IR Assignment 2
Jeremiah Malsawmkima Rokhum
2021533

Question 1:
I have used the VGG16 model. This is the code which I used. Rotations, Resizing, Contrast change, Brightness change and geometrical orientation changes are done.

```
import numpy as np
import cv2
from io import BytesIO
import requests
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from keras.models import Model

# Load the ResNet50 model
base_model = VGG16(weights='imagenet', include_top=False)

# Create a new model with the VGG16 base model's output as the output
model = Model(inputs=base_model.input, outputs=base_model.output)

processed_images_list = []  # List to store processed images

for image_url in images:
   image_urls = eval(image_url)
   for url in image_urls:

       response = requests.get(url)
       image_bytes = BytesIO(response.content)
       image = cv2.imdecode(np.frombuffer(image_bytes.read(), np.uint8),
cv2.IMREAD_COLOR)
       if image is None:
           processed_images_list.append(np.zeros(100352,))
           print(f"Failed to read image from {image_url}")
           continue  # Skip to the next image if loading fails

       # Basic image pre-processing
       resized_image = cv2.resize(image, (224, 224))
       resized_image = cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB)  # Convert to
RGB

       # Apply random flips
       if np.random.rand() > 0.5:
           resized_image = np.fliplr(resized_image)
```

```
        # Apply contrast adjustments
        # Apply brightness and contrast adjustments
        alpha = 1.0 + np.random.uniform(-0.2, 0.2)  # Brightness adjustment factor
        beta = 50 + np.random.uniform(-20, 20)      # Contrast adjustment factor
        resized_image = cv2.convertScaleAbs(resized_image, alpha=alpha, beta=beta)

        # Apply geometric orientation (rotation)
        angle = np.random.uniform(-30, 30)
        rows, cols, _ = resized_image.shape
        rotation_matrix = cv2.getRotationMatrix2D((cols / 2, rows / 2), angle, 1)
        resized_image = cv2.warpAffine(resized_image, rotation_matrix, (cols, rows))

        resized_image = np.expand_dims(resized_image, axis=0)  # Add batch dimension
        resized_image = preprocess_input(resized_image)  # Preprocess the input for
VGG16

        # Normalize the images
        resized_image /= 255.0
        # Extract features using the VGG16 model
        features = model.predict(resized_image)
        flattened_features = features.flatten()
        # Append the features to the list
        processed_images_list.append(flattened_features)

        # # Display the pre-processed image (optional)
        # print(processed_images_list)
```

I have normalised using StandardScalar()

```
from sklearn.preprocessing import StandardScaler

# Initialize a scaler
scaler = StandardScaler()

# Suppose 'all_features' is a list or array of all extracted feature vectors
dataset_features_normalized = scaler.fit_transform(processed_images_list)
```

Then, I used pickle to dump the normalised features

```
import pickle
with open('dataset_features_normalized.pkl', 'wb') as f:
```

```
    pickle.dump(dataset_features_normalized, f)
```

Question 2:

```
from collections import defaultdict
import math



# Calculate term frequencies (TF)
def calculate_tf(texts):
    tfs = []
    for text in texts:
        tf_dict = defaultdict(int)
        for word in text.split():
            tf_dict[word] += 1
        # tf_dict[word] = tf_dict[word] / len(text.split())
        tfs.append(tf_dict)
    return tfs

# Calculate document frequencies (DF)
def calculate_df(tfs):
    df = defaultdict(int)
    for tf in tfs:
        for word in tf.keys():
            df[word] += 1
    return df

# Calculate TF-IDF
def calculate_tfidf(tfs, dfs, N, preprocess_texts):
    tfidf = []
    for i in range(len(tfs)):
        tfidf_dict = {}
        for word, count in tfs[i].items():
            tfidf_dict[word] = count * math.log10(N / dfs[word]) /
len(preprocessed_texts[i].split())
        tfidf.append(tfidf_dict)
    return tfidf

def calculate_idfs(dfs,N):
    idfs = defaultdict(int)
    for i in dfs:
```

```
        idfs[i] = math.log10(N / dfs[i])
    return idfs



tfs = calculate_tf(preprocessed_texts)
dfs = calculate_df(tfs)
N = len(preprocessed_texts)  # Number of documents
idfs = calculate_idfs(dfs,N)
tfidf_scores = calculate_tfidf(tfs, dfs, N, preprocessed_texts)
print(tfidf_scores)
with open('tfidf_scores.pkl', 'wb') as f:
    pickle.dump(tfidf_scores, f)
```

This is the function which I have used for term frequency, inverse document frequency and to calculate tfidf.

I have also dumped the scores in a pickle file

```
def cosine_similarity_between_two_entity(img1, img2):
    if(np.linalg.norm(img1) == 0 or np.linalg.norm(img2) == 0):
        return 0
    return np.dot(img1, img2) / (np.linalg.norm(img1) * np.linalg.norm(img2))
```

This is the function which I used for cosine similarity between two entity. It works for both images and documents
It's also being normalized.

```
img = input()
text = input()
img = preprocess_picture(img)
if img is None:
    print(f"Failed to read image from asdasd ad {img}")
    exit()
text = preprocess_text(text)
import ast
text = text.split()
def calculate_tf_for_new_corpus(texts):
    # tfs = []
    # for text in texts:
    #     tf_dict = defaultdict(int)
```

```
    #     for word in text.split():
    #         tf_dict[word] += 1
    #     # tf_dict[word] = tf_dict[word] / len(text.split())
    #     tfs.append(tf_dict)
    # return tfs
    tf_1 = defaultdict()
    for i in range(len(texts)):
        if texts[i] in tf_1:
            tf_1[texts[i]] += 1
        else:
            tf_1[texts[i]] = 1

    for i in tf_1:
        tf_1[i] = tf_1[i] / len(texts)
        tf_1[i] = tf_1[i] * idfs[i]
    return tf_1
# print(text)

Newtfs = calculate_tf_for_new_corpus(text)
# print(Newtfs)
#convert newtfs into list while maintaining it's index as it is in text
# Newtfs = list(Newtfs.values())
# print(Newfs)
newtfs = [Newtfs[i] for i in text]

Similarities_for_review = []
for i in tfidf_scores:
    Similarities_for_review.append((cosine_similarity_between_two_entity(newtfs, [i[j]
if j in i else 0 for j in text ])))

# get top 3 highest Similarities for review
Composite_Similarity = {}

def using_text_retrieval(top_3_review,df,Similarities_for_review, img):
    top_3_review = sorted(top_3_review)
    print(top_3_review)
    topCosinescorefortext = []
    image_urls = []
    review_text = []
    topCosinescoreforimg = []
    for i in top_3_review:
        topCosinescorefortext.append(Similarities_for_review[i])
```

```
        image_urls.append(df['Image'][i])
        a = df['Image'][i]
        review_text.append(df['Review Text'][i])
        for j in ast.literal_eval(a):
            print("Image link in top 3 text  ", j)
            print("img", img)
            list_urls = j
            image = preprocess_picture(list_urls)
            topCosinescoreforimg.append(cosine_similarity_between_two_entity(img,
image))
    image_urls = sorted(image_urls,reverse = True)
    review_text = sorted(review_text,reverse = True)
    topCosinescoreforimg = sorted(topCosinescoreforimg,reverse = True)
    topCosinescorefortext = sorted(topCosinescorefortext,reverse = True)
    print(topCosinescoreforimg)
    print(topCosinescorefortext)
    print("USING TEXT RETRIEVAL")
    for i in range(3):
        print("Top ",i+1," Review")
        print("Image URL: ",image_urls[i])
        print("Review Text: ",review_text[i])
        print("Cosine Similarity Score for Image: ",topCosinescoreforimg[i])
        print("Cosine Similarity Score for Text: ",topCosinescorefortext[i])
        print("Composite Similarity Score: ",(topCosinescoreforimg[i] +
topCosinescorefortext[i])/2)
        Composite_Similarity[(topCosinescoreforimg[i] + topCosinescorefortext[i])/2] =
[image_urls[i],review_text[i]]
        print("\n")



# #give me top 3 highest Similarities for review
top_3_review = sorted(range(len(Similarities_for_review)), key=lambda i:
Similarities_for_review[i])[-3:]

# print(top_3_review)
# print("USING TEXT RETRIEVAL")
# top_3_review_words = [reviews[i] for i in top_3_review]
# print(top_3_review)

# for i in top_3_review:
```

```
#     print(df['Image'][i])
#     print(df['Review Text'][i])

using_text_retrieval(top_3_review,df,Similarities_for_review, img)
```

This is the code that I have for retrieving using text.

```
def using_img_retrieval(img,top_3, Similarities,dataset_features_normalized,
tfidf_scores,text):
   topCosinescorefortext = []
   image_urls = []
   review_text = []
   topCosinescoreforimg = []
   for i in top_3:
       topCosinescoreforimg.append(Similarities[i])
   Composite_similarity_score = []
   for i in range(len(top_3)):
       top_3[i] = retrieve_index(top_3[i])
   for i in top_3:
       image_urls.append(df['Image'][i])
       review_text.append(df['Review Text'][i])
   for i in range(3):
       print("Top ",i+1," Review")
       print("Image URL: ",image_urls[i])
       a = review_text[i]
       a = preprocess_text(a)
       print("Review Text: ",review_text[i])
       print("Cosine Similarity Score for Image: ",topCosinescoreforimg[i])

       a1=calculate_tf_for_new_corpus(a.split())
       # print("text:",text)
       text1=calculate_tf_for_new_corpus((text  ))
       # print(a1)
       # print(text1)
       # print(a1, text1)
       Similarities_for_review = []
       # print(([text1[x] for x in text1], [a1[x] if x in a1 else 0 for x in text1 ]
))
       Similarities_for_review.append((cosine_similarity_between_two_entity([text1[x]
for x in text1], [a1[x] if x in a1 else 0 for x in text1 ] )))
       print("Cosine Similarity Score for Text: ",Similarities_for_review)
```

```
        Composite_similarity_score.append((Similarities_for_review[-1] +
topCosinescoreforimg[i])/2)
        print("Composite similarity score: ",(Similarities_for_review[-1] +
topCosinescoreforimg[i])/2)
        Composite_Similarity[Similarities_for_review[-1] + topCosinescoreforimg[i]/2] =
[image_urls[i],review_text[i]]
```

This is the code I have for retrieving images

I received a better score on my image retrieval. The reasons are given below:
Feature Representation: Images often have high-dimensional feature representations (e.g., color histograms, texture features, deep learning-based features like CNN activations), which can capture rich visual information. These features may encode various aspects of the image, such as shape, texture, and color distribution, making them highly discriminative for retrieval tasks.

Semantic Gap: Textual data might suffer from what is often referred to as the "semantic gap." This gap refers to the inherent difficulty in bridging the difference between low-level features (like individual words or phrases) and high-level semantics (the actual meaning conveyed by the text). Images, on the other hand, might have a smaller semantic gap, as visual features can directly capture certain visual aspects of the content.

Perceptual Content: Images inherently contain perceptual content that humans can interpret quickly and intuitively. Features extracted from images often encapsulate this perceptual content, making them suitable for similarity comparisons based on human perception. Textual similarity, on the other hand, might rely more on syntactic and semantic analysis, which can be more complex and prone to ambiguity.

Dimensionality of Feature Space: The feature space for images tends to be very high-dimensional due to the complexity and richness of visual content. While this might seem like a challenge, high dimensionality can actually be beneficial for similarity calculations, as it allows for finer discrimination between different images. Textual feature spaces can also be high-dimensional, but they may not capture the nuances of meaning as effectively as visual features capture the nuances of visual content.

Availability of Pre-trained Models: With the rise of deep learning, pre-trained models for image feature extraction, such as convolutional neural networks (CNNs), have become widely available and easily accessible. These models are often trained on large-scale datasets and can capture diverse visual patterns effectively. Conversely, while there are pre-trained models for text (e.g., word embeddings, language models), capturing semantic similarity in text can be more challenging due to the complexity of language.

The challenges I faced in text retrieval was:

1. **Out of vocabulary word: Traditional models struggle with out-of-vocabulary words, i.e., words not seen during training. Techniques like subword tokenization and character-level embeddings can help mitigate this issue.**
2. **Semantic Understanding: Textual content can be ambiguous, and understanding the semantic meaning accurately is challenging. Improvements can be made through the use of advanced natural language processing (NLP) techniques such as contextual embeddings and language models trained on large corpora.**

For image retrieval it was:

1. **Semantic gap: The semantic gap between low-level visual features and high-level semantic concepts makes retrieval challenging. Bridging this gap through techniques like deep learning-based feature extraction and end-to-end learning can improve retrieval accuracy.**
2. **High-dimensional feature spaces pose challenges for similarity calculations. Dimensionality reduction techniques (e.g., PCA, t-SNE) can help reduce computational complexity and improve retrieval efficiency.**
3. **It only worked well because the images were domain specific.**