NANDHA ENGINEERING COLLEGE (AUTONOMOUS), ERODE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DEEP LEARNING

ASSIGNMENT I

ACADEMIC YEAR: 2024-2025

CLASS: III - CSE 'B'

MARKS: 20 marks

SEM : V

TEAM 12(22CS103, 22CS104, 22CS106,22CS107,22CS108)

S.No	QUESTION	Marks
1	Develop a neural network that performs imageto-image translation tasks, such as converting sketches into detailed color images or translating grayscale images into color images.	10
2	Design an autoencoder-based system for collaborative filtering in recommendation systems. The autoencoder should	10

Faculty signature **Student signature**

1.Develop a neural network that performs image-to-imageTranslation tasks, such as converting sketches into detailedColor images or translating grayscale images into color images.

CODING:

```
Import numpy as np
Import tensorflow as tf
From tensorflow.keras import layers, models
Grayscale_dir = ‘flowers_grey’
Color_dir = ‘flowers_colour’
Image_size = (128, 128)
Grayscale_dataset = tf.keras.preprocessing.image_dataset_from_directory(
   Grayscale_dir,
   Label_mode=None,
   Color_mode=’grayscale’,
   Image_size=image_size,
   Batch_size=32,
Shuffle = False
)
Color_dataset = tf.keras.preprocessing.image_dataset_from_directory(
   Color_dir,
   Label_mode=None,
   Color_mode=’rgb’,
   Image_size=image_size,
   Batch_size=32,
Shuffle=False
)
Grayscale_dataset = grayscale_dataset.map(lambda x: x / 255.0)
Color_dataset = color_dataset.map(lambda x: x / 255.0)
For gray_batch, color_batch in zip(grayscale_dataset, color_dataset):
```

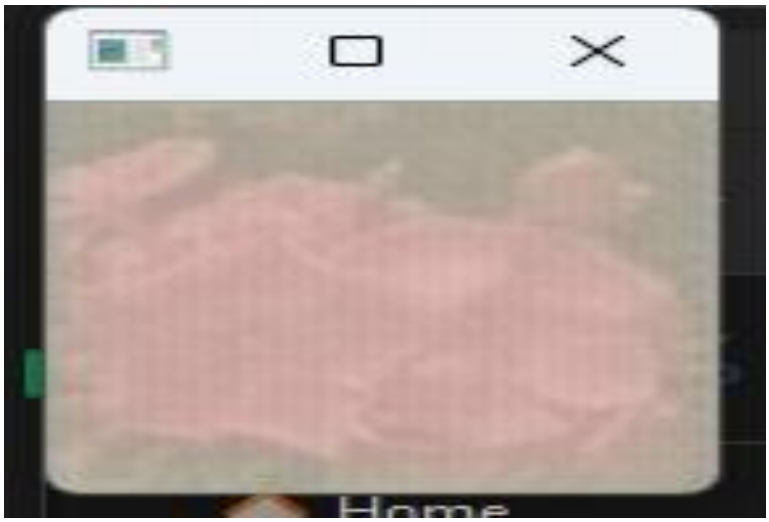
```
 Print(“Grayscale batch shape:”, gray_batch.shape)
    Print(“Color batch shape:”, color_batch.shape)
    Break
Def build_generator():
    Inputs = layers.Input(shape=(128, 128, 1))
    # Encoder (downsampling)
    X = layers.Conv2D(64, 4, strides=2, padding=’same’)(inputs)
    X = layers.LeakyReLU()(x)
    X = layers.Conv2D(128, 4, strides=2, padding=’same’)(x)
    X = layers.BatchNormalization()(x)
    X = layers.LeakyReLU()(x)
    X = layers.Conv2D(256, 4, strides=2, padding=’same’)(x)
    X = layers.BatchNormalization()(x)
    X = layers.LeakyReLU()(x)
    # Decoder (upsampling)
    X = layers.Conv2DTranspose(128, 4, strides=2, padding=’same’)(x)
    X = layers.BatchNormalization()(x)
    X = layers.ReLU()(x)
    X = layers.Conv2DTranspose(64, 4, strides=2, padding=’same’)(x)
    X = layers.BatchNormalization()(x)
    X = layers.ReLU()(x)
    X = layers.Conv2DTranspose(3, 4, strides=2, padding=’same’, activation=’tanh’)(x)
    Return models.Model(inputs=inputs, outputs=x)
Generator = build_generator()
Generator.summary()
Loss_object = tf.keras.losses.MeanSquaredError()
Generator_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
@tf.function
```

```
Def train_step(input_image, target_image):
    With tf.GradientTape() as gen_tape:
        Gen_output = generator(input_image, training=True)
        Gen_loss = loss_object(target_image, gen_output)3
    Generator_gradients = gen_tape.gradient(gen_loss, generator.trainable_variables)
    Generator_optimizer.apply_gradients(zip(generator_gradients,
generator.trainable_variables))
    Return gen_loss
Def train(dataset, epochs):
    For epoch in range(epochs):
        For input_image, target_image in dataset:
            Gen_loss = train_step(input_image, target_image)
        Print(f"Epoch {epoch + 1}, Generator Loss: {gen_loss}")
# Combine datasets
Dataset = tf.data.Dataset.zip((grayscale_dataset, color_dataset))
# Train the model
Train(dataset, epochs=50)
# Save the trained model
Generator.save('grayscale_to_color_model.h5')
Test
Import tensorflow as tf
Import cv2
Import numpy as np
Import matplotlib.pyplot as plt
Model = tf.keras.models.load_model('grayscale_to_color_model.h5')
Def load_and_preprocess_image(img_path, target_size=(128, 128)):
  Img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)    Img = cv2.resize(img,
target_size)
```

```
    Img = img / 255.0
    Img = np.expand_dims(img, axis=-1)
    Img = np.expand_dims(img, axis=0)
    Return img
Def colorize_image(model, grayscale_img):
    Colorized_img = model.predict(grayscale_img)
    Colorized_img = (colorized_img + 1) / 2.0
    Return colorized_img
Def save_or_display_image(colorized_img):
    Colorized_img = np.squeeze(colorized_img, axis=0)
    Colorized_img = cv2.cvtColor(colorized_img, cv2.COLOR_RGB2BGR)
    Cv2.imshow('Colorized Image', colorized_img)
    Cv2.waitKey(0)
    Cv2.destroyAllWindows()
Grayscale_img = load_and_preprocess_image('0002.png')
Colorized_img = colorize_image(model, grayscale_img)
Save_or_display_image(colorized_img)
```

Test image :

Output image:



2.Design an autoencoder-based system for collaborative filtering in recommendation systems. The autoencoder should learn to predict user preferences based on their past Interactions and those of similar users.

CODING:

```
Import pandas as pd

From sklearn.preprocessing import LabelEncoder

From keras.layers import Input, Dense

From keras.models import Model

Import numpy as np

# Load the dataset

Df = pd.read_csv('imdb_dataset.csv')

# Select the relevant features

Features = ['Movie Name', 'IMDB Rating', 'Genre', 'Cast', 'Director', 'Metascore']

Df = df[features]

# Encode categorical features

Le = LabelEncoder()

Df['Genre'] = le.fit_transform(df['Genre'])

Df['Cast'] = le.fit_transform(df['Cast'])

Df['Director'] = le.fit_transform(df['Director'])

# Create a user-item interaction matrix

User_item_matrix = pd.pivot_table(df, values='IMDB Rating', index='Movie Name', columns='Genre')

# Normalize the ratings

User_item_matrix = user_item_matrix.apply(lambda x: (x – x.mean()) / x.std())

# Define the autoencoder architecture

Input_dim = user_item_matrix.shape[1]

Latent_dim = 10

Input_layer = Input(shape=(input_dim,))

Encoder_layer = Dense(latent_dim, activation='relu')(input_layer)
```

```
Decoder_layer = Dense(input_dim, activation='sigmoid')(encoder_layer)
Autoencoder = Model(input_layer, decoder_layer)
# Define the encoder model
Encoder = Model(input_layer, encoder_layer)
# Define the decoder model
Decoder_input = Input(shape=(latent_dim,))
Decoder_output = autoencoder.layers[-1](decoder_input)
Decoder = Model(decoder_input, decoder_output)
# Compile the autoencoder
Autoencoder.compile(loss='binary_crossentropy', optimizer='adam')
# Compile the encoder and decoder
Encoder.compile(loss='binary_crossentropy', optimizer='adam')
Decoder.compile(loss='binary_crossentropy', optimizer='adam')
# Train the autoencoder
Autoencoder.fit(user_item_matrix, user_item_matrix, epochs=10, batch_size=32, verbose=2)
# Define a function to make recommendations
Def make_recommendations(movie_name, num_recs):
    # Get the movie's latent representation
    Movie_latent = encoder.predict(user_item_matrix.loc[movie_name].values.reshape(1, -1))
    # Reshape movie_latent to have shape (1981, 10)
    Movie_latent = movie_latent.reshape(1981, 10)
    # Calculate the cosine similarity between the input movie and all other movies
    Similarities = np.dot(user_item_matrix.values.T, movie_latent) / (
            Np.linalg.norm(user_item_matrix.values, axis=1) * np.linalg.norm(movie_latent,
axis=1))
    # Get the top-N recommended movies
    Recommended_indices = np.argsort(-similarities)[:num_recs]
```

```
  # Map the indices back to the original movie names
    Recommended_movies = user_item_matrix.index[recommended_indices]
    Return recommended_movies
# Test the recommendation function
Movie_name = ’10 Things I Hate About You’
Num_recs = 5
Recommended_movies = make_recommendations(movie_name, num_recs)
Print(recommended_movies)
```

Test:

```
Import pandas as pd
Import numpy as np
From sklearn.feature_extraction.text import TfidfVectorizer
From sklearn.metrics.pairwise import cosine_similarity
From ast import literal_eval
Def load_and_preprocess_data(file_path):
    Df = pd.read_csv(file_path)
    Df = df[[‘Movie Name’, ‘IMDB Rating’, ‘Genre’, ‘Director’, ‘Cast’]]
    Df[‘Genre’] = df[‘Genre’].fillna(‘’)
    Df[‘Director’] = df[‘Director’].fillna(‘’)
    Df[‘Cast’] = df[‘Cast’].fillna(‘’)
    Df[‘combined_features’] = df[‘Genre’] + ‘ ‘ + df[‘Director’] + ‘ ‘ + df[‘Cast’]
    Return df
Def create_tfidf_matrix(df):
    Tfidf = TfidfVectorizer(stop_words=’english’)
    Tfidf_matrix = tfidf.fit_transform(df[‘combined_features’])
    Return tfidf_matrix
Def compute_cosine_similarity(tfidf_matrix):
    Cosine_sim = cosine_similarity(tfidf_matrix)
```

```
    Return cosine_sim
Def get_recommendations(title, df, cosine_sim):
    Idx = df.index[df[‘Movie Name’] == title].tolist()[0]
    Sim_scores = list(enumerate(cosine_sim[idx]))
    Sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    Sim_scores = sim_scores[1:11]  # Top 10 similar movies
    Movie_indices = [i[0] for i in sim_scores]
    Recommendations = df[‘Movie Name’].iloc[movie_indices].tolist()
    Return recommendations
Def main():
    Df = load_and_preprocess_data(‘imdb_dataset.csv’)  # Replace with your file path
    Tfidf_matrix = create_tfidf_matrix(df)
    Cosine_sim = compute_cosine_similarity(tfidf_matrix)
    Movie_title = “Epic Movie”
    Recommendations = get_recommendations(movie_title, df, cosine_sim)
    Print(f”Recommendations for ‘{movie_title}’:”)
    For i, movie in enumerate(recommendations, 1):
        Print(f”{i}. {movie}”)
If __name__ == “__main__”:
    Main()
```

DATASET :

```
G:\python\recommender\.venv\Scripts\python.exe G:\python\recommender\main.py
Recommendations for 'Epic Movie':
1. Thank You for Smoking
2. Brother Bear
3. Kick-Ass
4. The Core
5. Alvin and the Chipmunks: The Squeakquel
6. I Heart Huckabees
7. Up in the Air
8. Jay and Silent Bob Strike Back
9. American Pie 2
10. The Blind Side

Process finished with exit code 0
```

TEST IMAGE:

OUTPUT IMAGE: