In [141]:
```python
import cv2
import os
import random
import numpy as np
from matplotlib import pyplot as plt
import uuid
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Layer, Conv2D, Dense, MaxPooling2D, Input
import tensorflow as tf
from tensorflow.keras.metrics import Precision, Recall
```

In [142]:
```python
cwd = os.getcwd()
```

In [143]:
```python
print(cwd)
```

```
/home/kumar.amit1/ondemand/data/sys/myjobs/projects/default/1/data
```

In [144]:
```python
#!unzip data.zip
```

In [145]:
```python
POS_PATH ='data_crop/positive_crop'
NEG_PATH ='data_crop/negative_crop'
ANC_PATH ='data_crop/anchor_crop'
```

Loading and Processing Image

In [146]:
```python
# Get Image directory
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 [147]:
```
# Sacle and Resize
def preprocess_img(file_path):

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

    # Preprocessing steps - resizing the image to be 100x100x3
    img = tf.image.resize(img, (224,224))
    # Scale image to be between 0 and 1
    img = img / 255.0

    # Return image
    return img
```

Creating Labels

In [66]:
```
positives = tf.data.Dataset.zip((anchor, positive, tf.data.Dataset.from_tenso
negatives = tf.data.Dataset.zip((anchor, negative, tf.data.Dataset.from_tenso
data = positives.concatenate(negatives)
```

In [67]:
```
samples = data.as_numpy_iterator()
```

In [68]:
```
exmp = samples.next()
print(exmp)
```

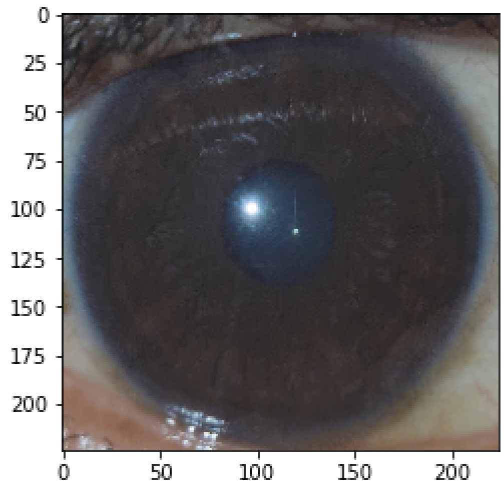
```
(b'data_crop/anchor_crop/RE2327.jpg', b'data_crop/positive_crop/RE300.jpg',
1.0)
```

Train and Test Partiton

In [69]:
```
def preprocess_twin(input_img, validation_img, label):
    return(preprocess(input_img), preprocess(validation_img), label)
```

In [70]:
```python
res = preprocess_twin(*exampple)
plt.imshow(res[1])
print(res[2])
```

```
1.0
```



In [71]:
```python
# Build dataloader pipeline
data = data.map(preprocess_twin)
data = data.cache()
data = data.shuffle(buffer_size=10000)
```

In [72]:
```python
# Training partition
train_data = data.take(round(len(data)*.7))
train_data = train_data.batch(16)
train_data = train_data.prefetch(8)
```

In [73]:
```python
#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)
```

Model Engineering

Embedding Model

In [108]:
```
def form_embedding_R50():
    #efficeint net B4 architecture
    inputlayer = Input(shape=(224,224,3), name='input_layer')
    m1 = tf.keras.applications.ResNet50(include_top=True,
                                        weights=None,
                                        input_tensor=None,
                                        input_shape= None ,
                                        pooling=None,
                                        #classes=2,
                                        classifier_activation='sigmoid',
                                        )(inputlayer)

    return Model(inputs=[inputlayer],outputs = [m1],name='ResNet50')
```

In [109]:
```
# Efficenetnet embedding
embedding_R50 = form_embedding_R50()
```

distance Layer

In [110]:
```
# 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)
```

Siamese network

In [111]:
```
def make_siamese_model():

    # Anchor image input in the network
    input_image = Input(name='input_img', shape=(224,224,3))

    # Validation image in the network
    validation_image = Input(name='validation_img', shape=(224,224,3))

    # Combine siamese distance components
    siamese_layer = L1Dist()
    siamese_layer._name = 'distance'
    #distances = siamese_layer(embedding_efB4(input_image), embedding_efB4(va
    distances = siamese_layer(embedding_R50(input_image), embedding_R50(valid

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

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

In [112]:
```
siamese_model = make_siamese_model()
```

In [113]:
```
siamese_model.summary()
```

```
Model: "SiameseNetwork"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected t
o
==================================================================================================
input_img (InputLayer)          [(None, 224, 224, 3) 0
__________________________________________________________________________________________________
validation_img (InputLayer)     [(None, 224, 224, 3) 0
__________________________________________________________________________________________________
ResNet50 (Functional)           (None, 1000)         25636712    input_img
[0][0]
                                                                 validation_
img[0][0]
__________________________________________________________________________________________________
distance (L1Dist)               (None, 1000)         0           ResNet50[0]
[0]
                                                                 ResNet50[1]
[0]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 1)            1001        distance[0]
[0]
==================================================================================================
Total params: 25,637,713
Trainable params: 25,584,593
Non-trainable params: 53,120
__________________________________________________________________________________________________
```

Training

In [114]:
```
# Loss and Optimier
binary_cross_loss = tf.losses.BinaryCrossentropy()
opt = tf.keras.optimizers.Adam(1e-4)
```

In [115]:
```
# establish checkpoint
checkpoint_dir = './training_checkpoints_efB4'
checkpoint_prefix = os.path.join(checkpoint_dir, 'ckpt')
checkpoint = tf.train.Checkpoint(opt=opt, siamese_model=siamese_model)
```

In [116]:

```python
# Train Step Function
@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 [117]:

```python
# Training Loop
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)
```

Train The model

In [120]:
```
EPOCHS = 50
```

In [121]:
```
train(train_data, EPOCHS)
```

```
 Epoch 1/500
45/45 [==============================] - 4s 94ms/step
1.1170022 0.7485549 0.94871795

 Epoch 2/500
45/45 [==============================] - 4s 94ms/step
9.415203e-05 0.9101449 1.0

 Epoch 3/500
45/45 [==============================] - 4s 93ms/step
1.2741654 0.30882353 0.990566

 Epoch 4/500
45/45 [==============================] - 4s 92ms/step
1.5240006 0.0 0.0

 Epoch 5/500
45/45 [==============================] - 4s 93ms/step
```

Evaluating Model

In [129]:
```
# Batch of test data
test_input, test_val, y_true = test_data.as_numpy_iterator().next()
y_hat = siamese_model.predict([test_input, test_val])
```

In [130]:
```
# Post processing the results
[1 if prediction > 0.5 else 0 for prediction in y_hat ]
print(y_true)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1.]
```

Comparision Matrix

In [131]:
```
# Creating a metric object
m = Recall()

# Calculating the recall value
m.update_state(y_true, y_hat)

# Return Recall Result
m.result().numpy()
```

Out[131]:
```
1.0
```

In [132]:
```
r = Recall()
p = Precision()

for test_input, test_val, y_true in test_data.as_numpy_iterator():
    yhat = siamese_model.predict([test_input, test_val])
    r.update_state(y_true, yhat)
    p.update_state(y_true,yhat)

print(r.result().numpy(), p.result().numpy())
```

```
1.0 0.8791209
```

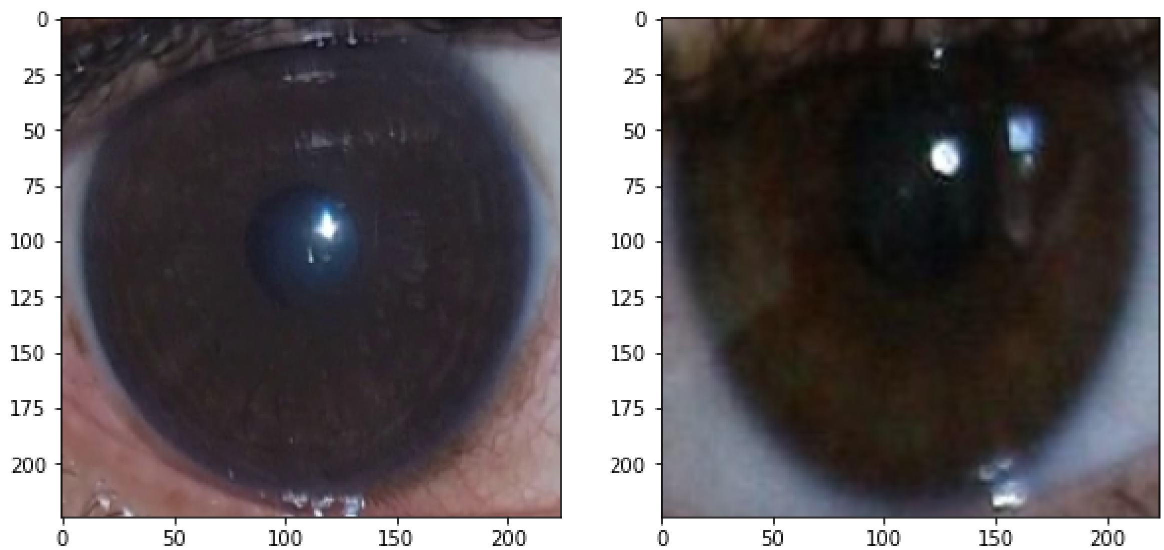
Result

In [137]:
```
# Set plot size
plt.figure(figsize=(10,8))

# Set first subplot
plt.subplot(1,2,1)
plt.imshow(test_input[1])

# Set second subplot
plt.subplot(1,2,2)
plt.imshow(test_val[1])

# Renders cleanly
plt.show()
```



Save Model

In [138]:
```
#Save model weights
siamese_model.save('siamesemodelv2R50.h5')
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have
yet to be built. `model.compile_metrics` will be empty until you train or ev
aluate the model.
```

In [139]:
```
# Reload model
siamese_model = tf.keras.models.load_model('siamesemodelv2R50.h5',
                                   custom_objects={'L1Dist':L1Dist, 'BinaryCr
```

```
WARNING:tensorflow:No training configuration found in the save file, so the
model was *not* compiled. Compile it manually.
```

In [140]:
```
# Make predictions with reloaded model
siamese_model.predict([test_input, test_val])
```

Out[140]:
```
array([[0.9999865 ],
       [0.54936284],
       [0.06182227],
       [0.23521104],
       [0.04769669],
       [0.9897593 ],
       [0.99999   ],
       [0.9999895 ],
       [0.9997813 ],
       [0.01169477],
       [0.9999987 ],
       [0.00250948],
       [0.9580088 ],
       [0.80119497]], dtype=float32)
```

In [ ]:
```

```