IR Assignment

```python
def adjust_brightness_and_contrast(image, brightness=0, contrast=0):
        # Alpha controls contrast; Beta controls brightness.
        alpha = 1 + contrast / 127
        beta = brightness
        adjusted_image = cv2.convertScaleAbs(image, alpha=alpha, beta=beta)
        return adjusted_image

def preprocess(image_url):
    response = requests.get(image_url)
    image = cv2.imdecode(np.frombuffer(response.content, np.uint8), -1)
    if image is None:
        return None
    image_height, image_width = image.shape[:2]


    #border removal
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    _,thresh = cv2.threshold(gray,1,255,cv2.THRESH_BINARY)
    contours,hierarchy = cv2.findContours(thresh,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
    x,y,w,h = cv2.boundingRect(contours[0])
    image = image[y:y+h,x:x+w]
    image = cv2.resize(image, (80, 80))


    # Random flip
    flip_direction = np.random.choice(["horizontal", "vertical"])
    if flip_direction == "horizontal":
        image = cv2.flip(image, 1)  # 1: Flip horizontally
    elif flip_direction == "vertical":
        image = cv2.flip(image, 0)  # 0: Flip vertically



    #pixel normalisation
    def normalize_image(image):
        normalized_image = cv2.normalize(image, None, alpha=0, beta=1, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_32F)
        return normalized_image

    image = normalize_image(image)


    #brightness and contrast adjustment
    def calculate_brightness_and_contrast(image):
        gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        mean, std_dev = cv2.meanStdDev(gray_image)
        return mean[0][0], std_dev[0][0]



    mean_brightness, std_dev_contrast = calculate_brightness_and_contrast(image)

    # Set a threshold for deciding whether to adjust brightness and contrast
    brightness_threshold = 20
    contrast_threshold = 10

    # Adjust brightness and contrast only if needed
    if mean_brightness < brightness_threshold or std_dev_contrast < contrast_threshold:
        brightness = 20
        contrast = 20
        adjusted_image = adjust_brightness_and_contrast(image, brightness=brightness, contrast=contrast)
    else:
        adjusted_image = image

    return image


image_url = data['Image'][200][0]
image = preprocess(image_url)
```

In the preprocessing of images first I have used cv2 libraries to do the following things:

adjust_brightness_and_contrast: This function adjusts the brightness and contrast of an image using the cv2.convertScaleAbs function from the OpenCV library.

preprocess: This function takes an image URL as input, downloads the image using requests.get, and decodes it into a NumPy array using cv2.imdecode. It then performs the following preprocessing steps:

Border removal: Finds the contours of the image and crops it to remove any surrounding borders.
Resizes the image to a fixed size of 80x80 pixels.
Random flip: Flips the image horizontally or vertically randomly using cv2.flip.
Pixel normalization: Normalizes the pixel values of the image to the range [0, 1].
calculate_brightness_and_contrast: This function calculates the mean brightness and standard deviation of the grayscale version of the image using cv2.meanStdDev.

Adjusting brightness and contrast:

It calculates the mean brightness and standard deviation of the image using the calculate_brightness_and_contrast function.
Sets thresholds for brightness and contrast.
If the mean brightness or standard deviation of the image is below the specified thresholds, it adjusts the brightness and contrast using the adjust_brightness_and_contrast function with predefined values.
If the image meets the criteria, it returns the adjusted image; otherwise, it returns the original image.
Finally, it returns the preprocessed image.

```python
import cv2
import torch
import torchvision.transforms as transforms
from torchvision.models import vgg16
from sklearn.preprocessing import normalize

# Load pre-trained VGG16 model
vgg16_model = vgg16(pretrained=True)
vgg16_model.eval()

# Define a function to extract features using VGG16
def extract_features_vgg16(image):
    img = cv2.resize(image, (224, 224))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_tensor = transforms.ToTensor()(img)
    img_tensor = torch.unsqueeze(img_tensor, 0)
    with torch.no_grad():
        features = vgg16_model.features(img_tensor)
    features = torch.nn.functional.normalize(features, p=2, dim=1)
    features = features.squeeze().numpy()
    return features.flatten()

# Define a function to extract normalized features using VGG16
def extract_features_normalized_vgg16(image):
    extracted_features = extract_features_vgg16(image)
    normalized_features = normalize([extracted_features], norm='l2')
    return normalized_features[0]
```

I have used VG16 for vectorization.

```python
#  preprorcess revierw texts
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer as ps

# Download the stopwords corpus if you haven't already
nltk.download('stopwords')
nltk.download('punkt')
stop_words = set(stopwords.words('english'))

def preprocess_review_text(review_text):

    review_text = str(review_text)
    review_text = review_text.lower()
    review_text = re.sub(r'[^\w\s]', '', review_text)
    # remove stop words
    review_text = ' '.join([word for word in review_text.split() if word not in stop_words])
    # do stemming
    review_text = ' '.join([ps().stem(word) for word in review_text.split()])
    return review_text



# data['Review Text'] = data['Review Text'].apply(preprocess_review_text)


import numpy as np
from collections import Counter
import math

def tokenize(text):
    # Split text into tokens (words)
    return text.split()

def calculate_tf(text):
    # Calculate term frequency (TF) for each word in the text
    tokens = tokenize(text)
    word_count = Counter(tokens)
    total_words = len(tokens)
    tf = {word: count / total_words for word, count in word_count.items()}
    return tf

def calculate_idf(documents):
    # Calculate inverse document frequency (IDF) for each word
    total_documents = len(documents)
    all_words = set([word for document in documents for word in tokenize(document)])
    idf = {}
    for word in all_words:
        doc_count = sum([1 for document in documents if word in tokenize(document)])
        idf[word] = math.log10(total_documents / doc_count)
    return idf

def calculate_tfidf(text, idf):
    # Calculate TF-IDF for each word in the text using precomputed IDF values

    tf = calculate_tf(text)
    tfidf = {word: tf[word] * idf[word] for word in tf}
    return tfidf
```

Preprocessing Review Texts:

preprocess_review_text(review_text): This function preprocesses review texts by performing the following steps:
Converts the text to lowercase.
Removes punctuation using regular expression re.sub.
Removes stopwords using NLTK's English stopwords list.
Stems words using the Porter stemming algorithm.
TF-IDF Calculation:

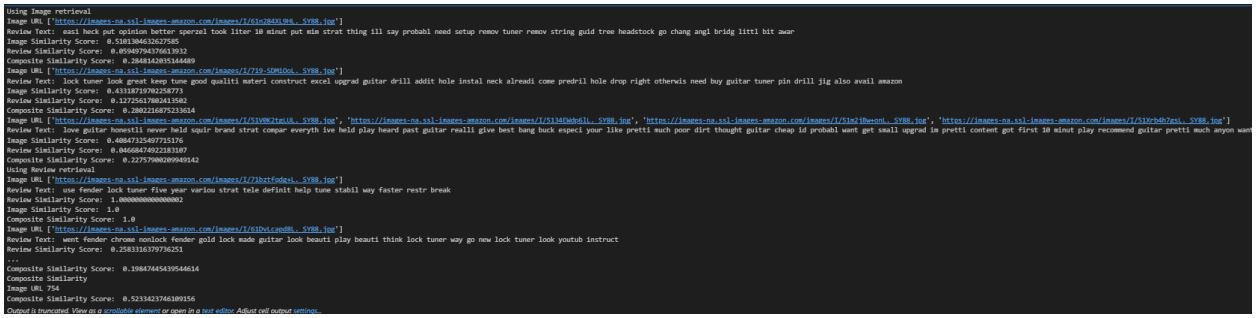
tokenize(text): Splits the text into tokens (words).
calculate_tf(text): Calculates the term frequency (TF) for each word in the text. TF is the ratio of the count of a word to the total number of words in the text.
calculate_idf(documents): Calculates the inverse document frequency (IDF) for each word. IDF is the logarithmically scaled inverse fraction of the documents that contain the word.
calculate_tfidf(text, idf): Calculates the TF-IDF value for each word in the text using precomputed IDF values.
Applying Preprocessing and TF-IDF Calculation to Data:

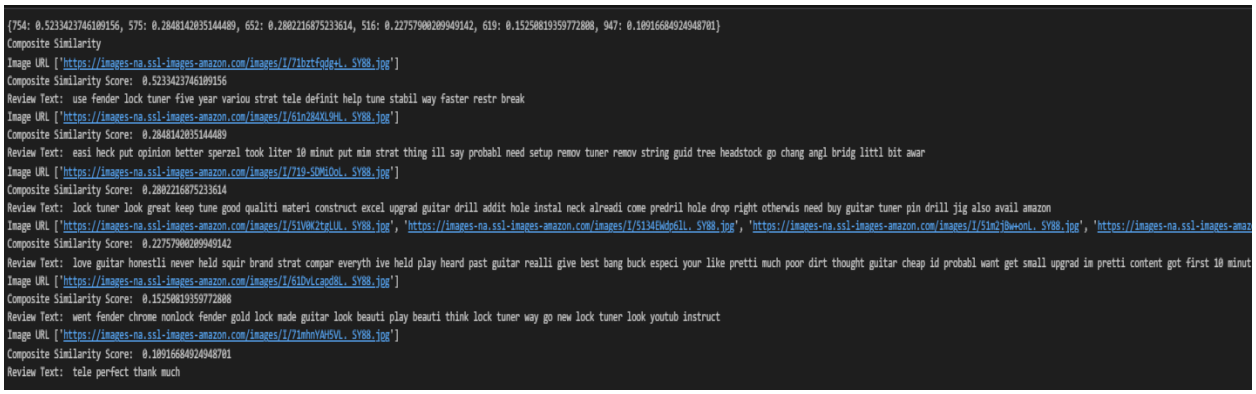
	Unnamed: 0	Image	Review Text	Image_Vectors
0	3452	[https://images-na.ssl-images-amazon.com/image...	love vintag spring vintag strat good tension g...	[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,...
1	1205	[https://images-na.ssl-images-amazon.com/image...	work great guitar bench mat rug enough abus ta...	[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 3.563436856848...
2	1708	[https://images-na.ssl-images-amazon.com/image...	use everyth acoust bass ukulel know smaller mo...	[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,...
3	2078	[https://images-na.ssl-images-amazon.com/image...	great price good qualiti didnt quit match radi...	[[0.008398145451137908, 0.002245986571792943 5,...
4	801	[https://images-na.ssl-images-amazon.com/image...	bought bass split time primari bass dean edg m...	[[0.015007795081277633, 0.009100475550438039, ...

My dataset after all these modifications looks like this.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
love	vintag	spring	strat	good	tension	great	stabil	float	bridg	want	way	go	work	guitar	bench	mat	rug	enough	abus	take	care	make	organ	workspac	much
0.060206	0.2326303	0.2873203	0.0911021	0.0458831	0.139794	0.03299	0.139794	0.1681919	0.0979014	0.062873	0.0757785	0.060673	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.0299237	0	0.0215152	0	0	0	0	0	0	0.027669	0.0238354	0.1173465	0.1173465	0.088946	0.0481698	0.1096904	0.1044052	0.1311859	0.038817	0.1000448	0.1304348	0.0410041
0	0	0	0	0	0	0	0	0	0	0	0	0	0.0104326	0.0089871	0	0	0	0.0181624	0	0	0	0	0	0	0
0	0	0	0	0.0573538	0	0.0412375	0	0	0	0	0	0	0	0	0	0	0	0.0923254	0	0	0	0	0	0	0
0	0	0	0	0	0	0.0107576	0	0	0	0.0205021	0	0	0	0	0	0	0	0	0	0	0	0.0194085	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0548214	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.0181117	0	0.0130224	0	0	0	0	0	0	0.0167471	0.0288533	0	0	0	0	0	0	0	0	0	0	0.0496366
0	0	0	0	0.0208559	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0498376	0	0	0	0	0	0	0	0.0541085	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0168681	0	0	0	0.0170447	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0349294
0	0	0	0	0.0372025	0	0.0133743	0	0	0	0	0.030721	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.0573538	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.007821	0	0	0	0	0	0	0	0	0	0.0186891	0	0	0	0.0125898	0	0.0136439	0	0	0	0	0.010717
0	0	0	0	0.0083932	0	0	0	0	0	0	0	0	0.0110987	0	0	0	0	0	0	0	0	0.0217754	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0217236	0	0	0	0	0	0	0
0	0	0	0	0	0	0.0353464	0	0	0	0	0	0	0	0.0195791	0	0	0	0.039568	0	0.0428807	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.0067701	0	0	0	0.0217634	0	0	0	0	0	0	0	0
0.0645064	0	0	0	0	0	0	0	0	0	0.033682	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.033682
0	0	0	0	0	0	0	0	0	0	0.0188619	0	0.0182019	0	0.0219285	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0.1822042	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0.0380654	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.0073218	0	0.0105287	0	0	0	0	0	0	0	0.0233282	0	0	0	0	0	0	0	0.0094978	0	0	0
0.0115781	0	0	0	0.0088237	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0153931	0	0	0	0	0.012091
0	0	0	0	0.0118663	0	0.0085319	0	0	0	0	0	0	0.0109722	0	0	0	0	0	0	0.020701	0	0	0	0	0.0162603
0.0291319	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0176843	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.159097	0.1370534	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.0101408	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0101408
0	0	0	0	0	0	0	0	0	0	0	0	0	0.1060647	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.0163176	0	0	0	0	0	0	0	0	0.011446	0	0	0

Also this is a Sample from my tfidf matrix.





Finally using these I have made a CLI which gives the following output.