Data Preparation for Siamese Network

The Siamese network expects data in the form of triplets, comprising an anchor image, a positive image (similar to the anchor), and a negative image (dissimilar to the anchor). Additionally, landmark coordinates corresponding to each image are provided to aid in feature extraction. This setup allows the network to learn embeddings such that the distance between embeddings of similar instances is minimized, while the distance between embeddings of dissimilar instances is maximized.

I have used below dataset from kaggle along with our personal photos.

1. https://www.kaggle.com/datasets/atulanandjha/lfwpeople
2. Own photos

In [1]:

```
import matplotlib.pyplot as plt
import numpy as np
import os
import random
import pandas as pd
import tensorflow as tf
from pathlib import Path
from keras import applications, layers, losses, ops, optimizers, metrics, Model, Sequent
from keras.applications import VGG16
import cv2
from tqdm.auto import tqdm

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from transformers import AutoImageProcessor, AutoModel
from transformers import TFAutoModel

from PIL import Image
import requests

import time

import glob
import random
from functools import lru_cache
import tensorflow_io as tfio
```

Expected Data format for Siamese Network

In [3]:

```
image1 = cv2.cvtColor(cv2.imread(anchor_image_url), cv2.COLOR_BGR2RGB)
image2 = cv2.cvtColor(cv2.imread(positive_image_url), cv2.COLOR_BGR2RGB)
image3 = cv2.cvtColor(cv2.imread(negative_image_url), cv2.COLOR_BGR2RGB)

# Create subplots with one row and two columns
plt.figure(figsize=(10, 5))
plt.subplot(1, 3, 1)  # Row 1, Column 1
plt.imshow(image1)
plt.title('Anchor Image')
plt.axis('off')

plt.subplot(1, 3, 2)  # Row 1, Column 2
plt.imshow(image2)
plt.title('Positive Image')
plt.axis('off')
plt.subplot(1, 3, 3)  # Row 1, Column 2
plt.imshow(image3)
plt.title('Negative Image')
plt.axis('off')

plt.show()
```

Siaseme Network

LFW Funnelled data augumentation

In [47]:
```
def augment_images_and_save(image_folder, output_folder, num_augmentations=5):
    """
    Augments images from a folder using TensorFlow's ImageDataGenerator and saves the au

    Parameters:
        image_folder (str): Path to the folder containing the input images.
        output_folder (str): Path to the folder where augmented images will be saved.
        num_augmentations (int): Number of augmentations to apply to each input image. D

    Returns:
        None
    """

    if not os.path.isdir(image_folder):
        print(f"Omitting - {image_folder}, Not a folder")
        return

    # Create output folder if it does not exist
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    # Initialize ImageDataGenerator for augmentation
    datagen = ImageDataGenerator(
        rotation_range=20,      # Random rotation between 0 and 20 degrees
        width_shift_range=0.1,  # Randomly shift width by up to 10%
        height_shift_range=0.1, # Randomly shift height by up to 10%
        shear_range=0.2,          # Shear intensity
        zoom_range=0.2,           # Random zoom between 80% and 120%
        horizontal_flip=True,     # Random horizontal flip
        brightness_range=[0.4,1.5],
        fill_mode='nearest' # Fill mode for pixels outside the input boundaries
    )

    # Loop through each image in the input folder
    for image_name in tqdm(os.listdir(image_folder)):

        if image_name.split(".")[-1] not in ["png", "jpeg", "jpg"]:
            continue

        # Read the input image
        image_path = os.path.join(image_folder, image_name)
        image = cv2.imread(image_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        image = np.expand_dims(image, axis=0)   # Add batch dimension

        # Generate augmented images
        augmented_images = [image[0]]
        for _ in range(num_augmentations):
            augmented_image = next(datagen.flow(image, batch_size=1))[0].astype(np.uint8
            augmented_images.append(augmented_image)

        # Save augmented images to the output folder
        base_name = os.path.splitext(image_name)[0]
        for i, augmented_image in enumerate(augmented_images):
            output_name = f"{base_name}_aug_{i+1}.jpg"
            output_path = os.path.join(output_folder, output_name)
            cv2.imwrite(output_path, cv2.cvtColor(augmented_image, cv2.COLOR_RGB2BGR))
```

In [45]:

```
LFW_BASE_DIR = "/Users/vignesh/Documents/george brown pgdm /DL2/FacialRecoData/archive/l
LFW_OUTPUT_DIR = "/Users/vignesh/Documents/george brown pgdm /DL2/FacialRecoData/archive
for dir in tqdm(os.listdir(LFW_BASE_DIR)):

    image_folder = f"{LFW_BASE_DIR}/{dir}"
    output_folder = f"{LFW_OUTPUT_DIR}/{dir}"

    augment_images_and_save(image_folder, output_folder, num_augmentations=3)
```

```
  0%|          | 0/5760 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/2 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/5 [00:00<?, ?it/s]
  0%|          | 0/3 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/3 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/2 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/5 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/4 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/2 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/2 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/15 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/3 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/3 [00:00<?, ?it/s]
  0%|          | 0/3 [00:00<?, ?it/s]
  0%|          | 0/2 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/2 [00:00<?, ?it/s]
  0%|          | 0/2 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/2 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/2 [00:00<?, ?it/s]
  0%|          | 0/2 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/14 [00:00<?, ?it/s]
  0%|          | 0/2 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/2 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/7 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/4 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/5 [00:00<?, ?it/s]
  0%|          | 0/2 [00:00<?, ?it/s]
  0%|          | 0/8 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/2 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
  0%|          | 0/1 [00:00<?, ?it/s]
```

Data Preparation

In [3]:

```
img_height, img_width = 224, 224
```

In [4]:

```
@lru_cache
def get_image_paths(parent_path):
    files = []
    files.extend(glob.glob(parent_path + '/**/*.jpeg', recursive=True))
    files.extend(glob.glob(parent_path + '/**/*.jpg', recursive=True))
    files.extend(glob.glob(parent_path + '/**/*.png', recursive=True))
    return files
```

In [5]:

```
def get_data_path(full_path):
    return "/".join(full_path.split("/")[:-1])
```

In [6]:

```
PARENT_FOLDER = "/Users/vignesh/Documents/george brown pgdm /DL2/FacialRecoData/Personal
LFW_PARENT_FOLDER = "/Users/vignesh/Documents/george brown pgdm /DL2/FacialRecoData/crop
```

In [7]:

```
image_paths = []
image_paths.extend(get_image_paths(PARENT_FOLDER))
image_paths.extend(get_image_paths(LFW_PARENT_FOLDER))
```

In [8]:

```
len(image_paths)
```

Out[8]:

```
53758
```

Color Correction

Color correction is needed because by default our siamese network model needs images in RGB but the images in the LFW dataset are in BGR color. So converting it to RGB and storing it back.

In [10]:

```
def check_color_correction(lfw_test_img_path, test_img_path):
    lfw_test_imge = cv2.imread(lfw_test_img_path)

    plt.figure(figsize=(10, 5))
    plt.subplot(2, 2, 1)   # Row 1, Column 1
    plt.imshow(lfw_test_imge)
    plt.title('Image without any conversion')
    plt.axis('off')

    plt.subplot(2, 2, 2)   # Row 1, Column 2
    plt.imshow(cv2.cvtColor(lfw_test_imge, cv2.COLOR_BGR2RGB))
    plt.title('Converted to rgb Image(COLOR_BGR2RGB)')
    plt.axis('off')

    test_imge = cv2.imread(test_img_path)

    plt.subplot(2, 2, 3)   # Row 2, Column 1
    plt.imshow(test_imge)
    plt.title('Image without any conversion')
    plt.axis('off')
    plt.subplot(2, 2, 4)   # Row 2, Column 2
    plt.imshow(cv2.cvtColor(test_imge, cv2.COLOR_BGR2RGB))
    plt.title('Converted to rgb Image(COLOR_BGR2RGB)')
    plt.axis('off')

    plt.show()
```

In [7]:

```
lfw_test_img_path = "/Users/vignesh/Documents/george brown pgdm /DL2/FacialRecoData/arch
test_img_path = "/Users/vignesh/Documents/george brown pgdm /DL2/FacialRecoData/Personal

check_color_correction(lfw_test_img_path, test_img_path)
```

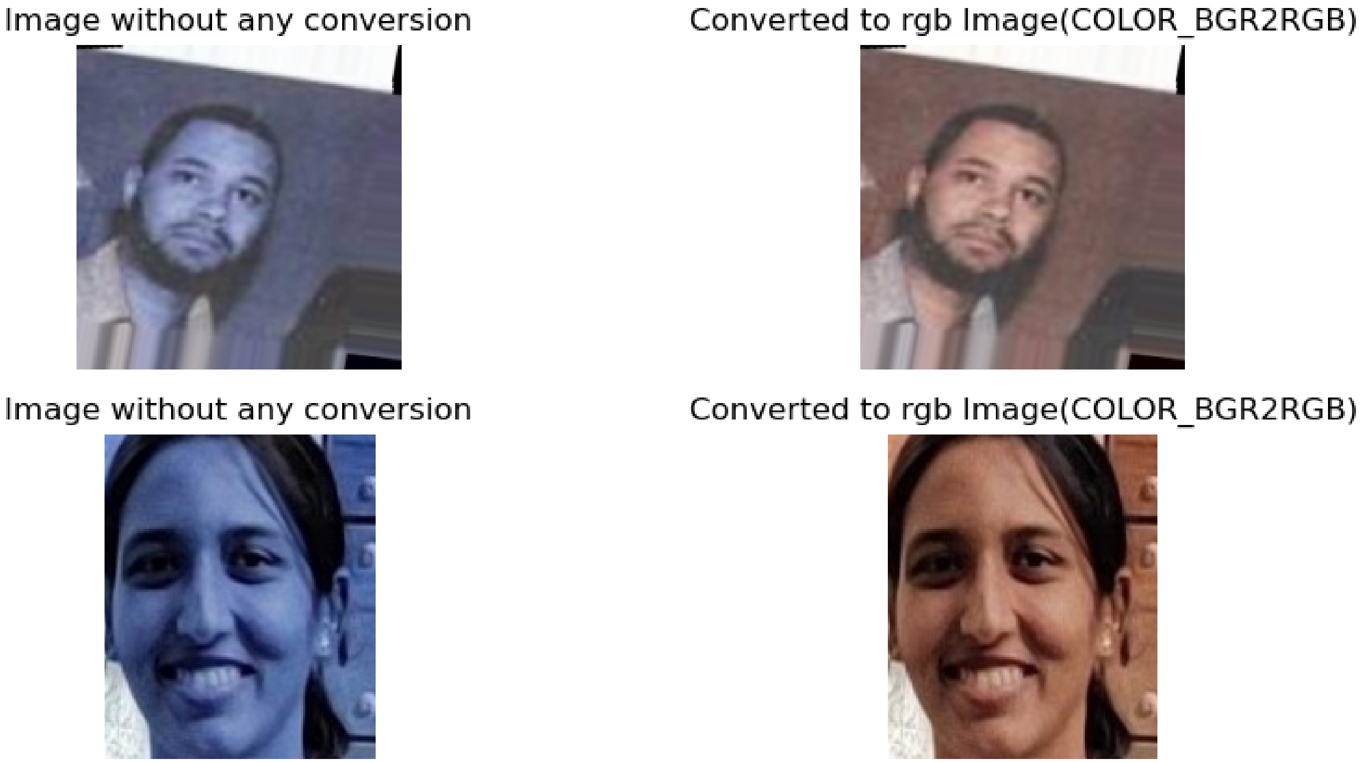


Image without any conversion

Converted to rgb Image(COLOR_BGR2RGB)

Image without any conversion

Converted to rgb Image(COLOR_BGR2RGB)

Image is in BGR format. But, Resnet / Mobilenet / Efficientnet needs in RGB format.

In [12]:

```
def convert_to_rgb(image_paths, new_parent_folder_name):

    """
    Resnet needs RGB input. So convert before fedding into the dataset.
    cannot to inside preprocess seciont because "tfio.experimental.color.bgr_to_rgb" won
    """

    new_image_paths = []
    for image_path in tqdm(image_paths):

        image = cv2.imread(image_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        splits = image_path.split("FacialRecoData")
        output_path = splits[0]+new_parent_folder_name+splits[1]

        os.makedirs(get_data_path(output_path), exist_ok=True)

        cv2.imwrite(output_path, image)

        new_image_paths.append(output_path)

    return new_image_paths
```

In [13]:

```
COLOR_CORRECTED_FOLDER = "FacialColorRecoData"
```

In [14]:
```
image_paths = convert_to_rgb(image_paths, COLOR_CORRECTED_FOLDER)
```

```
  0%|          | 0/53758 [00:00<?, ?it/s]
```

In [15]:
```
image_paths[:2]
```

Out[15]:
```
['/Users/vignesh/Documents/george brown pgdm /DL2/FacialColorRecoData/Personal/cropped_i
mages/0Madhou/Madhou 21.jpeg',
 '/Users/vignesh/Documents/george brown pgdm /DL2/FacialColorRecoData/Personal/cropped_i
mages/0Madhou/Madhou 37.jpeg']
```

In [16]:
```
print(len(image_paths))
```

```
53758
```

In [17]:
```
PARENT_FOLDER = f"/Users/vignesh/Documents/george brown pgdm /DL2/{COLOR_CORRECTED_FOLDE
LFW_PARENT_FOLDER = f"/Users/vignesh/Documents/george brown pgdm /DL2/{COLOR_CORRECTED_F
```

After color correction

In [18]:
```
lfw_test_img_path = f"/Users/vignesh/Documents/george brown pgdm /DL2/{COLOR_CORRECTED_F
test_img_path = f"/Users/vignesh/Documents/george brown pgdm /DL2/{COLOR_CORRECTED_FOLDE

check_color_correction(lfw_test_img_path, test_img_path)
```

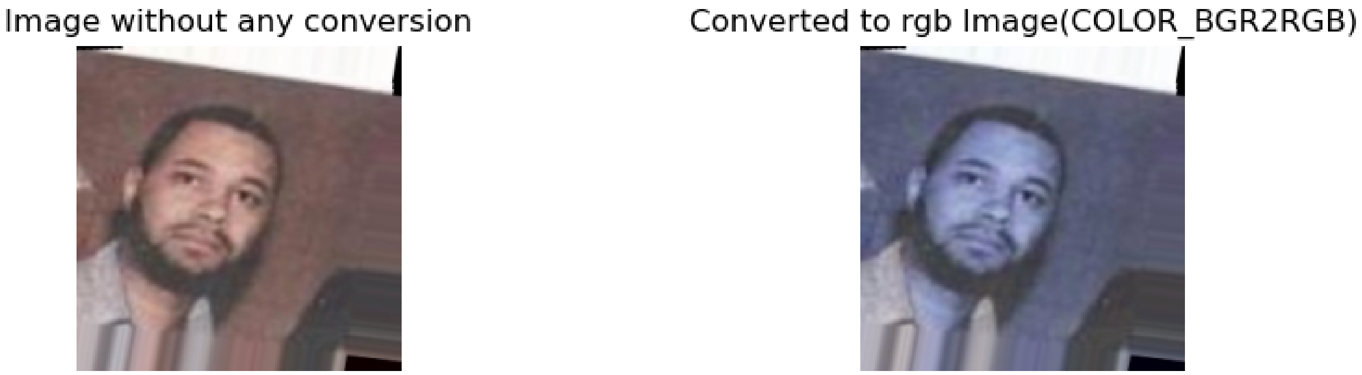


Image without any conversion
Converted to rgb Image(COLOR_BGR2RGB)

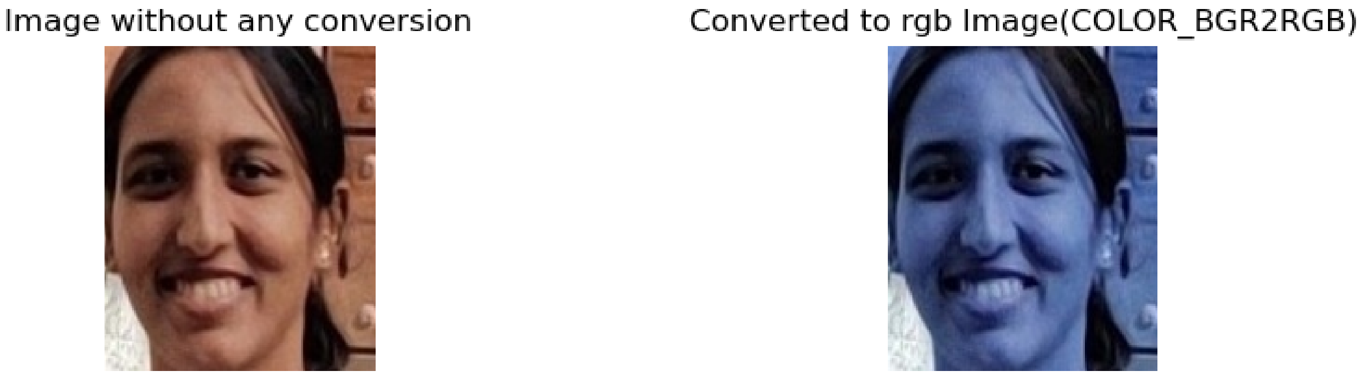


Image without any conversion
Converted to rgb Image(COLOR_BGR2RGB)

Now, raw image is in RGB format. So, we can proceed with further analysis

Triplets Preparation

In [9]:
```
unique_paths = list(set([get_data_path(image) for image in image_paths]))
```

In [10]:
```
def generate_random_int(max_limit, exclude_number):

    if exclude_number == -1:
        return random.randint(0, max_limit)
    while True:
        random_number = random.randint(0, max_limit)
        if random_number != exclude_number:
            return random_number
```

In [11]:

```
def generate_triplets(image_paths, unique_paths):
    """
    Generate triplets of images for training.

    Parameters:
        image_paths (list): A list of image paths.
        unique_paths (list): A list of unique parent paths.

    Returns:
        list: A list of triplets, each containing an anchor image path, a positive image
    """
    # Define a list to store triplets
    triplets = []

    # Loop through the image paths
    for anchor_image_path in tqdm(image_paths):
        """
        Generate triplets for anchor images by selecting positive and negative images.

        Parameters:
            anchor_image_path (str): The path to the anchor image.

        Returns:
            None
        """
        # Get the parent path of the anchor image
        anchor_parent_path = get_data_path(anchor_image_path)
        anchor_parent_path_index = unique_paths.index(get_data_path(anchor_image_path))

        # Get all similar images within the same parent path as the anchor image
        all_anchor_similar_images = get_image_paths(anchor_parent_path)

        # Get the index of the anchor image within the list of similar images
        anchor_image_index = all_anchor_similar_images.index(anchor_image_path)

        # Select a positive image randomly from similar images (excluding the anchor ima
        positive_image_index = generate_random_int(len(all_anchor_similar_images)-1, anc
        positive_image_path = all_anchor_similar_images[positive_image_index]

        # Select a negative parent path randomly (excluding the parent path of the ancho
        negative_parent_path_index = generate_random_int(len(unique_paths)-1, anchor_par
        negative_parent_path = unique_paths[negative_parent_path_index]

        # Get all images within the selected negative parent path
        all_negative_images = get_image_paths(negative_parent_path)

        # Select a negative image randomly from all negative images
        negative_image_path_index = generate_random_int(len(all_negative_images)-1,-1)
        negative_image_path = all_negative_images[negative_image_path_index]

        # Append the triplet (anchor image, positive image, negative image) to the list
        triplets.append((anchor_image_path, positive_image_path, negative_image_path))

    return triplets
```

In [12]:

```
triplets = generate_triplets(image_paths, unique_paths)
```

```
  0%|          | 0/53758 [00:00<?, ?it/s]
```

In [13]:

```
triplets_df = pd.DataFrame(triplets, columns=["anchor", "positive", "negative"])
```

```
triplets_df["anchor_names"] = triplets_df["anchor"].apply(lambda x: x.split("/")[-1])
triplets_df.sort_values("anchor_names", inplace=True)
triplets_df.to_csv("gbctriplets.csv", index=False)
```

In [2]:

```
triplets_df.head()
```

Out[2]:

	anchor	positive	negative	anchor
0	/Users/vignesh/Documents/george brown pgdm /DL...	/Users/vignesh/Documents/george brown pgdm /DL...	/Users/vignesh/Documents/george brown pgdm /DL...	
1	/Users/vignesh/Documents/george brown pgdm /DL...	/Users/vignesh/Documents/george brown pgdm /DL...	/Users/vignesh/Documents/george brown pgdm /DL...	
2	/Users/vignesh/Documents/george brown pgdm /DL...	/Users/vignesh/Documents/george brown pgdm /DL...	/Users/vignesh/Documents/george brown pgdm /DL...	10_a
3	/Users/vignesh/Documents/george brown pgdm /DL...	/Users/vignesh/Documents/george brown pgdm /DL...	/Users/vignesh/Documents/george brown pgdm /DL...	10_au
4	/Users/vignesh/Documents/george brown pgdm /DL...	/Users/vignesh/Documents/george brown pgdm /DL...	/Users/vignesh/Documents/george brown pgdm /DL...	10_au

In [7]:

```
triplet_count = len(triplets_df)
print(triplet_count)
```

```
53758
```

In [32]:

```
def preprocess_image(image_path):


    image_string = tf.io.read_file(image_path)

    if tf.strings.split(image_path, sep=".")[-1] == "png":
        image = tf.image.decode_png(image_string)
    else:
        image = tf.image.decode_jpeg(image_string)

            # Convert image to grayscale
    image = tf.image.rgb_to_grayscale(image)
    image = tf.image.grayscale_to_rgb(image)  # to get 3 channels


    image = tf.image.convert_image_dtype(image, tf.float32)
    image = tf.image.resize(image, (img_height, img_width), method=tf.image.ResizeMethod

    return tf.keras.applications.vgg16.preprocess_input(image)

def preprocess_triplets(anchor, positive, negative):
    """
    Given the filenames corresponding to the three images, load and
    preprocess them.
    """

    return tf.stack(
        [preprocess_image(anchor),
        preprocess_image(positive),
        preprocess_image(negative)]
    )
```

In [33]:

```
anchor_dataset = tf.data.Dataset.from_tensor_slices(triplets_df["anchor"].to_list())
positive_dataset = tf.data.Dataset.from_tensor_slices(triplets_df["positive"].to_list())
negative_dataset = tf.data.Dataset.from_tensor_slices(triplets_df["negative"].to_list())

dataset = tf.data.Dataset.zip((anchor_dataset, positive_dataset, negative_dataset))
```

```
dataset= dataset.shuffle(buffer_size=1024)
dataset = dataset.map(preprocess_triplets)
```

In [34]:

```
def plot_triplets(triplet_images):

    print(type(triplet_images))
    # Create subplots with one row and two columns
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 3, 1)  # Row 1, Column 1
    plt.imshow(triplet_images[0], cmap="gray")
    plt.title('Anchor Image')
    plt.axis('off')

    plt.subplot(1, 3, 2)  # Row 1, Column 2
    plt.imshow(triplet_images[1], cmap="gray")
    plt.title('Positive Image')
    plt.axis('off')

    plt.subplot(1, 3, 3)  # Row 1, Column 2
    plt.imshow(triplet_images[2], cmap="gray")
    plt.title('Negative Image')
    plt.axis('off')

    plt.show()
```

In [46]:

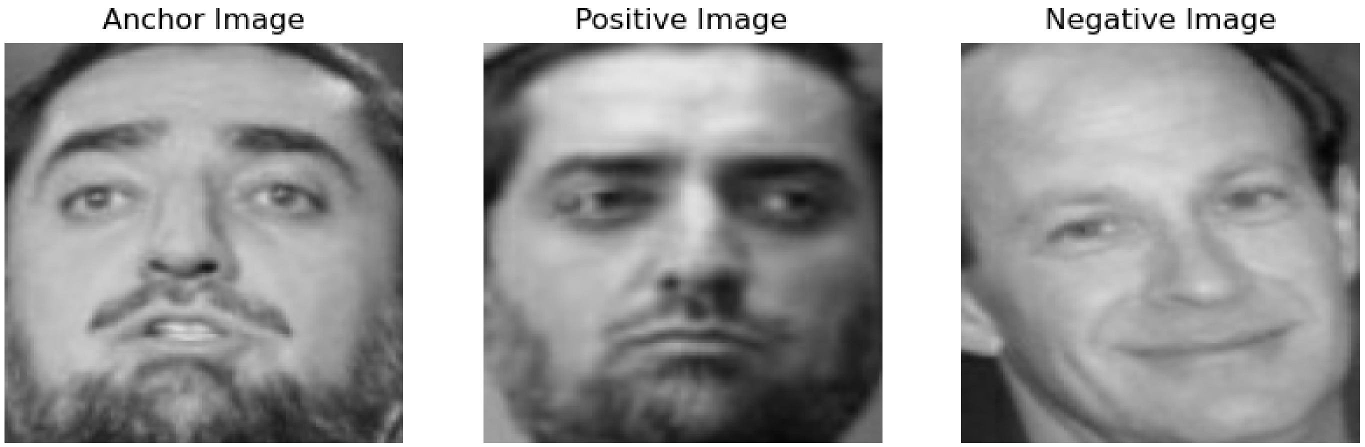
```
train_dataset = dataset.take(round(triplet_count*0.8))
val_dataset = dataset.skip(round(triplet_count*0.8))

train_dataset = train_dataset.batch(64, drop_remainder=False).prefetch(tf.data.AUTOTUNE)
val_dataset = val_dataset.batch(32, drop_remainder=False).prefetch(tf.data.AUTOTUNE)
```

In [140…

```
for batch in train_dataset.skip(0).take(1):
    for triplet in batch:
        plot_triplets(triplet)
```

```
<class 'tensorflow.python.framework.ops.EagerTensor'>
```



```
<class 'tensorflow.python.framework.ops.EagerTensor'>
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
<class 'tensorflow.python.framework.ops.EagerTensor'>
```

```
<class 'tensorflow.python.framework.ops.EagerTensor'>
```

```
<class 'tensorflow.python.framework.ops.EagerTensor'>
```

```
<class 'tensorflow.python.framework.ops.EagerTensor'>
```

Anchor Image

Positive Image

Negative Image

```
<class 'tensorflow.python.framework.ops.EagerTensor'>
```

Anchor Image

Positive Image

Negative Image

```
<class 'tensorflow.python.framework.ops.EagerTensor'>
```

Anchor Image

Positive Image

Negative Image

```
<class 'tensorflow.python.framework.ops.EagerTensor'>
```

Anchor Image

Positive Image

Negative Image

```
<class 'tensorflow.python.framework.ops.EagerTensor'>
```

Anchor Image
Positive Image
Negative Image

```
<class 'tensorflow.python.framework.ops.EagerTensor'>
```

Anchor Image
Positive Image
Negative Image

```
<class 'tensorflow.python.framework.ops.EagerTensor'>
```

Anchor Image
Positive Image
Negative Image

```
<class 'tensorflow.python.framework.ops.EagerTensor'>
```

Anchor Image
Positive Image
Negative Image

```
<class 'tensorflow.python.framework.ops.EagerTensor'>
```

```
<class 'tensorflow.python.framework.ops.EagerTensor'>
```

```
<class 'tensorflow.python.framework.ops.EagerTensor'>
```

```
<class 'tensorflow.python.framework.ops.EagerTensor'>
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
<class 'tensorflow.python.framework.ops.EagerTensor'>
```