NAME: VANSHAJ SHARMA

ROLLNO: MT23103

COURSE: M.TECH CSE

YEAR: FIRST YEAR

Libraries Used :-

```
import os
import csv
import cv2
import nltk
import string
import skimage
import requests
import numpy as np
import pandas as pd
from PIL import Image
from skimage import transform
from skimage import io
from keras.models import Model
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from skimage.exposure import adjust_gamma
from keras.applications.vgg16 import VGG16
from nltk.stem.porter import PorterStemmer
from sklearn.metrics.pairwise import cosine_similarity
from collections import OrderedDict
from io import BytesIO
from skimage.transform import rotate
import pickle
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')
from sklearn.preprocessing import normalize
import statistics
```

Data Structures used to maintain whole code

```
#****************** Data Structure ********************
# Array of Dictionaries

   #[ Dictionary
      # Product Id
      # Image URL
      # Image Features ]

# 2d matrix of terms vs documents


#*************** Array of Dictionaries *************

# Products
products = []

# images
images = []
```

Q1

```
#************************ Image pre processing ****************
def image_pre_processing(url):
    # loading image
    data = requests.get(url).content
    image_name = url.split("/")[-1:]
    f = open('images/'+ str(image_name[0]),'wb')
    f.write(data)
    f.close()

    image_Matrix = []
    try:
        image_Matrix = io.imread('images/'+ str(image_name[0]))
    except:
        return None


    # Convert BGR image to RGB
    image_rgb = cv2.cvtColor(image_Matrix, cv2.COLOR_BGR2RGB)

    # Image rotation parameter
    center = (image_rgb.shape[1] // 2, image_rgb.shape[0] // 2)
    angle = 30
```

```
    scale = 1

    # getRotationMatrix2D creates a matrix needed for transformation.
    rotation_matrix = cv2.getRotationMatrix2D(center, angle, scale)

    # We want matrix for rotation w.r.t center to 30 degree without scaling.
    image = cv2.warpAffine(image_rgb, rotation_matrix, (image_Matrix.shape[1],
image_Matrix.shape[0]))

    # brighten the image
    image = adjust_gamma(image,gamma=0.5,gain=1)

    # Resize image
    image = cv2.resize(image, (224, 224) )
    image = image.reshape(1,224,224,3)

    return image
```

a) For pre processing we used above function called image_pre_processing. It takes input the url of the image and returns the pre-processed image vector. Here we used request models get function to get image matrix or content of image. Then we convert image matrix into RGB from BGR because opne cv library requires RGB format of image colors. Then we use Open CV library to rotate , adjusting contrast and brightness and resizing. Here at last I am doing resizing because I wants all image of same dimesions.

```
#************************ Image features Extraction *****************************
def get_image_features(preprocessed_img):
    #***************** Feature extraction ******************
    # load the model
    feature_extraction_model = VGG16()

    # restructure the model
    model = Model(inputs=feature_extraction_model.inputs, outputs=feature_extraction_model.layers[-2].output)

    # extract features
    features = model.predict(preprocessed_img, verbose=0)
    features = normalize(features)

    return features
```

b) For image feature extraction we used above defined function called get_image_features. This function takes input the preprocessed image vector and returns the features extracted from it. In this function for feature extraction we used VGG16 a pre tarined model on image Net data set.

c) After extracting features using VGG16, we normalize these features using normalize function.

Q2

```
#************************ Text Pre-Processing ********************
def text_pre_processing(review):
    #******* Lowercase conversion *******
    lowercase_review = review.lower()

    #******* Tokenization ***************
    tokenize_review = word_tokenize(lowercase_review)

    #******* Punctuation Removal ********
    punctuation_free_review = []
    for token in tokenize_review:
        token = token.strip()
        translator = str.maketrans('', '', string.punctuation)
        newToken = token.translate(translator)
        punctuation_free_review.append(newToken)

    #****** Stopwords Removal ***********
    stopwords_free_review = []
    stopwordsList = stopwords.words("english")
    for word in punctuation_free_review:
        word = word.strip()
        if not word in stopwordsList:
            stopwords_free_review.append(word)

    #****** Stemming ********************
    porterStemmer = PorterStemmer()
    stemmed_review = []
    for word in stopwords_free_review:
        word = porterStemmer.stem(word)
        stemmed_review.append(word)

    #****** Lemmetization ***************
    wordnetLemmatizer = WordNetLemmatizer()
    Lemmtized_review = []
    for word in stemmed_review:
        word = wordnetLemmatizer.lemmatize(word)
        Lemmtized_review.append(word)

    #****** Whitespace removal **********
    whitespace_removed_review = []
    for word in Lemmtized_review:
        newToken = word.strip()
        if(newToken != ''):
```

```
            whitespace_removed_review.append(newToken)

    return whitespace_removed_review
```

a) For Text pre processing , we used function called text_pre_processing. This function takes input a review and returns the list of tokens generated from this review.

b)

```
def unique_terms(documents):
    uni_terms = []
    for document in documents:
        for term in document:
            if term not in uni_terms:
                uni_terms.append(term)
    return uni_terms
```

This above function takes the documents (the list of list of tokens of each review) and return the unique words among the all the reviews for further calculation of TF-IDF

```
def Term_Frequency(documents,terms_list):
    noOfDocuments = len(documents)
    noOfUniqueTerms = len(terms_list)

    term_doc_matrix = pd.DataFrame(np.zeros((noOfDocuments, noOfUniqueTerms)), columns=terms_list)

    for i in range(noOfDocuments):
        for term in documents[i]:
            term_doc_matrix[term][i] = term_doc_matrix[term][i] + 1

    return term_doc_matrix
```

The above function Term_Frequency is used to calculate term frequency for each term in the term list (calculated using unique_terms function) in respect of all the documents. This takes input the documents (the list of list of tokens of each review) and the term_list (which contains unique words) and it returns term_doc_matrix (a dataframe whose columns are representing each term in terms_list and row represents each document i.e product). This term_doc_matrix is having term frequncy of each term in column corresponding to each document in the row.

```python
def Inverse_Document_Frequency(documents,terms_list):
    idf = {}

    NoOfDocuments = len(documents)

    for term in terms_list:
        appearedInNoOfDoc = 0

        for document in documents:
            if term in document:
                appearedInNoOfDoc += 1

        # idf[term] =  np.log10(NoOfDocuments / appearedInNoOfDoc)

        if appearedInNoOfDoc > 0:
            idf[term] = np.log10(NoOfDocuments / appearedInNoOfDoc)
        else :
            idf[term] = 0

    return idf
```

The above function Inverse_Document_Frequency takes input the documents and terms_list and return the idf (list of inverse document frequency for each term in terms_list).

```python
def TF_IDF(term_doc_matrix,IDF,noOfDocuments, terms_list):
    tf_idf = term_doc_matrix.copy()

    for i in range(noOfDocuments):
        for term in terms_list:
            tf_idf[term][i] = tf_idf[term][i] * IDF[term]

    return tf_idf
```

The above function called TF_IDF used to calculate tf-idf. This function takes input the term_doc_matrix and IDF (that we calculated in above steps). In this function we multiply each row in term_doc_matrix with the IDF vector and produce tf_idf dataframe for all the rows (i.e for all the documents because each row represents each document and column represents term in wordlist).

Q3)

Helping functions:-

```
#************************* Cosine similarity *************************
# custom made cosine similarity
def customCosineSimilarity(vector1,vector2):
    v1 = vector1[0]
    v2 = vector2[0]
    if(len(v1) != len(v2)):
        return None

    product = np.dot(v1,v2)

    square_sum_vector_1 = 0
    for val in v1:
        square_sum_vector_1 += (val**2)

    square_sum_vector_2 = 0
    for val in v2:
        square_sum_vector_2 += (val**2)

    product_of_sqrt_square_sums_roots = np.sqrt(square_sum_vector_1) * np.sqrt(square_sum_vector_2)

    result = product/product_of_sqrt_square_sums_roots
    return result
```

This above function is custom made cosine similarity calculating function, This function takes two vectors vector1 and vector2 and returns the cosine similarity between these two vectors.

Here I used formula

$$cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

Here in function we did the same we took two vectors in input and calculated there dot product as **product** and calculated the L2 normalization as **product_of_sqrt_square_sums_roots.** Divided the **product** by **product_of_sqrt_square_sums_roots** to calculate the cosine similarity of two vectors named vector1 and vector2

```
def compute_cosine_similarity(Query,Doc):
    return customCosineSimilarity(Query, Doc)
```

This function above defined function compute_cosine_similarity uses cosine_similarity function to compute the cosine similairity between two vectors. It takes input Query vector and Doc vector and return the computed cosine similairty between the two vectors in range of [0 to 1].

```
#*************** Image Cosine similairity **********************
def ranking_basis_cosine_similarity_image(query_features):
    ranked_result = []

    for image in images:
        features = image["image_features"]
        score = compute_cosine_similarity(features,query_features)
        image["image_score"] = score


    images_d  = sorted(images, key=lambda d: d['image_score'],reverse=True)

    for image in images_d:
        product_temp = {}
        for product in products:
            if product["id"] == image["id"]:
                product_temp = product
                break
        product_temp["images_score"].append(image["image_score"])
        product_temp["image_ranked_urls"].append(image["image_url"])


    i = 0
    for image in images_d:
        if i == 3:
            break
        if image["id"] not in ranked_result:
            ranked_result.append(image["id"])
            i += 1

    return ranked_result
```

The above function is used to calculate the ranking of the images that are most similar to query image based on their features. For calculating similarity we use cosine similarity. This function takes input the features vector of the query image and it returns as output the ranked_result. These ranked result is having id of the top three ranked products based on the images features similairty matching.

```
def ranking_basis_cosine_similarity_text(tf_idf_d,tf_idf_q):
    cosine_similarity_dict = {}
    ranked_result = []

    for i in range(len(tf_idf_d)):
        X = np.array(tf_idf_d.iloc[i]).reshape(1,-1)
        # print(X)
        Y = [tf_idf_q]
        score = compute_cosine_similarity(X,Y)
        cosine_similarity_dict[products[i]["id"]] = score
        products[i]["text_score"] = score

    cosine_similarity_dict_d  = OrderedDict(sorted(cosine_similarity_dict.items(), key=lambda item: item[1], reverse=True))

    # print(cosine_similarity_dict_d)
    i = 0
    for key in cosine_similarity_dict_d:
        if( i == 3):
            break
        ranked_result.append(key)
        i += 1

    return ranked_result
```

This above function takes input the tf_idf vectors of all documents as tf_idf_d and tf_idf vector of the query's text review as tf_idf_q. It then calculates cosine similairity between the tf_idf of each document and tf-idf of query's text review and based on these cosine similarity scores to calculates the top three ranked product based on the text review cosine similarity matching.

These below functions that are ending with name Query are for calculating tf-idf and query image url image feature to find the ranking of products based these properties.

```
def term_frequency_Query(query,term_list):
    tf = {}
    for term in term_list:
        for q_term in query:
            if q_term == term:
                if q_term in tf:
                    tf[q_term] = tf[q_term] + 1
                else :
                    tf[q_term] = 1
        if term not in tf:
            tf[term] = 0
    return tf
```

This above function called term_function_Query is used to calculate the term frequency for the unique terms among all the documents based on the terms in query. This functions takes query (i.e list of tokens in the review text of query after pre-processing) and term_list (the unique terms among all documenst) as its parameter and returns the term frequency of all the terms in term_list in the query's review text.

```
def TF_IDF_QUERY(tf,IDF, terms_list):
    tf_idf = []
    # print(tf)
    # print(IDF)
    for term in terms_list:
        # print(term)
        tf_idf.append(tf[term] * IDF[term])
    return tf_idf
```

This function takes input the tf (i.e term frequency of terms in the query's review text but among all the terms in all the products text review) , IDF (i.e the inverse document frequency is for all the unique terms in all the products) and terms_list (i.e the unique terms among the review text of all the products). This function calculates the tf-idf from given tf and idf vectors according to the formula tf * idf by simply multiplying all the terms inside the vectors with each other and returns its result as the tf_idf.

```
#*************************** Input Output ******************************
def reading_CSV():
    with open('A2_Data.csv', mode ='r')as file:
        csvFile = csv.reader(file)
        i = 0
        for line in csvFile:
            if(i == 0):
                i += 1
                continue

            product = {}
            product["id"] = line[0]
            product["image_url"] = line[1].strip('][').split(', ')
            product["image_ranked_urls"] = []
            product["image_review"] = line[2]
            product["images_score"] = []
            product["text_score"] = 0
            product["composite_score"] = 0
            products.append(product)
            # print(product)
```

This above function is used to read csv file and fill an array products that was defined earlier in the discussion and this array product will be array of dictionaries where each dictionary represents each product that is it is having product id , image_url, image review and soon. This products array will be working as the main dataset that helps us getting properties corresponds to the product based on the product id.

```
def load_data_from_file(filename):
    file_data = open(filename, 'rb')
    data = pickle.load(file_data)
    file_data.close()
    return data

def save_data_to_file(filename, data):
    data_file= open(filename, 'wb')
    pickle.dump(data, data_file)
    data_file.close()
```

These above two functions load_data_from_file and save_data_to_file helps us in saving and loading data in and from pickle file.

```
def fill_images_feature_data():
    for product in products:
        print(product)
        for url in product["image_url"]:
            image = {}
            image["id"] = product["id"]
            image["image_url"] = url
            print(url)
            url = url[1:len(url)-1]
            pre_processed_img = image_pre_processing(url)
            if(pre_processed_img is None):
                continue
            feature = get_image_features(pre_processed_img)
            image["image_features"] = feature
            images.append(image)
```

This above function called fill_image_feature_data helps us to fill images array defined earlier in discussion under data structures section. The array images will be the array of dictionary where each dictionary represents one image which is having keys id , image_url and image_score etc. This functions traverse on the array of products and for each url in images_url for each product it create an image dictionary that is having keys image_features (i.e these are features for the image in url that are extracted using VGG16 model with the help of function get_image_features) and id (whose value is product id corresponds to this image) and image_url whose value will be the uel of this image for which we are making dictionary. After creating dictionary image for a url of a product, It will be appended in images array. So the images becomes the array of image which corresponds to single url.

a)

```
def mainProcess():
    reading_CSV()
```

→ First we prepare the products array.

```
documents = []

for product in products:
    document = text_pre_processing(product["image_review"])
    documents.append(document)

terms_list = unique_terms(documents)
term_doc_matrix = Term_Frequency(documents,terms_list)
idf = Inverse_Document_Frequency(documents,terms_list)
tf_idf = TF_IDF(term_doc_matrix,idf,len(documents),terms_list)
```

We then calculates the tf-idf for the text reviews corresponds to all the products. So for calculating tf-idf we first calculate the documents array that is having image-reviews (i.e array of tokens of text reviews) of all the products. Using this documents array we then calculate unique terms among all the text reviews of all the products as the terms_list using function called unique_terms. Then later using this terms_list and documents we calculate term frequency of each term in the terms list with respect to each product text review or we can say with respect to each document in documents as term_doc_matrix by using Term_Frequency. Later using documents and terms_list we will calculate the inverse document frequency of each term in terms_list with respect to each document in document array using function Inverse_Document_Frequency. Later using TF_IDF function we calculate tf-idf vectors for each document in documents array by passing parameters idf and term_doc_matrix calculated in earlier steps.

```
tf_idf_data_file = open('IF_IDF', 'wb')
pickle.dump(tf_idf, tf_idf_data_file)
tf_idf_data_file.close()

tf_idf = load_data_from_file('IF_IDF')
```

Then after calculated tf-idf for each documents we will store that result in file names IF_IDF using pickle library. The purpose of storing it is we can load the results directly without calculating each thing again and again.

Now after calculating tf_idf we will be calculating images array using fill images features

```
    # fill_images_feature_data()
```

Later we will save images array in file named images_features_V1 because it takes almost 2 hours of time to create images array.

```
# image_feature_data_file = open('image_features_V1', 'wb')
# pickle.dump(images, image_feature_data_file)
# image_feature_data_file.close()
```

```
global images
images_features_data = open('image_features_V1', 'rb')
images = pickle.load(images_features_data)
images_features_data.close()
```

Below we are taking input and calculating tf_idf and image features for input query text review and image url

```
print("Image and Text Query Input: ")
query_url = input("Image: ")
query_review = input("Review: ")

query_pre_processed_img = image_pre_processing(query_url)
query_img_feature = get_image_features(query_pre_processed_img)

query_review_doc = text_pre_processing(query_review)
query_tf = term_frequency_Query(query_review_doc,terms_list)
query_tf_idf = TF_IDF_QUERY(query_tf,idf,terms_list)
```

a) and b)

Using calculated image features and tf_idf for query we got vectors to compare with images and text reviews of the dataset using cosine similarity and find the top three ranked product based on text as well as image . Later we save the ranked results for text as well as image we got using similarity ranking functions

```
ranked_ids_image = ranking_basis_cosine_similarity_image(query_img_feature)
ranked_ids_text = ranking_basis_cosine_similarity_text(tf_idf,query_tf_idf)
ranked_ids_composite = ranking_based_composite_score(ranked_ids_image,ranked_ids_text)
```

```
save_data_to_file("ranked_id_images",ranked_ids_image)
save_data_to_file("ranked_ids_text",ranked_ids_text)
save_data_to_file("ranked_ids_composite",ranked_ids_composite)

ranked_ids_image = load_data_from_file("ranked_id_images")
ranked_ids_text = load_data_from_file("ranked_ids_text")
ranked_ids_composite = load_data_from_file("ranked_ids_composite")
```

Later we printed the ranked results .

First we printed the ranked results using image based retrieval. Here for calculating composite similarity score I take cosine similarity of the image which is used in ranking and text cosine similairty and find the composite similarity score.

For text based retrieval, for calculating composite cosine similarity score we take cosine similarity of image as mean of cosine similarity of all the images corresponds to given text for the product and cosine similarity of the text.

```
print("USING IMAGE RETRIEVAL:- ")
for id in ranked_ids_image:
    for product in products:
        if id == product["id"]:
            print("Image URL: " + str(product["image_url"]))
            print("Image Ranked URL: "+ str(product["image_ranked_urls"]))
            print("Review: "+ str(product["image_review"]))
            print("Cosine similarity of images: "+ str(product["images_score"][0]))
            print("Cosine similarity of text: "+ str(product["text_score"]))
            print("Composite score: "+ str(product["composite_score"]))
            print("\n")

print("USING TEXT RETRIEVAL:- ")
for id in ranked_ids_text:
    for product in products:
        if id == product["id"]:
            print("Image URL: " + str(product["image_url"]))
            print("Image Ranked URL: "+ str(product["image_ranked_urls"]))
            print("Review: "+ str(product["image_review"]))
            print("Cosine similarity of images: "+ str(product["images_score"]))
            print("Cosine similarity of images Average of URLs: "+ str(statistics.mean(product["images_score"])))
            print("Cosine similarity of image used in composite similarity: "+ str(statistics.mean(product["images_score"])))
            print("Cosine similarity of text: "+ str(product["text_score"]))
            print("Composite score: "+ str(product["composite_score"]))
            print("\n")
```

After calculating and printing composite similarity score in text and image based retrival, we print the ranking of all the products ranked above based on composite cosine similarity calculated previously

```
print("USING COMPOSITE RANK RETRIEVAL:- ")
for id in ranked_ids_composite:
    for product in products:
        if id == product["id"]:
            print("Image URL: " + str(product["image_url"]))
            print("Image Ranked URL: "+ str(product["image_ranked_urls"]))
            print("Review: "+ str(product["image_review"]))
            print("Cosine similarity of images: "+ str(product["images_score"]))
            print("Cosine similarity of images Average of URLs: "+ str(statistics.mean(product["images_score"])))
            print("Cosine similarity of image used in composite similarity, if it is the result of image based retrieval: "+ str(product["i
            print("Cosine similarity of image used in composite similarity, if it is the result of text based retrieval: "+ str(statistics.
            print("Cosine similarity of text: "+ str(product["text_score"]))
            print("Composite score: "+ str(product["composite_score"]))
            print("\n")
```

Output:-

```
Image and Text Query Input:
Image: https://images-na.ssl-images-amazon.com/images/I/71Y9QQZDpvL._SY88.jpg
Review: I love these great finish and quality
```

```
USING IMAGE RETRIEVAL:-
Image URL: ["'https://images-na.ssl-images-amazon.com/images/I/71Y9QQZDpvL._SY88.jpg'"]
Image Ranked URL: ["'https://images-na.ssl-images-amazon.com/images/I/71Y9QQZDpvL._SY88.jpg'"]
Review: I love these great finish and quality
Cosine similarity of images: 1.0
Cosine similarity of text: 1.0
Composite score: 1.0


Image URL: ["'https://images-na.ssl-images-amazon.com/images/I/712LuK8v2+L._SY88.jpg'"]
Image Ranked URL: ["'https://images-na.ssl-images-amazon.com/images/I/712LuK8v2+L._SY88.jpg'"]
Review: Hot damn!  I replaced the original neck position humbucker on my 1980 Les Paul (which I bought new) with a SPH90 Phat Cat.  What a difference!!  It is MUCH louder, and brighter, and all around sweeter and more tuneful that the original.  I used to rarely use the neck pickup alone.  Now I'll use it almost exclusively (though mixing in a bit of the bridge humbucker is sweet too).  I'm very glad I bought this and made the switch.  P.S. - I've had no problem with hum from this pickup - if you do, your problem is probably somewhere in your amp.
Cosine similarity of images: 0.69706404
Cosine similarity of text: 0.0
Composite score: 0.3485320210456848


Image URL: ["'https://images-na.ssl-images-amazon.com/images/I/61TrBfb23gL._SY88.jpg'", "'https://images-na.ssl-images-amazon.com/images/I/713koU-p-sL._SY88.jpg'"]
Image Ranked URL: ["'https://images-na.ssl-images-amazon.com/images/I/713koU-p-sL._SY88.jpg'", "'https://images-na.ssl-images-amazon.com/images/I/61TrBfb23gL._SY88.jpg'"]
Review: Rock solid, and everything you need to hang your instrument.  Someone's review stated that they were ugly.  Seriously?  I used to buy these for about $25 a piece, and they all look the same.  These get the job done at about 1/3 of the price of most music shops, and if I can trust my Gibson and Martin guitars on them, they are solid.
Cosine similarity of images: 0.6873503
Cosine similarity of text: 0.0
Composite score: 0.3436751365661621
```

```
USING TEXT RETRIEVAL:-
Image URL: ["'https://images-na.ssl-images-amazon.com/images/I/71Y9QQZDpvL._SY88.jpg'"]
Image Ranked URL: ["'https://images-na.ssl-images-amazon.com/images/I/71Y9QQZDpvL._SY88.jpg'"]
Review: I love these great finish and quality
Cosine similarity of images: [1.0]
Cosine similarity of images Average of URLs: 1.0
Cosine similarity of image used in composite similarity: 1.0
Cosine similarity of text: 1.0
Composite score: 1.0


Image URL: ["'https://images-na.ssl-images-amazon.com/images/I/71Du-FXA7hL._SY88.jpg'", "'https://images-na.ssl-images-amazon.com/images/I/71e+zuserVL._SY88.jpg'", "'https://images-na.ssl-images-amazon.com/images/I/71a7VdFNl0L._SY88.jpg'", "'https://images-na.ssl-images-amazon.com/images/I/71KFVLamjhL._SY88.jpg'"]
Image Ranked URL: ["'https://images-na.ssl-images-amazon.com/images/I/71a7VdFNl0L._SY88.jpg'", "'https://images-na.ssl-images-amazon.com/images/I/71e+zuserVL._SY88.jpg'", "'https://images-na.ssl-images-amazon.com/images/I/71KFVLamjhL._SY88.jpg'", "'https://images-na.ssl-images-amazon.com/images/I/71Du-FXA7hL._SY88.jpg'"]
Review: Sticks on great. It also holds on extremely tight. And when i take it of my finish is perfect
Cosine similarity of images: [0.43411082, 0.42316037, 0.41427967, 0.34002534]
Cosine similarity of images Average of URLs: 0.40289405
Cosine similarity of image used in composite similarity: 0.40289405
Cosine similarity of text: 0.29223288481044607
Composite score: 0.34756346734666743


Image URL: ["'https://images-na.ssl-images-amazon.com/images/I/61hEWhycZ9L._SY88.jpg'"]
Image Ranked URL: ["'https://images-na.ssl-images-amazon.com/images/I/61hEWhycZ9L._SY88.jpg'"]
Review: Works Great ...was easy to Install ....Love it
Cosine similarity of images: [0.4249453]
Cosine similarity of images Average of URLs: 0.4249453
Cosine similarity of image used in composite similarity: 0.4249453
Cosine similarity of text: 0.286925373786213
Composite score: 0.35593533432161906
```

```
USING COMPOSITE RANK RETRIEVAL:-
Image URL: ["'https://images-na.ssl-images-amazon.com/images/I/71Y9QQZDpvL._SY88.jpg'"]
Image Ranked URL: ["'https://images-na.ssl-images-amazon.com/images/I/71Y9QQZDpvL._SY88.jpg'"]
Review: I love these great finish and quality
Cosine similarity of images: [1.0]
Cosine similarity of images Average of URLs: 1.0
Cosine similarity of image used in composite similarity, if it is the result of image based retrieval: 1.0
Cosine similarity of image used in composite similarity, if it is the result of text based retrieval: 1.0
Cosine similarity of text: 1.0
Composite score: 1.0


Image URL: ["'https://images-na.ssl-images-amazon.com/images/I/61hEWhycZ9L._SY88.jpg'"]
Image Ranked URL: ["'https://images-na.ssl-images-amazon.com/images/I/61hEWhycZ9L._SY88.jpg'"]
Review: Works Great ...was easy to Install ....Love it
Cosine similarity of images: [0.4249453]
Cosine similarity of images Average of URLs: 0.4249453
Cosine similarity of image used in composite similarity, if it is the result of image based retrieval: 0.4249453
Cosine similarity of image used in composite similarity, if it is the result of text based retrieval: 0.4249453
Cosine similarity of text: 0.286925373786213
Composite score: 0.35593533432161906
```

```
Image URL: ["'https://images-na.ssl-images-amazon.com/images/I/712LuK8v2+L._SY88.jpg'"]
Image Ranked URL: ["'https://images-na.ssl-images-amazon.com/images/I/712LuK8v2+L._SY88.jpg'"]
Review: Hot damn!  I replaced the original neck position humbucker on my 1980 Les Paul (which I bought new) with a SPH90 Phat Cat.  What a difference!!  It
is MUCH louder, and brighter, and all around sweeter and more tuneful that the original.  I used to rarely use the neck pickup alone.  Now I'll use it almos
t exclusively (though mixing in a bit of the bridge humbucker is sweet too).  I'm very glad I bought this and made the switch.  P.S. - I've had no problem w
ith hum from this pickup - if you do, your problem is probably somewhere in your amp.
Cosine similarity of images: [0.69706404]
Cosine similarity of images Average of URLs: 0.69706404
Cosine similarity of image used in composite similarity, if it is the result of image based retrieval: 0.69706404
Cosine similarity of image used in composite similarity, if it is the result of text based retrieval: 0.69706404
Cosine similarity of text: 0.0
Composite score: 0.3485320210456848


Image URL: ["'https://images-na.ssl-images-amazon.com/images/I/71Du-FXA7hL._SY88.jpg'", "'https://images-na.ssl-images-amazon.com/images/I/71e+zuserVL._SY88
.jpg'", "'https://images-na.ssl-images-amazon.com/images/I/71a7VdFNl0L._SY88.jpg'", "'https://images-na.ssl-images-amazon.com/images/I/71KFVLamjhL._SY88.jpg
'"]
Image Ranked URL: ["'https://images-na.ssl-images-amazon.com/images/I/71a7VdFNl0L._SY88.jpg'", "'https://images-na.ssl-images-amazon.com/images/I/71e+zuserV
L._SY88.jpg'", "'https://images-na.ssl-images-amazon.com/images/I/71KFVLamjhL._SY88.jpg'", "'https://images-na.ssl-images-amazon.com/images/I/71Du-FXA7hL._S
Y88.jpg'"]
Review: Sticks on great. It also holds on extremely tight. And when i take it of my finish is perfect
Cosine similarity of images: [0.43411082, 0.42316037, 0.41427967, 0.34002534]
Cosine similarity of images Average of URLs: 0.40289405
Cosine similarity of image used in composite similarity, if it is the result of image based retrieval: 0.43411082
Cosine similarity of image used in composite similarity, if it is the result of text based retrieval: 0.40289405
Cosine similarity of text: 0.29223288481044607
Composite score: 0.34756346734666743
```

```
Image URL: ["'https://images-na.ssl-images-amazon.com/images/I/61TrBfb23gL._SY88.jpg'", "'https://images-na.ssl-images-amazon.com/images/I/713koU-p-sL._SY88
.jpg'"]
Image Ranked URL: ["'https://images-na.ssl-images-amazon.com/images/I/713koU-p-sL._SY88.jpg'", "'https://images-na.ssl-images-amazon.com/images/I/61TrBfb23g
L._SY88.jpg'"]
Review: Rock solid, and everything you need to hang your instrument.  Someone's review stated that they were ugly.  Seriously?  I used to buy these for abou
t $25 a piece, and they all look the same.  These get the job done at about 1/3 of the price of most music shops, and if I can trust my Gibson and Martin gu
itars on them, they are solid.
Cosine similarity of images: [0.6873503, 0.43994915]
Cosine similarity of images Average of URLs: 0.5636497
Cosine similarity of image used in composite similarity, if it is the result of image based retrieval: 0.6873503
Cosine similarity of image used in composite similarity, if it is the result of text based retrieval: 0.5636497
Cosine similarity of text: 0.0
Composite score: 0.3436751365661621
```

b. Observe which out of the two retrieval techniques gives a better similarity score and argue the reason.

Image based retrieval is better then text based retrieval because in image based retrieval we are using VGG16 here to extract feature for the image. Due to the usage of VGG16, no two images can have the same features because while extracting the features VGG16 consider many parameters. Also for similarity measure we are using cosine similairity which gives value in range of [0 to 1]. Consine similarity gives value closest to 1, when values in two vectors are mostly similar. Cosine similarity gives 1 when two vectors are equal. So VGG16 helps in differentiating images by giving different features for the different images and cosine similarity helps in identifying most similar vector. But on the other hand in text based

retrieval, we used tf-idf and cosine similarity to rank our results. In case of tf-idf, it doesn't consider the context of sentence, It only cosider the terms frequency and inverse document frequency, so due to which it can rank the document higher which is having the most words matching to query and lower to which is having less words matching to query. But what about the synonyms and same word with different spelling like color and colour, so in that case tf-idf gives bad results. So from the discussion we can say that image based retrieval is better than text based retrival because image based cosinder some context but text based not it is fully dependent on words. The same can be viewed in output mentioned in output section that image based retrieved most relevent documents than text based.

c. Discuss the challenges faced and potential improvements in the retrieval process.

Challenges faced:-

This method is slow because we have to first fetch the image from url and then calculate features for all the images first then find the cosine similarity for them and rank them.

Here I am maintaining the images array which is an array of dictionary that contains each image in form of dictionary having all the details about it like features , id and url.

So making this array by finding for the all the images was time taking for this small dataset of only 1000 rows but if we do the same for larger dataset then this will be drastically slow.

Also here, we have to calculate tf-idf for the review of all the products and then find the cosine similarity of these calculated tf-df with the tf-idf of review of the query. This also takes about 1 – 2 minutes for this small dataset of 1000 rows but if we do the same for larger dataset then we will face drastically slow behaviour.

This proposed method don't consider the length of the document.

Here we have to calculate the cosine similarity for the image features of all the images of each product to find the top 3 similar products based on image similarity. Also we have to calculate cosine similarity for tf-df of the review corresponds to each product. This is a very time consuming task. Since for efficient information retrieval system, system should be fast and responsive to the users. But the calculating cosine similarity for all the products property based on which we want to rank documents for the query is a bottleneck for the information retrieval system which are calculating cosine similarity for the property of each product.

Improvements:-

1) Probablistic rank retreival:-

Probabilistic ranking with feedback query improvement will be faster than proposed solution because it takes into consideration the user need and find the most relevent documents according to user need by considering direct and indirect feedback. Also the ranking can be done by evaluating the results using NDCG which consider the position of the ranked results. Probalistic ranking can be done using okapi BM25. Probalistic ranking also rank the results based on the users need by anaylizing users behaviour. It is faster than proposed solution because it don't required to calculate image features or tf-idf for text, it only requires the user's feedback.

2) Okapi BM25 also consider the document length while ranking.

3) To avoid calculating cosine similarity for all the products property , we can use WAND scoring to prune the computation of cosine similarity for irrelenvent documents.

We can speed the computation of cosine similarity by pruning

We can also do index elimination that is consider only documents that are having at least one query term

Can also do use the variation of index elimination like high tf-idf term only and documents containing many query terms,