\_length))

model.add(LSTM(64, return\_sequences=False))

model.add(Dropout(0.5))

model.add(Dense(32, activation='relu'))

model.add(Dense(len(set(labels)), activation=sigmoid))

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

model.fit(X\_train, y\_train, epochs=7, validation\_data=(X\_test, y\_test), batch\_size=32)

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

print(f'Test Accuracy: {accuracy}')

**def predict\_emotion(input\_text):**

    input\_seq = tokenizer.texts\_to\_sequences([input\_text])

    input\_padded = pad\_sequences(input\_seq, maxlen=max\_sequence\_length, padding='post')

    prediction = model.predict(input\_padded)

    predicted\_class = np.argmax(prediction, axis=1)

    emotion\_label = label\_encoder.inverse\_transform(predicted\_class)

    return emotion\_label[0]

**def gradio\_predict(text):**

    return predict\_emotion(text)

interface = gr.Interface(fn=gradio\_predict,

                         inputs="text",

                         outputs="text",

                         title="Emotion Prediction",

                         description="Enter a sentence and the model will predict the emotion.")

interface.launch()

**Ex No : 6         PARTS OF SPEECH TAGGING USING SEQUENCE TO SEQUENCE**

import numpy as np

from nltk.corpus import brown,treebank,conll2000

from keras.preprocessing.sequence import pad\_sequences

from keras.models import Sequential,Model

from keras.layers import Embedding,Dense,Input,LSTM,GRU,Bidirectional,SimpleRNN ,RNN

from tensorflow.keras.preprocessing.text import Tokenizer

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

from tensorflow.keras.utils import to\_categorical

import nltk

nltk.download('treebank')

nltk.download('brown')

nltk.download('conll2000')

nltk.download('universal\_tagset')

treebank\_corpus=treebank.tagged\_sents(tagset='universal')

brown\_corpus=brown.tagged\_sents(tagset='universal')

conll\_corpus=conll2000.tagged\_sents(tagset='universal')

tagged\_sentences=treebank\_corpus+brown\_corpus+conll\_corpus

X=[]

Y=[]

for sentence in tagged\_sentences:

    X\_sentence=[]

    Y\_sentence=[]

    for entity in sentence:

        X\_sentence.append(entity[0])

        Y\_sentence.append(entity[1])

    X.append(X\_sentence)

    Y.append(Y\_sentence)

word\_tokenizer=Tokenizer(oov\_token='<OOV>')

word\_tokenizer.fit\_on\_texts(Y)

tag\_tokenizer=Tokenizer(lower=False)

tag\_tokenizer.fit\_on\_texts(Y)

X\_sequences=word\_tokenizer.texts\_to\_sequences(X)

Y\_sequences=tag\_tokenizer.texts\_to\_sequences(Y)

maxlen=max([len(seq) for seq in X\_sequences])

X\_padded = pad\_sequences(X\_sequences, maxlen=maxlen, padding='post')

Y\_padded = pad\_sequences(Y\_sequences, maxlen=maxlen, padding='post')

num\_tags=len(tag\_tokenizer.word\_index)+1

Y\_onehot=[to\_categorical(seq,num\_classes=num\_tags) for seq in Y\_padded]

Y\_onehot = np.array(Y\_onehot)

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

vocab\_size = len(word\_tokenizer.word\_index) + 1

embedding\_dim = 128

hidden\_size = 256

num\_tags = len(tag\_tokenizer.word\_index) + 1

input\_seq = Input(shape=(maxlen,))

embedding\_layer = Embedding(input\_dim=vocab\_size, output\_dim=embedding\_dim, input\_length=maxlen)(input\_seq)

encoder\_output = Bidirectional(LSTM(hidden\_size, return\_sequences=True))(embedding\_layer)

output\_layer = Dense(num\_tags, activation='softmax')(encoder\_output)

model = Model(inputs=input\_seq, outputs=output\_layer)

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

model.fit(X\_train, Y\_train, epochs=10, batch\_size=32, validation\_data=(X\_test, Y\_test))

sentence = ["The", "dog", "eats", "snacks"]

X\_input = word\_tokenizer.texts\_to\_sequences([sentence])

X\_input\_padded = pad\_sequences(X\_input, maxlen=maxlen, padding='post')

predictions = model.predict(X\_input\_padded)

predicted\_indices = np.argmax(predictions, axis=-1)

predicted\_tags = []

for idx in predicted\_indices[0]:

    if idx != 0:

        predicted\_tags.append(tag\_tokenizer.index\_word.get(idx, "UNK"))

print("Input sentence: ", sentence)

print("Predicted POS Tags: ", predicted\_tags)

**Ex No : 7 MACHINE TRANSLATION USING ENCODER-DECODER MODEL**

**PROGRAM :**

import pandas as pd

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Model

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

from tensorflow.keras.preprocessing.text import Tokenizer

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

df = pd.read\_csv(r'/content/drive/MyDrive/general\_en\_ta 87k.csv')

df = df.iloc[:20000, 1:]

df.columns = ['English', 'Tamil']

df['Tamil'] = df['Tamil'].apply(lambda x: '<start> ' + x + ' <end>')

eng\_tokenizer = Tokenizer()

eng\_tokenizer.fit\_on\_texts(df['English'])

eng\_sequences = eng\_tokenizer.texts\_to\_sequences(df['English'])

tamil\_tokenizer = Tokenizer()

tamil\_tokenizer.fit\_on\_texts(df['Tamil'])

tamil\_sequences = tamil\_tokenizer.texts\_to\_sequences(df['Tamil'])

print(tamil\_tokenizer.word\_index)



num\_encoder\_tokens = len(eng\_tokenizer.word\_index) + 1

num\_decoder\_tokens = len(tamil\_tokenizer.word\_index) + 1

max\_encoder\_seq\_length = max([len(seq) for seq in eng\_sequences])

max\_decoder\_seq\_length = max([len(seq) for seq in tamil\_sequences])

encoder\_input\_data = pad\_sequences(eng\_sequences, maxlen=max\_encoder\_seq\_length, padding='post')

decoder\_input\_data = pad\_sequences(tamil\_sequences, maxlen=max\_decoder\_seq\_length, padding='post')

decoder\_target\_data = np.zeros\_like(decoder\_input\_data)

decoder\_target\_data[:, :-1] = decoder\_input\_data[:, 1:]

encoder\_input\_data = tf.keras.utils.to\_categorical(encoder\_input\_data, num\_encoder\_tokens)

decoder\_input\_data = tf.keras.utils.to\_categorical(decoder\_input\_data, num\_decoder\_tokens)

latent\_dim = 512

encoder\_inputs = Input(shape=(None, num\_encoder\_tokens))

encoder = LSTM(latent\_dim, return\_state=True, return\_sequences=True)

encoder\_outputs, state\_h, state\_c = encoder(encoder\_inputs)

encoder \_states = [state\_h, state\_c]

decoder\_inputs = Input(shape=(None, num\_decoder\_tokens))

decoder\_lstm = LSTM(latent\_dim, return\_sequences=True, return\_state=True)

decoder\_outputs, \_, \_ = decoder\_lstm(decoder\_inputs, initial\_state=encoder\_states)

decoder\_dense = Dense(num\_decoder\_tokens, activation='softmax')

decoder\_outputs = decoder\_dense(decoder\_outputs)

model = Model([encoder\_inputs, decoder\_inputs], decoder\_outputs)

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

model.fit([encoder\_input\_data, decoder\_input\_data], decoder\_target\_data,

          epochs=20, validation\_split=0.2)

model.save('English\_tamil\_translation\_model.h5')

encoder\_model = Model(encoder\_inputs, encoder\_states)

decoder\_state\_input\_h = Input(shape=(latent\_dim,))

decoder\_state\_input\_c = Input(shape=(latent\_dim,))

decoder\_states\_inputs = [decoder\_state\_input\_h, decoder\_state\_input\_c]

decoder\_outputs, state\_h, state\_c = decoder\_lstm(

     decoder\_inputs, initial\_state=decoder\_states\_inputs)

decoder\_states = [state\_h, state\_c]

decoder\_outputs = decoder\_dense(decoder\_outputs)

decoder\_model = Model(

    [decoder\_inputs] + decoder\_states\_inputs,

    [decoder\_outputs] + decoder\_states)

def decode\_sequence(input\_seq):

     states\_value = encoder\_model.predict(input\_seq)

     target\_seq = np.zeros((1, 1, num\_decoder\_tokens))

     target\_seq[0, 0, tamil\_tokenizer.word\_index['start']] = 1.0

     stop\_condition = False

     decoded\_sentence = ''

     while not stop\_condition:

         output\_tokens, h, c = decoder\_model.predict([target\_seq] + states\_value)

         sampled\_token\_index = np.argmax(output\_tokens[0, -1, :])

         sampled\_char = tamil\_tokenizer.index\_word.get(sampled\_token\_index, '')

         decoded\_sentence += ' ' + sampled\_char

         if sampled\_char == 'end' or len(decoded\_sentence) > max\_decoder\_seq\_length:

             stop\_condition = True

         target\_seq = np.zeros((1, 1, num\_decoder\_tokens))

         target\_seq[0, 0, sampled\_token\_index] = 1.0

         states\_value = [h, c]

     return decoded\_sentence

test\_sentence = "How are you?"

test\_sequence = eng\_tokenizer.texts\_to\_sequences([test\_sentence])

test\_sequence = pad\_sequences(test\_sequence, maxlen=max\_encoder\_seq\_length, padding='post')

test\_sequence = tf.keras.utils.to\_categorical(test\_sequence, num\_encoder\_tokens)

translated\_sentence = decode\_sequence(test\_sequence)

print(f'Translated sentence: {translated\_sentence}')

test\_sentence = "Seize him"

test\_sequence = eng\_tokenizer.texts\_to\_sequences([test\_sentence])

test\_sequence = pad\_sequences(test\_sequence, maxlen=max\_encoder\_seq\_length, padding='post')

test\_sequence = tf.keras.utils.to\_categorical(test\_sequence, num\_encoder\_tokens)

translated\_sentence = decode\_sequence(test\_sequence)

print(f'Translated sentence: {translated\_sentence}')

**8     IMAGE AUGMENTATION USING GANs**

import tensorflow as tf

from tensorflow.keras import layers

import numpy as np

import os

from PIL import Image

from tensorflow.keras import models

import matplotlib.pyplot as plt

image\_folder = '/content/drive/MyDrive/Gan\_image'

image\_size = (64, 64)

images = []

for filename in os.listdir(image\_folder):

    if filename.endswith(".jpeg") or filename.endswith(".png"):

        img = Image.open(os.path.join(image\_folder, filename))

        img = img.resize(image\_size).convert('L')

        img = np.array(img) / 255.0

        images.append(img)

images = np.array(images).reshape(-1, image\_size[0], image\_size[1], 1)

print("Shape of images:", images.shape )

def make\_generator\_model():

    model = models.Sequential()

    model.add(layers.Dense(8\*8\*256, use\_bias=False, input\_shape=(100,)))

    model.add(layers.BatchNormalization())

    model.add(layers.LeakyReLU())

    model.add(layers.Reshape((8, 8, 256)))

    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(2, 2), padding='same', use\_bias=False))

    model.add(layers.BatchNormalization())

    model.add(layers.LeakyReLU())

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

    model.add(layers.BatchNormalization())

    model.add(layers.LeakyReLU())

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

    return model

def make\_discriminator\_model():

    model = models.Sequential()

    model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same', input\_shape=[64, 64, 1]))

    model.add(layers.LeakyReLU())

    model.add(layers.Dropout(0.3))

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

    model.add(layers.LeakyReLU())

    model.add(layers.Dropout(0.3))

    model.add(layers.Flatten())

    model.add(layers.Dense(1))

    return model

generator = make\_generator\_model()

discriminator = make\_discriminator\_model()

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

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

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

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

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

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

    total\_loss = real\_loss + fake\_loss

    return total\_loss

def generator\_loss(fake\_output):

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

@tf.function

def train\_step(images):

    noise = tf.random.normal([BATCH\_SIZE, noise\_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\_of\_generator = gen\_tape.gradient(gen\_loss, generator.trainable\_variables)

    gradients\_of\_discriminator = disc\_tape.gradient(disc\_loss, discriminator.trainable\_variables)

    generator\_optimizer.apply\_gradients(zip(gradients\_of\_generator, generator.trainable\_variables))

    discriminator\_optimizer.apply\_gradients(zip(gradients\_of\_discriminator, discriminator.trainable\_variables))

def train(dataset, epochs):

    for epoch in range(epochs):

        for image\_batch in dataset:

            train\_step(image\_batch)

        if epoch % 400 == 0:

            generate\_and\_save\_images(generator, epoch + 1, seed)

def generate\_and\_save\_images(model, epoch, test\_input):

    predictions = model(test\_input, training=False)

    fig = plt.figure(figsize=(40, 40))

    for i in range(predictions.shape[0]):

        plt.subplot(4, 4, i + 1)

        plt.imshow(predictions[i, :, :, 0] \* 127.5 + 127.5, cmap='gray')

        plt.axis('off')

    plt.savefig(f'/content/drive/MyDrive/Gan\_image/epoch\_{epoch}.png')

    plt.show()

BATCH\_SIZE = 64

noise\_dim = 100

seed = tf.random.normal([16, noise\_dim])

train\_dataset = tf.data.Dataset.from\_tensor\_slices(images).shuffle(buffer\_size=1024).batch(BATCH\_SIZE)

train(train\_dataset, 10000)

|  |  |  |  |
| --- | --- | --- | --- |
|  | **4 A  LANGUAGE MODELLING USING RNN**  import pandas as pd  import numpy as np  import tensorflow as tf  from tensorflow.keras.preprocessing.text import Tokenizer  from tensorflow.keras.preprocessing.sequence import pad\_sequences  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Embedding, SimpleRNN, Dense  from tensorflow.keras.utils import to\_categorical  corpus = [      "Life is a mystery.",      "Go with the flow.",      "Just get over it.",      "Everything will work out in the end.",      "I need a break from everything."]  tokenizer = Tokenizer()  tokenizer.fit\_on\_texts(corpus)  total\_words = len(tokenizer.word\_index) + 1  input\_sequences=[]  for line in corpus:      token\_list = tokenizer.texts\_to\_sequences([line])[0]      for i in range(1, len(token\_list)):          n\_gram\_sequence = token\_list[:i+1]          input\_sequences.append(n\_gram\_sequence)  max\_sequence\_len = max([len(seq) for seq in input\_sequences])  input\_sequences = pad\_sequences(input\_sequences, maxlen = max\_sequence\_len, padding = 'pre')  X,y = input\_sequences[:, :-1], input\_sequences[:, -1]  y = to\_categorical(y, num\_classes = total\_words)  model = Sequential()  model.add(Embedding(total\_words, 100, input\_length = max\_sequence\_len-1))  model.add(SimpleRNN(150, return\_sequences = False))  model.add(Dense(total\_words, activation = 'softmax'))  model.compile(loss = 'categorical\_crossentropy', optimizer = 'adam', metrics = ['accuracy'])  history = model.fit(X, y, epochs = 100, verbose = 1)  def predict\_next\_word(model, tokenizer, text, max\_sequence\_len):      token\_list = tokenizer.texts\_to\_sequences([text])[0]      token\_list = pad\_sequences([token\_list], maxlen = max\_sequence\_len - 1, padding = 'pre')      predicted = model.predict(token\_list, verbose = 0)      predicted\_word\_index = np.argmax(predicted, axis = -1)[0]      for word, index in tokenizer.word\_index.items():          if index == predicted\_word\_index:              return word  seed\_text = "I need a"  next\_word = predict\_next\_word(model,tokenizer, seed\_text, max\_sequence\_len)  print(f"Next word after '{seed\_text}': {next\_word}")  seed\_text = "Life is"  next\_word = predict\_next\_word(model, tokenizer, seed\_text, max\_sequence\_len)  print(f"Next word after '{seed\_text}' : {next\_word} | **4 B LANGUAGE MODELLING USING RNN**  import pandas as pd import os import numpy as np  import tensorflow as tf  from tensorflow.keras.preprocessing.sequence import pad\_sequences  from tensorflow.keras.layers import Embedding, LSTM, Dense, Bidirectional  from tensorflow.keras.preprocessing.text import Tokenizer  from tensorflow.keras.models import Sequential  medium\_data = pd.read\_csv("Downloads/medium\_data.csv/medium\_data.csv")  medium\_data.head()  print("Number of records:", medium\_data.shape[0])  print("Number of fields:", medium\_data.shape[1])  medium\_data["title"]  medium\_data['title'] = medium\_data['title'].apply(lambda x: x.replace(u'\xa0', u' '))  medium\_data['title'] = medium\_data['title'].apply(lambda x: x.replace('\u200a', ' '))  tokenizer = Tokenizer(oov\_token = '<oov>')  tokenizer.fit\_on\_texts(medium\_data['title'])  total\_words = len(tokenizer.word\_index) + 1  print("Total number of words : ", total\_words)  input\_sequences = []  for line in medium\_data['title']:      token\_list = tokenizer.texts\_to\_sequences([line])[0]      for i in range(1, len(token\_list)):          n\_gram\_sequences = token\_list[:i+1]          input\_sequences.append(n\_gram\_sequences)  print("Total input sequences: " , len(input\_sequences))  max\_sequence\_len = max([len(x) for x in input\_sequences])  input\_sequences = np.array(pad\_sequences(input\_sequences, maxlen = max\_sequence\_len, padding = 'pre'))  input\_sequences[1]  xs, labels = input\_sequences[:,:-1], input\_sequences[:,-1]  ys = tf.keras.utils.to\_categorical(labels, num\_classes = total\_words) print(xs[5]) print(labels[5]) print(ys[5][14])  model = Sequential()  model.add(Embedding(total\_words, 100, input\_length = max\_sequence\_len - 1))  model.add(SimpleRNN(150))  model.add(Dense(total\_words, activation = 'softmax'))  model.compile(loss = 'categorical\_crossentropy', optimizer = 'adam', metrics = ['accuracy'])  history = model.fit(xs, ys, epochs = 1, verbose = 1)  print(model)  seed\_text = "A Beginner's Guide"  next\_words = 2  for \_ in range(next\_words):      token\_list = tokenizer.texts\_to\_sequences([seed\_text])[0]      token\_list = pad\_sequences([token\_list], maxlen = max\_sequence\_len - 1, padding = 'pre')      predicted = model.predict(token\_list, verbose = 0)      predicted\_word\_index = np.argmax(predicted, axis = -1)      output\_word = ""      for word, index in tokenizer.word\_index.items():          if index == predicted\_word\_index:              output\_word = word              break      seed\_text += " " + output\_word  print(seed\_text) | **8 IMAGEAUGMENTATION USING GANs**  from tensorflow.keras import layers  from PIL import Image  from tensorflow.keras import models plt np  image\_folder = '/content/drive/MyDrive/Gan\_image'  image\_size = (64, 64)  images = []  for filename in os.listdir(image\_folder):      if filename.endswith(".jpeg") or filename.endswith(".png"):          img = Image.open(os.path.join(image\_folder, filename))          img = img.resize(image\_size).convert('L')          img = np.array(img) / 255.0  images.append(img)  images = np.array(images).reshape(-1, image\_size[0], image\_size[1], 1)  print("Shape of images:", images.shape )  def make\_generator\_model():  model = models.Sequential()      model.add(layers.Dense(8\*8\*256, use\_bias=False, input\_shape=(100,))) model.add(layers.BatchNormalization())      model.add(layers.LeakyReLU())      model.add(layers.Reshape((8, 8, 256)))      model.add(layers.Conv2DTranspose(128, (5, 5), strides=(2, 2), padding='same', use\_bias=False))      model.add(layers.BatchNormalization())      model.add(layers.LeakyReLU())      model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use\_bias=False))      model.add(layers.BatchNormalization())      model.add(layers.LeakyReLU())      model.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same', use\_bias=False, activation='tanh'))      return model  def make\_discriminator\_model():      model = models.Sequential()      model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same', input\_shape=[64, 64, 1]))      model.add(layers.LeakyReLU())      model.add(layers.Dropout(0.3))      model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))      model.add(layers.LeakyReLU())  model.add(layers.Dropout(0.3))  model.add(layers.Flatten())  model.add(layers.Dense(1))  return model  generator = make\_generator\_model()  discriminator = make\_discriminator\_model()  generator\_optimizer = tf.keras.optimizers.Adam(1e-4)  discriminator\_optimizer = tf.keras.optimizers.Adam(1e-4)  cross\_entropy = tf.keras.losses.BinaryCrossentropy(from\_logits=True)  def discriminator\_loss(real\_output, fake\_output):      real\_loss = cross\_entropy(tf.ones\_like(real\_output), real\_output)      fake\_loss = cross\_entropy(tf.zeros\_like(fake\_output), fake\_output)      total\_loss = real\_loss + fake\_loss  return total\_loss  def generator\_loss(fake\_output):      return cross\_entropy(tf.ones\_like(fake\_output), fake\_output)  @tf.function |
|  | X\_input = word\_tokenizer.texts\_to\_sequences([sentence])  X\_input\_padded = pad\_sequences(X\_input, maxlen=maxlen, padding='post')  predictions = model.predict(X\_input\_padded)  predicted\_indices = np.argmax(predictions, axis=-1)  predicted\_tags = []  for idx in predicted\_indices[0]:      if idx != 0:          predicted\_tags.append(tag\_tokenizer.index\_word.get(idx, "UNK"))  print("Input sentence: ", sentence)  print("Predicted POS Tags: ", predicted\_tags) | decoder\_state\_input\_c = Input(shape=(latent\_dim,))  decoder\_states\_inputs = [decoder\_state\_input\_h, decoder\_state\_input\_c]  decoder\_outputs, state\_h, state\_c = decoder\_lstm(      decoder\_inputs, initial\_state=decoder\_states\_inputs)  decoder\_states = [state\_h, state\_c]  decoder\_outputs = decoder\_dense(decoder\_outputs)  decoder\_model = Model(      [decoder\_inputs] + decoder\_states\_inputs,      [decoder\_outputs] + decoder\_states)  def decode\_sequence(input\_seq):      states\_value = encoder\_model.predict(input\_seq)      target\_seq = np.zeros((1, 1, num\_decoder\_tokens))      target\_seq[0, 0, tamil\_tokenizer.word\_index['start']] = 1.0      stop\_condition = False      decoded\_sentence = ''      while not stop\_condition:          output\_tokens, h, c = decoder\_model.predict([target\_seq] + states\_value)          sampled\_token\_index = np.argmax(output\_tokens[0, -1, :])          sampled\_char = tamil\_tokenizer.index\_word.get(sampled\_token\_index, '')          decoded\_sentence += ' ' + sampled\_char          if sampled\_char == 'end' or len(decoded\_sentence) > max\_decoder\_seq\_length:              stop\_condition = True          target\_seq = np.zeros((1, 1, num\_decoder\_tokens))          target\_seq[0, 0, sampled\_token\_index] = 1.0          states\_value = [h, c]      return decoded\_sentence  test\_sentence = "How are you?"  test\_sequence = eng\_tokenizer.texts\_to\_sequences([test\_sentence])  test\_sequence = pad\_sequences(test\_sequence, maxlen=max\_encoder\_seq\_length, padding='post')  test\_sequence = tf.keras.utils.to\_categorical(test\_sequence, num\_encoder\_tokens)  translated\_sentence = decode\_sequence(test\_sequence)  print(f'Translated sentence: {translated\_sentence}') | X\_sequences = tokenizer.texts\_to\_sequences(X\_preprocessed)  X\_padded = pad\_sequences(X\_sequences, maxlen=100)  validation\_sequences = tokenizer.texts\_to\_sequences(validation\_X\_preprocessed)  validation\_padded = pad\_sequences(validation\_sequences, maxlen=100)  X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X\_padded, Y\_encoded, test\_size=0.2, random\_state=42)  model = Sequential()  model.add(Embedding(input\_dim=5000, output\_dim=128, input\_length=100))  model.add(LSTM(128, dropout=0.1, return\_sequences=True))  model.add(LSTM(128, dropout=0.1))  model.add(Dense(4, activation='softmax'))  model.compile(loss='sparse\_categorical\_crossentropy',                optimizer='adam',                metrics=['accuracy'])  early\_stopping = EarlyStopping(monitor='val\_loss', patience=3, restore\_best\_weights=True)  history = model.fit(X\_padded, Y\_encoded,                      epochs=10,                      batch\_size=64,                      validation\_data=(validation\_padded, validation\_Y\_encoded),                      callbacks=[early\_stopping])  loss, accuracy = model.evaluate(X\_test, Y\_test)  print(f'Test Loss: {loss}')  print(f'Test Accuracy: {accuracy}')  def predict(text):      text\_preprocessed = preprocess\_text(text)      text\_sequence = tokenizer.texts\_to\_sequences([text\_preprocessed])      text\_padded = pad\_sequences(text\_sequence, maxlen=100)      res = model.predict(text\_padded)      predicted\_class = res.argmax(axis=1)[0]      predicted\_label = encoder.inverse\_transform([predicted\_class])[0]      confidence = res[0][predicted\_class]      return predicted\_label, confidence  iface = gr.Interface(      fn=predict,      inputs=gr.Textbox(lines=2, placeholder="Type your text here..."),      outputs=["text", "number"],      title="Text Classification Model",      description="Enter a sentence to get its predicted classification and confidence score."  )  iface.launch() |
| def train\_step(images):      noise = tf.random.normal([BATCH\_SIZE, noise\_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\_of\_generator = gen\_tape.gradient(gen\_loss, generator.trainable\_variables)      gradients\_of\_discriminator = disc\_tape.gradient(disc\_loss, discriminator.trainable\_variables)      generator\_optimizer.apply\_gradients(zip(gradients\_of\_generator, generator.trainable\_variables))      discriminator\_optimizer.apply\_gradients(zip(gradients\_of\_discriminator, discriminator.trainable\_variables))  def train(dataset, epochs):  for epoch in range(epochs):          for image\_batch in dataset:  train\_step(image\_batch)          if epoch % 400 == 0:              generate\_and\_save\_images(generator, epoch + 1, seed)  def generate\_and\_save\_images(model, epoch, test\_input):      predictions = model(test\_input, training=False)      fig = plt.figure(figsize=(40, 40))      for i in range(predictions.shape[0]):  plt.subplot(4, 4, i + 1)          plt.imshow(predictions[i, :, :, 0] \* 127.5 + 127.5, cmap='gray')  plt.axis('off')      plt.savefig(f'/content/drive/MyDrive/Gan\_image/epoch\_{epoch}.png')  plt.show()  BATCH\_SIZE = 64  noise\_dim = 100  seed = tf.random.normal([16, noise\_dim])  train\_dataset = tf.data.Dataset.from\_tensor\_slices(images).shuffle(buffer\_size=1024).batch(BATCH\_SIZE)  train(train\_dataset, 10000) | **5 A SENTIMENT ANALYSIS USING LSTM**  import numpy as np  import pandas as pd  from tensorflow.keras.preprocessing.text import Tokenizer  from tensorflow.keras.preprocessing.sequence import pad\_sequences  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import LabelEncoder  from sklearn.metrics import classification\_report, accuracy\_score  data = pd.read\_csv('/content/drive/MyDrive/IMDB Dataset.csv')  texts = data['review'].values  labels = data['sentiment'].values  label\_encoder = LabelEncoder()  labels = label\_encoder.fit\_transform(labels)  max\_vocab\_size = 5000  max\_sequence\_length = 100  tokenizer = Tokenizer(num\_words=max\_vocab\_size, oov\_token="<OOV>")  tokenizer.fit\_on\_texts(texts)  word\_index = tokenizer.word\_index  sequences = tokenizer.texts\_to\_sequences(texts)  padded\_sequences = pad\_sequences(sequences, maxlen=max\_sequence\_length, padding='post')  X\_train, X\_test, y\_train, y\_test = train\_test\_split(padded\_sequences, labels, test\_size=0.2, random\_state=42)  model = Sequential()  model.add(Embedding(input\_dim=max\_vocab\_size, output\_dim=64, input\_length=max\_sequence\_length))  model.add(LSTM(64, return\_sequences=False))  model.add(Dropout(0.5))  model.add(Dense(32, activation='relu'))  model.add(Dense(len(set(labels)), activation=sigmoid))  model.compile(loss='sparse\_categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])  model.fit(X\_train, y\_train, epochs=7, validation\_data=(X\_test, y\_test), batch\_size=32)  loss, accuracy = model.evaluate(X\_test, y\_test)  print(f'Test Accuracy: {accuracy}')  **def predict\_emotion(input\_text):**      input\_seq = tokenizer.texts\_to\_sequences([input\_text])      input\_padded = pad\_sequences(input\_seq, maxlen=max\_sequence\_length, padding='post')      prediction = model.predict(input\_padded)      predicted\_class = np.argmax(prediction, axis=1)      emotion\_label = label\_encoder.inverse\_transform(predicted\_class)      return emotion\_label[0]  **def gradio\_predict(text):**      return predict\_emotion(text)  interface = gr.Interface(fn=gradio\_predict,                           inputs="text",                           outputs="text",                           title="Emotion Prediction",                           description="Enter a sentence and the model will predict the emotion.")  interface.launch() | **3 FACE RECOGNITION USING CNN**  from tf.keras.preprocessing.image import ImageDataGenerator import os gr  data\_dir='C:/Users/VarshaK/Downloads/lfw\_facedata/lfw-deepfunneled/lfw-deepfunneled' datagen=ImageDataGenerator(rescale=1./255,  validation\_split=0.2)  train\_data=datagen.flow\_from\_directory(  directory=data\_dir,  target\_size=(128,128), batch\_size=32,  class\_mode='categorical',  subset='training' )  val\_data=datagen.flow\_from\_directory(  directory=data\_dir, target\_size=(128,128),  batch\_size=32,  class\_mode='categorical',  subset='validation' )  pip install pillow  images,labels=next(train\_data)  plt.imshow(images[3])  class\_indices=train\_data.class\_indices  index\_to\_class={v: k for k,v in class\_indices.items()}  predicted\_label=3 person\_name=index\_to\_class[predicted\_label]  print('predicted person:', person\_name)  model = models.Sequential()  model.add(layers.Conv2D(32,(3,3), activation ='relu', input\_shape=(128,128,3))) model.add(layers.MaxPooling2D(2,2))  model.add(layers.Conv2D(64, (3,3), activation='relu'))  model.add(layers.MaxPooling2D(2,2))  model.add(layers.Conv2D(128, (3,3), activation ='relu'))  model.add(layers.MaxPooling2D((2,2)))  model.add(layers.Flatten())  model.add(layers.Dense(444, activation = 'relu'))  model.add(layers.Dropout(0,2))  model.add(layers.Dense(5749, activation='softmax'))  model.summary()  model.compile(optimizer = 'adam', loss = 'categorical\_crossentropy', metrics =['accuracy']) model.fit(train\_data, validation\_data = val\_data, epochs = 20)  import cv2 import numpy as np  import matplotlib.pyplot as plt  img\_path = "C:/Users/Varsha  K/Downloads/lfw\_facedata/lfw-deepfunneled/lfw-deepfunneled/Aaron\_Guiel/Aaron\_Guie l\_0001.jpg"  img = cv2.imread(img\_path, cv2.IMREAD\_COLOR)  if img is not None:  img\_resized = cv2.resize(img, (128, 128))  img\_resized\_rgb = cv2.cvtColor(img\_resized, cv2.COLOR\_BGR2RGB) img\_normalized=img\_resized /255  plt.imshow(img\_normalized) plt.axis('off')  plt.show()  img\_expanded = np.expand\_dims(img\_normalized, axis=0) result = model.predict(img\_expanded)  print("Prediction result:", result)  else: print("Error: Image not found or could not be loaded.")  predicted\_label = np.argmax(result)  predicted\_label  index\_to\_class[predicted\_label]  def preprocess\_image(image):  image = cv2.cvtColor(image, cv2.IMREAD\_COLOR)  #image = cv2.imread(image, cv2.IMREAD\_COLOR)  image = cv2.resize(image, (128,128))  image = image/255. image =np.expand\_dims(image, axis=0)  return image  def predict(image):  image = np.array(image)  image = preprocess\_image(image)  prediction = model.predict(image)  predicted\_label = np.argmax(result)  return index\_to\_class[predicted\_label]  interface = gr.Interface(fn=predict, inputs=gr.Image(type="numpy"),  outputs=gr.Textbox(),  live=True, interface.launch(share=True) |  |
| **5 B SENTIMENT ANALYSIS USING LSTM**  import re from nltk.corpus import stopwords  from nltk.tokenize import word\_tokenize  from nltk.stem import WordNetLemmatizer  from sklearn.preprocessing import LabelEncoder  from sklearn.model\_selection import train\_test\_split  from tensorflow.keras.preprocessing.text import Tokenizer  from tensorflow.keras.preprocessing.sequence import pad\_sequences  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Embedding, LSTM,Dense  from tensorflow.keras.callbacks import EarlyStopping  data = pd.read\_csv("/content/drive/MyDrive/twitter\_training.csv")  validation\_data = pd.read\_csv("/content/drive/MyDrive/twitter\_validation.csv")  X = data.iloc[:,3:] X.head() X.columns = ['text']  X['text'] = X['text'].fillna('') y = data.iloc[:,2:3] y.head()  y.columns = ['label'] nltk.download('stopwords')  nltk.download('punkt') nltk.download('wordnet')  from nltk.corpus import stopwords  from nltk.tokenize import word\_tokenize  from nltk.stem import WordNetLemmatizer  lemmatizer = WordNetLemmatizer()  stop\_words = set(stopwords.words('english'))  def preprocess\_text(text):      if isinstance(text, str):          text = text.lower()          text = re.sub(r'[^a-zA-Z\s]', '', text)          tokens = word\_tokenize(text)          tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop\_words]          return ' '.join(tokens)      else:          return ''  X\_preprocessed = [preprocess\_text(text) for text in X['text']]  tokenizer = Tokenizer(num\_words=5000)  tokenizer.fit\_on\_texts(X\_preprocessed)  X\_sequences = tokenizer.texts\_to\_sequences(X\_preprocessed)  X\_padded = pad\_sequences(X\_sequences, maxlen=100)  y['label'].unique() encoder = LabelEncoder()  Y\_encoded = encoder.fit\_transform(y) print(Y\_encoded)  label\_mapping = dict(zip(encoder.classes\_, encoder.transform(encoder.classes\_)))  print(label\_mapping)  reverse\_label\_mapping = {v: k for k, v in label\_mapping.items()} reverse\_label\_mapping  X\_padded.shape Y\_encoded.shape  validation\_X = validation\_data.iloc[:, 3:]  validation\_X.columns = ['text']  validation\_X['text'] = validation\_X['text'].fillna('')  validation\_X\_preprocessed = [preprocess\_text(text) for text in validation\_X['text']] encoder = LabelEncoder()  Y\_encoded = encoder.fit\_transform(y)  validation\_y = validation\_data.iloc[:, 2:3]  validation\_y.columns = ['label']  validation\_Y\_encoded = encoder.transform(validation\_y)  tokenizer = Tokenizer(num\_words=5000)  tokenizer.fit\_on\_texts(X\_preprocessed) | **7 MACHINE TRANSLATION USING ENCODER-DECODER MODEL**  from tensorflow.keras.models import Model  from tensorflow.keras.layers import Input, LSTM, Dense  from tensorflow.keras.preprocessing.text import Tokenizer  from tensorflow.keras.preprocessing.sequence import pad\_sequences  df = pd.read\_csv(r'/content/drive/MyDrive/general\_en\_ta 87k.csv') df = df.iloc[:20000, 1:]  df.columns = ['English', 'Tamil']  df['Tamil'] = df['Tamil'].apply(lambda x: '<start> ' + x + ' <end>')  eng\_tokenizer = Tokenizer()  eng\_tokenizer.fit\_on\_texts(df['English'])  eng\_sequences = eng\_tokenizer.texts\_to\_sequences(df['English'])  tamil\_tokenizer = Tokenizer()  tamil\_tokenizer.fit\_on\_texts(df['Tamil'])  tamil\_sequences = tamil\_tokenizer.texts\_to\_sequences(df['Tamil'])  print(tamil\_tokenizer.word\_index)  num\_encoder\_tokens = len(eng\_tokenizer.word\_index) + 1  num\_decoder\_tokens = len(tamil\_tokenizer.word\_index) + 1  max\_encoder\_seq\_length = max([len(seq) for seq in eng\_sequences])  max\_decoder\_seq\_length = max([len(seq) for seq in tamil\_sequences])  encoder\_input\_data = pad\_sequences(eng\_sequences, maxlen=max\_encoder\_seq\_length, padding='post')  decoder\_input\_data = pad\_sequences(tamil\_sequences, maxlen=max\_decoder\_seq\_length, padding='post')  decoder\_target\_data = np.zeros\_like(decoder\_input\_data)  decoder\_target\_data[:, :-1] = decoder\_input\_data[:, 1:]  encoder\_input\_data = tf.keras.utils.to\_categorical(encoder\_input\_data, num\_encoder\_tokens)  decoder\_input\_data = tf.keras.utils.to\_categorical(decoder\_input\_data, num\_decoder\_tokens)  latent\_dim = 512  encoder\_inputs = Input(shape=(None, num\_encoder\_tokens))  encoder = LSTM(latent\_dim, return\_state=True, return\_sequences=True)  encoder\_outputs, state\_h, state\_c = encoder(encoder\_inputs)  encoder\_states = [state\_h, state\_c]  decoder\_inputs = Input(shape=(None, num\_decoder\_tokens))  decoder\_lstm = LSTM(latent\_dim, return\_sequences=True, return\_state=True)  decoder\_outputs, \_, \_ = decoder\_lstm(decoder\_inputs, initial\_state=encoder\_states)  decoder\_dense = Dense(num\_decoder\_tokens, activation='softmax')  decoder\_outputs = decoder\_dense(decoder\_outputs)  model = Model([encoder\_inputs, decoder\_inputs], decoder\_outputs)  model.compile(optimizer='rmsprop', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])  model.fit([encoder\_input\_data, decoder\_input\_data], decoder\_target\_data,            epochs=20, validation\_split=0.2)  model.save('English\_tamil\_translation\_model.h5')  encoder\_model = Model(encoder\_inputs, encoder\_states)  decoder\_state\_input\_h = Input(shape=(latent\_dim,)) | **6 PARTS OF SPEECH TAGGING USING SEQUENCE TO SEQUENCE**  import numpy as np  from nltk.corpus import brown,treebank,conll2000  from keras.preprocessing.sequence import pad\_sequences  from keras.models import Sequential,Model  from keras.layers import Embedding,Dense,Input,LSTM,GRU,Bidirectional,SimpleRNN ,RNN  from tensorflow.keras.preprocessing.text import Tokenizer  from sklearn.model\_selection import train\_test\_split  from tensorflow.keras.utils import to\_categorical  nltk.download('treebank')  nltk.download('brown')  nltk.download('conll2000')  nltk.download('universal\_tagset')  treebank\_corpus=treebank.tagged\_sents(tagset='universal')  brown\_corpus=brown.tagged\_sents(tagset='universal')  conll\_corpus=conll2000.tagged\_sents(tagset='universal')  tagged\_sentences=treebank\_corpus+brown\_corpus+conll\_corpus X=[] Y=[]  for sentence in tagged\_sentences:      X\_sentence=[] Y\_sentence=[]      for entity in sentence:          X\_sentence.append(entity[0])          Y\_sentence.append(entity[1])      X.append(X\_sentence) Y.append(Y\_sentence)  word\_tokenizer=Tokenizer(oov\_token='<OOV>')  word\_tokenizer.fit\_on\_texts(Y)  tag\_tokenizer=Tokenizer(lower=False)  tag\_tokenizer.fit\_on\_texts(Y)  X\_sequences=word\_tokenizer.texts\_to\_sequences(X)  Y\_sequences=tag\_tokenizer.texts\_to\_sequences(Y)  maxlen=max([len(seq) for seq in X\_sequences])  X\_padded = pad\_sequences(X\_sequences, maxlen=maxlen, padding='post')  Y\_padded = pad\_sequences(Y\_sequences, maxlen=maxlen, padding='post')  num\_tags=len(tag\_tokenizer.word\_index)+1  Y\_onehot=[to\_categorical(seq,num\_classes=num\_tags) for seq in Y\_padded]  Y\_onehot = np.array(Y\_onehot)  X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X\_padded, Y\_onehot, test\_size=0.2, random\_state=42)  vocab\_size = len(word\_tokenizer.word\_index) + 1  embedding\_dim = 128  hidden\_size = 256  num\_tags = len(tag\_tokenizer.word\_index) + 1  input\_seq = Input(shape=(maxlen,))  embedding\_layer = Embedding(input\_dim=vocab\_size, output\_dim=embedding\_dim, input\_length=maxlen)(input\_seq)  encoder\_output = Bidirectional(LSTM(hidden\_size, return\_sequences=True))(embedding\_layer)  output\_layer = Dense(num\_tags, activation='softmax')(encoder\_output)  model = Model(inputs=input\_seq, outputs=output\_layer)  model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])  model.fit(X\_train, Y\_train, epochs=10, batch\_size=32, validation\_data=(X\_test, Y\_test))  sentence = ["The", "dog", "eats", "snacks"] |  |