Facial_Verification_with_a_Siamese_Network

December 3, 2023

1 Translating Research into Application: Development of a Facial Recognition System Utilizing Siamese Neural Network Architecture

Abstract: The forthcoming project endeavors to construct a robust facial recognition application designed for authentication purposes within an application framework. The initial phase involves the creation of a deep learning model utilizing TensorFlow, mirroring the principles elucidated in the research paper titled "Siamese Neural Networks for One-shot Image Recognition." Following the completion of model training, the subsequent step entails seamless integration into a Kivy application, facilitating real-world authentication capabilities.

2 Importing libraries

The initial stage involves the incorporation of pertinent libraries for our facial recognition project. Noteworthy among these are TensorFlow, OpenCV, Matplotlib, NumPy, and additional resources essential for the project's implementation.

[1]:
```
import os
import cv2
import io
import random
import time
import numpy as np
from contextlib import redirect_stdout
import matplotlib.pyplot as plt
```

[25]:
```
# Import tensorflow dependencies - Functional API
import tensorflow as tf
from tensorflow.keras.layers import Dropout
from tensorflow.keras import regularizers
from sklearn.metrics import precision_score, recall_score
from tensorflow.keras.models import Model
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import Layer, Conv2D, Dense, MaxPooling2D, Input,␣
↪Flatten
```

2.1 Create folder structure

[3]:
```
# Setup paths
POS_PATH = os.path.join('data', 'positive')
NEG_PATH = os.path.join('data', 'negative')
ANC_PATH = os.path.join('data', 'anchor')
```

[ ]:
```
# Make the directories
os.makedirs(POS_PATH)
os.makedirs(NEG_PATH)
os.makedirs(ANC_PATH)
```

3 Data Collection

3.1 Untar labelled faces in the Wild Dataset

- Labelled Faces in the Wild: http://vis-www.cs.umass.edu/lfw/

[ ]:
```
#!tar -xf lfw.tgz
```

[ ]:
```
# Move lfw images to the following repository data/negative
for directory in os.listdir('lfw'):
  for file_ in os.listdir(os.path.join('lfw', directory)):
    EX_PATH = os.path.join('lfw', directory, file_)
    NEW_PATH = os.path.join(NEG_PATH, file_)
    os.replace(EX_PATH, NEW_PATH)
```

It is pertinent to acknowledge that subsequent to the relocation of data from the *lfw* directory to the *negative* directory, the retention of the *lfw* directory becomes unnecessary due to its vacuity.

3.2 Collect positive and anchor classes

To gather anchor images, we will transition to the *Jupyter Lab* environment to address certain challenges associated with sharing the webcam in the *Google Colab* environment. Once a sufficient number of images have been gathered, we can return to *Google Colab* to progress with the project. Subsequently, we can leverage the `GPU` resources provided by *Google Colab* to expedite the model training process.

[ ]:
```
# Import uuid library to generate unique image names
import uuid
```

[ ]:
```
# Establish a connection to the webcam
cap = cv2.VideoCapture(0)
while cap.isOpened():
  _, frame = cap.read()

  # Cut down frame to 250x250px
  frame = frame[120:120+250, 200:200+250, :]
```

```
  # Collect anchores
  if cv2.waitKey(1) & 0xFF == ord('a'):
    # Create the unique file path
    imgname = os.path.join(ANC_PATH, '{}.jpg'.format(uuid.uuid1()))
    # Write out anchor image
    cv2.imwrite(imgname, frame)
  if cv2.waitKey(1) & 0xFF == ord('p'):
    # Create the unique file path
    imgname = os.path.join(POS_PATH, '{}.jpg'.format(uuid.uuid1()))
    # Write out anchor image
    cv2.imwrite(imgname, frame)

  cv2.imshow('Image Collection', frame)
  # Breaking gracefully
  if cv2.waitKey(1) & 0xFF == ord('q'):
    break

# Release the webcam & close the image show frame
cap.release()
cv2.destroyAllWindows()
```

3.2.1 Data Augmentation

[ ]:
```
def data_augmentation(img):
    """
    Apply data augmentation techniques to an input image.

    Args:
    - img: Input image tensor.

    Returns:
    - augmented_data: List of augmented images.
    """
    augmented_data = []

    # Check if the image is successfully loaded
    if img is not None:
        # Convert the NumPy array to a TensorFlow tensor
        for i in range(10):
            if i % 2 == 0:
                # Random brightness, contrast, and JPEG quality adjustments
                img = tf.image.stateless_random_brightness(img, max_delta=0.02,␣
↪seed=(1, 2))
                img = tf.image.stateless_random_contrast(img, lower=0.6,␣
↪upper=1, seed=(1, 2))
```

```
                img = tf.image.stateless_random_jpeg_quality(img,␣
↪min_jpeg_quality=90, max_jpeg_quality=100, seed=(np.random.randint(105), np.
↪random.randint(105)))
            else:
                # Additional contrast, flip left-right, saturation, and random␣
↪rotation (simulated)
                img = tf.image.stateless_random_contrast(img, lower=0.6,␣
↪upper=1, seed=(1, 2))
                img = tf.image.stateless_random_flip_left_right(img, seed=(np.
↪random.randint(105), np.random.randint(105)))
                img = tf.image.stateless_random_flip_up_down(img, seed=(np.
↪random.randint(105), np.random.randint(105)))

                # Simulating rotation with flip operations
                img = tf.image.transpose(img)

            augmented_data.append(img)
    else:
        print("Error: Image not loaded successfully.")

    return augmented_data
```

Visualize one sample

[ ]:
```
img_path = r"C:\Users\gavvi\Desktop\Programming\Computer Vision Projects\Facial␣
↪Verification Project\data\anchor\5a33aedf-901d-11ee-9879-646c8087a76e.jpg"
```

[ ]:
```
img = cv2.imread(img_path)
augmented_images_try = data_augmentation(img)
```

[ ]:
```
fig, axes = plt.subplots(3, 3, figsize=(10, 10))
axes = axes.flatten()

# Loop through the images and plot them
for i in range(len(augmented_images_try) -1):
    # Display each image on the corresponding subplot
    axes[i].imshow(cv2.cvtColor(augmented_images_try[i].numpy(), cv2.
↪COLOR_BGR2RGB))
    axes[i].axis('off')  # Turn off axis labels

# Adjust layout to prevent overlapping
plt.tight_layout()
plt.suptitle("Augmented images:", fontsize=25)
# Show the plot
plt.show()
```

Add the augmented data to our image dataset The subsequent phase involves processing all images within the positive and anchor directory to apply data augmentation techniques, thereby enhancing the generalization and robustness of our prospective model.

[ ]:
```
for file_name in os.listdir(os.path.join(POS_PATH)):
    img_path = os.path.join(POS_PATH, file_name)
    img = cv2.imread(img_path)
    # Pass img throw data_augmentation() function
    augmented_images = data_augmentation(img)

    # Add the new data to the positive images directory
    for image in augmented_images:
```

```
        cv2.imwrite(os.path.join(POS_PATH, '{}.jpg'.format(uuid.uuid1())),␣
↪image.numpy())
```

[ ]:
```
for file_name in os.listdir(os.path.join(ANC_PATH)):
    img_path = os.path.join(ANC_PATH, file_name)
    img = cv2.imread(img_path)
    # Pass img throw data_augmentation() function
    augmented_images = data_augmentation(img)

    # Add the new data to the anchor images directory
    for image in augmented_images:
        cv2.imwrite(os.path.join(ANC_PATH, '{}.jpg'.format(uuid.uuid1())),␣
↪image.numpy())
```

4 Data Preparation

4.1 Get image direcories

The next step is to create three datasets using `tf.data.Dataset`, one for each anchor, positive and negative images directories. We will take only the first *3000* images of each direcory to it's coresponding dataset.

[8]:
```
anchor = tf.data.Dataset.list_files(ANC_PATH+'/*.jpg').take(3000)
positive = tf.data.Dataset.list_files(POS_PATH+'/*.jpg').take(3000)
negative = tf.data.Dataset.list_files(NEG_PATH+'/*.jpg').take(3000)
```

[9]:
```
len(anchor), len(positive), len(negative)
```

[9]:
```
(3000, 3000, 3000)
```

4.2 Preprocessing - scale and resize

The next steps include:

- Resizing the image dimensions to 105x105 pixels is imperative within the context of our project, as it aligns with the specified dimensions required by the model outlined in the pertinentpappern. This adjustment ensures the images conform to the prescribed size parameters, contributing to the fidelity and compatibility with the underlying model architecture detailed in the associatedpappern
- Normalizin (scalling)g the image such that its pixel values are constrained within the range of [0,1]..

[10]:
```
def preprocess(file_path):
    try:
        byte_img = tf.io.read_file(file_path)
        img = tf.io.decode_jpeg(byte_img)

        # Resizing the image to 105x105x3
```

```
        img = tf.image.resize(img, (105, 105))

        # Normalizing pixel values
        img = img / 255.0

        return img

    except Exception as e:
        print(f"Error processing image {file_path}: {str(e)}")
        return None
```

[ ]:
```
img = preprocess("data/negative/Aaron_Peirsol_0003.jpg")
```

[ ]:
```
img.numpy().max(), img.numpy().min()
```

[ ]:
```
(1.0, 0.0)
```

[ ]:
```
img
```

[ ]:
```
<tf.Tensor: shape=(105, 105, 3), dtype=float32, numpy=
array([[[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.        ]],

       …,

       [[0.86399627, 0.9585587 , 0.85351646],
```

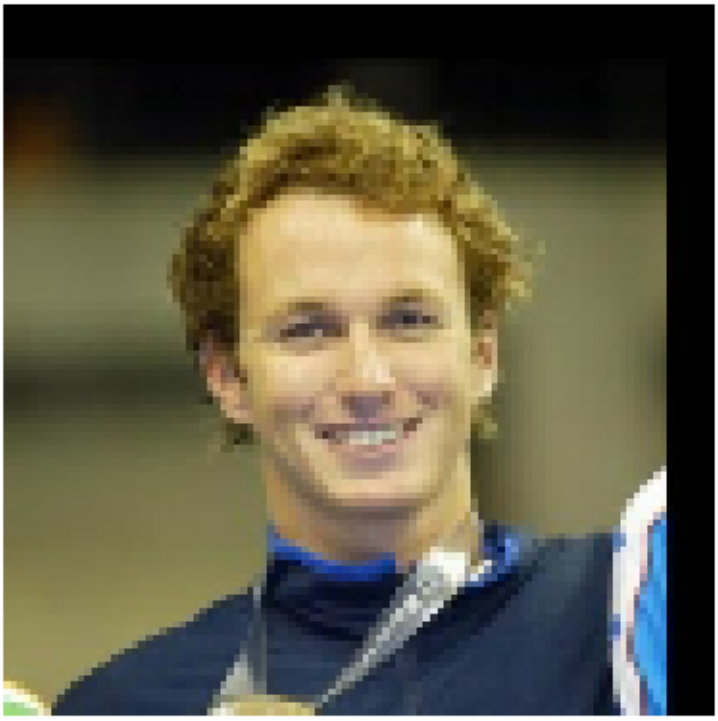
```
 [0.9433704 , 0.99787915, 0.93412006],
 [0.9824106 , 0.9921013 , 0.90370154],
 …,
 [0.        , 0.        , 0.        ],
 [0.        , 0.        , 0.        ],
 [0.        , 0.        , 0.        ]],

[[0.64537805, 0.85423505, 0.56246483],
 [0.630592  , 0.8112044 , 0.5380548 ],
 [0.71448547, 0.85690916, 0.6453443 ],
 …,
 [0.        , 0.        , 0.        ],
 [0.        , 0.        , 0.        ],
 [0.        , 0.        , 0.        ]],

[[0.62259567, 0.8630252 , 0.48804852],
 [0.6259571 , 0.86088437, 0.48917565],
 [0.6310102 , 0.8567805 , 0.5200636 ],
 …,
 [0.        , 0.        , 0.        ],
 [0.        , 0.        , 0.        ],
 [0.        , 0.        , 0.        ]]], dtype=float32)>
```

[ ]:
```
plt.imshow(img)
plt.axis(False)
```

[ ]:
```
(-0.5, 104.5, 104.5, -0.5)
```



4.3 Generate labelled dataset

Subsequently, we will generate a labeled dataset wherein the pairing of an anchor and a positive image results in an output of 1.0, while the combination of an anchor and a negative image produces an output of 0.0

(anchor, positive) => 1 (anchor, negative) => 0.

[11]:
```
positives = tf.data.Dataset.zip((anchor, positive, tf.data.Dataset.
  ↪from_tensor_slices(tf.ones(len(anchor)))))
negatives = tf.data.Dataset.zip((anchor, negative, tf.data.Dataset.
  ↪from_tensor_slices(tf.zeros(len(anchor)))))
data = positives.concatenate(negatives)
```

Visualize one representation of the data:

[ ]:
```
samples = data.as_numpy_iterator()

sample = samples.next()
sample
```

[ ]:
```
(b'data\\anchor\\5a210a65-9023-11ee-bedc-646c8087a76e.jpg',
 b'data\\positive\\3c3bed33-9023-11ee-a726-646c8087a76e.jpg',
 1.0)
```

Oour objective is to formulate a function aimed at preprocessing the values encapsulated within the variable`dat` .. This involves the utilization of a previously defined function denoted as`preprocess()`.

[12]:
```
def preprocess_twin(input_img, validation_img, label):
    return(preprocess(input_img), preprocess(validation_img), label)
```

[ ]:
```
preprocess_twin(*sample)
```

[ ]:
```
(<tf.Tensor: shape=(105, 105, 3), dtype=float32, numpy=
 array([[[0.31587523, 0.3315615 , 0.33548307],
         [0.32184872, 0.3369748 , 0.33305323],
         [0.3392157 , 0.34705883, 0.34313726],
         …,
         [0.48860875, 0.37880483, 0.32819796],
         [0.46703348, 0.35722956, 0.31017074],
         [0.46395892, 0.354155  , 0.30709618]],

        [[0.31575963, 0.33872882, 0.33116582],
         [0.334994  , 0.35744298, 0.3501401 ],
         [0.3620582 , 0.37774447, 0.3738229 ],
         …,
         [0.48620778, 0.3765306 , 0.32124847],
         [0.47873148, 0.36892757, 0.32078832],
         [0.48263305, 0.3728291 , 0.31792715]],

        [[0.3313014 , 0.34343958, 0.33596992],
         [0.34628516, 0.35842335, 0.35120714],
         [0.37437865, 0.38651684, 0.38259527],
         …,
         [0.49803922, 0.38743278, 0.32606822],
         [0.49794585, 0.38966253, 0.32956517],
         [0.49197012, 0.38394025, 0.32371616]],

        …,

        [[0.3026255 , 0.32400736, 0.3083211 ],
         [0.3198279 , 0.3413632 , 0.32567692],
         [0.32448313, 0.35122266, 0.33232626],
         …,
         [0.6942311 , 0.63309103, 0.5778156 ],
         [0.6840603 , 0.6213152 , 0.5660397 ],
         [0.688422  , 0.62567693, 0.57469654]],

        [[0.30785647, 0.31653994, 0.31177804],
         [0.29915965, 0.31538612, 0.30984393],
         [0.3137855 , 0.33367345, 0.31798717],
         …,
```

```
         [0.6811991 , 0.61117107, 0.55598897],
         [0.68857545, 0.61854744, 0.5633654 ],
         [0.6854342 , 0.61540616, 0.56078434]],

        [[0.33613667, 0.33613667, 0.33613667],
         [0.3085034 , 0.31634653, 0.31242496],
         [0.31763595, 0.32902715, 0.32510558],
         …,
         [0.6958917 , 0.62530345, 0.5704015 ],
         [0.6901961 , 0.61960787, 0.5647059 ],
         [0.69411767, 0.62352943, 0.5686275 ]]], dtype=float32)>,
<tf.Tensor: shape=(105, 105, 3), dtype=float32, numpy=
array([[[0.42745098, 0.34509805, 0.27058825],
         [0.44601175, 0.3471522 , 0.2787982 ],
         [0.49337068, 0.38394025, 0.3112045 ],
         …,
         [0.74454886, 0.6579008 , 0.58553857],
         [0.7351007 , 0.64490455, 0.5815993 ],
         [0.7475257 , 0.6573296 , 0.5945845 ]],

        [[0.42296916, 0.33782178, 0.25546885],
         [0.45186073, 0.3532613 , 0.27507004],
         [0.480799  , 0.37096173, 0.29766575],
         …,
         [0.7344271 , 0.6475256 , 0.58344   ],
         [0.74847937, 0.6481392 , 0.58933574],
         [0.7596638 , 0.6585434 , 0.6       ]],

        [[0.40773866, 0.316142  , 0.23201501],
         [0.45440173, 0.34823927, 0.26987463],
         [0.49434218, 0.37630165, 0.2958917 ],
         …,
         [0.7413454 , 0.6529234 , 0.59607846],
         [0.7520341 , 0.6636121 , 0.61035746],
         [0.7422969 , 0.65618694, 0.6025099 ]],

        …,

        [[0.6976879 , 0.5155217 , 0.4527766 ],
         [0.65970385, 0.47576365, 0.41896757],
         [0.64005375, 0.45771858, 0.39986652],
         …,
         [0.3348829 , 0.3833422 , 0.3929594 ],
         [0.3502601 , 0.39003602, 0.39367747],
         [0.34724557, 0.39038283, 0.37469655]],

        [[0.6493931 , 0.4690009 , 0.40989727],
```

```
          [0.635054  , 0.45544213, 0.39635846],
          [0.58336663, 0.4138989 , 0.34965986],
          …,
          [0.32452318, 0.37298253, 0.38259968],
          [0.32995194, 0.36972785, 0.3733693 ],
          [0.34565824, 0.38151258, 0.36946777]],

         [[0.6205438 , 0.4519163 , 0.38917118],
          [0.59495795, 0.42633054, 0.36358544],
          [0.56676   , 0.39813256, 0.33361343],
          …,
          [0.33548087, 0.38394022, 0.39355737],
          [0.3483727 , 0.38814858, 0.39179006],
          [0.35051355, 0.38580766, 0.37404296]]], dtype=float32)>,
 1.0)
```

[13]:
```
data = data.map(preprocess_twin)
data = data.cache()
data = data.shuffle(buffer_size=10000)
```

[ ]:
```
data
```

[ ]:
```
<_ShuffleDataset element_spec=(TensorSpec(shape=(105, 105, None),
dtype=tf.float32, name=None), TensorSpec(shape=(105, 105, None),
dtype=tf.float32, name=None), TensorSpec(shape=(), dtype=tf.float32,
name=None))>
```

[ ]:
```
samples = data.as_numpy_iterator()
sample = samples.next()

plt.imshow(sample[1])
plt.axis(False)
```

[ ]:
```
(-0.5, 104.5, 104.5, -0.5)
```

[ ]:
```
print(f'The label should be zero because it is negative image: \nLabel ->␣
↪{sample[2]}')
```

```
The label should be zero because it is negative image:
Label -> 0.0
```

4.4 Build train and test partition

[14]:
```
# Training partition
train_data = data.take(round(len(data)*.7))
train_data = train_data.batch(32)
train_data = train_data.prefetch(16)
```

[ ]:
```
train_samples = train_data.as_numpy_iterator()
train_sample = train_samples.next()

len(train_sample[0])
```

[ ]:
```
32
```

[15]:
```
# Testing partition
test_data = data.skip(round(len(data)*.7))
test_data = test_data.take(round(len(data)*.3))
test_data = test_data.batch(32)
```

```
test_data = test_data.prefetch(16)
```

4.5 Building the Siamese Neural Network Model

Following the establishment of our training and testing datasets, the subsequent task involves the construction of the Siamese neural network in accordance with the principles delineated in the scholarly article titled "Siamese Neural Networks for One-shot Image Recognition." (research papper: https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf)

This endeavor unfolds through several sequential steps:

1. **Embedding Layer Construction:**
   - The primary phase entails the creation of the embedding layer. This layer is instrumental in generating embedding vectors for both the anchor image and a secondary image, which may assume roles as either positive or negative instances.
2. **L1Dist Model Generation**
   - Subsequently, the development of the L1Dist model ensues. This model is designed to compute the similarity between the two vector embeddings by straightforwardly calculating the absolute distance existing between the respective vectors.
3. **Siamese Model Formulaton:**
   - The final phase encompasses the construction of the Siamese model. This model is tasked with generating embeddings for both input images. It proceeds to evaluate their similarity using the previously established L1Dist model. Ultimately, a Dense layer with a sigmoid activation function is employed to ascertain whether the input image is verified or not.

This methodological framework aligns with the referenced research paper, providing a systematic approach to the instantiation of a Siamese neural network for one-shot image recognition.

4.5.1 Building the embedding layer

[16]:
```
def make_embedding():
    inp = Input(shape=(105, 105, 3), name='input_image')

    # First block
    conv1 = Conv2D(64, (10, 10), activation='relu',␣
 ↪kernel_regularizer=regularizers.l2(0.001))(inp)
    batch_norm1 = BatchNormalization()(conv1)
    max_pool1 = MaxPooling2D((2, 2), padding='same')(batch_norm1)

    # Second block
    conv2 = Conv2D(128, (7, 7), activation='relu',␣
 ↪kernel_regularizer=regularizers.l2(0.001))(max_pool1)
    batch_norm2 = BatchNormalization()(conv2)
    max_pool2 = MaxPooling2D((2, 2), padding='same')(batch_norm2)

    # Third block
    conv3 = Conv2D(128, (4, 4), activation='relu',␣
 ↪kernel_regularizer=regularizers.l2(0.001))(max_pool2)
```

```
    batch_norm3 = BatchNormalization()(conv3)
    max_pool3 = MaxPooling2D((2, 2), padding='same')(batch_norm3)

    # Fourth block
    conv4 = Conv2D(256, (4, 4), activation='relu',␣
 ↪kernel_regularizer=regularizers.l2(0.001))(max_pool3)
    batch_norm4 = BatchNormalization()(conv4)
    fl = Flatten()(batch_norm4)

    dl = Dense(4096, activation='relu', kernel_regularizer=regularizers.l2(0.
 ↪01))(fl)

    return Model(inputs=[inp], outputs=[dl], name='embedding')
```

[17]:
```
embedding = make_embedding()

embedding.summary()
```

```
Model: "embedding"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_image (InputLayer)    [(None, 105, 105, 3)]     0

 conv2d (Conv2D)             (None, 96, 96, 64)        19264

 batch_normalization (Batch  (None, 96, 96, 64)        256
 Normalization)

 max_pooling2d (MaxPooling2  (None, 48, 48, 64)        0
 D)

 conv2d_1 (Conv2D)           (None, 42, 42, 128)       401536

 batch_normalization_1 (Bat  (None, 42, 42, 128)       512
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 21, 21, 128)       0
 g2D)

 conv2d_2 (Conv2D)           (None, 18, 18, 128)       262272

 batch_normalization_2 (Bat  (None, 18, 18, 128)       512
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 9, 9, 128)         0
 g2D)
```

```
 conv2d_3 (Conv2D)           (None, 6, 6, 256)         524544

 batch_normalization_3 (Bat  (None, 6, 6, 256)         1024
 chNormalization)

 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 4096)              37752832

=================================================================
Total params: 38962752 (148.63 MB)
Trainable params: 38961600 (148.63 MB)
Non-trainable params: 1152 (4.50 KB)
_________________________________________________________________
```

4.5.2 Building the distance layer

[18]:
```
# Siamese L1 Distance class
class L1Dist(Layer):
    def __init__(self, **kwargs):
        super().__init__()

    def call(self, input_embedding, validation_embedding):
        return tf.math.abs(input_embedding - validation_embedding)
```

4.5.3 Building Siamese Model

[19]:
```
def make_siamese_model():

    # Handle inputs:
    # Anchor image input in the network
    input_image = Input(name='input_image', shape=(105,105,3))

    # Validation image in the network
    validation_image = Input(name='validation_image', shape=(105,105,3))

    # Combine siamese distance
    siamese_layer = L1Dist()
    siamese_layer._name = 'distance'
    distances = siamese_layer(embedding(input_image),␣
 ↪embedding(validation_image))

    # Classification layer
    classifier = Dense(1, activation='sigmoid', name='classifier')(distances)

    return Model(inputs=[input_image, validation_image], outputs=classifier,␣
 ↪name='Siamese_Network')
```

[ ]:
```
siamese_model = make_siamese_model()

siamese_model.summary()
```

```
Model: "Siamese_Network"
__________________________________________________________________________________
__________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================
==================
 input_image (InputLayer)    [(None, 105, 105, 3)]        0         []

 validation_image (InputLay  [(None, 105, 105, 3)]        0         []
 er)

 embedding (Functional)      (None, 4096)                 3896275
['input_image[0][0]',
                                                          2
'validation_image[0][0]']

 distance (L1Dist)           (None, 4096)                 0
['embedding[0][0]',
'embedding[1][0]']

 classifier (Dense)          (None, 1)                    4097
['distance[0][0]']

==================================================================================
==================
Total params: 38966849 (148.65 MB)
Trainable params: 38965697 (148.64 MB)
Non-trainable params: 1152 (4.50 KB)
__________________________________________________________________________________
__________________
```

4.6 Train the model using Google Collab interface

In order to expedite the training process and leverage GPU acceleration, we will transition from the Jupyter Lab interface to Google Colab. Before initiating the training, it's essential to clone our project to ensure all directories and images required for training are available. Subsequently, we'll execute all the previous cells necessary for running the training process.

[4]:
```
!git clone https://github.com/Gavision97/Computer-Vision.git
```

```
Cloning into 'Computer-Vision'…
remote: Enumerating objects: 20396, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 20396 (delta 1), reused 6 (delta 1), pack-reused 20386
```

```
Receiving objects: 100% (20396/20396), 311.37 MiB | 20.98 MiB/s, done.
Resolving deltas: 100% (2/2), done.
Updating files: 100% (20437/20437), done.
```

[5]:
```
cd "Computer-Vision"
```

```
/content/Computer-Vision
```

[6]:
```
cd "Facial Verification Project"
```

```
/content/Computer-Vision/Facial Verification Project
```

Set GPU Growth Avoid OOM errors by setting GPU Memory Consumption Growth

[7]:
```
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
```

[23]:
```
from tensorflow.keras.metrics import Recall, Precision
import io
from contextlib import redirect_stdout
```

4.6.1 Establish checkpoints

[20]:
```
checkpoint_dir = './training_checkpoints'
os.makedirs(checkpoint_dir, exist_ok=True)
checkpoint_prefix = os.path.join(checkpoint_dir, 'ckpt')
```

[21]:
```
EPOCHS = 100
```

4.6.2 Train the model

[26]:
```
siamese_model = make_siamese_model()
siamese_model.compile(
    optimizer=tf.keras.optimizers.Adam(1e-3),
    loss=tf.losses.BinaryCrossentropy(),
)

desired_loss_threshold = 0.01
min_epochs = 50

for epoch in range(EPOCHS):
    # Reset metrics at the beginning of each epoch
    siamese_model.reset_metrics()

    # Lists to store true labels and predicted labels for each batch
    true_labels = []
    predicted_labels = []
```

```
    # Iterate over batches in the training dataset
    for batch in train_data:
        # Train the model on the current batch
        loss = siamese_model.train_on_batch(batch[:2], batch[2])

        # Collect true labels and predicted labels
        true_labels.extend(batch[2])
        predicted_labels.extend(siamese_model.predict_on_batch(batch[:2]).
↪flatten())

    # Convert lists to numpy arrays
    true_labels = np.array(true_labels)
    predicted_labels = np.array(predicted_labels)

    # Calculate precision and recall using sklearn metrics
    precision = precision_score(true_labels, (predicted_labels > 0.5).
↪astype(int))
    recall = recall_score(true_labels, (predicted_labels > 0.5).astype(int))

    if epoch % 10 == 0:
        print(f"Epoch: {epoch}, Loss: {loss}, Recall: {recall}, Precision:␣
↪{precision}")

    # Save checkpoints
    if (epoch + 1) % 10 == 0:
        siamese_model.save_weights(checkpoint_prefix.format(epoch=epoch + 1))

    # Check if the minimum number of epochs has been reached
    if epoch >= min_epochs - 1 and loss < desired_loss_threshold:
        print(f"Training stopped. Loss below threshold␣
↪({desired_loss_threshold}) and at least {min_epochs} epochs completed.")
        siamese_model.save_weights(checkpoint_prefix.format(epoch=epoch + 1))
        break

print(f"Epoch: {epoch+1}, Loss: {loss}, Recall: {recall}, Precision:␣
↪{precision}")
```

```
Epoch: 0, Loss: 2.1126108169555664, Recall: 0.4358851674641148, Precision:
0.8141197497765862
Epoch: 10, Loss: 0.3297222852706909, Recall: 0.7992333493052228, Precision:
0.9569707401032702
Epoch: 20, Loss: 0.6592720150947571, Recall: 0.9560699102503543, Precision:
0.7524163568773234
Epoch: 30, Loss: 0.17207109928131104, Recall: 0.975563009103977, Precision:
0.9318077803203662
Epoch: 40, Loss: 0.1198141872882843, Recall: 0.9985556090515166, Precision:
0.9342342342342342
```

```
Epoch: 50, Loss: 0.08846653252840042, Recall: 0.9961941008563273, Precision:
0.9156099693922168
Epoch: 60, Loss: 0.1926010251045227, Recall: 0.9919239904988123, Precision:
0.9538602101416171
Epoch: 70, Loss: 0.06231403350830078, Recall: 1.0, Precision: 0.9892873777363763
Epoch: 80, Loss: 0.19533447921276093, Recall: 0.9436485195797517, Precision:
0.8035786905246035
Epoch: 90, Loss: 0.08153785765171051, Recall: 0.994847775175644, Precision:
0.9672131147540983
Epoch: 100, Loss: 0.07429824769496918, Recall: 0.928030303030303, Precision:
0.974155069582505
```

[27]:
```
# Save the entire model to a single file
siamese_model.save("_siamese_model_.h5")
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079:
UserWarning: You are saving your model as an HDF5 file via `model.save()`. This
file format is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

Now that we have completed training the model and achieved excellent training evaluation results (e.g., low loss ~0 and precision/recall both equal to 1), and we've saved the model, we can return to the Jupyter Lab environment since no further GPU is needed.

4.7 Real time test

The next step is to conduct a real-time test using the webcam's feed to obtain live input images. We will then use these input images for verification against a set of positive samples. Specifically, we will select a number of images from our positive directory, say 50 images, and iterate through each of these images. During each iteration, we will predict whether the input image and each of the selected images are a match or no, and finally if the person is verified or nott.

To achieve this, we will establish a verification threshold. If the prediction surpasses this threshold, we will consider it a match (assigned a label of 1); otherwise, it will be labeled as 0. Additionally, a detection threshold will be defined. If the prediction exceeds this threshold, it signifies that the individual captured in the real-time image has been successfully verified.

[ ]:
```
# Reaload the model
siamese_model = tf.keras.models.load_model('_siamese_model_.h5',
                                           custom_objects={'L1Dist':L1Dist,
                                                           'BinaryCrossentropy':
 ↪tf.losses.BinaryCrossentropy})
```

[ ]:
```
def verify(model, detection_threshold, verification_threshold):

    input_image = preprocess(os.path.join('application_data', 'input_image',␣
 ↪'input_image.jpg'))
    res = []
```

```
    for image in os.listdir(os.path.join('application_data',␣
↪'verification_image')):
        validation_image = preprocess(os.path.join('application_data',␣
↪'verification_image', image))

        result = model.predict(list(np.expand_dims([input_image,␣
↪validation_image], axis=1)), verbose=0)
        res.append(result)

    detection = np.sum(np.array(res) > detection_threshold)
    verification = detection / len(os.path.join('application_data',␣
↪'verification_image'))
    verified = verification > verification_threshold

    return res, verified
    # Detection Threshold: Metric above which a prediction a prediction is␣
↪considered positive
    # Verefication Threshold: Proportion of positive predictions / total␣
↪positive samples
```

4.8 OpenCV real time verification

[ ]:
```
cap = cv2.VideoCapture(0)
while cap.isOpened():
    ret, frame = cap.read()

    # Cut down frame to 250x250px
    frame = frame[120:120+250, 200:200+250, :]

    cv2.imshow('Verification', frame)

    # Verification trigger
    if cv2.waitKey(10) & 0xFF == ord('v'):
        # Save the input image inside the input image directory
        cv2.imwrite(os.path.join('application_data', 'input_image',␣
↪'input_image.jpg'), frame)
        # Execute verification
        res, verified = verify(siamese_model, 0.85, 0.5)
        print(verified)

    if cv2.waitKey(10) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

```
True
True
True
False
False
False
False
False
True
True
False
False
True
True
```

[ ]: