Setup

installing dependencies

In [163]:

```
# !pip install tensorflow
```

In [164]:

```
# !pip install opencv-python matplotlib
```

In [165]:

```
# !pip install --upgrade pip
```

Importing dependencies

In [166]:

```
import cv2
import os
import random
import numpy as np
from matplotlib import pyplot as plt
```

In [167]:

```
# plt.imshow??
```

In [168]:

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Layer, Conv2D, Dense, MaxPooling2D, Input, Flatten
```

In [169]:

```
import tensorflow as tf
```

crating seperate folder structure for data

In [170]:

```
POS_PATH = os.path.join('data', 'positive')
NEG_PATH = os.path.join('data', 'negative')
ANC_PATH = os.path.join('data', 'anchor')
```

In [171]:

```
# import os

# # making the directories
# os.makedirs(POS_PATH)
# os.makedirs(NEG_PATH)
# os.makedirs(ANC_PATH)
```

Collecting the data, positive and negative anchors

In [172]:

```
# import os
# 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)
# #         print(EX_PATH)
#         os.replace(EX_PATH, NEW_PATH)
# #         shutil.move(EX_PATH, NEG_PATH)
```

In [173]:

```
# os.listdir('lfw')
```

In [185]:

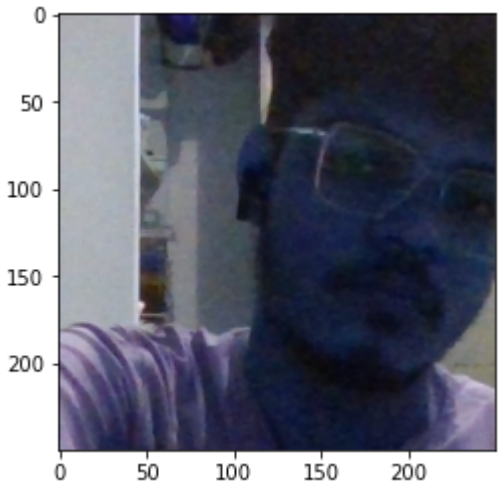
```
cap = cv2.VideoCapture(0)
import uuid

while cap.isOpened():

    ret, frame = cap.read()
    frame = frame[100:100+250, 200:200+250,:]

    if cv2.waitKey(1) & 0XFF == ord('a'):
        imgname = os.path.join(ANC_PATH, '{}.jpg'.format(uuid.uuid1()))
        cv2.imwrite(imgname, frame)

    if cv2.waitKey(1) & 0XFF == ord('p'):
        imgname = os.path.join(POS_PATH, '{}.jpg'.format(uuid.uuid1()))
        cv2.imwrite(imgname, frame)

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

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

In [175]:

```
plt.imshow(frame)
```

Out[175]:

```
<matplotlib.image.AxesImage at 0x1e59f32d340>
```



preprocessing the collected images

get image directories

In [220]:

```
anchor = tf.data.Dataset.list_files(ANC_PATH+'\*.jpg').take(300)
positive = tf.data.Dataset.list_files(POS_PATH+'\*.jpg').take(300)
negative = tf.data.Dataset.list_files(NEG_PATH+'\*.jpg').take(300)
```

In [240]:

```
ANC_PATH+'\*.jpg'
```

Out[240]:

```
'data\\anchor\\*.jpg'
```

In [241]:

```
dir_test = anchor.as_numpy_iterator()
dir_test = dir_test.next()
dir_test
```

Out[241]:

```
b'data\\anchor\\3e214f49-2bb9-11ee-b309-204ef6489e9c.jpg'
```

In [242]:

```
# import numpy as np
# from PIL import Image
# # plt.imshow()
# image = Image.open('data\\anchor\\6273d927-2ba8-11ee-86fa-204ef6489e9c.jpg')
# np.array(dir_test)
```

In [244]:

```
def preprocess(file_path):

    byte_img = tf.io.read_file(file_path)
    img = tf.io.decode_jpeg(byte_img)
    img = tf.image.resize(img, (100,100))
    img = img / 255.0

    return img
```

In [246]:

```
# preprocess(dir_test)
```

In [247]:

```
# creating the dataset

positives = tf.data.Dataset.zip((anchor, positive, tf.data.Dataset.from_tensor_slices(tf
negatives = tf.data.Dataset.zip((anchor, negative, tf.data.Dataset.from_tensor_slices(tf
data = positives.concatenate(negatives)
```

In [248]:

```
sample = data.as_numpy_iterator()
```

In [249]:

```
eg = sample.next()
eg
```

Out[249]:

```
(b'data\\anchor\\f4a6b971-2bb8-11ee-affa-204ef6489e9c.jpg',
 b'data\\positive\\b28fdf55-2bba-11ee-a888-204ef6489e9c.jpg',
 1.0)
```

In [ ]:

```

```

splitting into train and test

In [252]:

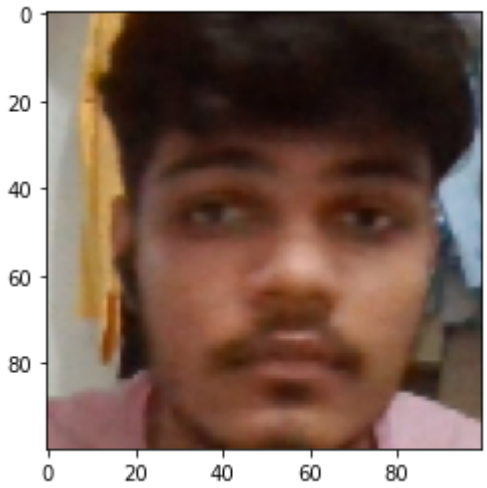
```
def preprocess_twin(input_img, validation_img, label):
    return (preprocess(input_img) , preprocess(validation_img) , label)
```

In [260]:

```
# preprocess_twin(*eg)
res = preprocess_twin(*eg)
plt.imshow(res[0])
```

Out[260]:

```
<matplotlib.image.AxesImage at 0x1e5a42ee760>
```

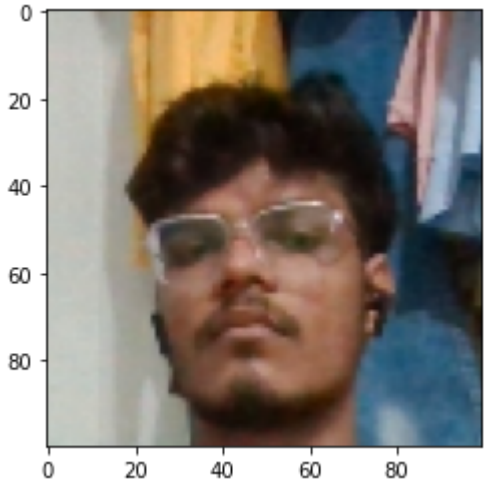


In [261]:

```
plt.imshow(res[1])
```

Out[261]:

```
<matplotlib.image.AxesImage at 0x1e5a4346160>
```



In [263]:

```
print(res[2])
```

```
1.0
```

In [264]:

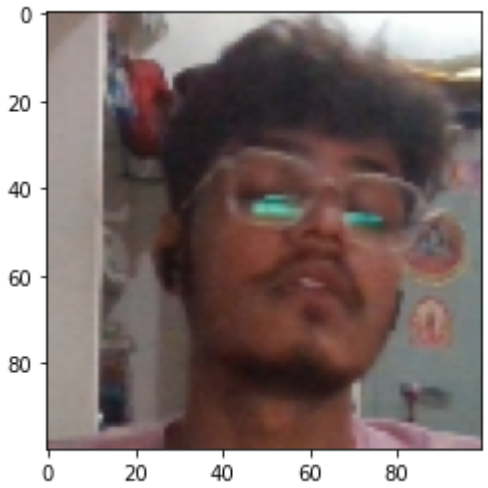
```
data = data.map(preprocess_twin)
data = data.cache()
data = data.shuffle(buffer_size = 1024)
```

In [269]:

```
plt.imshow(data.as_numpy_iterator().next()[0])
```

Out[269]:

```
<matplotlib.image.AxesImage at 0x1e5a443d1f0>
```

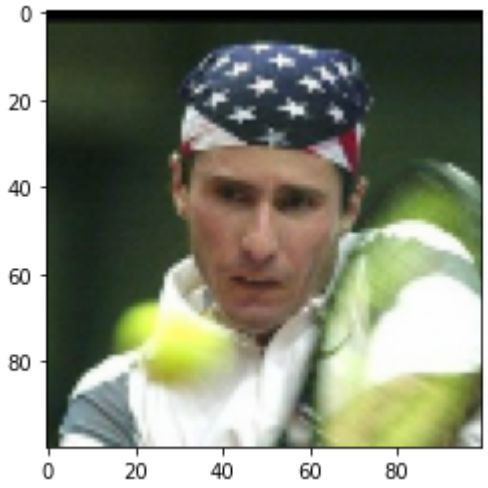


In [270]:

```
plt.imshow(data.as_numpy_iterator().next()[1])
```

Out[270]:

```
<matplotlib.image.AxesImage at 0x1e5a4493880>
```



In [271]:

```
data.as_numpy_iterator().next()[2]
```

Out[271]:

```
0.0
```

In [273]:

```
# training partitian
train_data = data.take(round(len(data)*0.7))
train_data = train_data.batch(16)
train_data = train_data.prefetch(8)
```

In [274]:

```
train_data
```

Out[274]:

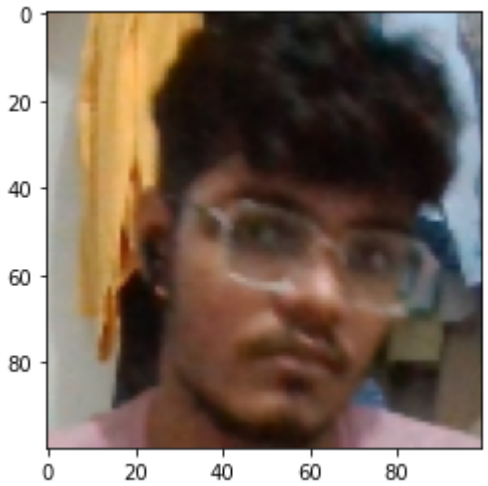
```
<PrefetchDataset element_spec=(TensorSpec(shape=(None, 100, 100, None), d
type=tf.float32, name=None), TensorSpec(shape=(None, 100, 100, None), dty
pe=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.float32, na
me=None))>
```

In [283]:

```
train_samples = train_data.as_numpy_iterator()
train_sample = train_samples.next()
plt.imshow(train_sample[0][6])
```

Out[283]:

```
<matplotlib.image.AxesImage at 0x1e5b0784940>
```



In [284]:

```
#  we have batch of 16
len(train_sample[0])
```

Out[284]:

```
16
```

In [291]:

```
# testing partitian
test_data = data.skip(round(len(data)*0.7))
test_data = test_data.batch(16)
test_data = test_data.prefetch(8)
# test_data = data.take(round(len(data)*0.3)) # no sense writing it
```

In [292]:

```
test_data
```

Out[292]:

```
<PrefetchDataset element_spec=(TensorSpec(shape=(None, 100, 100, None), d
type=tf.float32, name=None), TensorSpec(shape=(None, 100, 100, None), dty
pe=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.float32, na
me=None))>
```

In [293]:

```
round(len(data)*0.7)
```

Out[293]:

```
420
```

Modeling

In [302]:

```
def make_embedding():

    # first block
    inp = Input(shape = (100, 100, 3), name = 'input_image')
    c1 = Conv2D(64, (10, 10), activation = 'relu')(inp)
    m1 = MaxPooling2D(64, (2, 2), padding = 'same')(c1)

    # second block
    c2 = Conv2D(128, (7,7), activation = 'relu')(m1)
    m2 = MaxPooling2D(64, (2, 2), padding = 'same')(c2)

    # third block
    c3 = Conv2D(128, (4,4), activation = 'relu')(m2)
    m3 = MaxPooling2D(64, (2, 2), padding = 'same')(c3)

    # fouth layer
    c4 = Conv2D(256, (4,4), activation = 'relu')(m3)
    f1 = Flatten()(c4)
    d1 = Dense(4096, activation = 'sigmoid')(f1)

    return Model(inputs = [inp], outputs = [d1], name = 'embedding')
```

In [312]:

```
embedding = make_embedding()
embedding.summary()
```

```
Model: "embedding"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_image (InputLayer)    [(None, 100, 100, 3)]     0

 conv2d_20 (Conv2D)          (None, 91, 91, 64)        19264

 max_pooling2d_16 (MaxPoolin  (None, 46, 46, 64)       0
 g2D)

 conv2d_21 (Conv2D)          (None, 40, 40, 128)       401536

 max_pooling2d_17 (MaxPoolin  (None, 20, 20, 128)      0
 g2D)

 conv2d_22 (Conv2D)          (None, 17, 17, 128)       262272

 max_pooling2d_18 (MaxPoolin  (None, 9, 9, 128)        0
 g2D)

 conv2d_23 (Conv2D)          (None, 6, 6, 256)         524544

 flatten_3 (Flatten)         (None, 9216)              0

 dense_3 (Dense)             (None, 4096)              37752832

=================================================================
Total params: 38,960,448
Trainable params: 38,960,448
Non-trainable params: 0
_________________________________________________________________
```

now creating the siamese distance layer

In [308]:

```
# will create a custom nn layer
class L1Dist(Layer):

    def __init__(self, **kwargs):
        super().__init__()

    # the crux of the paper lies here(charaterstics of the siamese nn)!
    def call(self, input_embedding, validation_embedding):
        return tf.math.abs(input_embedding - validation_embedding)
```

In [310]:

```
l1 = L1Dist()
l1
```

Out[310]:

```
<__main__.L1Dist at 0x1e5b0b10e80>
```

In [339]:

```
def make_siamese_model():

    input_image = Input(name = 'input_img', shape = (100, 100, 3))
    validation_image = Input(name = 'validation_img', shape = (100, 100, 3))

    siamese_layer = L1Dist(name = 'distance')
#     siamese_layer.name = 'distance'
    distances = siamese_layer(embedding(input_image), embedding(validation_image))

    # final classification layer
    classifier = Dense(1, activation = 'sigmoid')(distances)

    return Model(inputs = [input_image, validation_image], outputs = classifier, name =
```

In [341]:

```
siamese_model = make_siamese_model()
siamese_model.summary()
```

```
Model: "SiameseNetwork"
__________________________________________________________________________
_______________________
 Layer (type)                Output Shape              Param #     Connecte
d to
==========================================================================
========================
 input_img (InputLayer)      [(None, 100, 100, 3  0           []
                             )]

 validation_img (InputLayer)  [(None, 100, 100, 3  0           []
                             )]

 embedding (Functional)      (None, 4096)              38960448    ['input_
img[0][0]',
                                                                  'valida
tion_img[0][0]']

 l1_dist_25 (L1Dist)         (None, 4096)              0           ['embedd
ing[28][0]',
                                                                  'embedd
ing[29][0]']

 dense_18 (Dense)            (None, 1)                 4097        ['l1_dis
t_25[0][0]']

==========================================================================
========================
Total params: 38,964,545
Trainable params: 38,964,545
Non-trainable params: 0
__________________________________________________________________________
_______________________
```

Training the model

setting up loss functions and optimizer

In [344]:

```
binary_cross_loss = tf.losses.BinaryCrossentropy() # recommended set logits = true when
opt = tf.keras.optimizers.Adam(1e-4)
```

creating the checkpoints

In [345]:

```
checkpoint_dir = "./training_checkpoints"
checkpoint_prefix = os.path.join(checkpoint_dir, 'ckpt')
checkpoint = tf.train.Checkpoint(opt = opt, siamese_model = siamese_model)
```

creating the training step

In [369]:

```
test_batch = test_data.as_numpy_iterator().next()
len(test_batch)

# understanding the data

# np.array(test_batch).shape
# x = test_batch[:2]
# np.array(x).shape
# len(test_batch[1])

# np.array(sample[0]).shape
# len(sample[0][0][0][0])
```

Out[369]:

```
3
```

In [374]:

```
@tf.function # this line will make the follwing function will integrate the following fu
def train_step(batch):

    # recording all of our operations
    with tf.GradientTape() as tape:

        # getting anchor and positive images
        X = batch[:2]
        y = batch[2]

        # forward pass
        yhat = siamese_model(X, training = True) # setting training = true madetory for
        # calculate loss
        loss = binary_cross_loss(y, yhat)

    print(loss)

    # calculating gradients
    grad = tape.gradient(loss, siamese_model.trainable_variables)

    # calcualtes updated weights and apply to the siamese network
    opt.apply_gradients(zip(grad, siamese_model.trainable_variables))

    return loss
```

creating a loop for training the model

In [375]:

```
def train(data, EPOCHS):

    for epoch in range(1, EPOCHS + 1):

        print('\n Epoch {}/{}'.format(epoch, EPOCHS))
        progbar = tf.keras.utils.Progbar(len(data)) # to inculcate the progression!

        for idx, batch in enumerate(data):
            train_step(batch)
            progbar.update(idx + 1)

        if epoch % 10 == 0:
            checkpoints.save(file_prefix = checkpoint_prefix)
```

Training the model

In [376]:

```
EPOCHS = 50
train(train_data, EPOCHS)
```

...

lets get to google collab to train the model

In [ ]: