IR Assignment 2

Tanuj Kamboj
2021497

Code Chunks marked in .pynb file:

1. The code snippet provided is intended to perform image feature extraction using the VGG16 model. However, it's incomplete, and the part responsible for image feature extraction is missing. This process typically involves loading images, preprocessing them, passing them through the VGG16 model, and extracting features from a specific layer. Once the image features are extracted, they can be further processed or used for various tasks such as image classification, object detection, or similarity comparison.
   The code snippet includes a step to upload a CSV file using the files.upload() function from the Google Colab library. This function allows users to upload files from their local system directly to a Google Colab notebook environment.

2. This code reads data from a CSV file named "A2_Data.csv" and processes it to fetch and save images. The CSV file contains information about images, including their URLs. The script downloads these images from the URLs and saves them to a folder named "extracted_images" in the local file system. Each image is saved with a unique filename based on its index in the CSV file.
3. This code extracts features from images using the pre-trained ResNet50 model. It takes a folder path as input, reads images from that folder, processes them, and extracts features using ResNet50. These features are then normalized and stored in a dictionary where the keys are file names and values are the corresponding normalized feature vectors. Finally, the normalized features are saved to a file named "image_features_resnet.pkl" using Pickle for later use.
4. This code preprocesses text data from a CSV file ("A2_Data.csv") and calculates TF-IDF (Term Frequency-Inverse Document Frequency) scores for each term in the text documents. TF-IDF is a numerical statistic that reflects the importance of a term in a document relative to a collection of documents. The preprocessing steps include lowercasing, removing special characters, tokenizing, removing stopwords, stemming, and lemmatization. After preprocessing, TF (Term Frequency) and DF (Document Frequency) are

calculated for each term. TF-IDF scores are then computed using the formula:
TF-IDF = TF × IDF
where TF is the term frequency in the document. IDF is the inverse document frequency, calculated as the logarithm of the total number of documents divided by the document frequency of the term. The calculate_tf_idf function returns a dictionary containing TF-IDF scores for each document. The normalize_tf_idf function normalizes the TF-IDF scores to a range of 0 to 1. Finally, the TF-IDF scores for the first document are printed. If any document has no terms after preprocessing (as in document 828), it's assigned an empty TF-IDF dictionary.

5. This code downloads an image from a given URL and saves it to a folder named "input_images" with the filename "input_image1.jpg". If the folder doesn't exist, it creates one. Here's a breakdown:
Function Definition: Defines a function download_image to download and save an image from a URL.
Folder Creation: Checks if the "input_images" folder exists. If not, it creates the folder.
Image Download: Downloads the image from the provided URL using the download_image function and saves it with the filename "input_image1.jpg" in the "input_images" folder.

6. This code snippet extracts the top 3 file names with the highest cosine similarity from a set of image features compared to the input image features. It then extracts data from rows in a DataFrame based on the row numbers associated with these filenames. Here's a summary of what the code does: Extract Image Features: The function extract_image_features extracts features from an input image using a pre-trained ResNet50 model. These features are then normalized. Compute **Cosine Similarity**: The cosine_similarity function calculates the cosine similarity between two vectors. Find **Top 3 Similarities:** The top_3_similarity function computes the cosine similarity between the input image and all images in the dataset. It then selects the top 3 images with the highest similarity.
   Extract Data: The extract_data function retrieves data from the DataFrame for the specified row numbers.
   Example Usage: The code demonstrates how to use these functions to extract data for specific rows based on the top similar images.
7. This code attempts to find the cosine similarity between the input text and the text reviews associated with the top 3 images that have the highest cosine similarity with the input image. Here's a brief explanation of what the code does:
   Preprocess Text: The preprocess_text function preprocesses the input text by converting it to lowercase,

removing special characters, tokenizing it, removing stopwords, and performing stemming and lemmatization.
Compute Cosine Similarity: The cosine_similarity_text function computes the cosine similarity between two text vectors. It handles cases where one vector is shorter than the other. Find Index for Input Text: The get_index function finds the index of the input text in the TF-IDF scores data. It compares the TF-IDF scores of each review with the TF-IDF scores of the input text and returns the index of the matching review.
Calculate Similarity with Image Text: For each review in the extracted data, the code computes the cosine similarity between the TF-IDF vector of the review and the TF-IDF vector of the input text. Output Results: The code prints the details of the top 3 reviews along with their cosine similarity scores with the input text and the average similarity between the text and image.

8. This script calculates the cosine similarity between the input text and the text reviews associated with the top 3 images having the highest cosine similarity with the input text. Here's a summary of what it does:
Preprocess Text: Preprocesses the input text by converting it to lowercase, removing special characters, tokenizing it, removing stopwords, stemming, and lemmatizing.
Compute Cosine Similarity (Text): Defines a function to compute cosine similarity between two text vectors.

Handles cases where one vector is shorter than the other.

Find Index for Input Text: Find the index of the input text in the TF-IDF scores data by comparing the TF-IDF scores of each review with the TF-IDF scores of the input text.

Calculate Similarity with Image Text: Computes the cosine similarity between the TF-IDF vector of each review text and the TF-IDF vector of the input text.

Find Top 3 Similarities (Text): Identifies the top 3 reviews with the highest cosine similarity scores to the input text.

Extract Data: Extracts the image filenames and review texts associated with the top 3 reviews. Compute Cosine Similarity (Image): Calculates the average cosine similarity between the input image and each of the top 3 images.

Output Results: Prints the image filenames, review texts, cosine similarity scores between the input text and reviews, cosine similarity scores between the input image and images, and the average similarity between the text and image for each of the top 3 reviews. This script is designed to provide insights into the similarity between the input text and reviews as well as the similarity between the input image and the images associated with the top 3 reviews.

Let's break down the approach, methodologies, assumptions, and results for each problem based on the provided information:

Problem 1: Extracting Image Features

Approach:
- Utilized a pre-trained ResNet50 model to extract image features.
- Removed the classification head from the model.
- Resized the images to match the ResNet50 input size (224x224).
- Preprocessed the images using the preprocess_input function from Keras.

Methodologies:
- Loaded the pre-trained ResNet50 model.
- Removed the classification head to retain only the feature extraction layers.
- Processed each image by resizing and preprocessing it.
- Extracted features using the predict function of the model.
- Flattened the feature vectors to make them compatible with further processing.

Assumptions:
- Assumes that the pre-trained ResNet50 model provides meaningful feature representations for images.

- Assumes that resizing the images to 224x224 pixels does not significantly distort the content.
- Assumes that the preprocess_input function adequately prepares the images for feature extraction.

Results:
- Extracted image features stored in a dictionary format, with filenames as keys and feature vectors as values.
- Saved the extracted features using the Pickle module for future use.

Problem 2: Calculating TF-IDF Scores for Text Data

Approach:
- Preprocessed the text data by lowercasing, removing special characters, tokenizing, removing stopwords, stemming, and lemmatizing.
- Calculated the Term Frequency-Inverse Document Frequency (TF-IDF) scores for each term in the text corpus.

Methodologies:
- Preprocessed the text using various techniques like lowercasing, removing special characters, tokenizing, removing stopwords, stemming, and lemmatizing.
- Calculated the document frequency (DF) and term frequency (TF) for each term.
- Used the TF and DF to calculate the TF-IDF scores for each term in the corpus.

Assumptions:
- Assumes that lowercasing, removing special characters, tokenizing, removing stopwords, stemming, and lemmatizing are effective text preprocessing techniques.
- Assumes that the TF-IDF scores adequately capture the importance of terms in the text corpus.

Results:
- Obtained TF-IDF scores for each term in the text corpus stored in a dictionary format.
- The TF-IDF scores can be used to represent the importance of terms in individual documents relative to the entire corpus.

Problem 3: Finding Similarities Between Image Features and Input Vector

Approach:
- Calculated the cosine similarity between the input vector (image feature) and each feature vector from the dataset.
- Identified the top 3 files with the highest cosine similarity scores.

Methodologies:
- Utilized the cosine similarity metric to measure the similarity between feature vectors.
- Compared the input vector with each feature vector in the dataset.

- Sorted the similarity scores to identify the top 3 files with the highest similarity.

Assumptions:
- Assumes that cosine similarity is an appropriate metric for comparing image features.
- Assumes that higher cosine similarity scores indicate greater similarity between images.

Results:
- Identified the top 3 files with the highest cosine similarity scores to the input vector.
- Provided filenames and their corresponding similarity scores as output.

Problem 4: Finding Similarities Between TF-IDF Vectors and Input Vector (Text)

Approach:
- Calculated the cosine similarity between the input vector (TF-IDF vector of the input text) and TF-IDF vectors of each review text.
- Identified the top 3 reviews with the highest cosine similarity scores.

Methodologies:
- Utilized the cosine similarity metric to measure the similarity between TF-IDF vectors.

- Compared the input vector with TF-IDF vectors of each review text.
- Sorted the similarity scores to identify the top 3 reviews with the highest similarity.

Assumptions:
- Assumes that cosine similarity is an appropriate metric for comparing TF-IDF vectors.
- Assumes that higher cosine similarity scores indicate greater similarity between texts.

Results:
- Identified the top 3 reviews with the highest cosine similarity scores to the input text.
- Provided review indices and their corresponding similarity scores as output.

Problem 5: Combining Image and Text Similarities

Approach:
- Combined the cosine similarity scores from both image features and TF-IDF vectors to compute an average similarity score for each review.
- Identified the top 3 reviews with the highest average similarity scores.

Methodologies:

- Calculated the average similarity score for each review by averaging the cosine similarity scores from image features and TF-IDF vectors.
- Combined the similarity scores to obtain an overall similarity measure.

Assumptions:
- Assumes that combining similarity scores from both image features and TF-IDF vectors provides a comprehensive measure of similarity.
- Assumes that averaging the scores adequately captures the overall similarity between the input and reviews.

Results:
- Identified the top 3 reviews with the highest average similarity scores.
- Provided review indices, filenames, review texts, and their corresponding average similarity scores as output.

Overall, these approaches leverage machine learning and natural language processing techniques to extract features, compute similarities, and identify relevant content based on input data.