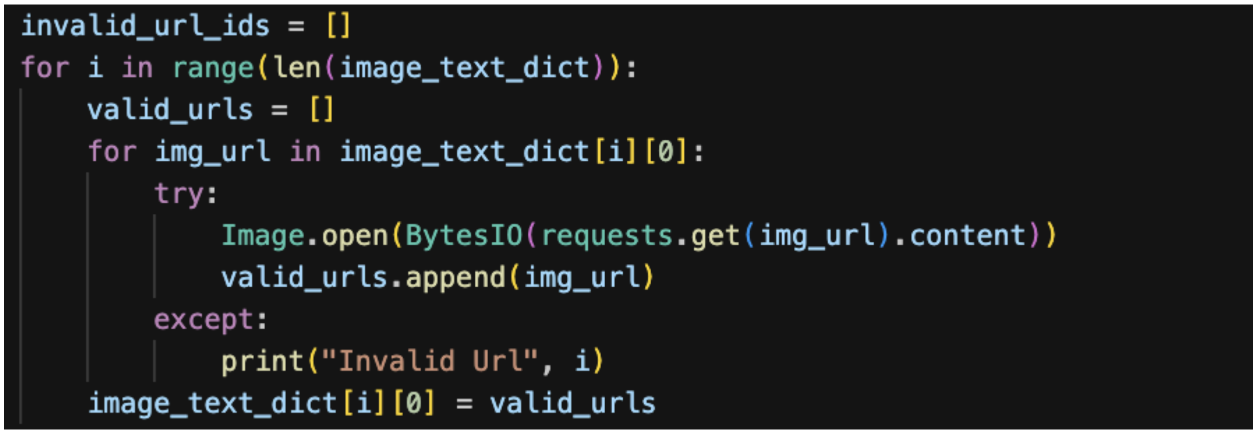
IR Assignment 2

Invalid Url Removal



```
invalid_url_ids = []
for i in range(len(image_text_dict)):
    valid_urls = []
    for img_url in image_text_dict[i][0]:
        try:
            Image.open(BytesIO(requests.get(img_url).content))
            valid_urls.append(img_url)
        except:
            print("Invalid Url", i)
    image_text_dict[i][0] = valid_urls
```

I have deleted invalid URLs from the dataset with the help of the above try and except block.

Image Processing

1. Resize the image to VGG model input size (224,224).
2. Increased the brightness of the image by a factor of 1.2 .
3. Introduced random horizontal and vertical flips.

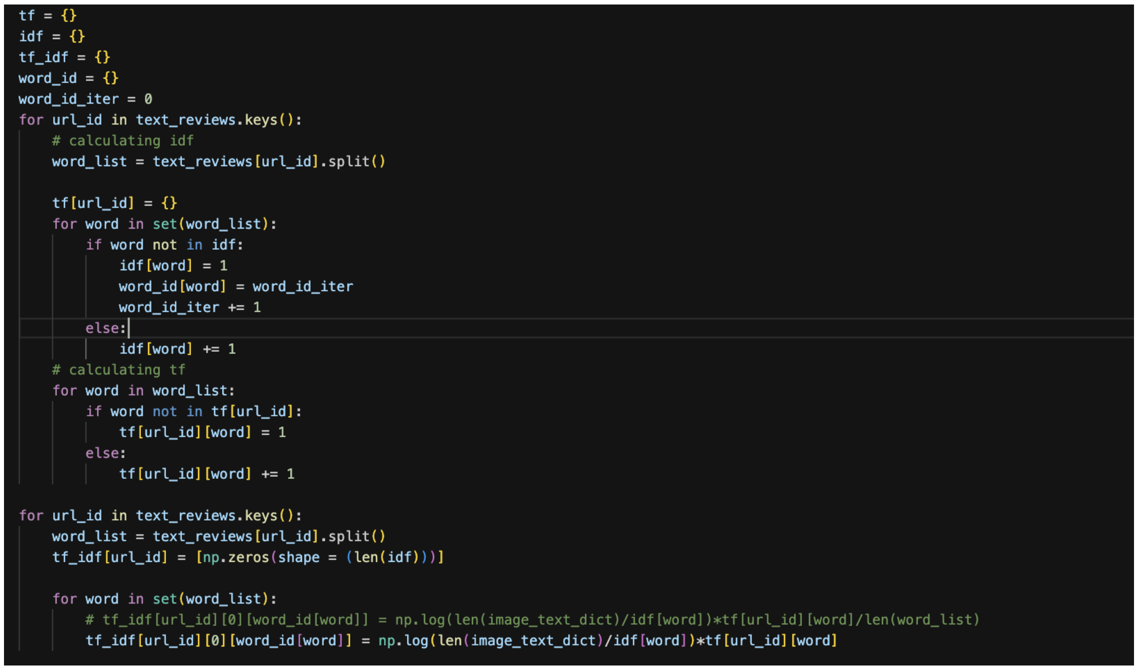
Text Processing

1. Tokenized the words using nltk.
2. Removed stop words and punctuation.
3. Used lemmatization as a normalization technique.

Image Feature Extraction

1. Used VGG model.
2. Clipped the model after the “fc2” layer, thus extracted a feature vector with 4096 dimensions.

Tf-Idf Calculation

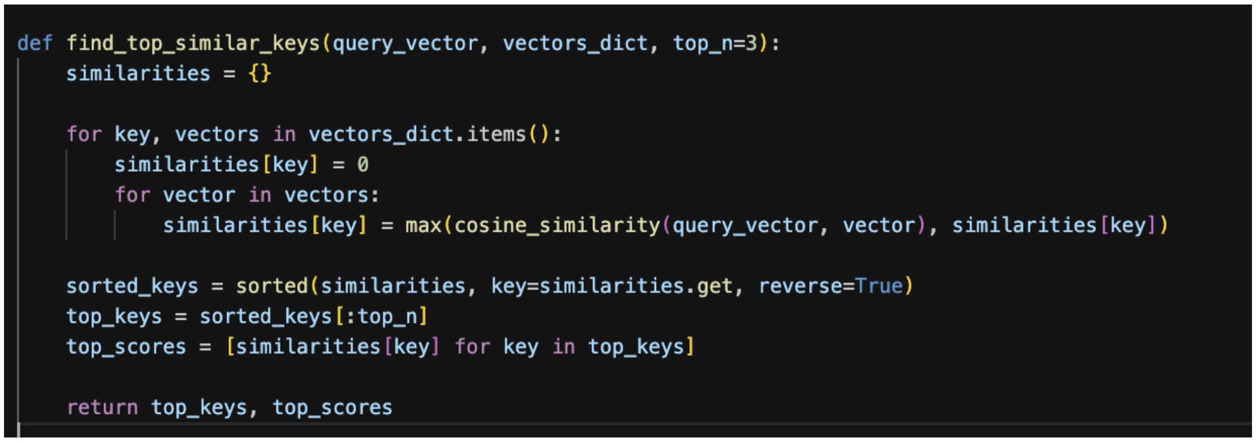


```python
tf = {}
idf = {}
tf_idf = {}
word_id = {}
word_id_iter = 0
for url_id in text_reviews.keys():
    # calculating idf
    word_list = text_reviews[url_id].split()

    tf[url_id] = {}
    for word in set(word_list):
        if word not in idf:
            idf[word] = 1
            word_id[word] = word_id_iter
            word_id_iter += 1
        else:
            idf[word] += 1
    # calculating tf
    for word in word_list:
        if word not in tf[url_id]:
            tf[url_id][word] = 1
        else:
            tf[url_id][word] += 1

for url_id in text_reviews.keys():
    word_list = text_reviews[url_id].split()
    tf_idf[url_id] = [np.zeros(shape = (len(idf)))]

    for word in set(word_list):
        # tf_idf[url_id][0][word_id[word]] = np.log(len(image_text_dict)/idf[word])*tf[url_id][word]/len(word_list)
        tf_idf[url_id][0][word_id[word]] = np.log(len(image_text_dict)/idf[word])*tf[url_id][word]
```

Text Vector Creation

I have created a one dimensional vector of the vocabulary size where each index represents a word of the vocabulary which are initialized as zeros and for the words present in the text review their corresponding indices contain the tf-idf for the word in that document.

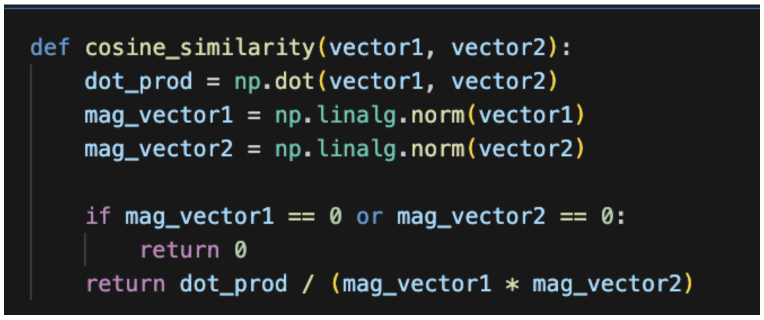
Retrieval(Image/Text)



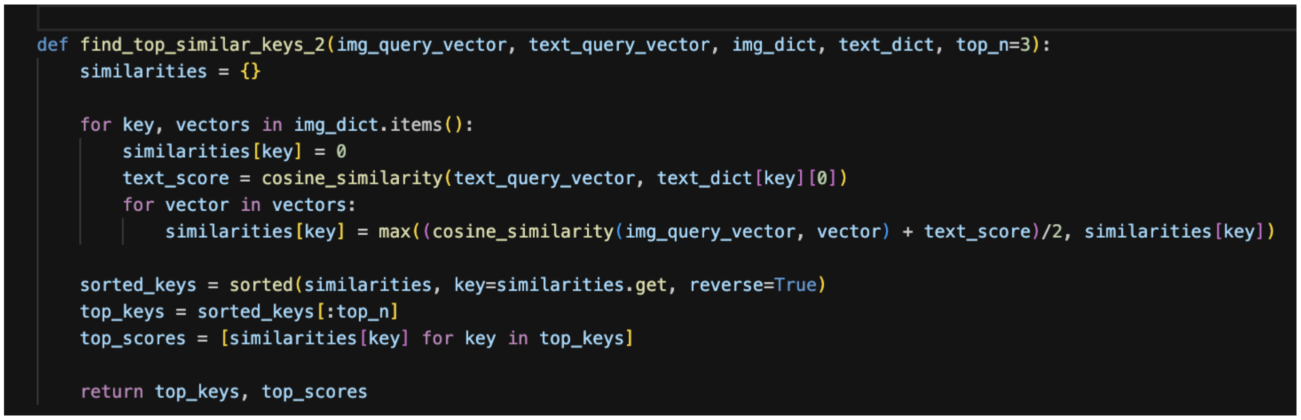
```python
def find_top_similar_keys(query_vector, vectors_dict, top_n=3):
    similarities = {}

    for key, vectors in vectors_dict.items():
        similarities[key] = 0
        for vector in vectors:
            similarities[key] = max(cosine_similarity(query_vector, vector), similarities[key])

    sorted_keys = sorted(similarities, key=similarities.get, reverse=True)
    top_keys = sorted_keys[:top_n]
    top_scores = [similarities[key] for key in top_keys]

    return top_keys, top_scores
```

I have used cosine similarity as a score for ranking.

Combined Retrieval



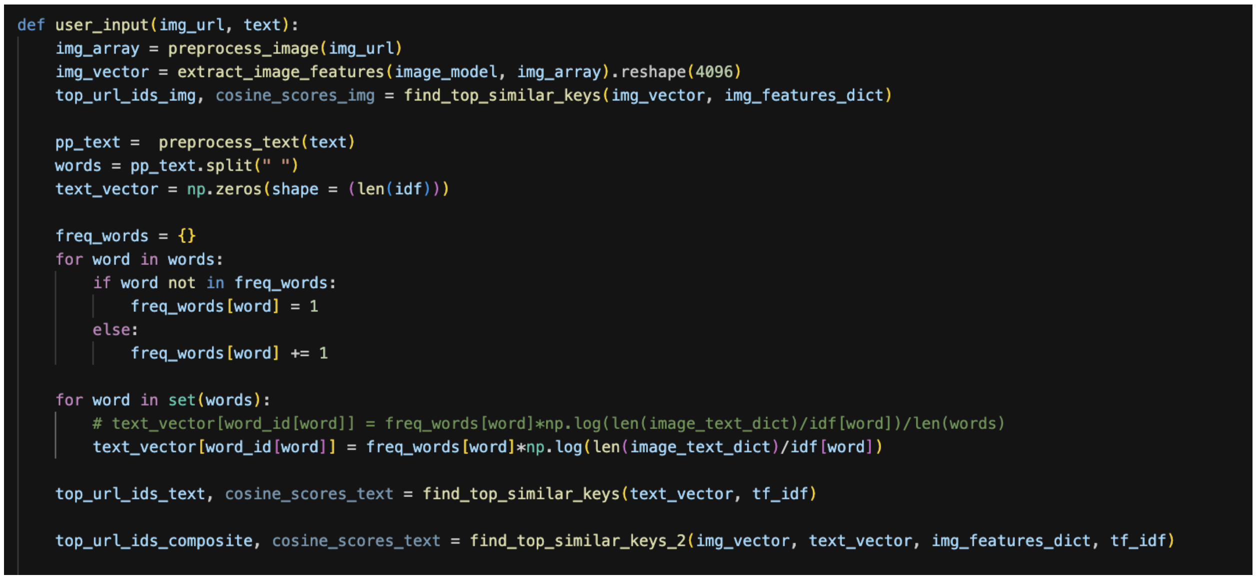
```python
def cosine_similarity(vector1, vector2):
    dot_prod = np.dot(vector1, vector2)
    mag_vector1 = np.linalg.norm(vector1)
    mag_vector2 = np.linalg.norm(vector2)

    if mag_vector1 == 0 or mag_vector2 == 0:
        return 0
    return dot_prod / (mag_vector1 * mag_vector2)
```



```python
def find_top_similar_keys_2(img_query_vector, text_query_vector, img_dict, text_dict, top_n=3):
    similarities = {}

    for key, vectors in img_dict.items():
        similarities[key] = 0
        text_score = cosine_similarity(text_query_vector, text_dict[key][0])
        for vector in vectors:
            similarities[key] = max((cosine_similarity(img_query_vector, vector) + text_score)/2, similarities[key])

    sorted_keys = sorted(similarities, key=similarities.get, reverse=True)
    top_keys = sorted_keys[:top_n]
    top_scores = [similarities[key] for key in top_keys]

    return top_keys, top_scores
```

For the combined retrieval I have used an average of the text and image cosine scores.

Query Function



```python
def user_input(img_url, text):
    img_array = preprocess_image(img_url)
    img_vector = extract_image_features(image_model, img_array).reshape(4096)
    top_url_ids_img, cosine_scores_img = find_top_similar_keys(img_vector, img_features_dict)

    pp_text =  preprocess_text(text)
    words = pp_text.split(" ")
    text_vector = np.zeros(shape = (len(idf)))

    freq_words = {}
    for word in words:
        if word not in freq_words:
            freq_words[word] = 1
        else:
            freq_words[word] += 1

    for word in set(words):
        # text_vector[word_id[word]] = freq_words[word]*np.log(len(image_text_dict)/idf[word])/len(words)
        text_vector[word_id[word]] = freq_words[word]*np.log(len(image_text_dict)/idf[word])

    top_url_ids_text, cosine_scores_text = find_top_similar_keys(text_vector, tf_idf)

    top_url_ids_composite, cosine_scores_text = find_top_similar_keys_2(img_vector, text_vector, img_features_dict, tf_idf)
```

The image is downloaded from the url and processed in the same way as training image data while for the text review, term frequency is calculated from the given text while the inverse document frequency is used from training corpus.
Top 3 ranking pairs are returned with the help of find_top_similar_keys() and find_top_similar_keys_2() function.

Result and Analysis

1. Image retrieval performs better than text retrieval. The reason behind is possibly image feature vector being more informative and representative than the tf-idf vector as text may not be representative or the non-semantic nature of tf-idf.
2. Challenges faced and Improvement - tf-idf does not take into account the document length which can be the reason behind the poor performance of text retrieval, so use of models like BM25 will be a better design choice.