IR ASSIGNMENT - 02
REPORT

Question - 01

a. Use basic image pre-processing techniques as altering contrast, resizing, geometrical orientation, random flips, brightness and exposure or any other relevant operation.
b. Use a pre-trained Convolutional Neural Network Architecture as ResNet, VGG16, Inception-v3, MobileNet (or any other CNN , preferably pre-trained on ImageNet Dataset), to extract relevant features from the images in the given training Set. Choose only one of the networks for your final pipeline.
c. Normalize the extracted features.

Part-a

```
# Define transformations
preprocess = transforms.Compose([
    transforms.ToPILImage(),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.2),  # Adjust brightness, contrast, saturation, and hue
    transforms.RandomVerticalFlip(p=0.5),  # Random vertical flip with 50% probability
    transforms.RandomRotation(30),  # Random rotation within the range of -30 to 30 degrees
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])
```

- The code begins by importing necessary libraries such as pandas for data manipulation, urllib for fetching images from URLs, OpenCV (cv2) for image processing, NumPy for numerical operations, and PyTorch's torchvision for image transformations.

- It defines a function preprocess_image(url) to process images fetched from URLs. Within this function, it applies a series of transformations using PyTorch's **torchvision.transforms.Compose()**, including color adjustments, random vertical flipping, random rotation, resizing, and conversion to a PyTorch tensor.

- Iterating through URLs extracted from a DataFrame column named 'Image', it preprocesses each image, handling exceptions if any occur during processing, and stores the successfully processed images in a list named **processed_images.**

- After processing all images, it stacks them into a single numpy array using **torch.stack()** to facilitate further analysis. Finally, it prints the shape of the processed images array to verify the dimensions.

Part- b

```python
resnet = models.resnet50(pretrained=True)
resnet.eval()  # Set the model to evaluation mode

# Function to extract features using ResNet
def extract_features(image):
    with torch.no_grad():
        image=image.unsqueeze(0)
        features = resnet(image)
    return features
```

```python
# Initialize an empty list to store all features
all_features = []

# Iterate over each row in the DataFrame
for _, row in df.iterrows():
    # Iterate over each image URL in the row
    for url in eval(row['Image']):
        # Preprocess the image
        image = preprocess_image(url)
        if image is not None:
            # Extract features
            features = extract_features(image)
            features=np.array(features)
            all_features.append([row['r_id'], url, features])

# for features in all_features normalize the features

for i, element in enumerate(all_features):
    features = element[2]
    # Normalize the features
    features = features - np.mean(features) / np.std(features) + 0.01
    all_features[i][2] = features
print(all_features[0][2].shape)
print(all_features)
```

- The code initializes a ResNet-50 model pretrained on ImageNet and sets it to evaluation mode. It defines a function **extract_features(image)** to extract features from an image using the ResNet model.

- It iterates over each row in the DataFrame and for each image URL in the row, preprocesses the image and extracts features using the defined function. Extracted features, along with relevant information like **'r_id'** and URL, are stored in a list named **all_features**.

- After extracting features for all images, it iterates through all_features, normalizes the features by subtracting the mean and dividing by the standard deviation, and stores the normalized features back into the list.

- Finally, it prints the shape of features for the first element in all_features and the entire all_features list, which contains information about **'r_id'**, **URL,** and normalized features for each image.

Question - 02

2. Text Feature Extraction [25]
    a. Implement relevant pre-processing techniques as Lower-Casing, Tokenization, removing punctuations, Stop Word Removal, Stemming and Lemmatization on the given text reviews in the data
    b. Calculate the Term Frequency-Inverse Document Frequency (TF-IDF) scores for the textual reviews.

```python
def preprocess_text(text):
    if isinstance(text, str):
        # Lowercasing
        text = text.lower()

        # Remove punctuation
        text = ''.join([char for char in text if char not in string.punctuation])

        # Tokenization
        tokens = word_tokenize(text)

        # Stop word removal
        stop_words = set(stopwords.words('english'))
        tokens = [word for word in tokens if word not in stop_words]

        # Stemming
        stemmer = PorterStemmer()
        tokens = [stemmer.stem(word) for word in tokens]

        # Lemmatization
        lemmatizer = WordNetLemmatizer()
        tokens = [lemmatizer.lemmatize(word) for word in tokens]

        return tokens
    else:
        return []  # Return an empty list for non-string inputs
```

```
word_to_idx = dict()

for doc in data["Review Text"]:
    for word in doc:
        if word not in word_to_idx:
            word_to_idx[word] = len(word_to_idx)

idx_to_word = dict()
for k, v in word_to_idx.items():
    idx_to_word[v] = k

tfidf_mat = np.zeros(((len(word_to_idx), 1000)), dtype=float)

# store frequency of each word in tfidf[word][document]

for i in range(1000):
    doc =  data["Review Text"][i]
    for word in doc:
        tfidf_mat[word_to_idx[word]][i] += 1

# divide tf by frequency across column
tfidf_mat = tfidf_mat/(0.00000001 + np.sum(tfidf_mat, axis=0))

# calculate and store idf value for each word

idf_values = {}

for word, idx in word_to_idx.items():

    num_docs = 0
    for doc in data["Review Text"]:
        if word in doc:
            num_docs += 1

    idf_value = (1 + math.log( (1000) / (1 + num_docs) ))
    idf_values[idx] = idf_value

    tfidf_mat[idx] *= idf_value

print(tfidf_mat)
```

- It first preprocesses the text data using the preprocess_text function and doest the following :
  tokenizing, lowercasing, removing punctuation, stop words, and applying stemming and lemmatization.

- It then constructs a word-to-index mapping (word_to_idx) and an index-to-word mapping (idx_to_word) to uniquely identify each word in the documents.

- A TF-IDF matrix (tfidf_mat) is initialized as a numpy array with dimensions (number of unique words, number of documents), initialized with zeros. For each document in the first 1000 documents (based on the loop), it iterates through the words and calculates the term frequency (TF) of each word in that document, updating the corresponding entry in the TF-IDF matrix.

- The TF-IDF values are then normalized by dividing each column by the sum of its elements, ensuring that the TF-IDF values are on a comparable scale.

- IDF (Inverse Document Frequency) values are calculated for each word based on the number of documents containing that word. The IDF values are applied to the TF-IDF matrix for each word.

- The final TF-IDF matrix is printed, representing the weighted importance of each word in each document, where higher values indicate greater importance.

Question - 03

3. Image Retrieval and Text Retrieval [25]
    a. For the input (image, review) pair, find the most similar images (*preferably your top three*) to your input based on extracted image features/embeddings using a similarity measure (cosine similarity) and a suitable data-structure.
    b. For the input (image, review) pair, find the most similar reviews (*preferably your top three*) to your input review based on TF-IDF scores using a similarity measure (Cosine Similarity)
    c. Save your results using Python's **pickle** module to save and load your results.

- For question 3 I used the same functions I used before for taking input and preprocessing the image and text
- The used the pkl files to fetch the images
- The functions written will use cosine similarity to fetch the images from the data base

```
top_three = []

# Calculate cosine similarity for each product
for i in range(len(image_features)):
    product_id = image_features[i][0]
    product_url = image_features[i][1]
    product_features = image_features[i][2]

    # Calculate cosine similarity
    cosine_similarity = np.dot(extract_preprocess.flatten(), product_features.flatten()) / (np.linalg.norm(extract_preprocess)

    # Append product ID, product URL, and cosine similarity
    top_three.append([product_id, product_url, cosine_similarity])

# Sort the list based on cosine similarity (in descending order)
top_three.sort(key=lambda x: x[2], reverse=True)

# Get the top three entries
top_three = top_three[:3]

# Print the top three entries with product ID, product URL, and cosine similarity
print("Top three cosine similarities:")
for entry in top_three:
    print(f"Product ID: {entry[0]}, Product URL: {entry[1]}, Cosine Similarity: {entry[2]}")
```

Question - 04

4. Combined Retrieval (Text and Image)
    a. Get a composite similarity score (average) for the pairs generated in 3a) and 3b)
    b. Rank the pairs based on the composite similarity score.

- For this, I took the average of the outputs from the image review cosine similarity and retrieved the images

```python
# caluclate the composit score of the product of top_three_review and top_three
# Create a list to store top three entries
top_three_composite = []
for i in range(3):
    # take average of the cosine similarity in top_three_review and top_three
    composite_score = (top_three_review[i][2]+top_three[i][2])/2
    top_three_composite.append((top_three[i][0],top_three[i][1],composite_score))
print("Top three composite scores:")
for entry in top_three_composite:
    print(f" Product URL: {entry[1]}, Cosine Similarity: {entry[2]}")
```

Question - 05

5. Results and Analysis
    a. Present the top-ranked (image, review) pairs along with the cosine similarity scores.
    b. Observe which out of the two retrieval techniques gives a better similarity score and argue the reason.
    c. Discuss the challenges faced and potential improvements in the retrieval process.

a.

```
USING IMAGE RETRIVAL
Top three cosine similarities:
Top three cosine similarities:
Product ID: 590, Product URL: https://images-na.ssl-images-amazon.com/images/I/61jFeizn-KL._SY88.jpg, Cosine Similarity: 0.9531001448631287
Product ID: 2236, Product URL: https://images-na.ssl-images-amazon.com/images/I/71Sj6Aza1DL._SY88.jpg, Cosine Similarity: 0.9238896369934082
Product ID: 3647, Product URL: https://images-na.ssl-images-amazon.com/images/I/71eMcAlg-gL._SY88.jpg, Cosine Similarity: 0.919009804725647
USING REVIEW RETRIVAL
Top three cosine similarities:
 Product URL: https://images-na.ssl-images-amazon.com/images/I/71fSAGI0MfL._SY88.jpg, Cosine Similarity: 0.43938225507736206
 Product URL: https://images-na.ssl-images-amazon.com/images/I/61i4V8xiOSL._SY88.jpg, Cosine Similarity: 0.42433175444602966
 Product URL: https://images-na.ssl-images-amazon.com/images/I/71mjJmOZYfL._SY88.jpg, Cosine Similarity: 0.4168797433376312
USING COMPOSITE RETRIVAL
Top three composite scores:
 Product URL: https://images-na.ssl-images-amazon.com/images/I/61jFeizn-KL._SY88.jpg, Cosine Similarity: 0.6962411999702454
 Product URL: https://images-na.ssl-images-amazon.com/images/I/71Sj6Aza1DL._SY88.jpg, Cosine Similarity: 0.6741107106208801
 Product URL: https://images-na.ssl-images-amazon.com/images/I/71eMcAlg-gL._SY88.jpg, Cosine Similarity: 0.6679447889328003
```

b.

In this scenario, leveraging image cosine similarity yields superior results due to its ability to comprehensively assess the visual composition of images. By utilizing features extracted from CNNs, it effectively captures intricate visual elements such as objects, shapes, and patterns, facilitating accurate recognition. Furthermore, the inclusion of augmented data accounts for variations, enhancing the technique's robustness. Conversely, employing TF-IDF cosine similarity for text typically yields minimal cosine similarity scores. This is primarily attributed to the vast vocabulary size, leading to

sparse representations that struggle to adequately capture semantic similarities between texts.

c.
Scalability poses a significant challenge with the use of CNNs for feature extraction, particularly evident in the considerable time required to process augmented image datasets. Scaling up to even larger datasets would demand extensive computational resources and time, presenting a formidable obstacle. Additionally, issues regarding data quality and noise introduce further complexities. Suboptimal data quality and noisy datasets can markedly impact the accuracy of results. Employing advanced data cleaning and preprocessing methodologies, coupled with the integration of hybrid scores from multi-modal retrieval techniques, holds promise for enhancing outcomes in such scenarios.

Question - 06

Overall output:

```
USING IMAGE RETRIVAL
Top three cosine similarities:
Top three cosine similarities:
Product ID: 590, Product URL: https://images-na.ssl-images-amazon.com/images/I/61jFeizn-KL._SY88.jpg, Cosine Similarity: 0.9531001448631287
Product ID: 2236, Product URL: https://images-na.ssl-images-amazon.com/images/I/71Sj6Aza1DL._SY88.jpg, Cosine Similarity: 0.9238896369934082
Product ID: 3647, Product URL: https://images-na.ssl-images-amazon.com/images/I/71eMcAlg-gL._SY88.jpg, Cosine Similarity: 0.919009804725647
USING REVIEW RETRIVAL
Top three cosine similarities:
 Product URL: https://images-na.ssl-images-amazon.com/images/I/71fSAGI0MfL._SY88.jpg, Cosine Similarity: 0.43938225507736206
 Product URL: https://images-na.ssl-images-amazon.com/images/I/61i4V8xiOSL._SY88.jpg, Cosine Similarity: 0.42433175444602966
 Product URL: https://images-na.ssl-images-amazon.com/images/I/71mjJmOZYfL._SY88.jpg, Cosine Similarity: 0.4168797433376312
USING COMPOSITE RETRIVAL
Top three composite scores:
 Product URL: https://images-na.ssl-images-amazon.com/images/I/61jFeizn-KL._SY88.jpg, Cosine Similarity: 0.6962411999702454
 Product URL: https://images-na.ssl-images-amazon.com/images/I/71Sj6Aza1DL._SY88.jpg, Cosine Similarity: 0.6741107106208801
 Product URL: https://images-na.ssl-images-amazon.com/images/I/71eMcAlg-gL._SY88.jpg, Cosine Similarity: 0.6679447889328003
```