**WEEK-1**

**AIM:** To create and explore NumPy arrays (tensors) of different dimensions — scalar (0D), 1D tensor (vector), 2D tensor (matrix), and higher-order tensors (3D, 4D, 5D). The program also prints their shapes, number of dimensions, and values.

**DESCRIPTION:**

**Importing NumPy:**

1. The program begins by importing the NumPy library as np, which is used for numerical operations and handling arrays of multiple dimensions.

**Creating a Scalar (0D Tensor):**

1. A scalar value 7 is converted into a NumPy array using np.array(7).
2. It is a 0-dimensional array.
3. The shape of a scalar is (), indicating no dimensions.
4. The number of dimensions (ndim) is 0.

**Creating a 1D Tensor (Vector):**

1. A one-dimensional array [1, 2, 3] is created using np.array([1, 2, 3]).
2. It is a 1D array (also called a vector).
3. The shape is (3,), indicating 3 elements in a single row.
4. The number of dimensions (ndim) is 1.

**Creating a 2D Tensor (Matrix):**

1. A two-dimensional array [[1, 2, 3], [4, 5, 6]] is created using np.array.
2. This is a 2D array (also called a matrix).
3. The shape is (2, 3), meaning 2 rows and 3 columns.
4. The number of dimensions (ndim) is 2.

**Creating a 3D Tensor:**

1. A three-dimensional array [[[1, 2], [3, 4]], [[5, 6], [7, 8]]] is created using np.array.
2. This is a 3D tensor (array with 3 axes).
3. The shape is (2, 2, 2), representing 2 blocks of 2×2 matrices.
4. The number of dimensions (ndim) is 3.

**Creating a 4D Tensor:**

1. A four-dimensional array is created with 2 sets of 2×2×2 tensors using np.array.
2. This is a 4D tensor (array with 4 axes).
3. The shape is (2, 2, 2, 2).
4. The number of dimensions (ndim) is 4.

**Creating a 5D Tensor:**

1. A five-dimensional array is created with 2 groups of 2×2×2×2 tensors using np.array.
2. This is a 5D tensor (array with 5 axes).
3. The shape is (2, 2, 2, 2, 2).
4. The number of dimensions (ndim) is 5.

**Printing Shapes and Dimensions:**

1. For each array, the program prints its shape using .shape.
2. The number of dimensions is displayed using .ndim.

**PROGRAM:**

import numpy as np

# Scalar (0D Tensor)

scalar = np.array(7)

print("Scalar Shape:", scalar.shape)

print("Scalar Dimensions:", scalar.ndim)

# 1D Tensor (Vector)

tensor1d = np.array([1, 2, 3])

print("1D Tensor Shape:", tensor1d.shape)

print("1D Tensor Dimensions:", tensor1d.ndim)

# 2D Tensor (Matrix)

tensor2d = np.array([[1, 2, 3], [4, 5, 6]])

print("2D Tensor Shape:", tensor2d.shape)

print("2D Tensor Dimensions:", tensor2d.ndim)

# 3D Tensor

tensor3d = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

print("3D Tensor Shape:", tensor3d.shape)

print("3D Tensor Dimensions:", tensor3d.ndim)

# 4D Tensor

tensor4d = np.array([

[

[[1, 2], [3, 4]],

[[5, 6], [7, 8]]

],

[

[[9, 10], [11, 12]],

[[13, 14], [15, 16]]

]

])

print("4D Tensor Shape:", tensor4d.shape)

print("4D Tensor Dimensions:", tensor4d.ndim)

# 5D Tensor

tensor5d = np.array([

[

[

[[1, 2], [3, 4]],

[[5, 6], [7, 8]]

],

[

[[9, 10], [11, 12]],

[[13, 14], [15, 16]]

]

],

[

[

[[17, 18], [19, 20]],

[[21, 22], [23, 24]]

],

[

[[25, 26], [27, 28]],

[[29, 30], [31, 32]]

]

]

])

print("5D Tensor Shape:", tensor5d.shape)

print("5D Tensor Dimensions:", tensor5d.ndim)

# Print All Array Values

print("\n--- Array Values ---")

print("Scalar (0D):", scalar)

print("1D Tensor:\n", tensor1d)

print("2D Tensor:\n", tensor2d)

print("3D Tensor:\n", tensor3d)

print("4D Tensor:\n", tensor4d)

print("5D Tensor:\n", tensor5d)

**OUTPUT:**

Scalar Shape: ()

Scalar Dimensions: 0

1D Tensor Shape: (3,)

1D Tensor Dimensions: 1

2D Tensor Shape: (2, 3)

2D Tensor Dimensions: 2

3D Tensor Shape: (2, 2, 2)

3D Tensor Dimensions: 3

4D Tensor Shape: (2, 2, 2, 2)

4D Tensor Dimensions: 4

5D Tensor Shape: (2, 2, 2, 2, 2)

5D Tensor Dimensions: 5

--- Array Values ---

Scalar (0D): 7

1D Tensor:

[1 2 3]

2D Tensor:

[[1 2 3]

[4 5 6]]

3D Tensor:

[[[1 2]

[3 4]]

[[5 6]

[7 8]]]

4D Tensor:

[[[[ 1 2]

[ 3 4]]

[[ 5 6]

[ 7 8]]]

[[[ 9 10]

[11 12]]

[[13 14]

[15 16]]]]

5D Tensor:

[[[[[ 1 2]

[ 3 4]]

[[ 5 6]

[ 7 8]]]

[[[ 9 10]

[11 12]]

[[13 14]

[15 16]]]]

[[[[17 18]

[19 20]]

[[21 22]

[23 24]]]

[[[25 26]

[27 28]]

[[29 30]

[31 32]]]]]

**WEEK-2**

**AIM**: Implement multi-layer perceptron algorithm for MNIST Hand written Digit Classification.

**DESCRIPTION:**

**Importing Required Libraries:**

1. tensorflow is imported for building and training the neural network.
2. matplotlib.pyplot is imported for visualizing accuracy trends.

**Loading the MNIST Dataset:**

1. The MNIST dataset is loaded using tf.keras.datasets.mnist.load\_data().
2. It contains 60,000 training and 10,000 testing grayscale images of digits (0–9).

**Preprocessing the Data:**

1. Pixel values are normalized by dividing by 255.0 to bring them into the [0, 1] range.
2. Labels are converted to one-hot encoding using tf.keras.utils.to\_categorical().

**Creating the Neural Network Model:**

1. A Sequential model is defined using tf.keras.Sequential().
2. Layers used:
   1. Flatten(input\_shape=(28, 28)): Converts 2D input into a 1D vector.
   2. Dense(128, activation='relu'): First hidden layer with 128 neurons.
   3. Dense(64, activation='relu'): Second hidden layer with 64 neurons.
   4. Dense(10, activation='softmax'): Output layer for 10 digit classes.

**The model is compiled with:**

1. Optimizer: adam
2. Loss function: categorical\_crossentropy
3. Metric: accuracy

**Training the Model:**

1. The model is trained using .fit() for 10 epochs.
2. 10% of training data is used as validation set.

**Evaluating the Model:**

1. The model is evaluated on the test set using .evaluate().
2. The test accuracy is printed.

**Visualizing Accuracy:**

1. A line graph is plotted showing both training accuracy and validation accuracy over each epoch using matplotlib.pyplot.

**PROGRAM:**

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense,Flatten

from tensorflow.keras.datasets import mnist

from tensorflow.keras.utils import to\_categorical

import matplotlib.pyplot as plt

# Load The MNIST Data Set

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

# Preprocess data

x\_train=x\_train.astype('float32')/255.0 # normalize to[0,1]

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

y\_train=to\_categorical(y\_train,10) #one-hot encode

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

# Build The Sequential Model

model=Sequential([

Flatten(input\_shape=(28,28)), # Flatten 28\*28 image

Dense(128,activation='relu'), # 1st Hidden layer

Dense(64,activation='relu'), # 2nd Hidden layer

Dense(10,activation='softmax')

]) #output layer(10 classes )

# Compile the model

model.compile(optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy'])

# Train The model

history=model.fit(x\_train,y\_train,

epochs=10,

batch\_size=128,

validation\_split=0.1,

verbose=2)

# Evaluate on test data

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

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

# Plot Training History

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

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

plt.axhline(y=test\_accuracy, color='green', linestyle='--', label=f'Test Accuracy: {test\_accuracy:.4f}')

plt.title('MLP MNIST Classification Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

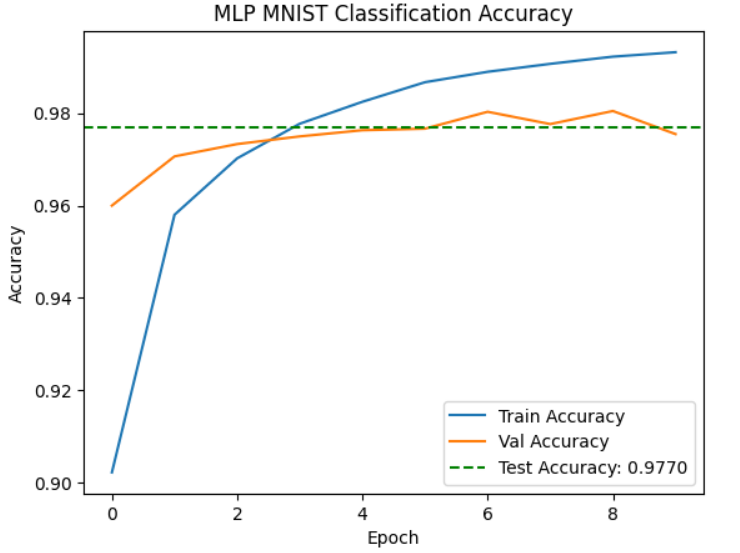
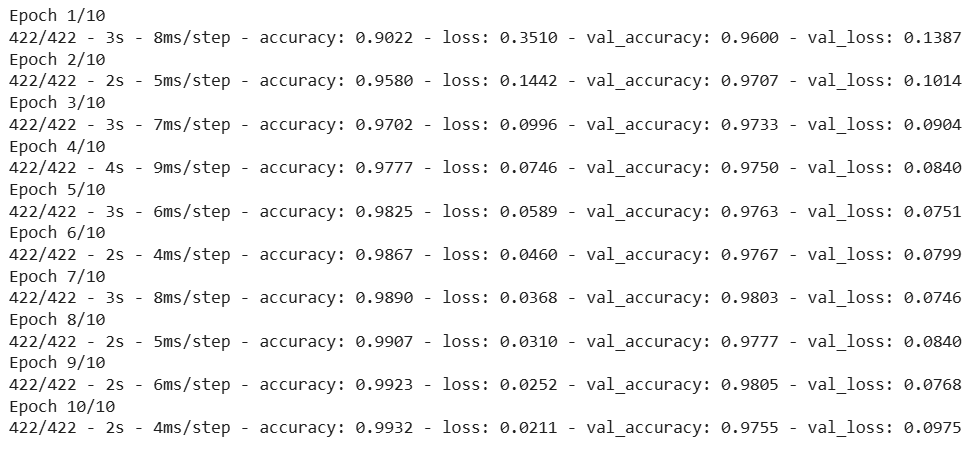
plt.legend()

plt.show()

**OUTPUT:**Test Accuracy: 0.9770







**WEEK** -3

**AIM:** To design a neural network for classifying movie reviews (Binary Classification)

**DESCRIPTION:**

**1.Loading IMDB Dataset:**

1. The dataset is loaded from keras.datasets.imdb with num\_words=1000, meaning only the top 1,000 most frequent words are considered.
2. train\_data and test\_data contain sequences of word indices representing movie reviews.
3. train\_labels and test\_labels are binary (0 = negative, 1 = positive reviews).

**2. Vectorizing Sequences:**

1. A custom vectorize\_sequences() function is defined to convert sequences of integers into binary bag-of-words vectors.
2. Each review becomes a 10,000-dimensional vector, where each index indicates the presence (1) or absence (0) of a word.
3. The same function is applied to both training and testing data.

**3.Preparing Labels:**

1. train\_labels and test\_labels are converted to NumPy float32 arrays for compatibility with the model.

**4. Model Architecture:**

1. A **Sequential model** is defined with:
   1. Two hidden layers: Dense(16, activation='relu')
   2. One output layer: Dense(1, activation='sigmoid') for binary classification.

**5. Compiling the Model:**

1. **Optimizer:** adam
2. **Loss function:** binary\_crossentropy (used for binary classification)
3. **Metric:** accuracy

**6. Training the Model:**

1. The model is trained for **10 epochs** with **batch size 512**.
2. **20% of training data** is used as validation data.

**7.Evaluating on Test Data:**

1. The trained model is evaluated on the test dataset.
2. Test loss and accuracy are printed.

**8.Visualizing Accuracy:**

1. A **line graph** is plotted showing:
   1. **Training accuracy** per epoch
   2. **Validation accuracy** per epoch
   3. A horizontal line indicating the final **test accuracy**

**PROGRAM:**

import numpy as np

import tensorflow as tf

from tensorflow.keras.datasets import imdb

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

import matplotlib.pyplot as plt

(train\_data,train\_labels),(test\_data,test\_labels)=imdb.load\_data(num\_words=1000)

import numpy as np

def vectorize\_sequences(sequences,dimension=10000):

results=np.zeros((len(sequences),dimension))

for i,sequence in enumerate(sequences):

results[i,sequence]=1

return results

# Vectorixw Training Dara

X\_train =vectorize\_sequences(train\_data)

# Vectorize Testing Data

X\_test=vectorize\_sequences(test\_data)

X\_train[0]

X\_train.shape

y\_train=np.asarray(train\_labels).astype('float32')

y\_test=np.asarray(test\_labels).astype('float32')

model=Sequential([

Dense(16,activation='relu',input\_shape=(10000,)),

Dense(16,activation='relu'),

Dense(1,activation='sigmoid')

])

model.compile(optimizer='adam',

loss='binary\_crossentropy',

metrics=['accuracy'])

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

epochs=10,

batch\_size=512,

validation\_split=0.2,)

# Evaluate on test data

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

print(f"Test Loss: {test\_loss:.4f}")

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

# Plot training & validation accuracy

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

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

plt.axhline(y=test\_accuracy, color='green', linestyle='--', label=f'Test Accuracy: {test\_accuracy:.4f}')

plt.title('Model Accuracy Over Epochs')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

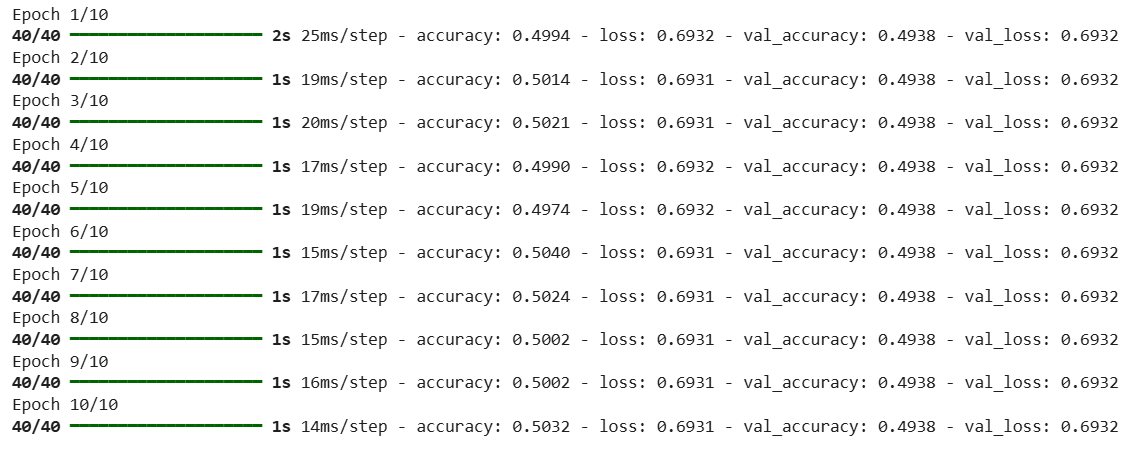
plt.grid(True)

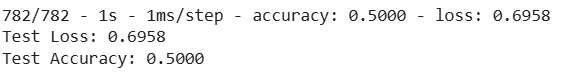
plt.show()

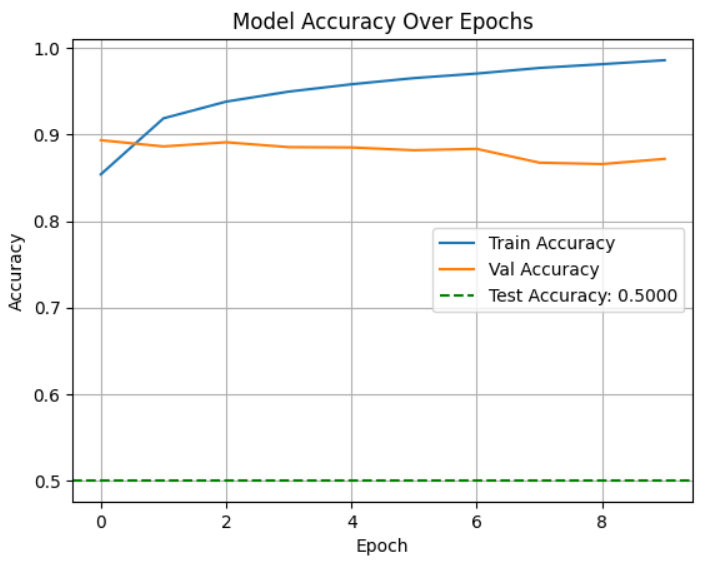
**OUTPUT:**

****

****

****





model.save("imdb\_sentiment\_model.h5")



from tensorflow.keras.models import load\_model

# load the model

loaded\_model =load\_model("imdb\_sentiment\_model.h5")



word\_index=imdb.get\_word\_index()

def encode\_review(text,word\_index,max\_words=10000):

  tokens=text.lower().split()

  encoded=[word\_index.get(word,0) for word in tokens]

  return encoded

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

def preprocess\_text(text):

  encoded=encode\_review(text,word\_index)

  vectorized=np.zeros((1,10000))

  for index in encoded:

    if index<10000:

      vectorized[0,index]=1

    return vectorized

**# Example 1 prediction**

sample\_text="This movie was fantastic and full of emotional depth"

x\_input=preprocess\_text(sample\_text)

prediction=loaded\_model.predict(x\_input)

**# Interpret Prediction**

label="Positive" if prediction>=0.5 else "Negative"

print(f"Sentiment :{label} (Confidence :{prediction[0][0]:.4f})")

**1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** 38ms/step

Sentiment :Positive (Confidence :0.5706)

**# Example 2 prediction**

sample\_text="Top-notch music and an emotional climax that leaves you in awe!"

x\_input=preprocess\_text(sample\_text)

prediction=loaded\_model.predict(x\_input)

**# Interpret Prediction**

label="Positive" if prediction>=0.5 else "Negative"

print(f"Sentiment :{label} (Confidence :{prediction[0][0]:.4f})")

**1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** 90ms/step

Sentiment :Positive (Confidence :0.5336)

**# Example 3 prediction**

sample\_text="I Kept checking my watch waiting for it to end!"

x\_input=preprocess\_text(sample\_text)

prediction=loaded\_model.predict(x\_input)

**# Interpret Prediction**

label="Positive" if prediction>=0.5 else "Negative"

print(f"Sentiment :{label} (Confidence :{prediction[0][0]:.4f})")

**1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** 56ms/step

Sentiment :Negative (Confidence :0.4108)

**# Example 4 prediction**

sample\_text="Some parts were really engaging,but others were painfully slow."

x\_input=preprocess\_text(sample\_text)

prediction=loaded\_model.predict(x\_input)

**# Interpret Prediction**

label="Positive" if prediction>=0.5 else "Negative"

print(f"Sentiment :{label} (Confidence :{prediction[0][0]:.4f})")

**1/1** ━━━━━━━━━━━━━━━━━━━━ **0s** 37ms/step

Sentiment :Positive (Confidence :0.5826)

**WEEK-4**

**AIM:** To build and compare various autoencoders (FC, CNN, Sparse, RNN, VAE, Denoising, Contractive) on the MNIST dataset.

**DESCRIPTION:**

This program demonstrates various autoencoder architectures in deep learning applied to the MNIST dataset. It includes multiple models: a Fully Connected Autoencoder for basic dimensionality reduction, a Sparse Autoencoder that enforces sparsity in hidden activations, a Recurrent Autoencoder that captures sequential dependencies, a Variational Autoencoder (VAE) that learns probabilistic latent representations, and a Convolutional Autoencoder that leverages spatial features of images. Additionally, it implements a Denoising Autoencoder which reconstructs clean images from noisy inputs and a Contractive Autoencoder that adds a regularization term to learn robust features. The program preprocesses MNIST images, trains each model, predicts reconstructions, and visualizes the original and reconstructed images side by side to compare the performance of each autoencoder type.

**PROGRAM:**

import numpy as np

import matplotlib.pyplot as plt

from keras.datasets import mnist

from keras.layers import Input, Dense, Conv2D, MaxPooling2D, UpSampling2D, Flatten, Reshape, LSTM, RepeatVector, TimeDistributed

from keras.models import Model

from keras import regularizers

from tensorflow.keras import layers

from tensorflow.keras import backend as K

from keras.layers import GaussianNoise

**# Load MNIST dataset**

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

**# Normalize data**

x\_train = x\_train.astype('float32') / 255

x\_test = x\_test.astype('float32') / 255

**# Flatten data for fully connected models**

x\_train\_fc = x\_train.reshape((len(x\_train), np.prod(x\_train.shape[1:])))

x\_test\_fc = x\_test.reshape((len(x\_test), np.prod(x\_test.shape[1:])))

input\_shape = (28, 28, 1)



**# Fully Connected Autoencoder**

input\_fc = Input(shape=(784,))

encoded\_fc = Dense(32, activation='relu')(input\_fc)

decoded\_fc = Dense(784, activation='sigmoid')(encoded\_fc)

autoencoder\_fc = Model(input\_fc, decoded\_fc)

**# Sparse Autoencoder**

input\_sparse = Input(shape=(784,))

encoded\_sparse = Dense(32, activation='relu', activity\_regularizer=regularizers.l1(10e-5))(input\_sparse)

decoded\_sparse = Dense(784, activation='sigmoid')(encoded\_sparse)

autoencoder\_sparse = Model(input\_sparse, decoded\_sparse)

**# Recurrent Autoencoder**

input\_rnn = Input(shape=(28, 28))

encoded\_rnn = LSTM(32)(input\_rnn)

decoded\_rnn = RepeatVector(28)(encoded\_rnn)

decoded\_rnn = LSTM(32, return\_sequences=True)(decoded\_rnn)

decoded\_rnn = TimeDistributed(Dense(28, activation='sigmoid'))(decoded\_rnn)

autoencoder\_rnn = Model(input\_rnn, decoded\_rnn)

**# Variational Autoencoder**

input\_vae = Input(shape=(784,))

encoded\_vae = Dense(256, activation='relu')(input\_vae)

z\_mean = Dense(2)(encoded\_vae)

z\_log\_var = Dense(2)(encoded\_vae)

**# Sampling function**

def sampling(args):

    z\_mean, z\_log\_var = args

    epsilon = K.random\_normal(shape=(K.shape(z\_mean)[0], 2))

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

z = layers.Lambda(sampling)([z\_mean, z\_log\_var])

decoded\_vae = Dense(784, activation='sigmoid')(z)

autoencoder\_vae = Model(input\_vae, decoded\_vae)

**# Convolutional Autoencoder**

input\_cnn = Input(shape=input\_shape)

x\_cnn = Conv2D(16, (3, 3), activation='relu', padding='same')(input\_cnn)

x\_cnn = MaxPooling2D((2, 2), padding='same')(x\_cnn)

x\_cnn = Conv2D(8, (3, 3), activation='relu', padding='same')(x\_cnn)

x\_cnn = MaxPooling2D((2, 2), padding='same')(x\_cnn)

x\_cnn = Conv2D(8, (3, 3), activation='relu', padding='same')(x\_cnn)

encoded\_cnn = MaxPooling2D((2, 2), padding='same')(x\_cnn)

x\_cnn = Conv2D(8, (3, 3), activation='relu', padding='same')(encoded\_cnn)

x\_cnn = UpSampling2D((2, 2))(x\_cnn)

x\_cnn = Conv2D(8, (3, 3), activation='relu', padding='same')(x\_cnn)

x\_cnn = UpSampling2D((2, 2))(x\_cnn)

x\_cnn = Conv2D(16, (3, 3), activation='relu')(x\_cnn)

x\_cnn = UpSampling2D((2, 2))(x\_cnn)

decoded\_cnn = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x\_cnn)

autoencoder\_cnn = Model(input\_cnn, decoded\_cnn)

**# Denoising Autoencoder**

input\_dae = Input(shape=(784,))

noisy\_input = GaussianNoise(0.2)(input\_dae)

encoded\_dae = Dense(64, activation='relu')(noisy\_input)

decoded\_dae = Dense(784, activation='sigmoid')(encoded\_dae)

autoencoder\_dae = Model(input\_dae, decoded\_dae)

**# Prepare noisy data**

noise\_factor = 0.5

x\_train\_noisy = x\_train\_fc + noise\_factor \* np.random.normal(loc=0.0, scale=1.0, size=x\_train\_fc.shape)

x\_test\_noisy = x\_test\_fc + noise\_factor \* np.random.normal(loc=0.0, scale=1.0, size=x\_test\_fc.shape)

x\_train\_noisy = np.clip(x\_train\_noisy, 0., 1.)

x\_test\_noisy = np.clip(x\_test\_noisy, 0., 1.)

**# Contractive Autoencoder**

lambda\_contractive = 1e-4

input\_cae = Input(shape=(784,))

encoded\_cae = Dense(64, activation='relu', activity\_regularizer=regularizers.l2(lambda\_contractive))(input\_cae)

decoded\_cae = Dense(784, activation='sigmoid')(encoded\_cae)

autoencoder\_cae = Model(input\_cae, decoded\_cae)

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

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

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

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

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

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

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

**# Pick a test image**

x = x\_test[0]

**# Predictions**

x\_fc\_pred = autoencoder\_fc.predict(np.expand\_dims(x.flatten(), axis=0)).reshape(28, 28)

x\_cnn\_pred = autoencoder\_cnn.predict(np.expand\_dims(x.reshape(28,28,1), axis=0)).reshape(28, 28)

x\_sparse\_pred = autoencoder\_sparse.predict(np.expand\_dims(x.flatten(), axis=0)).reshape(28, 28)

x\_rnn\_pred = autoencoder\_rnn.predict(np.expand\_dims(x, axis=0)).reshape(28, 28)

x\_vae\_pred = autoencoder\_vae.predict(np.expand\_dims(x.flatten(), axis=0)).reshape(28, 28)

x\_dae\_pred = autoencoder\_dae.predict(np.expand\_dims(x.flatten(), axis=0)).reshape(28, 28)

x\_cae\_pred = autoencoder\_cae.predict(np.expand\_dims(x.flatten(), axis=0)).reshape(28, 28)

**# Plot results**

fig, axs = plt.subplots(1, 8, figsize=(22, 3))

axs[0].imshow(x, cmap='gray'); axs[0].set\_title('Original')

axs[1].imshow(x\_fc\_pred, cmap='gray'); axs[1].set\_title('Fully Connected AE')

axs[2].imshow(x\_cnn\_pred, cmap='gray'); axs[2].set\_title('Convolutional AE')

axs[3].imshow(x\_sparse\_pred, cmap='gray'); axs[3].set\_title('Sparse AE')

axs[4].imshow(x\_rnn\_pred, cmap='gray'); axs[4].set\_title('Recurrent AE')

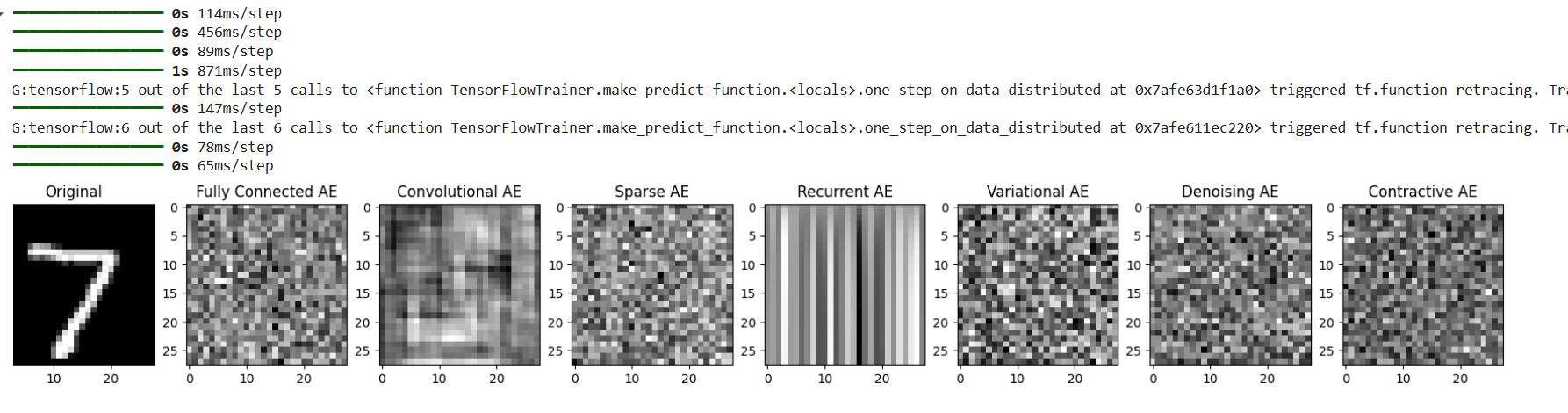
axs[5].imshow(x\_vae\_pred, cmap='gray'); axs[5].set\_title('Variational AE')

axs[6].imshow(x\_dae\_pred, cmap='gray'); axs[6].set\_title('Denoising AE')

axs[7].imshow(x\_cae\_pred, cmap='gray'); axs[7].set\_title('Contractive AE')

plt.show()

**OUTPUT:**



**WEEK-5**

**AIM:** This program evaluates and compares the training performance and accuracy of an MLP model on the MNIST dataset using different optimization algorithms.

**DESCRIPTION:** This program compares the performance of various optimization algorithms on the MNIST handwritten digits dataset. After loading and normalizing the data, a simple multilayer perceptron (MLP) with two hidden layers is used for classification. Optimizers such as SGD, Adam, and RMSprop are tested individually by compiling and training the model for each. Training accuracy, validation accuracy, training loss, and validation loss are recorded and stored for analysis. Finally, the results are visualized through accuracy and loss plots, providing insights into which optimizer performs best for this classification task

**PROGRAM:**

**# 1. Import packages**

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.datasets import mnist

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras import optimizers

**# 2. Load & normalize data**

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

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

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

**# One-hot encode labels**

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

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

**# 3. Form a dictionary of optimizers (removed nadam, ftrl)**

optimizer\_dict = {

    "adam": optimizers.Adam(learning\_rate=0.001),

    "sgd": optimizers.SGD(learning\_rate=0.01, momentum=0.9),

    "rmsprop": optimizers.RMSprop(learning\_rate=0.001),

    "adagrad": optimizers.Adagrad(learning\_rate=0.01),

    "adadelta": optimizers.Adadelta(learning\_rate=1.0)

}

**# 4. Function to build MLP model**

def build\_mlp():

    model = Sequential([

        Flatten(input\_shape=(28, 28)),

        Dense(512, activation='relu'),

        Dense(256, activation='relu'),

        Dense(10, activation='softmax')

    ])

    return model

**# 5. Train & store results**

results = {}

histories = {}

for opt\_name, opt in optimizer\_dict.items():

    print(f"\nTraining with optimizer: {opt\_name}")

    model = build\_mlp()

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

    history = model.fit(

        x\_train, y\_train,

        epochs=3, batch\_size=128,

        validation\_split=0.1, verbose=1

    )

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

    results[opt\_name] = {"accuracy": test\_acc, "loss": test\_loss}

    histories[opt\_name] = history.history

**# 6. Display final test results**

print("\nFinal Test Results:")

for opt\_name, res in results.items():

    print(f"{opt\_name}: Accuracy = {res['accuracy']:.4f}, Loss = {res['loss']:.4f}")

**# 7. Plot accuracy & loss for each optimizer**

for opt\_name, history in histories.items():

    plt.figure(figsize=(10,4))

**# Accuracy plot**

    plt.subplot(1, 2, 1)

    plt.plot(history['accuracy'], label='Train Acc')

    plt.plot(history['val\_accuracy'], label='Val Acc')

    plt.title(f"{opt\_name.upper()} - Accuracy")

    plt.xlabel("Epochs")

    plt.ylabel("Accuracy")

    plt.legend()

**# Loss plot**

    plt.subplot(1, 2, 2)

    plt.plot(history['loss'], label='Train Loss')

    plt.plot(history['val\_loss'], label='Val Loss')

    plt.title(f"{opt\_name.upper()} - Loss")

    plt.xlabel("Epochs")

    plt.ylabel("Loss")

    plt.legend()

    plt.tight\_layout()

    plt.show()

**OUTPUT:**

Final Test Results:

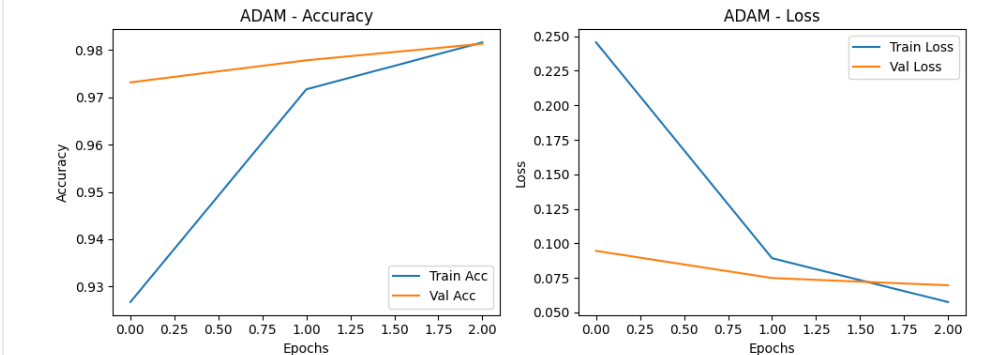
adam: Accuracy = 0.9762, Loss = 0.0753

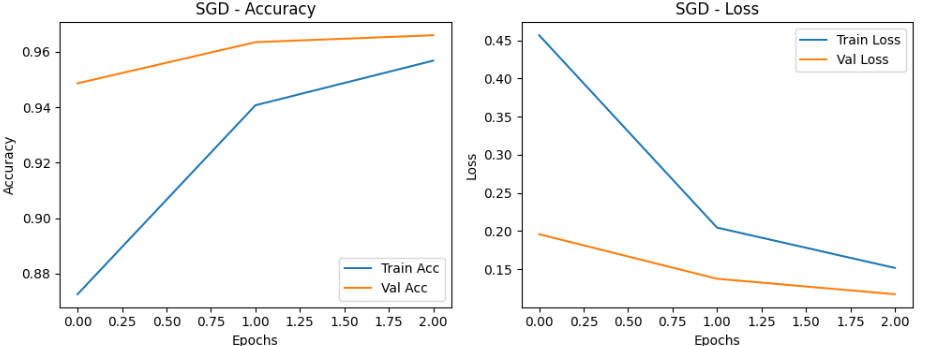
sgd: Accuracy = 0.9584, Loss = 0.1387

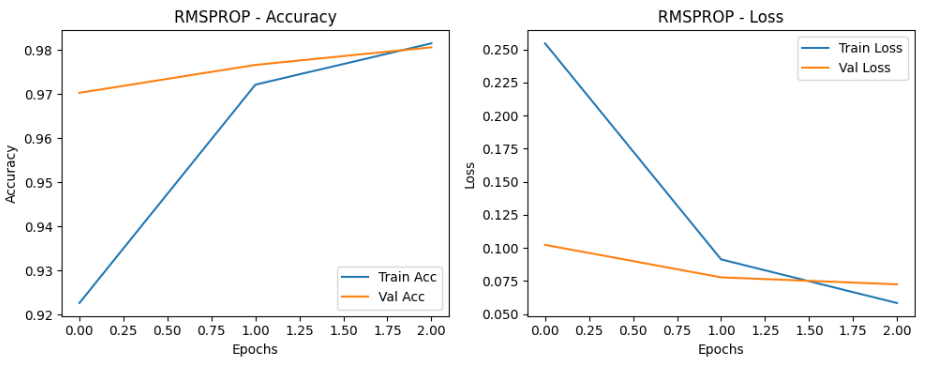
rmsprop: Accuracy = 0.9773, Loss = 0.0728

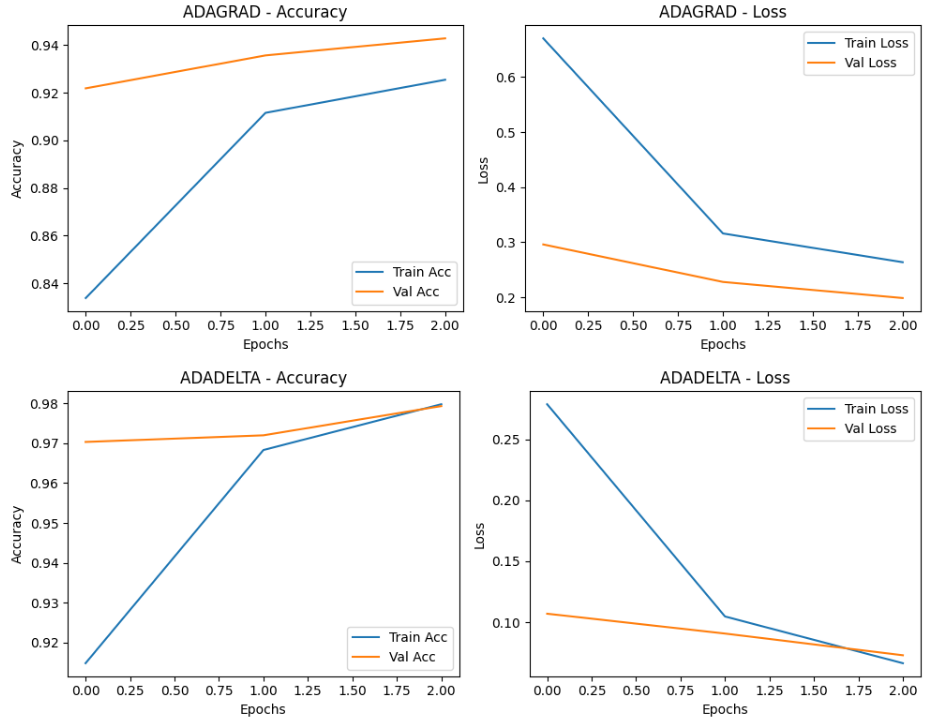
adagrad: Accuracy = 0.9333, Loss = 0.2310

adadelta: Accuracy = 0.9780, Loss = 0.0757









**WEEK-6**

**AIM:** Design a neural Network for classifying news wires (Multi class classification) using Reuters dataset.

**DESCRIPTION:** This program builds and trains a neural network to perform multi-class classification on the Reuters newswire dataset. The dataset is preprocessed by tokenizing the text into integer sequences and converting them into one-hot encoded vectors. A feedforward neural network is designed with fully connected layers and a softmax output layer to predict the topic category of each newswire. The model is trained and evaluated using metrics such as accuracy and loss, enabling effective classification of news articles into multiple categories.

**PROGRAM:**

import numpy as np

import tensorflow as tf

from tensorflow.keras.datasets import reuters

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

import matplotlib.pyplot as plt

(train\_data, train\_labels), (test\_data, test\_labels) = reuters.load\_data(num\_words=10000)

print(f"Number of training samples: {len(train\_data)}")

print(f"Number of test samples: {len(test\_data)}")



def vectorize\_sequences(sequences, dimension=10000):

    results = np.zeros((len(sequences), dimension))

    for i, sequence in enumerate(sequences):

        results[i, sequence] = 1

    return results

X\_train = vectorize\_sequences(train\_data)

X\_test = vectorize\_sequences(test\_data)

from tensorflow.keras.utils import to\_categorical

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

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

model = Sequential([

    Dense(64, activation='relu', input\_shape=(10000,)),

    Dense(64, activation='relu'),

    Dense(46, activation='softmax')  # 46 categories => softmax

])

model.compile(optimizer='adam',

loss='categorical\_crossentropy',

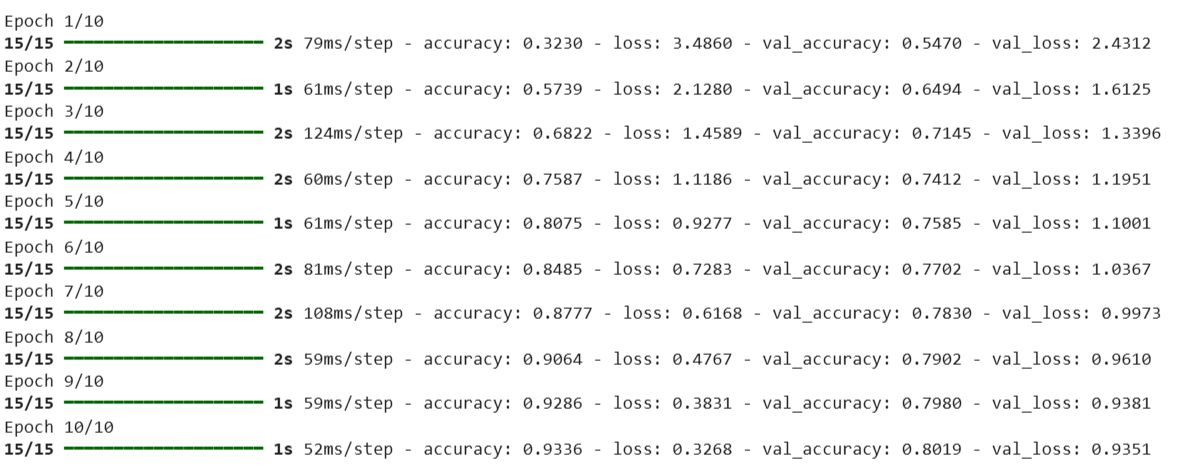
metrics=['accuracy'])

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

                    epochs=10,

                    batch\_size=512,

                    validation\_split=0.2)



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

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

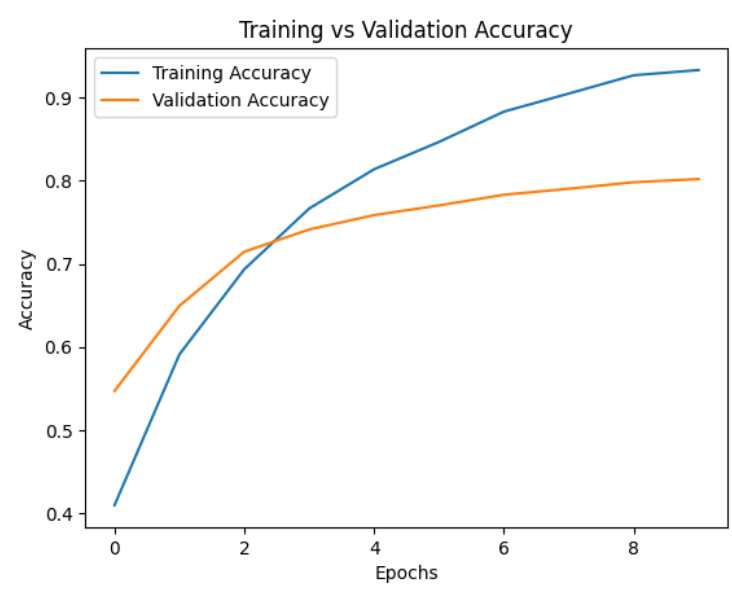
plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

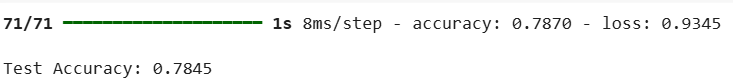
plt.title("Training vs Validation Accuracy")

plt.show()



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

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



predictions = model.predict(X\_test)

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

print(f"Predicted class for first test sample: {predicted\_class}")



import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

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

**# Predict class probabilities**

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

**# Convert probabilities to predicted class index**

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

**# Convert one-hot labels back to class index**

y\_true = np.argmax(y\_test, axis=1)



**# Count frequency of each true label**

(unique, counts) = np.unique(y\_true, return\_counts=True)

sorted\_indices = np.argsort(-counts)  # descending order

top\_classes = unique[sorted\_indices[:10]]

**# Filter predictions and labels to only top 10 classes**

mask = np.isin(y\_true, top\_classes)

y\_true\_top10 = y\_true[mask]

y\_pred\_top10 = y\_pred[mask]

**# Build confusion matrix**

cm\_top10 = confusion\_matrix(y\_true\_top10, y\_pred\_top10, labels=top\_classes)

**# Normalize (row-wise)**

cm\_top10\_normalized = cm\_top10.astype('float') / cm\_top10.sum(axis=1)[:, np.newaxis]

**# Plot with seaborn**

plt.figure(figsize=(10, 8))

sns.heatmap(cm\_top10\_normalized, annot=True, fmt=".2f", cmap="Blues",

            xticklabels=top\_classes, yticklabels=top\_classes)

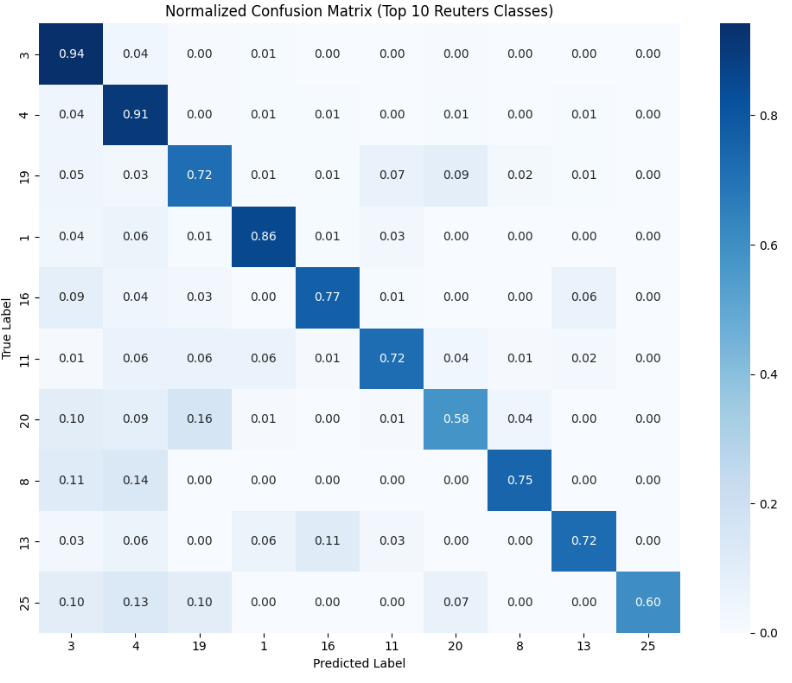
plt.title("Normalized Confusion Matrix (Top 10 Reuters Classes)")

plt.xlabel("Predicted Label")

plt.ylabel("True Label")

plt.tight\_layout()

plt.show()



model.save("reuters\_news\_model.h5")

from tensorflow.keras.models import load\_model

model = load\_model("reuters\_news\_model.h5")

**# Get the mapping from word -> index**

word\_index = reuters.get\_word\_index()

reverse\_word\_index = dict([(value, key) for (key, value) in word\_index.items()])



def encode\_new\_text(text, word\_index, max\_words=10000):

    # Basic preprocessing: lowercase and split

    tokens = text.lower().split()

    # Convert to word indices (0 for unknown)

    encoded = [word\_index.get(word, 0) for word in tokens]

    # Create binary bag-of-words vector

    vectorized = np.zeros((1, max\_words))

    for index in encoded:

        if index < max\_words:

            vectorized[0, index] = 1

    return vectorized

new\_text = "The stock market saw a major shift as tech companies surged in the second quarter"

x\_input = encode\_new\_text(new\_text, word\_index)

prediction = model.predict(x\_input)

predicted\_class = np.argmax(prediction[0])

print(f"Predicted topic class: {predicted\_class} (Confidence: {np.max(prediction[0]):.4f})")

****

from tensorflow.keras.datasets import reuters

\_, train\_labels = reuters.load\_data(num\_words=10000)[0]

num\_classes = np.max(train\_labels) + 1

print("Number of classes:", num\_classes)

****

**WEEK-7**

**AIM:** Use a pre-trained convolution neural network (VGG16) for image classification.

**DESCRIPTION:** The **VGG16 model** is one of the most widely used pre-trained convolutional neural networks (CNNs) developed by the **Visual Geometry Group (VGG)** at the University of Oxford. It was introduced in **the 2014 ILSVRC (ImageNet Large Scale Visual Recognition Challenge)** and is known for its deep architecture and simplicity.

Using **a pre-trained VGG16** means leveraging the model that has already been trained on a massive dataset like **ImageNet** (over **1.2 million images and 1,000 object categories).** This approach allows us to **transfer the learned features** to new image classification tasks without having to train a model from scratch, which saves time, computation, and data requirements.

**PROGRAM:**

from keras.models import Sequential

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

vgg16\_custom = Sequential()

vgg16\_custom.add(Conv2D(64, (3, 3), activation='relu', padding='same', input\_shape=(224, 224, 3)))

vgg16\_custom.add(Conv2D(64, (3, 3), activation='relu', padding='same'))

vgg16\_custom.add(MaxPooling2D((2, 2)))

vgg16\_custom.add(Conv2D(128, (3, 3), activation='relu', padding='same'))

vgg16\_custom.add(Conv2D(128, (3, 3), activation='relu', padding='same'))

vgg16\_custom.add(MaxPooling2D((2, 2)))

vgg16\_custom.add(Conv2D(256, (3, 3), activation='relu', padding='same'))

vgg16\_custom.add(Conv2D(256, (3, 3), activation='relu', padding='same'))

vgg16\_custom.add(Conv2D(256, (3, 3), activation='relu', padding='same'))

vgg16\_custom.add(MaxPooling2D((2, 2)))

vgg16\_custom.add(Conv2D(512, (3, 3), activation='relu', padding='same'))

vgg16\_custom.add(Conv2D(512, (3, 3), activation='relu', padding='same'))

vgg16\_custom.add(Conv2D(512, (3, 3), activation='relu', padding='same'))

vgg16\_custom.add(MaxPooling2D((2, 2)))

vgg16\_custom.add(Conv2D(512, (3, 3), activation='relu', padding='same'))

vgg16\_custom.add(Conv2D(512, (3, 3), activation='relu', padding='same'))

vgg16\_custom.add(Conv2D(512, (3, 3), activation='relu', padding='same'))

vgg16\_custom.add(MaxPooling2D((2, 2)))

vgg16\_custom.add(Flatten())

vgg16\_custom.add(Dense(4096, activation='relu'))

vgg16\_custom.add(Dropout(0.5))

vgg16\_custom.add(Dense(4096, activation='relu'))

vgg16\_custom.add(Dropout(0.5))

vgg16\_custom.add(Dense(3, activation='softmax'))

vgg16\_custom.summary()

**OUTPUT:**

****

**WEEK-8**

**AIM:** Build a Convolution Neural Network for simple image (dogs and Cats) Classification

**DESCRIPTION:** A Convolutional Neural Network (CNN) is a specialized type of deep learning model designed to analyze and classify visual data. In this task, we build a CNN from scratch to classify images of cats and dogs into their respective categories. CNNs are widely used for image classification because they can automatically learn and extract important features such as edges, textures, and shapes directly from raw image data.

The dataset typically contains two folders — one with cat images and another with dog images. By training the CNN on these labeled images, the model learns to distinguish between the two classes.

**PROGRAM:**

import numpy as np

import os

import matplotlib.pyplot as plt

import pickle

import cv2

import random

DIRECTORY = f"/kaggle/input/dog-vs-cat-fastai/dogscats/train"

CATEGORIES = ['cats', 'dogs']

import os

import cv2

img\_size = 100

data = []

for category in CATEGORIES:

folder = os.path.join(DIRECTORY, category)

print(folder)

label = 0 if category == "cats" else 1

for i, img in enumerate(os.listdir(folder)):

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

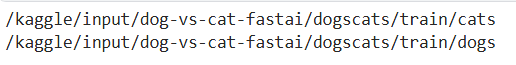
img\_arr = cv2.imread(img\_path)

if img\_arr is None:

continue

img\_arr = cv2.resize(img\_arr, (img\_size, img\_size))

data.append([img\_arr, label])



random.shuffle(data)

X = [features for features, label in data]

y = [label for features, label in data]

sample\_img = X[0]

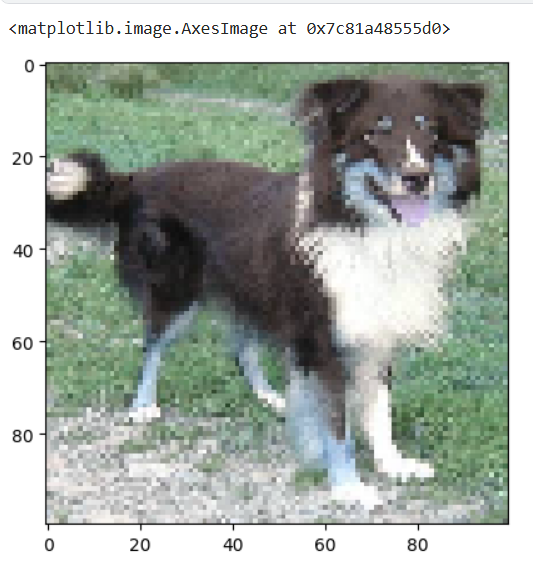
sample\_img.shape



len(data)



plt.imshow(sample\_img)

****

import pickle

pickle.dump(X, open('x.pkl', 'wb'))

pickle.dump(y, open('y.pkl', 'wb'))

X = np.array(X) / 255.0

y = np.array(y)

from tensorflow.keras.models import Sequential

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

model = Sequential()

model.add(Conv2D(32, (3,3), activation='relu', input\_shape = X.shape[1:]))

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

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

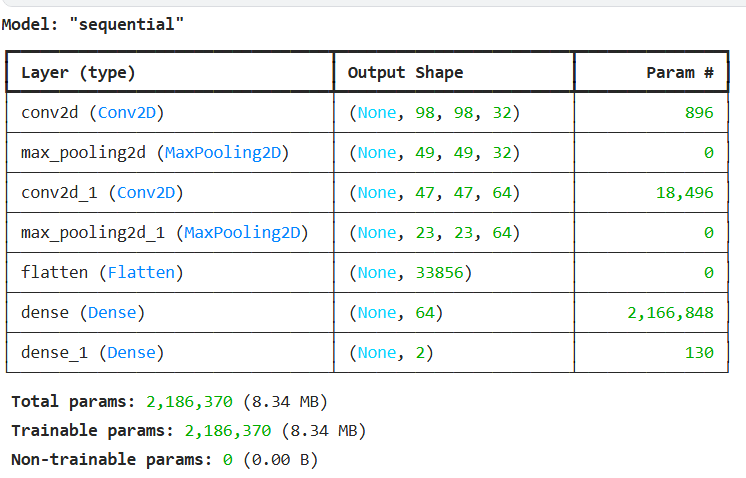
model.add(MaxPooling2D((2,2)))

model.add(Flatten())

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

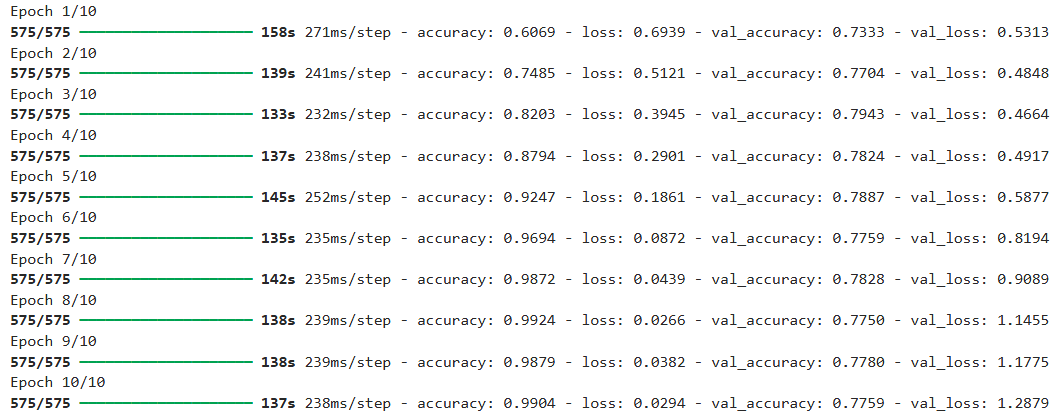
model.add(Dense(2, activation ='softmax'))

model.summary()



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

history = model.fit(X, y, epochs=10, validation\_split=0.2)



**WEEK-9**

**AIM:** Implement a Recurrent Neural Network for IMDB Movie review classification problem.

**DESCRIPTION:**

The **IMDB Movie Review Classification problem** is a classic **natural language processing (NLP)** task where the goal is to determine whether a given movie review is **positive** or **negative**. It is a **binary sentiment analysis problem** and is widely used to test deep learning models for text data.

A **Recurrent Neural Network (RNN)** is an ideal choice for this task because it is specifically designed to handle **sequential data**, such as sentences or text. Unlike traditional neural networks, RNNs have the ability to **retain memory** of previous inputs, which allows them to capture the context and meaning of words based on their order.

**PROGRAM:**

from keras.datasets import imdb

from keras.preprocessing import sequence

max\_features = 10000

maxlen = 500

batch\_size = 32

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

print(len(x\_train),'train sequences')

print(len(x\_test),'test sequences')



print ('pad sequences(sample x time)')



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

x\_train = sequence.pad\_sequences(x\_train,maxlen = maxlen)

x\_test = sequence.pad\_sequences(x\_test,maxlen = maxlen)

print('x\_train shape:',x\_train.shape)

print('x\_test shape:',x\_test.shape)



from keras.layers import Dense, Embedding, SimpleRNN

from keras.models import Sequential

model = Sequential()

model.add(Embedding(10000,32))

model.add(SimpleRNN(32))

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

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

history = model.fit(x\_train,y\_train,epochs = 10,batch\_size = 128,validation\_split = 0.2)



import matplotlib.pyplot as plt

accuracy = history.history['accuracy']

val\_accuracy = history.history['val\_accuracy']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(1,len(accuracy)+1)

plt.plot(epochs , accuracy , 'bo', label = 'training accuracy')

plt.plot(epochs,val\_accuracy,'b' ,label = 'validation accuracy')

plt.title("training and validation accuracy")

plt.legend()

plt.figure()

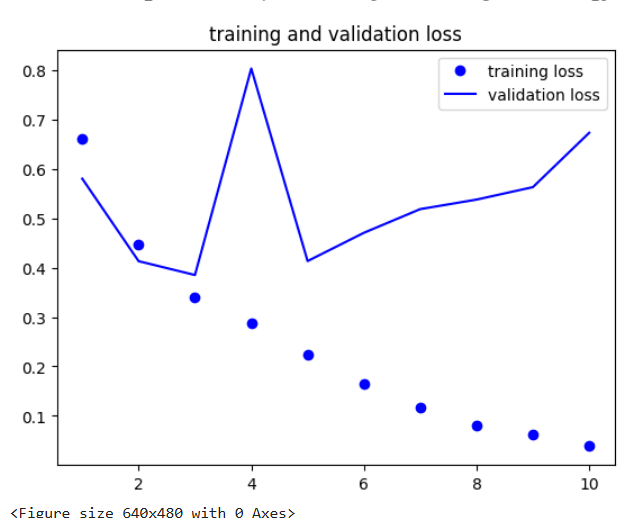
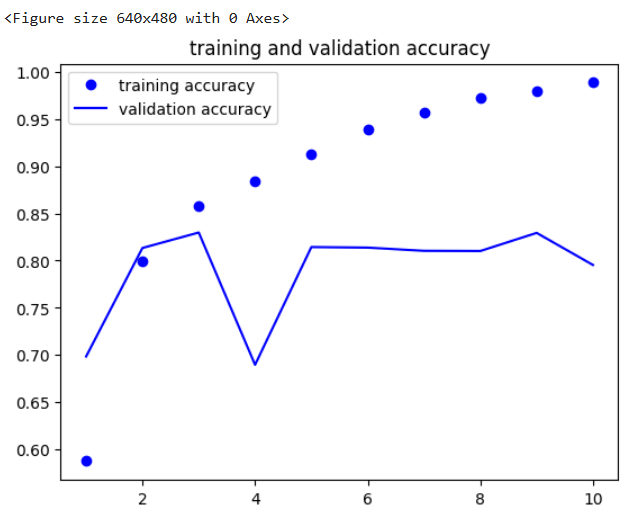
plt.plot(epochs , loss , 'bo', label = 'training loss')

plt.plot(epochs,val\_loss,'b' ,label = 'validation loss')

plt.title("training and validation loss")

plt.legend()

plt.figure()



# LSTM

from keras.layers import Dense

from keras.layers import LSTM

model = Sequential()

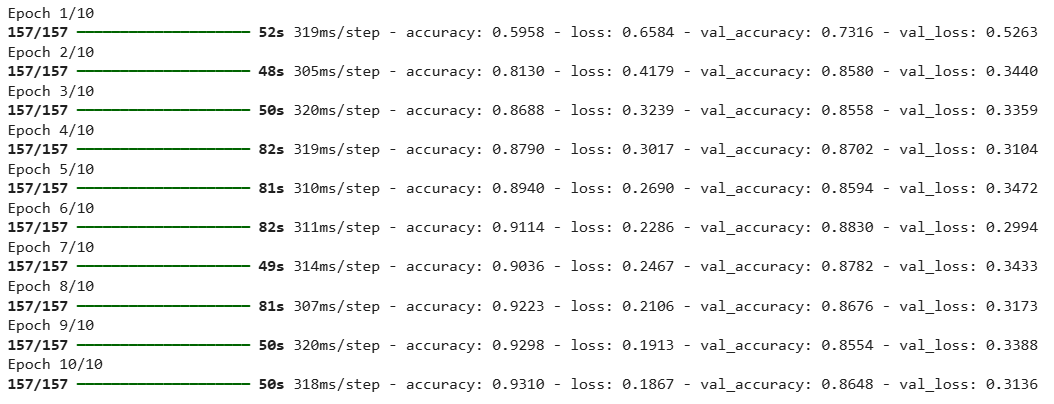
model.add(Embedding(10000,32))

model.add(LSTM(32))

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

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

history = model.fit(x\_train,y\_train,epochs = 10,batch\_size = 128,validation\_split = 0.2)



import matplotlib.pyplot as plt

accuracy = history.history['accuracy']

val\_accuracy = history.history['val\_accuracy']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(1,len(accuracy)+1)

plt.plot(epochs , accuracy , 'bo', label = 'training accuracy')

plt.plot(epochs,val\_accuracy,'b' ,label = 'validation accuracy')

plt.title("training and validation accuracy")

plt.legend()

plt.figure()

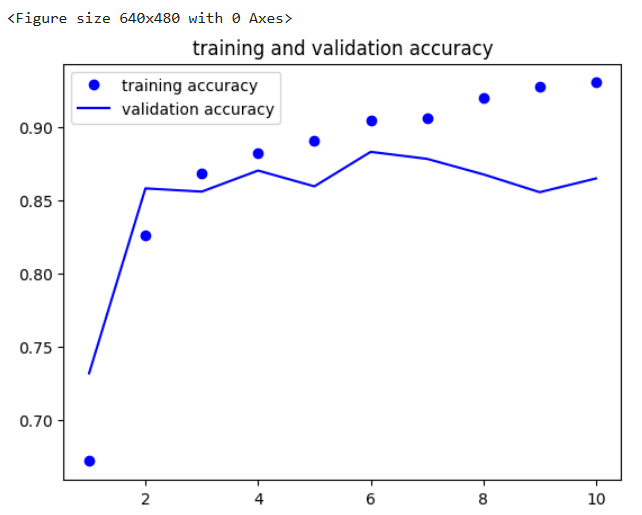
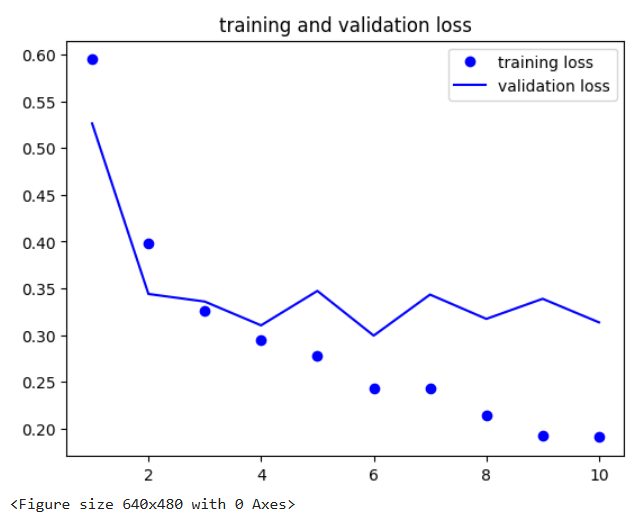
plt.plot(epochs , loss , 'bo', label = 'training loss')

plt.plot(epochs,val\_loss,'b' ,label = 'validation loss')

plt.title("training and validation loss")

plt.legend()

plt.figure()

**WEEK-10**

**AIM:** To implement a deep learning-based NLP pipeline that performs sentiment analysis, text summarization, and text-to-speech conversion on both online and local datasets using pre-trained transformer models.

**DESCRIPTION:**

This program leverages deep learning models from the Hugging Face Transformers library to process textual data and generate audio outputs. The pipeline includes three main DL tasks:

1. Sentiment Analysis:
   1. Uses a pre-trained transformer model (DistilBERT) fine-tuned on the SST-2 dataset.
   2. Classifies text into positive or negative sentiment based on learned deep representations.
2. Text Summarization:
   1. Uses a sequence-to-sequence transformer model (BART) to generate concise summaries from longer text inputs.
   2. Applies deep learning to understand semantic context and extract key information.
3. Text-to-Speech (TTS):
   1. Uses a deep learning-based TTS model (Bark) to convert summarized text into natural-sounding speech.
   2. Outputs are saved as WAV files for playback or further processing.
4. Dataset Handling:
   1. Processes online datasets like CNN/DailyMail and local text datasets defined in the code.
   2. For each text, the pipeline applies DL models sequentially to predict sentiment, summarize, and generate speech.

**PROGRAM:**

**1)SENTIMENT ANALYSIS USING DISTILBERT PRE-TRAINED TRANSFORMER MODEL**

from transformers import pipeline

classifier = pipeline("sentiment-analysis", model="distilbert-base-uncased-finetuned-sst-2-english")

# Sample online text (from URL or string)

text = "Artificial Intelligence is transforming the world at a rapid pace."

# Sentiment analysis

result = classifier(text)

print(" Sentiment Analysis Result:", result)

**OUTPUT:**

Device set to use cuda:0

Sentiment Analysis Result: [{'label': 'POSITIVE', 'score': 0.9995410442352295}]

**2)TEXT-TO-SPEECH CONVERSION USING SUNO BARK TRANSFORMER MODEL**

from transformers import pipeline

import numpy as np

import soundfile as sf

import sounddevice as sd

tts = pipeline("text-to-speech", model="suno/bark-small")

speech = tts(summary[0]['summary\_text'])

audio = np.array(speech['audio']).astype(np.float32).flatten()

rate = speech['sampling\_rate']

# Save and play audio

sf.write("summary\_speech.wav", audio, rate)

print("Audio saved as 'summary\_speech.wav'")

**OUTPUT:**

Device set to use cuda:0

The attention mask and the pad token id were not set. As a consequence, you may observe unexpected behavior. Please pass your input's `attention\_mask` to obtain reliable results.

Setting `pad\_token\_id` to `eos\_token\_id`:10000 for open-end generation.

Audio saved as 'summary\_speech.wav'

**3) SENTIMENT ANALYSIS, SUMMARIZATION, AND TEXT-TO-SPEECH ON ONLINE CNN/DAILYMAIL DATASET**

from datasets import load\_dataset

# Load an online text dataset

dataset = load\_dataset("cnn\_dailymail", "3.0.0", split="train[:1%]")  # small subset

sample\_text = dataset[0]['article']

print("Sample Text:", sample\_text[:500])  # print first 500 chars

# Sentiment analysis

sent\_result = classifier(sample\_text[:200])  # first 200 chars for demo

print("Sentiment:", sent\_result)

# Summarization

summary\_text = summarizer(sample\_text[:200], max\_length=50, min\_length=20, do\_sample=False)

print("Summary:", summary\_text[0]['summary\_text'])

# TTS

speech\_online = tts(summary\_text[0]['summary\_text'])

audio\_online = np.array(speech\_online['audio']).astype(np.float32).flatten()

rate\_online = speech\_online['sampling\_rate']

sf.write("online\_dataset\_speech.wav", audio\_online, rate\_online)

print("Online dataset audio saved as 'online\_dataset\_speech.wav'")

**OUTPUT:**

Sample Text: LONDON, England (Reuters) -- Harry Potter star Daniel Radcliffe gains access to a reported £20 million ($41.1 million) fortune as he turns 18 on Monday, but he insists the money won't cast a spell on him. Daniel Radcliffe as Harry Potter in "Harry Potter and the Order of the Phoenix" To the disappointment of gossip columnists around the world, the young actor says he has no plans to fritter his cash away on fast cars, drink and celebrity parties. "I don't plan to be one of those people who, as s

Sentiment: [{'label': 'POSITIVE', 'score': 0.9193909764289856}]

The attention mask and the pad token id were not set. As a consequence, you may observe unexpected behavior. Please pass your input's `attention\_mask` to obtain reliable results.

Setting `pad\_token\_id` to `eos\_token\_id`:10000 for open-end generation.

Summary: Harry Potter star Daniel Radcliffe gains access to a reported $41.1 million fortune. The 18-year-old insists the money won't cast a spell on him.

Online dataset audio saved as 'online\_dataset\_speech.wav'

**4) SENTIMENT ANALYSIS, SUMMARIZATION, AND TEXT-TO-SPEECH ON LOCAL TEXT DATASET**

# Step 2: Import libraries

from transformers import pipeline

import numpy as np

import soundfile as sf

import sounddevice as sd

# Step 3: Define local dataset directly in code

local\_texts = [

    "I love this movie! It was amazing and full of surprises.",

    "The weather today is terrible. I hate getting wet in the rain.",

    "Artificial intelligence will change the future of technology.",

    "My favorite food is pizza, I could eat it every day.",

    "The traffic jam made me late and very frustrated."

]

print("Local dataset defined with", len(local\_texts), "texts")

**OUTPUT:**

Local dataset defined with 5 texts

# Step 4: Initialize pipelines

# Sentiment analysis

classifier = pipeline("sentiment-analysis", model="distilbert-base-uncased-finetuned-sst-2-english")

# Summarization

summarizer = pipeline("summarization", model="facebook/bart-large-cnn")

# Text-to-Speech

tts = pipeline("text-to-speech", model="suno/bark-small")

**OUTPUT:**

Device set to use cuda:0

Device set to use cuda:0

Device set to use cuda:0

**5) SENTIMENT ANALYSIS, SUMMARIZATION, AND TEXT-TO-SPEECH FOR EACH TEXT IN LOCAL DATASET**

# Step 5: Process the local dataset

for i, text in enumerate(local\_texts):

    # Sentiment analysis

    sentiment = classifier(text)[0]['label']

    # Summarization

    summary = summarizer(text, max\_length=50, min\_length=10, do\_sample=False)[0]['summary\_text']

    # Text-to-Speech

    speech = tts(summary)

    audio = np.array(speech['audio']).astype(np.float32).flatten()

    rate = speech['sampling\_rate']

    # Save audio

    filename = f"local\_text\_{i}\_speech.wav"

    sf.write(filename, audio, rate)

    # Print results

    print(f"📄 Text {i}: {text}")

    print(f"Sentiment: {sentiment}")

    print(f"Summary: {summary}")

    print(f"Audio saved as: {filename}\n")

**OUTPUT:**

Your max\_length is set to 50, but your input\_length is only 15. Since this is a summarization task, where outputs shorter than the input are typically wanted, you might consider decreasing max\_length manually, e.g. summarizer('...', max\_length=7)

The attention mask and the pad token id were not set. As a consequence, you may observe unexpected behavior. Please pass your input's `attention\_mask` to obtain reliable results.

Setting `pad\_token\_id` to `eos\_token\_id`:10000 for open-end generation.

Your max\_length is set to 50, but your input\_length is only 16. Since this is a summarization task, where outputs shorter than the input are typically wanted, you might consider decreasing max\_length manually, e.g. summarizer('...', max\_length=8)

Text 0: I love this movie! It was amazing and full of surprises.

Sentiment: POSITIVE

Summary: "It was amazing and full of surprises. I love this movie!"

Audio saved as: local\_text\_0\_speech.wav

The attention mask and the pad token id were not set. As a consequence, you may observe unexpected behavior. Please pass your input's `attention\_mask` to obtain reliable results.

Setting `pad\_token\_id` to `eos\_token\_id`:10000 for open-end generation.

Your max\_length is set to 50, but your input\_length is only 11. Since this is a summarization task, where outputs shorter than the input are typically wanted, you might consider decreasing max\_length manually, e.g. summarizer('...', max\_length=5)

Text 1: The weather today is terrible. I hate getting wet in the rain.

Sentiment: NEGATIVE

Summary: The weather today is terrible. I hate getting wet in the rain.

Audio saved as: local\_text\_1\_speech.wav

The attention mask and the pad token id were not set. As a consequence, you may observe unexpected behavior. Please pass your input's `attention\_mask` to obtain reliable results.

Setting `pad\_token\_id` to `eos\_token\_id`:10000 for open-end generation.

Your max\_length is set to 50, but your input\_length is only 15. Since this is a summarization task, where outputs shorter than the input are typically wanted, you might consider decreasing max\_length manually, e.g. summarizer('...', max\_length=7)

Text 2: Artificial intelligence will change the future of technology.

Sentiment: POSITIVE

Summary: Artificial intelligence will change the future of technology, experts say.

Audio saved as: local\_text\_2\_speech.wav

The attention mask and the pad token id were not set. As a consequence, you may observe unexpected behavior. Please pass your input's `attention\_mask` to obtain reliable results.

Setting `pad\_token\_id` to `eos\_token\_id`:10000 for open-end generation.

Your max\_length is set to 50, but your input\_length is only 12. Since this is a summarization task, where outputs shorter than the input are typically wanted, you might consider decreasing max\_length manually, e.g. summarizer('...', max\_length=6)

Text 3: My favorite food is pizza, I could eat it every day.

Sentiment: POSITIVE

Summary: My favorite food is pizza, I could eat it every day. My favorite movie is The Godfather: Part II.

Audio saved as: local\_text\_3\_speech.wav

The attention mask and the pad token id were not set. As a consequence, you may observe unexpected behavior. Please pass your input's `attention\_mask` to obtain reliable results.

Setting `pad\_token\_id` to `eos\_token\_id`:10000 for open-end generation.

Text 4: The traffic jam made me late and very frustrated.

Sentiment: NEGATIVE

Summary: The traffic jam made me late and very frustrated.

Audio saved as: local\_text\_4\_speech.wav

**WEEK-11**

**AIM:** Implement one hot encoding of words or characters

**DESCRIPTION:** One-Hot Encoding is a representation technique used in Natural Language Processing (NLP) to convert words or characters into numerical vectors that a machine learning model can understand.  
In this method, each unique word or character in the text is represented by a binary vector — a sequence of 0s and 1s — where only one position is marked as 1 (indicating the presence of that specific word or character), and all other positions are 0.

**PROGRAM:**

**A) ONE-HOT ENCONDING OF WODS**

# Import numpy for array handling

import numpy as np

# Step 1: Create a simple document (list of words)

words = ["the", "cat", "sat", "on", "the", "mat"]

# Step 2: Create the vocabulary (unique words)

vocab = sorted(set(words))

print("Vocabulary:", vocab)

**OUTPUT:**

Vocabulary: ['cat', 'mat', 'on', 'sat', 'the']

# Step 3: Create a dictionary mapping word -> index

word\_to\_index = {word: i for i, word in enumerate(vocab)}

print("\nWord to Index Mapping:", word\_to\_index)

**OUTPUT:**

Word to Index Mapping: {'cat': 0, 'mat': 1, 'on': 2, 'sat': 3, 'the': 4}

# Step 4: Create a one-hot encoded matrix

num\_words = len(words)        # Total number of words in the document

vocab\_size = len(vocab)       # Size of the vocabulary

# Create an empty matrix of zeros

one\_hot\_matrix = np.zeros((num\_words, vocab\_size), dtype=int)

# Fill the matrix

for i, word in enumerate(words):

    index = word\_to\_index[word]   # Get index of the word from the dictionary

    one\_hot\_matrix[i, index] = 1  # Set that position to 1

# Step 5: Display results

print

("\nOne-Hot Encoded Matrix:")

print(one\_hot\_matrix)

**OUTPUT:**

[[0 0 0 0 1]

[1 0 0 0 0]

[0 0 0 1 0]

[0 0 1 0 0]

[0 0 0 0 1]

[0 1 0 0 0]]

# Step 6: Show what each row means

for i, word in enumerate(words):

    print(f"{word:>5} → {one\_hot\_matrix[i]}")

**OUTPUT:**

the → [0 0 0 0 1]

cat → [1 0 0 0 0]

sat → [0 0 0 1 0]

on → [0 0 1 0 0]

the → [0 0 0 0 1]

mat → [0 1 0 0 0]

**B) ONE HOT ENCODING OF CHARACTERS**

import numpy as np

text = "hello"

# Create vocabulary of unique characters

chars = sorted(set(text))

print("Characters:", chars)

**OUTPUT:**

Characters: ['e', 'h', 'l', 'o']

# Create mapping character -> index

char\_to\_index = {ch: i for i, ch in enumerate(chars)}

print("Character to Index Mapping:", char\_to\_index)

**OUTPUT:**

Character to Index Mapping: {'e': 0, 'h': 1, 'l': 2, 'o': 3}

# Create one-hot encoded matrix

one\_hot\_chars = np.zeros((len(text), len(chars)), dtype=int)

for i, ch in enumerate(text):

  index = char\_to\_index[ch]

  one\_hot\_chars[i, index] = 1

print("\nOne-Hot Encoded Chars:")

print(one\_hot\_chars)

**OUTPUT:**

One-Hot Encoded Chars:

[[0 1 0 0]

[1 0 0 0]

[0 0 1 0]

[0 0 1 0]

[0 0 0 1]]

print("\nOne-Hot Encoded Matrix:")

for i, ch in enumerate(text):

  print(f"{ch} → {one\_hot\_chars[i]}")

**OUTPUT:**

One-Hot Encoded Matrix:

h → [0 1 0 0]

e → [1 0 0 0]

l → [0 0 1 0]

l → [0 0 1 0]

o → [0 0 0 1]

**WEEK-12**

**AIM:** Implement word embeddings for IMDB dataset

**DESCRIPTION:** Word Embeddings are dense vector representations of words in a continuous vector space where words with similar meanings are located close to each other.  
Unlike one-hot encoding (which is sparse and high-dimensional), word embeddings capture semantic relationships and contextual meaning of words.

In this experiment, we implement word embeddings using the IMDB Movie Reviews dataset, which is commonly used for sentiment analysis (classifying reviews as positive or negative).

**PROGRAM:**

# Import libraries

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

# num\_words = 10000 means we keep only the top 10,000 most frequent words

vocab\_size = 10000

(x\_train, y\_train), (x\_test, y\_test) = keras.datasets.imdb.load\_data(num\_words=vocab\_size)



print("Training samples:", len(x\_train))

print("Test samples:", len(x\_test))

print("Example review (as integers):", x\_train[0][:10])

****

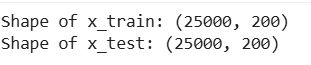
maxlen = 200 # we’ll use only first 200 words of each review

x\_train = keras.preprocessing.sequence.pad\_sequences(x\_train, maxlen=maxlen)

x\_test = keras.preprocessing.sequence.pad\_sequences(x\_test, maxlen=maxlen)

print("Shape of x\_train:", x\_train.shape)

print("Shape of x\_test:", x\_test.shape)

****

model = keras.Sequential([

layers.Embedding(input\_dim=vocab\_size, output\_dim=16, input\_length=maxlen),

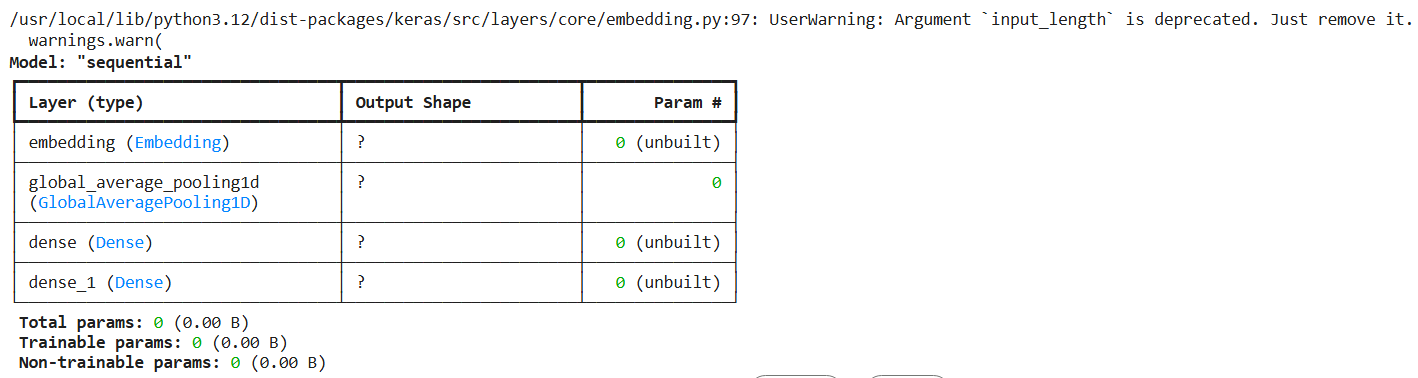
layers.GlobalAveragePooling1D(), # average all embeddings

layers.Dense(16, activation='relu'),

layers.Dense(1, activation='sigmoid') # binary output (positive or negative)

])

model.summary()



model.compile(optimizer='adam',

        loss='binary\_crossentropy',

        metrics=['accuracy'])

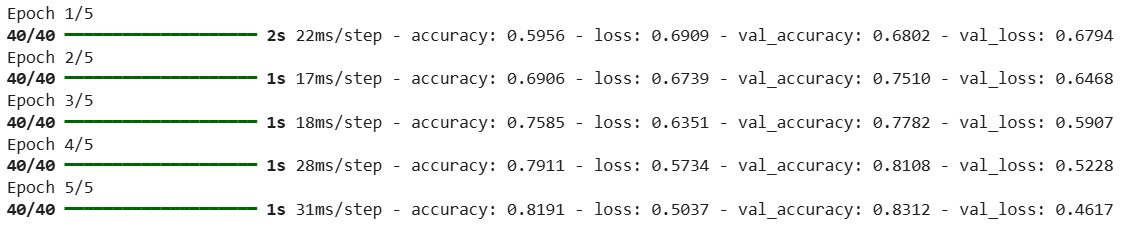
history = model.fit(x\_train, y\_train,

        epochs=5,

        batch\_size=512,

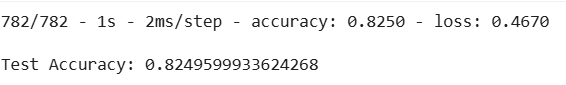
        validation\_split=0.2,

        verbose=1)



results = model.evaluate(x\_test, y\_test, verbose=2)

print("\nTest Accuracy:", results[1])



embedding\_layer = model.layers[0]

embeddings = embedding\_layer.get\_weights()[0]

print("\nEmbedding matrix shape:", embeddings.shape)



# Code to Check the Embedding for Any Word

from tensorflow.keras.datasets import imdb

# Load the IMDB word index dictionary

word\_index = imdb.get\_word\_index()

# Because Keras IMDB adds special reserved indices (0, 1, 2, 3),

# we need to shift existing indices by 3

reverse\_word\_index = {v + 3: k for k, v in word\_index.items()}

reverse\_word\_index[0] = "<PAD>"

reverse\_word\_index[1] = "<START>"

reverse\_word\_index[2] = "<UNK>"

reverse\_word\_index[3] = "<UNUSED>"

# ---- Enter any word you want to check ----

word = "director" # try "great", "bad", "terrible", "love", etc.

# Get its index

index = word\_index.get(word)

if index is not None and index + 3 < embeddings.shape[0]:

    print(f"Word: {word}")

    print(f"Index in vocabulary: {index + 3}")

    print("Embedding vector:\n", embeddings[index + 3])

else:

    print(f"'{word}' not found in the vocabulary (maybe too rare).")

