In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:

```
!mkdir -p ~/.kaggle
!mv kaggle.json ~/.kaggle/
```

In [5]:

```
!kaggle datasets download -d salader/dogs-vs-cats
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fix this, you
can run 'chmod 600 /root/.kaggle/kaggle.json'
Downloading dogs-vs-cats.zip to /content
 99% 1.05G/1.06G [00:11<00:00, 154MB/s]
100% 1.06G/1.06G [00:11<00:00, 98.3MB/s]
```

In [6]:

```
import zipfile
zip_ref = zipfile.ZipFile('/content/dogs-vs-cats.zip','r')
zip_ref.extractall('/content')
zip_ref.close()
```

In [7]:

```
import tensorflow as tf
from tensorflow import keras
from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, BatchNormalization, Dropo
ut
from keras import Sequential
```

In [8]:

```
train_data = keras.utils.image_dataset_from_directory(
    directory = '/content/train',
    labels="inferred",
    label_mode="int",
    batch_size=32,
    image_size=(256, 256)
)

test_data = keras.utils.image_dataset_from_directory(
    directory = '/content/test',
    labels="inferred",
    label_mode="int",
    batch_size=32,
    image_size=(256, 256)
)
```

```
Found 20000 files belonging to 2 classes.
Found 5000 files belonging to 2 classes.
```

In [9]:

```
type(train_data)
```

Out[9]:

```
tensorflow.python.data.ops.prefetch_op._PrefetchDataset
```

In [10]:

```
for images, labels in train_data.take(1):
```

```
    print("Batch of images:", images.shape)
    print("Batch of labels:", labels)
```

```
Batch of images: (32, 256, 256, 3)
Batch of labels: tf.Tensor([0 1 1 1 0 0 0 0 1 0 1 1 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 0 1 1
0 1], shape=(32,), dtype=int32)
```

In [11]:

```
# we need to normalize our image data
def normalize(image,label):
  image = tf.cast(image/255, tf.float32)
  return image, label
```

In [12]:

```
train_data = train_data.map(normalize)
```

In [13]:

```
test_data = test_data.map(normalize)
```

In [14]:

```
model = Sequential()

model.add(Conv2D(32, kernel_size=(3,3),padding='valid', activation='relu', input_shape=(
256,256,3)))
model.add(MaxPooling2D(pool_size=(2,2),strides=2,padding='valid'))

model.add(Conv2D(64, kernel_size=(3,3), padding='valid', activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2),strides=2,padding='valid'))

model.add(Conv2D(128, kernel_size=(3,3),padding='valid', activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2),strides=2,padding='valid'))

model.add(Flatten())

model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1,activation='sigmoid'))
```

In [15]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 254, 254, 32)      896

 max_pooling2d (MaxPooling2  (None, 127, 127, 32)      0
 D)

 conv2d_1 (Conv2D)           (None, 125, 125, 64)      18496

 max_pooling2d_1 (MaxPoolin  (None, 62, 62, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 60, 60, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 30, 30, 128)       0
 g2D)

 flatten (Flatten)           (None, 115200)            0

 dense (Dense)               (None, 128)               14745728

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 1)                 65
```

```
dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 14847297 (56.64 MB)
Trainable params: 14847297 (56.64 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [16]:

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [17]:

```
history = model.fit(train_data,epochs=8, validation_data=test_data)
```

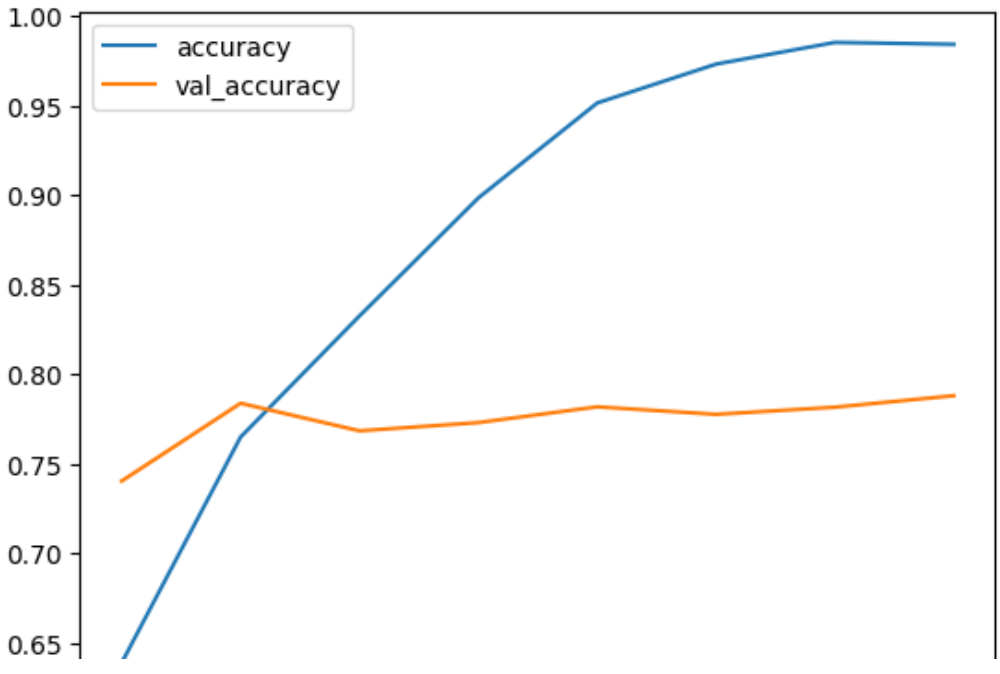
```
Epoch 1/8
625/625 [==============================] - 89s 132ms/step - loss: 0.6248 - accuracy: 0.63
90 - val_loss: 0.5241 - val_accuracy: 0.7404
Epoch 2/8
625/625 [==============================] - 78s 124ms/step - loss: 0.4811 - accuracy: 0.76
49 - val_loss: 0.4617 - val_accuracy: 0.7838
Epoch 3/8
625/625 [==============================] - 70s 110ms/step - loss: 0.3722 - accuracy: 0.83
25 - val_loss: 0.5073 - val_accuracy: 0.7684
Epoch 4/8
625/625 [==============================] - 53s 84ms/step - loss: 0.2393 - accuracy: 0.898
4 - val_loss: 0.5847 - val_accuracy: 0.7730
Epoch 5/8
625/625 [==============================] - 57s 91ms/step - loss: 0.1258 - accuracy: 0.951
3 - val_loss: 0.8162 - val_accuracy: 0.7818
Epoch 6/8
625/625 [==============================] - 55s 87ms/step - loss: 0.0744 - accuracy: 0.973
0 - val_loss: 1.2892 - val_accuracy: 0.7776
Epoch 7/8
625/625 [==============================] - 57s 90ms/step - loss: 0.0491 - accuracy: 0.985
1 - val_loss: 1.2086 - val_accuracy: 0.7816
Epoch 8/8
625/625 [==============================] - 57s 90ms/step - loss: 0.0475 - accuracy: 0.984
1 - val_loss: 1.2306 - val_accuracy: 0.7880
```

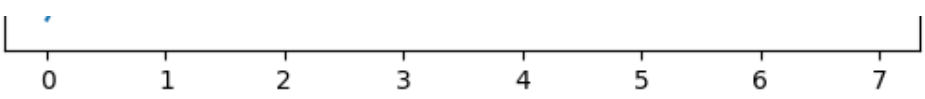
In [18]:

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.legend()
```

Out[18]:

```
<matplotlib.legend.Legend at 0x7dd58fc1bb80>
```



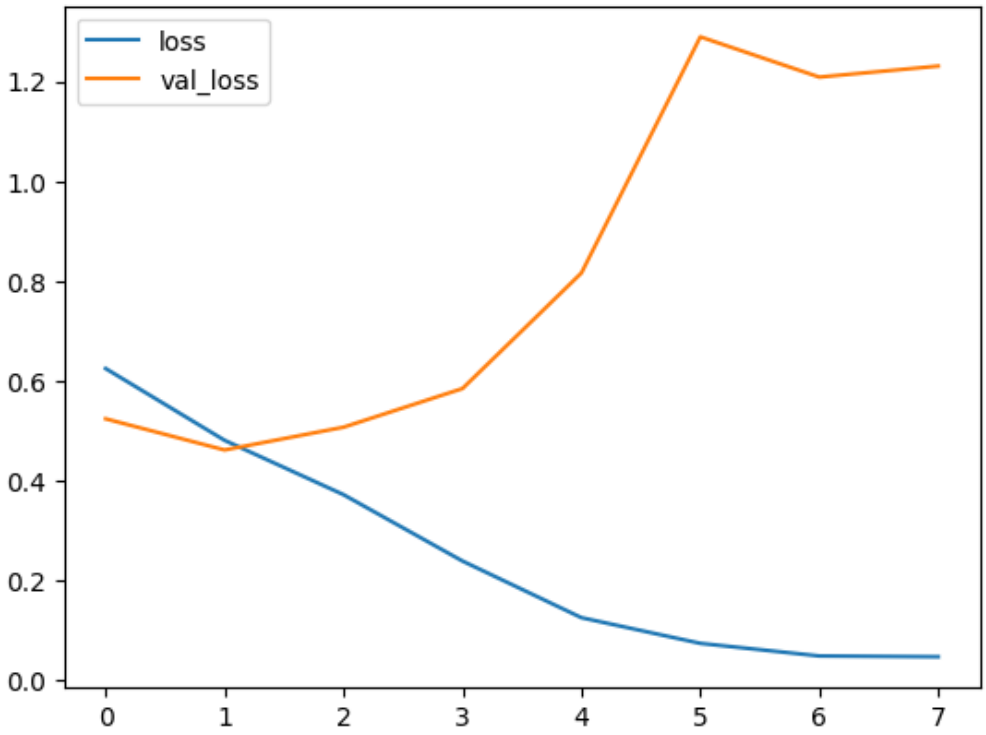


In [19]:

```
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.legend()
```

Out[19]:

```
<matplotlib.legend.Legend at 0x7dd53c107d60>
```



In [20]:

```
# overfitting
# 1-->> Try to reduce overfitting using Dropout layer
```

In [21]:

```
# model1 = Sequential()

# model1.add(Conv2D(32, kernel_size=(3,3),padding='valid', activation='relu', input_shape
=(256,256,3)))
# model1.add(Dropout(0.3))
# model1.add(MaxPooling2D(pool_size=(2,2),strides=2,padding='valid'))

# model1.add(Conv2D(64, kernel_size=(3,3), padding='valid', activation='relu'))
# model1.add(Dropout(0.25))
# model1.add(MaxPooling2D(pool_size=(2,2),strides=2,padding='valid'))

# model1.add(Conv2D(128, kernel_size=(3,3),padding='valid', activation='relu'))
# model1.add(Dropout(0.2))
# model1.add(MaxPooling2D(pool_size=(2,2),strides=2,padding='valid'))

# model1.add(Flatten())

# model1.add(Dense(128, activation='relu'))
# model1.add(Dense(64, activation='relu'))
# model1.add(Dense(1,activation='sigmoid'))
```

In [22]:

```
# model1.compile(
#     optimizer='adam',
```

```
#     loss='binary_crossentropy',
#     metrics=['accuracy']
# )
```

In [23]:

```
# history1 = model1.fit(train_data,epochs=10, validation_data=test_data)
```

In [24]:

```
import cv2
```

In [25]:

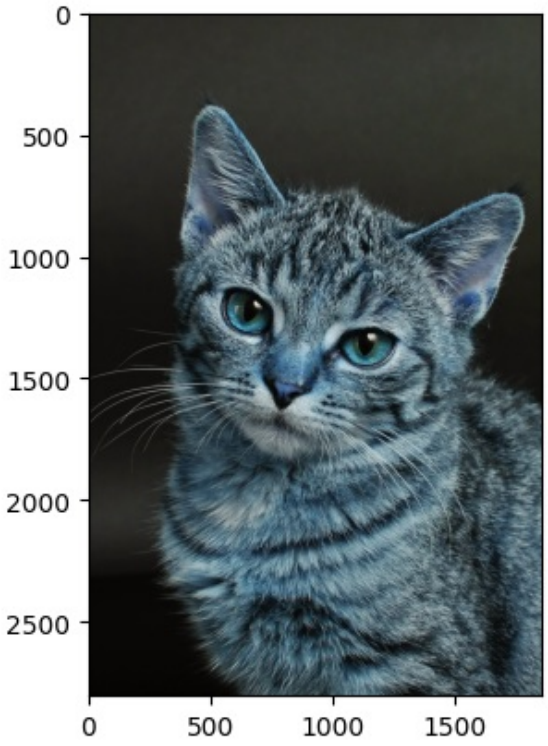
```
test_img1 = cv2.imread('/content/cat.jpg')
```

In [26]:

```
plt.imshow(test_img1)
```

Out[26]:

```
<matplotlib.image.AxesImage at 0x7dd4b41dac80>
```



In [27]:

```
image1 = cv2.imread('/content/cat1.jpeg') #cat image
image2 = cv2.imread('/content/dog2.jpeg') # dog image
```

In [28]:

```
image1 = cv2.resize(image1, (256,256))
image2 = cv2.resize(image2, (256,256))
```

In [29]:

```
image1 = image1.reshape((1,256,256,3))
image2 = image2.reshape((1,256,256,3))
```

In [31]:

```
prediction_cat = model.predict(image1)
prediction_dog = model.predict(image2)

print("Prediction of cat image is :", prediction_cat[0][0])
```

```
print("Prediction of dog image is :", prediction_dog[0][0])
```

```
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 45ms/step
Prediction of cat image is : 0.0
Prediction of dog image is : 1.0
```

In [32]:

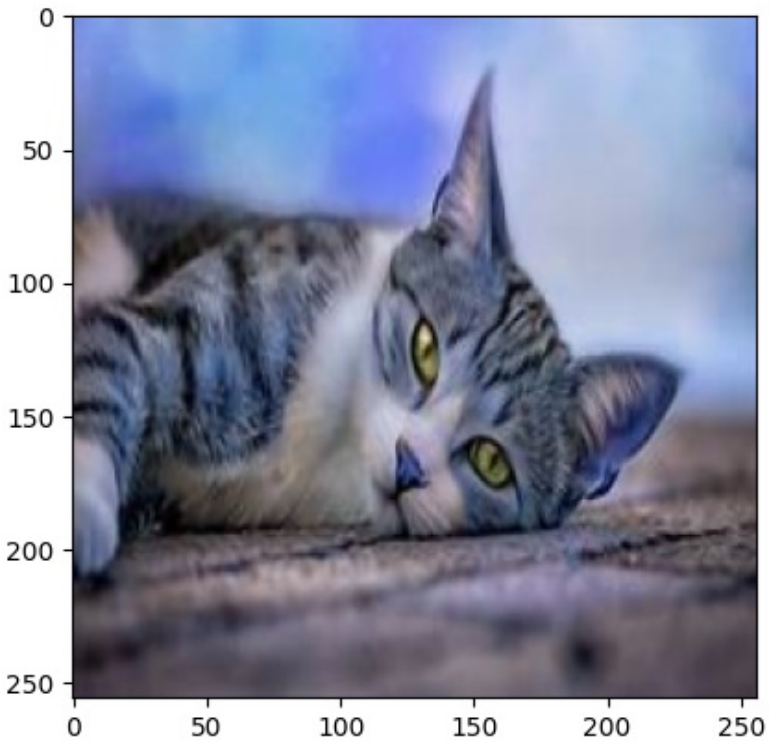
```
def Cat_or_Dog(image):
  image = cv2.resize(image,(256,256))
  temp_image = image
  image = image.reshape((1,256,256,3))
  prediction = model.predict(image)[0][0]
  plt.imshow(temp_image)

  if prediction == 1:
    print("The Image in the picture is DOG□□□")
  else:
    print("The Image in the picture is CAT□😹")
```

In [33]:

```
image = cv2.imread('/content/cat2.jpeg')
Cat_or_Dog(image)
```

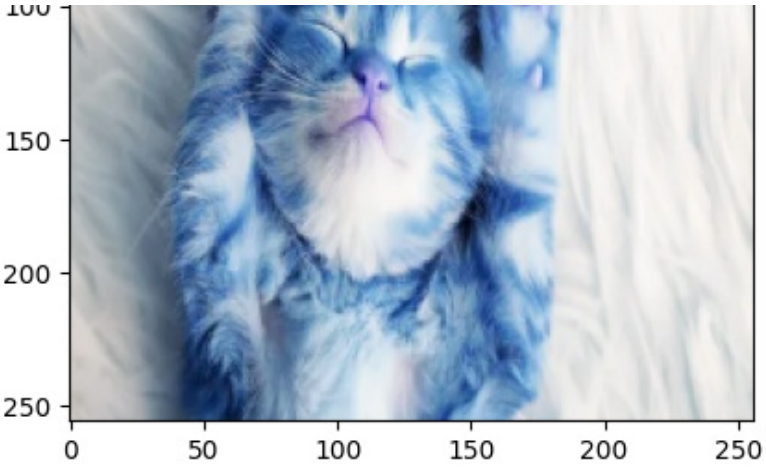
```
1/1 [==============================] - 0s 20ms/step
The Image in the picture is CAT□😹
```



In [34]:

```
image = cv2.imread('/content/cat3.jpeg')
Cat_or_Dog(image)
```

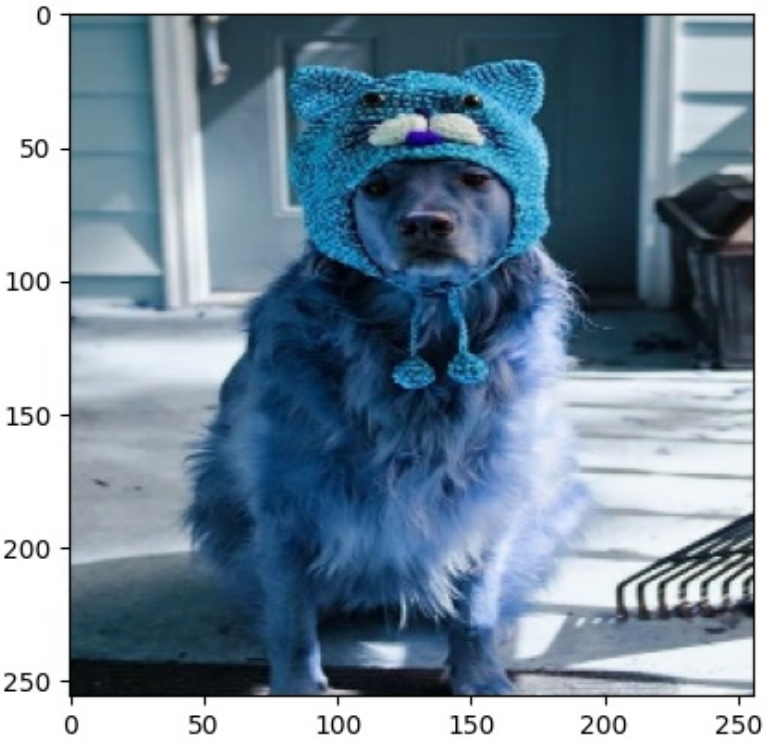
```
1/1 [==============================] - 0s 18ms/step
The Image in the picture is CAT□😹
```



In [35]:

```
image = cv2.imread('/content/dog3.jpg')
Cat_or_Dog(image)
```

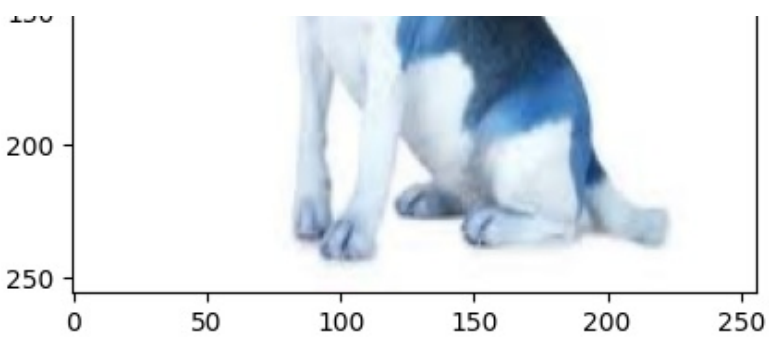
```
1/1 [==============================] - 0s 34ms/step
The Image in the picture is DOG□□□
```



In [37]:

```
image = cv2.imread('/content/dim.jpeg')
Cat_or_Dog(image)
```

```
1/1 [==============================] - 0s 22ms/step
The Image in the picture is DOG□□□
```



In [38]:

```
import pickle
pickle.dump(model,open('model.pkl','wb'))
```

In [ ]:

```

```