In [5]:

```
pip install tensorflow
```

...

In [6]:

```
pip install tensorflow-gpu
```

...

In [1]:

```
pip install opencv-python
```

```
Requirement already satisfied: opencv-python in c:\users\ferdi\anaconda3\lib\site-packages (4.6.0.66)
Requirement already satisfied: numpy>=1.19.3 in c:\users\ferdi\anaconda3\lib\site-packages (from opencv-python) (1.19.5)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:

```
pip install matplotlib
```

...

In [7]:

```
# Import standard dependencies
import cv2
import os
import random
import numpy as np
from matplotlib import pyplot as plt
```

In [8]:

```
# Import tensorflow dependencies - Functional API
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Layer, Conv2D, Dense, MaxPooling2D, Input, Flatten
import tensorflow as tf
```

In [9]:

```
# Avoid OOM errors by setting GPU Memory Consumption Growth
import tensorflow as tf
gpus =  tf.config.experimental.list_physical_devices('GPU')
print(len(gpus))
```

```
0
```

In [2]:

```
gpus
```

Out[2]:

```
[]
```

In [10]:

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

In [ ]:

```

```

In [11]:

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

...

In [12]:

```
def data_aug(img):
    data = []
    for i in range(9):
        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=(1,3))
        # img = tf.image.stateless_random_crop(img, size=(20,20,3), seed=(1,2))
        img = tf.image.stateless_random_flip_left_right(img, seed=(np.random.randint(100),np.random.randint(100)))
        img = tf.image.stateless_random_jpeg_quality(img, min_jpeg_quality=90, max_jpeg_quality=100, seed=(np.random.randint(100),np.rando
        img = tf.image.stateless_random_saturation(img, lower=0.9,upper=1, seed=(np.random.randint(100),np.random.randint(100)))

        data.append(img)

    return data
```

In [13]:

```
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)
```

In [14]:

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

In [15]:

```
print(dir_test.next())
```

```
b'data\\anchor\\Potato___Early_blight288.jpg'
```

In [16]:

```
def preprocess(file_path):

    # Read in image from file path
    byte_img = tf.io.read_file(file_path)
    # Load in the image
    img = tf.io.decode_jpeg(byte_img)

    # Preprocessing steps - resizing the image to be 100x100x3
    img = tf.image.resize(img, (105,105))
    # Scale image to be between 0 and 1
    img = img / 256.0

    # Return image
    return img
```

In [17]:

```
img = preprocess('data\\anchor\\Potato___Early_blight692.jpg')
```

img.numpy().max()

In [18]:

```
img.numpy().max()
```

Out[18]:

```
0.9360081
```

In [123]:

```
plt.imshow(img)
```

Out[123]:

```
<matplotlib.image.AxesImage at 0x1fd41660f10>
```

In [19]:

```
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)
```

In [20]:

```
samples = data.as_numpy_iterator()
```

In [21]:

```
exampple = samples.next()
```

In [22]:

```
exampple
```

Out[22]:

```
(b'data\\anchor\\Potato___Early_blight578.jpg',
 b'data\\positive\\Potato_Late_blight638.jpg',
 1.0)
```

TRAIN KISMI

In [23]:

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

In [24]:

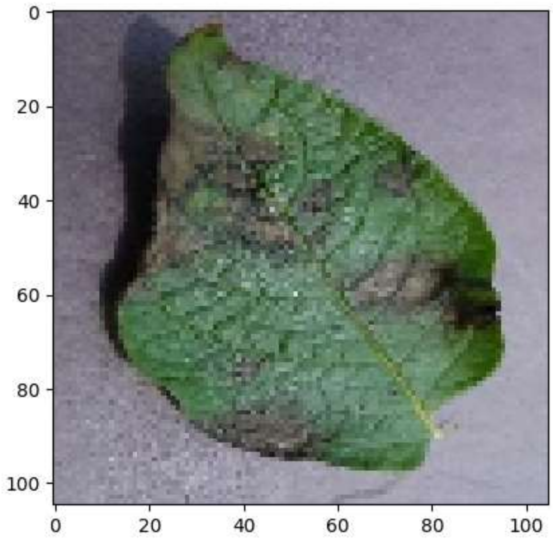
```
res = preprocess_twin(*exampple)
```

In [25]:

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

Out[25]:

```
<matplotlib.image.AxesImage at 0x2b2e4e976a0>
```



In [26]:

```
res[2]
```

Out[26]:

```
1.0
```

In [27]:

```
# Build dataloader pipeline
data = data.map(preprocess_twin)
data = data.cache()
data = data.shuffle(buffer_size=10000)
```

In [28]:

```
# Training partition
train_data = data.take(round(len(data)*.7))
train_data = train_data.batch(16)
train_data = train_data.prefetch(8)
```

In [29]:

```
# Testing partition
test_data = data.skip(round(len(data)*.7))
test_data = test_data.take(round(len(data)*.3))
test_data = test_data.batch(16)
test_data = test_data.prefetch(8)
```

MODELLEME KISMI

In [30]:

```
inp = Input(shape=(105,105,3), name='input_image')
c1 = Conv2D(64, (10,10), activation='relu')(inp)
m1 = MaxPooling2D(64, (2,2), padding='same')(c1)
c2 = Conv2D(128, (7,7), activation='relu')(m1)
m2 = MaxPooling2D(64, (2,2), padding='same')(c2)
c3 = Conv2D(128, (4,4), activation='relu')(m2)
m3 = MaxPooling2D(64, (2,2), padding='same')(c3)
c4 = Conv2D(256, (4,4), activation='relu')(m3)
f1 = Flatten()(c4)
d1 = Dense(4096, activation='sigmoid')(f1)
```

c1 = Conv2D(64, (10,10), activation='relu')(inp)

m1 = MaxPooling2D(64, (2,2), padding='same')(c1)

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

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

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

In [31]:

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

In [32]:

```
mod.summary()
```

```
Model: "embedding"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_image (InputLayer)     [(None, 105, 105, 3)]     0
_________________________________________________________________
conv2d (Conv2D)              (None, 96, 96, 64)        19264
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 48, 48, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 42, 42, 128)       401536
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 21, 21, 128)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 18, 18, 128)       262272
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 9, 9, 128)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 6, 6, 256)         524544
_________________________________________________________________
flatten (Flatten)            (None, 9216)              0
_________________________________________________________________
dense (Dense)                (None, 4096)              37752832
=================================================================
Total params: 38,960,448
Trainable params: 38,960,448
Non-trainable params: 0
_________________________________________________________________
```

In [33]:

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

    # First block
    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)

    # Final embedding block
    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 [34]:

```
embedding = make_embedding()
```

In [35]:

```
embedding.summary()
```

```
Model: "embedding"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_image (InputLayer)     [(None, 105, 105, 3)]     0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 96, 96, 64)        19264
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 48, 48, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 42, 42, 128)       401536
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 21, 21, 128)       0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 18, 18, 128)       262272
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 9, 9, 128)         0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 6, 6, 256)         524544
_________________________________________________________________
flatten_1 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              37752832
=================================================================
Total params: 38,960,448
Trainable params: 38,960,448
Non-trainable params: 0
_________________________________________________________________
```

In [36]:

```
# Siamese L1 Distance class
class L1Dist(Layer):

    # Init method - inheritance
    def __init__(self, **kwargs):
        super().__init__()

    # Magic happens here - similarity calculation
    def call(self, input_embedding, validation_embedding):
        return tf.math.abs(input_embedding - validation_embedding)
```

In [37]:

```
l1 = L1Dist()
```

In [38]:

```
input_image = Input(name='input_img', shape=(105,105,3))
validation_image = Input(name='validation_img', shape=(105,105,3))
```

In [39]:

```
inp_embedding = embedding(input_image)
val_embedding = embedding(validation_image)
```

In [40]:

```
siamese_layer = L1Dist()
```

In [41]:

```
distances = siamese_layer(inp_embedding, val_embedding)
```

In [42]:

```
classifier = Dense(1, activation='sigmoid')(distances)
```

In [43]:

```
classifier
```

Out[43]:

```
<tf.Tensor 'dense_2/Sigmoid:0' shape=(None, 1) dtype=float32>
```

In [44]:

```
siamese_network = Model(inputs=[input_image, validation_image], outputs=classifier, name='SiameseNetwork')
```

In [45]:

```
siamese_network.summary()
```

```
Model: "SiameseNetwork"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_img (InputLayer)          [(None, 105, 105, 3) 0
__________________________________________________________________________________________________
validation_img (InputLayer)     [(None, 105, 105, 3) 0
__________________________________________________________________________________________________
embedding (Functional)          (None, 4096)         38960448    input_img[0][0]
                                                                 validation_img[0][0]
__________________________________________________________________________________________________
l1_dist_1 (L1Dist)              (None, 4096)         0           embedding[0][0]
                                                                 embedding[1][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 1)            4097        l1_dist_1[0][0]
==================================================================================================
Total params: 38,964,545
Trainable params: 38,964,545
Non-trainable params: 0
__________________________________________________________________________________________________
```

In [46]:

```
def make_siamese_model():

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

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

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

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

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

In [47]:

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

```
Model: "SiameseNetwork"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_img (InputLayer)          [(None, 105, 105, 3) 0
__________________________________________________________________________________________________
validation_img (InputLayer)     [(None, 105, 105, 3) 0
__________________________________________________________________________________________________
embedding (Functional)          (None, 4096)         38960448    input_img[0][0]
                                                                 validation_img[0][0]
__________________________________________________________________________________________________
distance (L1Dist)               (None, 4096)         0           embedding[2][0]
                                                                 embedding[3][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 1)            4097        distance[0][0]
==================================================================================================
Total params: 38,964,545
Trainable params: 38,964,545
Non-trainable params: 0
__________________________________________________________________________________________________
```

In [48]:

```
binary_cross_loss = tf.losses.BinaryCrossentropy()
```

In [49]:

```
opt = tf.keras.optimizers.Adam(1e-4) # 0.0001
```

In [50]:

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

In [51]:

```
test_batch = train_data.as_numpy_iterator()
batch_1 = test_batch.next()
X = batch_1[:2]
y = batch_1[2]
```

In [52]:

```
y
```

Out[52]:

```
array([1., 0., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 0., 1.],
      dtype=float32)
```

In [54]:

```
np.array(X)
```

...

In [55]:

```
@tf.function
def train_step(batch):

    # Record all of our operations
    with tf.GradientTape() as tape:
        # Get anchor and positive/negative image
        X = batch[:2]
        # Get label
        y = batch[2]

        # Forward pass
        yhat = siamese_model(X, training=True)
        # Calculate loss
        loss = binary_cross_loss(y, yhat)
    print(loss)

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

    # Calculate updated weights and apply to siamese model
    opt.apply_gradients(zip(grad, siamese_model.trainable_variables))

    # Return loss
    return loss
```

In [56]:

```
# Import metric calculations
from tensorflow.keras.metrics import Precision, Recall
```

In [57]:

```
def train(data, EPOCHS):
    # Loop through epochs
    for epoch in range(1, EPOCHS+1):
        print('\n Epoch {}/{}'.format(epoch, EPOCHS))
        progbar = tf.keras.utils.Progbar(len(data))

        # Creating a metric object
        r = Recall()
        p = Precision()

        # Loop through each batch
        for idx, batch in enumerate(data):
            # Run train step here
            loss = train_step(batch)
            yhat = siamese_model.predict(batch[:2])
            r.update_state(batch[2], yhat)
            p.update_state(batch[2], yhat)
            progbar.update(idx+1)
        print(loss.numpy(), r.result().numpy(), p.result().numpy())

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

In [58]:

```
EPOCHS = 10
```

In [ ]:

```
train(train_data, EPOCHS)
```

```

 Epoch 1/10
Tensor("binary_crossentropy/weighted_loss/value:0", shape=(), dtype=float32)
Tensor("binary_crossentropy/weighted_loss/value:0", shape=(), dtype=float32)
49/50 [============================>.] - ETA: 34s Tensor("binary_crossentropy/weighted_loss/value:0", shape=(), dtype=float
32)
50/50 [==============================] - 1680s 34s/step
0.07182552 0.9957386 0.89641947

 Epoch 2/10
50/50 [==============================] - 1577s 32s/step
3.1034079 1.0 0.89258313

 Epoch 3/10
50/50 [==============================] - 1503s 30s/step
0.04339584 0.99712646 0.88746804

 Epoch 4/10
18/50 [=========>....................] - ETA: 16:14
```

```
MODEL EĞİTİMİNİ TAMAMLAYAMADIK
```

In [ ]:

```

```