Question 1:

(Part a and b)

```python
# Step 1: Image Preprocessing
def preprocess_image(img):
    # Altering contrast
    img = exposure.adjust_gamma(img, gamma=1.5)
    # Resizing
    img = cv2.resize(img, (224, 224))
    # Random flips
    if np.random.rand() < 0.5:
        img = cv2.flip(img, 1)  # Horizontal flip
    # Brightness and exposure adjustments (optional)
    # img = cv2.convertScaleAbs(img, alpha=1.2, beta=10)  # Example of brightness
    return img
```

✓ 0.0s Python

Using the above code and a function called "preprocess_image", we can adjust exposure, orientation, resizing, etc as per requirement. Further preprocessing and conversion to Tensor model is done in the extract features function as follows-

```python
# Step 3: Feature Extraction
def extract_features(img_path, model):

    try:

        req = urllib.request.urlopen(img_path)
        arr = np.asarray(bytearray(req.read()), dtype=np.uint8)
        img = cv2.imdecode(arr, -1) # 'Load it as it is'

        # img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert BGR to RGB
        img = preprocess_image(img)
        transform = transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0
        ])
        img = transform(img)
        img = img.unsqueeze(0)  # Add batch dimension
        with torch.no_grad():
            features = model(img)
        features = features.squeeze().numpy()
        return features

    except:
        return None
```

Python

Once all the preprocessing is done, we use the resnet 50 pretrained model to extract the image features from the image. We initialize the base model as resnet50 model as shown below, and pass it as a paramet to the above function.

```python
# Step 2: Selecting a Pre-trained CNN
base_model = models.resnet50(pretrained=True)
base_model = nn.Sequential(*list(base_model.children())[:-1])
base_model.eval()
```

Python

```python
# Preprocess images and extract features
new_image_features = []
image_features = dict()
for index, row in tqdm(dataset.iterrows(), total=len(dataset)):
    image_index = row[0]
    # print(image_index)
    img_path_1 = row['Image']
    image_path_str = img_path_1.strip('[]')
    image_paths_list = image_path_str.split(', ')
    image_paths_list = [path.strip("'") for path in image_paths_list]

    # print(image_paths_list)
    image_features[image_index] = []
    for img_path in image_paths_list:
        try:
            features = extract_features(img_path, base_model)
            if features.shape == (2048,):
                image_features[image_index].append(features)
                new_image_features.append(features)
                # print(image_features)
                # print(new_image_features)
        except Exception as e:
            continue
```

36.8s   Python

(part c)

Once we obtain the image features:

```python
image_features
```

Pytho

```
{3452: [array([0.27560937, 0.60194117, 0.754934  , ..., 0.16488846, 0.60872906,
        0.22635223], dtype=float32)],
 1205: [array([0.69648486, 0.4313286 , 0.68877006, ..., 0.08553925, 0.3681998 ,
        0.47034553], dtype=float32),
  array([0.75058746, 0.18958028, 0.12760516, ..., 0.10283512, 0.84645057,
        0.04751762], dtype=float32),
  array([0.5142772 , 0.51244044, 0.47707045, ..., 0.5209865 , 0.8395843 ,
        0.35326934], dtype=float32)],
 1708: [array([0.38012242, 0.02712757, 0.6850916 , ..., 0.11624838, 0.6173863 ,
        0.16000004], dtype=float32)],
 2078: [array([0.31159744, 0.4188953 , 0.5229862 , ..., 0.07281115, 0.8788534 ,
        0.39201817], dtype=float32)],
 801: [array([0.5459123 , 0.7595159 , 1.0664445 , ..., 0.18190739, 0.5383692 ,
        0.12744005], dtype=float32)],
 126: [array([1.7978957 , 0.32043925, 1.3665035 , ..., 0.14168923, 0.1333295 ,
        0.3691537 ], dtype=float32),
  array([1.5082341 , 0.5774796 , 0.7863694 , ..., 0.16180122, 0.02562088,
        0.0155558 ], dtype=float32),
  array([2.0794737 , 0.55688345, 0.78360575, ..., 0.40662152, 0.07597207,
        0.10914455], dtype=float32),
  array([1.1646376 , 0.2654516 , 0.7812614 , ..., 0.00357867, 0.06182662,
        0.01296882], dtype=float32)],
 1329: [array([0.7798044 , 0.01953565, 0.82690346, ..., 0.36494425, 0.90894556,
        0.14576328], dtype=float32)],
 325: [array([1.0878496 , 1.1471065 , 0.83133113, ..., 0.3851972 , 0.5177554 ,
...
        0.23168479], dtype=float32)],
 1004: [array([0.4361916 , 0.22129954, 0.32629344, ..., 0.41548464, 0.01816885,
        0.13705188], dtype=float32)],
 1306: [array([0.8677874 , 0.539021  , 0.49936375, ..., 0.17891298, 0.6473086 ,
        1.2238015 ], dtype=float32)]}
```

We can then normalize it by calculating the mean and standard deviation and using the formula (features-mean)/standard deviation. We do this by creating another numpy array containing all the extracted features list and then calculating the mean and standard deviation using numpy functions.

```python
new_image_features
```

Python

```
[array([0.27560937, 0.60194117, 0.754934  , ..., 0.16488846, 0.60872906,
       0.22635223], dtype=float32),
 array([0.69648486, 0.4313286 , 0.68877006, ..., 0.08553925, 0.3681998 ,
       0.47034553], dtype=float32),
 array([0.75058746, 0.18958028, 0.12760516, ..., 0.10283512, 0.84645057,
       0.04751762], dtype=float32),
 array([0.5142772 , 0.51244044, 0.47707045, ..., 0.5209865 , 0.8395843 ,
       0.35326934], dtype=float32),
 array([0.38012242, 0.02712757, 0.6850916 , ..., 0.11624838, 0.6173863 ,
       0.16000004], dtype=float32),
 array([0.31159744, 0.4188953 , 0.5229862 , ..., 0.07281115, 0.8788534 ,
       0.39201817], dtype=float32),
 array([0.5459123 , 0.7595159 , 1.0664445 , ..., 0.18190739, 0.5383692 ,
       0.12744005], dtype=float32),
 array([1.7978957 , 0.32043925, 1.3665035 , ..., 0.14168923, 0.1333295 ,
       0.3691537 ], dtype=float32),
 array([1.5082341 , 0.5774796 , 0.7863694 , ..., 0.16180122, 0.02562088,
       0.0155558 ], dtype=float32),
 array([2.0794737 , 0.55688345, 0.78360575, ..., 0.40662152, 0.07597207,
       0.10914455], dtype=float32),
 array([1.1646376 , 0.2654516 , 0.7812614 , ..., 0.00357867, 0.06182662,
       0.01296882], dtype=float32),
 array([0.7798044 , 0.01953565, 0.82690346, ..., 0.36494425, 0.90894556,
       0.14576328], dtype=float32),
 array([1.0878496 , 1.1471065 , 0.83133113, ..., 0.3851972 , 0.5177554 ,
...
 array([0.5034915 , 0.9335299 , 0.19028728, ..., 0.11867103, 0.77325165,
       0.43720183], dtype=float32),
 array([0.4702217 , 0.072171  , 0.33403248, ..., 0.2428987 , 0.92079246,
       0.09167468], dtype=float32),
```

```python
extracted_features = np.array(new_image_features)
mean = np.mean(extracted_features, axis=0)
std = np.std(extracted_features, axis=0)
# normalized_features = (extracted_features - mean) / std
```

(variable) std: Any

Python

```python
normalized_features_dict = dict()
for i in image_features:
    normalized_features_dict[i] = []
    for j in image_features[i]:
        j = np.array(j)
        print(j)
        normalized_features_dict[i].append((j - mean) / std)
```
Python

The normalized feature dictionary is as follows-

```python
normalized_features_dict
```
Python

```
{3452: [array([-0.6000867 ,  0.59189063,  1.2436503 , ..., -0.37770134,
          0.2009331 , -0.39657906], dtype=float32)],
 1205: [array([ 0.6755112 ,  0.09948208,  1.0173811 , ..., -0.74885845,
         -0.32997066,  0.516514  ], dtype=float32),
  array([ 0.8394864 , -0.59823287, -0.9017059 , ..., -0.66795677,
          0.72563946, -1.0658296 ], dtype=float32),
  array([0.12327259, 0.3335807 , 0.29340494, ..., 1.2879525 , 0.710484  ,
         0.07838126], dtype=float32)],
 1708: [array([-0.28332645, -1.0670911 ,  1.0048014 , ..., -0.605216  ,
          0.22004157, -0.64488804], dtype=float32)],
 2078: [array([-0.49101335,  0.0635981 ,  0.45042878, ..., -0.8083944 ,
          0.79715997,  0.22339052], dtype=float32)],
 801: [array([ 0.2191529 ,  1.0466703 ,  2.3089626 , ..., -0.29809505,
          0.04563254, -0.76673687], dtype=float32)],
 126: [array([ 4.013689  , -0.22055802,  3.3351123 , ..., -0.48621607,
         -0.8483837 ,  0.13782513], dtype=float32),
  array([ 3.135777  ,  0.5212916 ,  1.3511539 , ..., -0.39214194,
         -1.0861217 , -1.18544   ], dtype=float32),
  array([ 4.867101  ,  0.46184862,  1.3417027 , ...,  0.75300866,
         -0.9749849 , -0.8352039 ], dtype=float32),
  array([ 2.0943978, -0.379259 ,  1.3336854, ..., -1.1322303, -1.0062072,
         -1.1951212], dtype=float32)],
 1329: [array([ 0.9280377, -1.0890023,  1.4897734, ...,  0.5580626,  0.8635804,
         -0.6981661], dtype=float32)],
 325: [array([ 1.8616673e+00,  2.1653039e+00,  1.5049152e+00, ...,
...
          0.11945669, -0.3766231 ], dtype=float32)],
 1004: [array([-0.11339089, -0.5066872 , -0.2222264 , ...,  0.7944661 ,
         -1.10257   , -0.73076665], dtype=float32)],
 1306: [array([ 1.1946983 ,  0.41029546,  0.3696442 , ..., -0.3121014 ,
          0.28608704,  3.3361628 ], dtype=float32)]}
```

Question 2

(part a)

```python
def preprocess_text(text):
    # Lowercase the text

    text = str(text).lower()

    # Tokenization
    tokens = word_tokenize(text)

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens = [token for token in tokens if token not in stop_words]

    # Remove punctuations
    tokens = [token for token in tokens if token not in string.punctuation]

    # Remove blank space tokens
    tokens = [token for token in tokens if token.strip()]

    #stemming and lemmatization
    porter = PorterStemmer()
    lemmatizer = WordNetLemmatizer()
    stemmed_tokens = [porter.stem(token) for token in tokens]
    lemmatized_tokens = [lemmatizer.lemmatize(token) for token in stemmed_tokens]

    return lemmatized_tokens
```

```python
count = 1
tokens_in_doc = []
total_tokens = set()
for i in dataset['Review Text']:
    print(count)
    print(preprocess_text(i))
    for j in preprocess_text(i):
        total_tokens.add(j)
    tokens_in_doc.append(preprocess_text(i))
    count += 1
```

Python

We define a preprocess function as done in assignment 1 and we preprocess all the documents and store their respective token in a list of lists called "tokens_in_doc".

(part b)

We calculate the term frequency of all the tokens in the documents using a self defined function tf() as follows -

```python
def tf(tokens_in_doc, total_tokens):
    tf_dict = {}
    for i in tokens_in_doc:
        for j in i:
            if j in tf_dict:
                tf_dict[j] += 1
            else:
                tf_dict[j] = 1
    for key in tf_dict:
        tf_dict[key] = tf_dict[key] / len(total_tokens)
    return tf_dict
```

Python

We also calculate the Inverse document frequency of all the terms using a self-defined function idf() as follows -

```python
def idf(tokens_in_doc, total_tokens):

    no_of_docs = len(tokens_in_doc)
    idf_dict = {}
    for i in total_tokens:
        count = 0
        for j in tokens_in_doc:
            if i in j:
                count += 1
        idf_dict[i] = np.log(no_of_docs/count)
    return idf_dict
```

Python

We combine the results from the 2 functions to calculate a dictionary that contains the tf-idf score of each term seen in all the documents -

```python
#2b - TF-IDF

def tf_idf(tokens_in_doc, total_tokens):
    tf_dict = tf(tokens_in_doc, total_tokens)
    idf_dict = idf(tokens_in_doc, total_tokens)
    tf_idf_dict = {}
    for i in total_tokens:
        tf_idf_dict[i] = tf_dict[i] * idf_dict[i]
    return tf_idf_dict
```

Python

```python
tf_idf
```

Pyth

```
{'everyon': 0.008416671890969478,
 'textur': 0.002467583124249431,
 'dug': 0.002467583124249431,
 'meteor': 0.0013714026759940711,
 'lm2596': 0.0013714026759940711,
 'rode': 0.006577082689532158,
 '5': 0.0323988556228174,
 'area': 0.010536863917079541,
 'news': 0.002467583124249431,
 'moveabl': 0.0013714026759940711,
 'reverend': 0.0013714026759940711,
 'bbe': 0.0013714026759940711,
 'benefit': 0.004384721793021438,
 'blast': 0.0034598826624860205,
 'mean': 0.010744157251182291,
 '.13': 0.0013714026759940711,
 'hidden': 0.019317583847979155,
 'unlik': 0.002467583124249431,
 'fx': 0.0034598826624860205,
 'wand': 0.0013714026759940711,
 'strang': 0.0034598826624860205,
 'tha': 0.0013714026759940711,
 'pride': 0.002467583124249431,
 'luck': 0.002467583124249431,
 '8-space': 0.0013714026759940711,
...
 'phone': 0.00525939782266035,
 '.74': 0.0013714026759940711,
 'p.o.': 0.0013714026759940711,
 'leg': 0.011639503688780814,
 ...}
```

We create a tf-idf matrix containing sparse matrices of each document with the tf-idf score according to each term in their position using the following self defined function-

```python
def create_review_tfidf_matrices(reviews_token, tfidf_tokens):
    review_tfidf_matrices = []

    for review in reviews_token:
        tfidf_vector = np.zeros(len(tfidf_tokens))

        for token in review:
            if token in tfidf_tokens:
                token_index = list(tfidf_tokens.keys()).index(token)
                tfidf_vector[token_index] = tfidf_tokens[token]

        tfidf_matrix = np.reshape(tfidf_vector, (1, -1))
        review_tfidf_matrices.append(tfidf_matrix)

    return review_tfidf_matrices
```

Python

```python
create_review_tfidf_matrices(tokens_in_doc, tf_idf_loaded)
```

Python

```
[array([[0., 0., 0., ..., 0., 0., 0.]]),
 array([[0., 0., 0., ..., 0., 0., 0.]]),
 array([[0., 0., 0., ..., 0., 0., 0.]]),
 array([[0., 0., 0., ..., 0., 0., 0.]]),
 array([[0., 0., 0., ..., 0., 0., 0.]]),
 array([[0., 0., 0., ..., 0., 0., 0.]]),
 array([[0., 0., 0., ..., 0., 0., 0.]]),
 array([[0., 0., 0., ..., 0., 0., 0.]]),
 array([[0., 0., 0., ..., 0., 0., 0.]]),
 array([[0., 0., 0., ..., 0., 0., 0.]]),
 array([[0., 0., 0., ..., 0., 0., 0.]]),
 array([[0., 0., 0., ..., 0., 0., 0.]]),
 array([[0.00841667, 0.        , 0.        , ..., 0.        , 0.        ,
         0.        ]]),
 array([[0., 0., 0., ..., 0., 0., 0.]]),
 array([[0., 0., 0., ..., 0., 0., 0.]]),
 array([[0., 0., 0., ..., 0., 0., 0.]]),
 array([[0., 0., 0., ..., 0., 0., 0.]]),
 array([[0., 0., 0., ..., 0., 0., 0.]]),
 array([[0., 0., 0., ..., 0., 0., 0.]]),
 array([[0., 0., 0., ..., 0., 0., 0.]]),
 array([[0., 0., 0., ..., 0., 0., 0.]]),
 array([[0., 0., 0., ..., 0., 0., 0.]]),
 array([[0., 0., 0., ..., 0., 0., 0.]]),
 array([[0., 0., 0., ..., 0., 0., 0.]]),
...
 array([[0., 0., 0., ..., 0., 0., 0.]]),
 array([[0., 0., 0., ..., 0., 0., 0.]]),
 array([[0., 0., 0., ..., 0., 0., 0.]]),
 array([[0., 0., 0., ..., 0., 0., 0.]]),
 array([[0., 0., 0., ..., 0., 0., 0.]])]
```

Question 3

We define a function called "calculate_similarity()" to calculate the cosine similarity between the extracted features of the input image and the features of each document using a loop. Using another function called "retrieve_top_similar_images()" we can retrieve the top n number of similar images as per the n number entered by the user.

```python
def calculate_similarity(input_image, extracted_features):
    similarities = {}
    input_features = extracted_features[input_image]
    for image, features in extracted_features.items():
        if image != input_image:
            # print(image, features)
            similarity = cosine_similarity([input_features], [features])[0][0]
            similarities[image] = similarity
    return similarities

def retrieve_top_similar_images(similarities, top_n=3):
    similar_images = sorted(similarities.items(), key=lambda x: x[1], reverse=Tru
    return similar_images
```

Python

We use a dictionary to store these results.

(part b)

For calculating the similar reviews, we use a self-defined function called "similar_reviews()" . In this function, the cosin_similarity function definition is written from scratch and the indexes of the sorted cosine similarity values is returned along with the list of similarities. The top n results are fetched as per user requirement.

```python
def similar_reviews(input_review_tokens, tf_idf_dict, review_tfidf_matrices):

    tfidf_vector = np.zeros(len(tf_idf_dict))

    for token in input_review_tokens:
        if token in tf_idf_dict:
            token_index = list(tf_idf_dict.keys()).index(token)
            tfidf_vector[token_index] = tf_idf_dict[token]

    input_matrix = np.reshape(tfidf_vector, (1, -1))

    similarities = []

    for review_matrix in review_tfidf_matrices:
        # print(input_matrix, review_matrix)
        similarity = np.dot(input_matrix, review_matrix.T) / (np.linalg.norm(inpu
        similarities.append(similarity[0][0])
        # print(similarity)


    #get indices of top 3 most similar reviews
    top_similarities = np.argsort(similarities)

    return top_similarities, similarities
```

Python

(part c)

```python
with open ('image_similarities.pkl', 'wb') as f:
    pickle.dump(image_similiarities, f)
```

Python

```python
with open('review_similarities.pkl', 'wb') as f:
    pickle.dump(review_similarities, f)
```

Python

```python
with open('index_to_cosine_similarity.pkl', 'wb') as f:
    pickle.dump(index_to_cosine_similarity, f)
```

Python

```python
with open('image_similarities.pkl', 'rb') as f:
    image_similarities_loaded = pickle.load(f)
```

Python

```python
with open('review_similarities.pkl', 'rb') as f:
    review_similarities_loaded = pickle.load(f)
```

Python

```python
with open('index_to_cosine_similarity.pkl', 'rb') as f:
    index_to_cosine_similarity_loaded = pickle.load(f)
```

Python

Question 4

(part a and b)

```
input_image = input("Enter the image link: ")
review = input("Enter a review: ")

input_preprocessed = preprocess_text(review)
print(input_preprocessed)

for i in index_to_image:
    if input_image in index_to_image[i]:

        for j in range(i+1):
            if j == input_image:
                break

        input_image_group_features = normalized_extracted_features_loaded[index_t
        # print(index_to_number[i])
        break

image_similiarities = calculate_similarity(input_image, link_to_features)

top_review_similarities, review_similarities = similar_reviews(input_preprocessed

n = int(input("Enter the number of top documents you want to receive: "))

#------------------------------------------------------------------------------

print("USING IMAGE RETRIEVAL")

top_image_similarities = retrieve_top_similar_images(image_similiarities, n)
print(top_image_similarities)
indexes_of_similar_images = []
index_to_cosine_similarity = dict()

for i in index_to_image:
    index_to_cosine_similarity[i] = []
    for j in index_to_image[i]:
```

```
for i in index_to_image:
    index_to_cosine_similarity[i] = []
    for j in index_to_image[i]:
        if j in image_similiarities:
            index_to_cosine_similarity[i].append(image_similiarities[j])


for i in top_image_similarities:
    # print(i)
    for j in index_to_image:
        if i[0] in index_to_image[j]:
            for k in range(len(index_to_image[j])):
                if index_to_image[j][k] == i[0]:
                    indexes_of_similar_images.append([j, k])


print(indexes_of_similar_images)

#print images and reviews in pairs
for i in indexes_of_similar_images:
    print(index_to_number[i[0]])
    print("Image URL: ", index_to_image[i[0]])
    print("Review: ", index_to_review[i[0]])
    print("Cosine similarity of image: ", index_to_cosine_similarity[i[0]][i[1]])
    print("Cosine similarity of text: ", review_similarities[i[0]])
    print("Composite similarity score: ", (index_to_cosine_similarity[i[0]][i[1]]

#----------------------------------------------------------------------------

print("USING REVIEW RETRIEVAL")

indexes_of_similar_reviews = []

for i in range(1, n+1):
    print(top_review_similarities[-1*i])
```

```python
indexes_of_similar_reviews = []

for i in range(1, n+1):
    print(top_review_similarities[-1*i])
    indexes_of_similar_reviews.append(top_review_similarities[-1*i])

for i in indexes_of_similar_reviews:
    print(index_to_number[i])
    print("Image URL: ", index_to_image[i])
    print("Review: ", index_to_review[i])
    if index_to_cosine_similarity[i] == []:
        print("Cosine similarity of image: ", 0)
    else:
        print("Cosine similarity of image: ", index_to_cosine_similarity[i][0])
    print("Cosine similarity of text: ", review_similarities[i])
    if index_to_cosine_similarity[i] == []:
        print("Composite similarity score: ", review_similarities[i])
    else:
        print("Composite similarity score: ", (index_to_cosine_similarity[i][0] +
```

Python

```
['use', 'fender', 'lock', 'tuner', 'five', 'year', 'variou', 'strat', 'tele', 'defini
USING IMAGE RETRIEVAL
[('https://images-na.ssl-images-amazon.com/images/I/719-SDMiOoL._SY88.jpg', 0.623047
[[655, 0], [578, 0], [541, 0], [997, 0]]
643
Image URL:  ['https://images-na.ssl-images-amazon.com/images/I/719-SDMiOoL._SY88.jpg'
Review:  These locking tuners look great and keep tune.  Good quality materials and c
Cosine similarity of image:  0.6230473
Cosine similarity of text:  0.12423014147223474
Composite similarity score:  0.3736387168523741
647
Image URL:  ['https://images-na.ssl-images-amazon.com/images/I/61n284XL9HL._SY88.jpg'
Review:  Easy as heck to put on, In my opinion better than sperzel. These took litera
Only thing ill say is you will probably need a setup after as removing these tuners,
Cosine similarity of image:  0.5665882
Cosine similarity of text:  0.1064584307850333
Composite similarity score:  0.33652332688276626
173
Image URL:  ['https://images-na.ssl-images-amazon.com/images/I/71dCrR30OvL._SY88.jpg'
Review:  Looking at these on a guitar when they don't have a single wrap on the post,
I have added a few pictures. The new tuners covered the holes on my Les Paul so it lc
Cosine similarity of image:  0.46467492
Cosine similarity of text:  0.2891321574181678
Composite similarity score:  0.37690353863092074
1547
...
Review:  Using this to get complete control over my signal going into in-ear monitors
Cosine similarity of image:  0.018175285
Cosine similarity of text:  0.4951050043306808
Composite similarity score:  0.256640144820117
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

We use the functions as defined in question 3 part a and b and we calculate the cosine similarity of the input image and input review with first only all the other image links and then with only the other reviews presented. The results presented are as shown above.

Once we obtain the cosine similarity scores, we can present an average of the score of the image and corresponding review as the composite score. In case of retrieval using review, we use the first image of every link and in the case where there is no image or corresponding features, the review score itself is presented as the composite score.

```python
combined_dict = {}
combined_dict.update(composite_similarity_reviews)
combined_dict.update(composite_similarity_images)
sorted_combined_dict = dict(sorted(combined_dict.items(), reverse= True, key=lamb

sorted_combined_dict
```

```
{654: 1.0000000000000002,
 2486: 0.40172361254307026,
 173: 0.37690353863092074,
 643: 0.3736387168523741,
 1547: 0.33806406010012613,
 647: 0.33652332688276626,
 1012: 0.2830697113843314,
 784: 0.256640144820117}
```

We combine the results from the top similarity based images and reviews we received and we sort the dictionary to get the indexes of the pair. We then rank them as such.

Question 5

(Part a)

```python
print("Ranked combined scores: ")
for i in sorted_combined_dict:
    for j in index_to_number:
        if i == index_to_number[j]:
            print("Image URL: ", index_to_image[j])
            print("Review: ", index_to_review[j])
            print("Composite similarity score: ", sorted_combined_dict[i])
            print("Cosine similarity of image: ", index_to_cosine_similarity[j])
            print("Cosine similarity of text: ", review_similarities[j])
```

Python

```
Ranked combined scores: 
Image URL:  ['https://images-na.ssl-images-amazon.com/images/I/71bztfqdg+L._SY88.jp
Review:  I have been using Fender locking tuners for about five years on various st
Composite similarity score:  1.0000000000000002
Cosine similarity of image:  []
Cosine similarity of text:  1.0000000000000002
Image URL:  ['https://images-na.ssl-images-amazon.com/images/I/81p58EEtEpL._SY88.jp
Review:  awesome! i had the old mini, but this orientation had more useful space! p
Composite similarity score:  0.40172361254307026
Cosine similarity of image:  [0.24182291]
Cosine similarity of text:  0.5616243117970704
Image URL:  ['https://images-na.ssl-images-amazon.com/images/I/71dCrR30OvL._SY88.jp
Review:  Looking at these on a guitar when they don't have a single wrap on the pos
I have added a few pictures. The new tuners covered the holes on my Les Paul so it 
Composite similarity score:  0.37690353863092074
Cosine similarity of image:  [0.46467492, 0.28396487, 0.13215923]
Cosine similarity of text:  0.2891321574181678
Image URL:  ['https://images-na.ssl-images-amazon.com/images/I/719-SDMiOoL._SY88.jp
Review:  These locking tuners look great and keep tune.  Good quality materials and
Composite similarity score:  0.3736387168523741
Cosine similarity of image:  [0.6230473]
Cosine similarity of text:  0.12423014147223474
Image URL:  ['https://images-na.ssl-images-amazon.com/images/I/51OFdOanSXL._SY88.jp
```

```
I have added a few pictures. The new tuners covered the holes on my Les Paul so it
Composite similarity score:  0.37690353863092074
Cosine similarity of image:  [0.46467492, 0.28396487, 0.13215923]
Cosine similarity of text:  0.2891321574181678
Image URL:  ['https://images-na.ssl-images-amazon.com/images/I/719-SDMiOoL._SY88.jp
Review:  These locking tuners look great and keep tune.  Good quality materials and
Composite similarity score:  0.3736387168523741
Cosine similarity of image:  [0.6230473]
Cosine similarity of text:  0.12423014147223474
Image URL:  ['https://images-na.ssl-images-amazon.com/images/I/51OFdOanSXL._SY88.jp
Review:  I really like the simplicity of this bridge. It adjusts easy for string he
Composite similarity score:  0.33806406010012613
Cosine similarity of image:  [0.41581237, 0.37464252]
Cosine similarity of text:  0.2603157470389363
Image URL:  ['https://images-na.ssl-images-amazon.com/images/I/61n284XL9HL._SY88.jp
Review:  Easy as heck to put on, In my opinion better than sperzel. These took lite
Only thing ill say is you will probably need a setup after as removing these tuners
Composite similarity score:  0.33652332688276626
Cosine similarity of image:  [0.5665882]
Cosine similarity of text:  0.1064584307850333
Image URL:  ['https://images-na.ssl-images-amazon.com/images/I/7168briC3cL._SY88.jp
Review:  ergonomics and useful.
Composite similarity score:  0.2830697113843314
Cosine similarity of image:  [-0.10514382]
Cosine similarity of text:  0.6712832454982504
Image URL:  ['https://images-na.ssl-images-amazon.com/images/I/715yNxVy3ML._SY88.jp
Review:  Using this to get complete control over my signal going into in-ear monito
Composite similarity score:  0.256640144820117
Cosine similarity of image:  [0.018175285]
Cosine similarity of text:  0.4951050043306808
```

The ranked pairs according to the composite scores are as shown above.

(part b)

Out of image-based retrieval and text-based retrieval, text-based retrieval gives better results due to a cumulation of results-

1. Not all the links provided have an image, therefore, the amount of data present for images is slightly skewed per index. However, there is a text review for every index.
2. There can be multiple pictures for the same index, however, there is only 1 review for 1 index.
3. The indexes for which there are no images present, the composite score for such retrieved documents is solely the score of the review. Therefore if the review ranks high, then the retrieved document automatically rates high.

(part c)

The retrieval process used here is retrieval based on composite scores (average) of the cosine scores of images and review pairs. The shortcomings of this retrieval method are-
1. Cosine similarity doesn't consider the length of documents. Longer documents may have lower cosine similarity scores even if they share substantial content.
2. Cosine similarity treats words or features as independent entities. It doesn't capture the semantic meaning of words, making it less effective in understanding context
3. Cosine similarity depends on vector normalization. Different normalization methods can yield different results, making comparisons sensitive to preprocessing choices

The potential improvements to such retrieval methods are -
1. Instead of using raw term frequencies, weight the terms using TF-IDF (Term Frequency-Inverse Document Frequency). TF-IDF adjusts for the document length by penalizing terms that occur frequently across all documents. This helps in mitigating the impact of document length on cosine similarity scores.
2. Utilize word embeddings such as Word2Vec, GloVe, or FastText to capture semantic meaning and contextual relationships between words. By representing words in a dense vector space, cosine similarity can then capture the semantic similarity between documents more effectively.
3. Instead of comparing individual words or features, represent entire sentences or documents as embeddings. Techniques like Doc2Vec or Universal Sentence Encoder can generate fixed-length vectors for variable-length texts, capturing semantic meaning and context more accurately than word-level embeddings.
4. Experiment with different normalization methods to understand their impact on cosine similarity scores. While L2 normalization is commonly used, consider alternatives like L1 normalization or min-max scaling. Additionally, explore techniques like length normalization, where cosine similarity is adjusted based on the length of the documents being compared, to mitigate the influence of document length.