In [ ]:
```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
```

In [ ]:
```
!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 /roo
t/.kaggle/kaggle.json'
Downloading dogs-vs-cats.zip to /content
100% 1.06G/1.06G [00:10<00:00, 116MB/s]
100% 1.06G/1.06G [00:10<00:00, 107MB/s]
```

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

In [ ]:
```
import tensorflow as tf
from tensorflow import keras
from keras import Sequential
from keras.layers import Dense,Conv2D,MaxPooling2D,Flatten,BatchNormalization,Dropout
```

In [ ]:
```
# generators
train_ds = keras.utils.image_dataset_from_directory(
    directory = '/content/train',
    labels='inferred',
    label_mode = 'int',
    batch_size=32,
    image_size=(256,256)
)

validation_ds = 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 [ ]:
```
# Normalize
def process(image,label):
    image = tf.cast(image/255. ,tf.float32)
    return image,label

train_ds = train_ds.map(process)
validation_ds = validation_ds.map(process)
```

In [ ]:
```
# create CNN model

model = Sequential()

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

model.add(Flatten())

model.add(Dense(128,activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(64,activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(1,activation='sigmoid'))
```

In [ ]:
```
model.summary()
```

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

 batch_normalization (Batch  (None, 254, 254, 32)      128
 Normalization)

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

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

 batch_normalization_1 (Bat  (None, 125, 125, 64)      256
 chNormalization)

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

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

 batch_normalization_2 (Bat  (None, 60, 60, 128)       512
 chNormalization)

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

 flatten (Flatten)           (None, 115200)            0

 dense (Dense)               (None, 128)               14745728

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 64)                8256

 dropout_1 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 14848193 (56.64 MB)
Trainable params: 14847745 (56.64 MB)
Non-trainable params: 448 (1.75 KB)
_________________________________________________________________
```

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

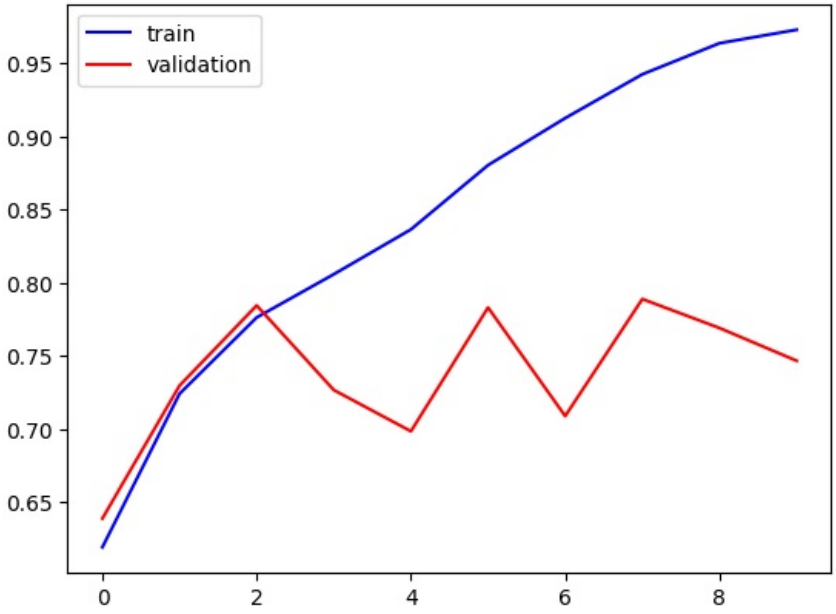
In [ ]:
```
history = model.fit(train_ds,epochs=10,validation_data=validation_ds)
```

```
Epoch 1/10
625/625 [==============================] - 78s 109ms/step - loss: 1.0535 - accuracy: 0.6191 - val_loss: 0.6123 -
val_accuracy: 0.6388
Epoch 2/10
625/625 [==============================] - 65s 103ms/step - loss: 0.5403 - accuracy: 0.7240 - val_loss: 0.5215 -
val_accuracy: 0.7296
Epoch 3/10
625/625 [==============================] - 64s 102ms/step - loss: 0.4728 - accuracy: 0.7762 - val_loss: 0.4573 -
val_accuracy: 0.7844
Epoch 4/10
625/625 [==============================] - 67s 107ms/step - loss: 0.4247 - accuracy: 0.8058 - val_loss: 0.5990 -
val_accuracy: 0.7266
Epoch 5/10
625/625 [==============================] - 64s 102ms/step - loss: 0.3660 - accuracy: 0.8364 - val_loss: 0.6744 -
val_accuracy: 0.6984
Epoch 6/10
625/625 [==============================] - 65s 104ms/step - loss: 0.2842 - accuracy: 0.8803 - val_loss: 0.5301 -
val_accuracy: 0.7830
Epoch 7/10
625/625 [==============================] - 67s 106ms/step - loss: 0.2107 - accuracy: 0.9125 - val_loss: 0.7736 -
val_accuracy: 0.7088
Epoch 8/10
625/625 [==============================] - 64s 102ms/step - loss: 0.1464 - accuracy: 0.9424 - val_loss: 0.7021 -
val_accuracy: 0.7888
Epoch 9/10
625/625 [==============================] - 63s 101ms/step - loss: 0.1000 - accuracy: 0.9637 - val_loss: 0.7793 -
val_accuracy: 0.7690
Epoch 10/10
625/625 [==============================] - 64s 102ms/step - loss: 0.0781 - accuracy: 0.9729 - val_loss: 0.8508 -
val_accuracy: 0.7466
```

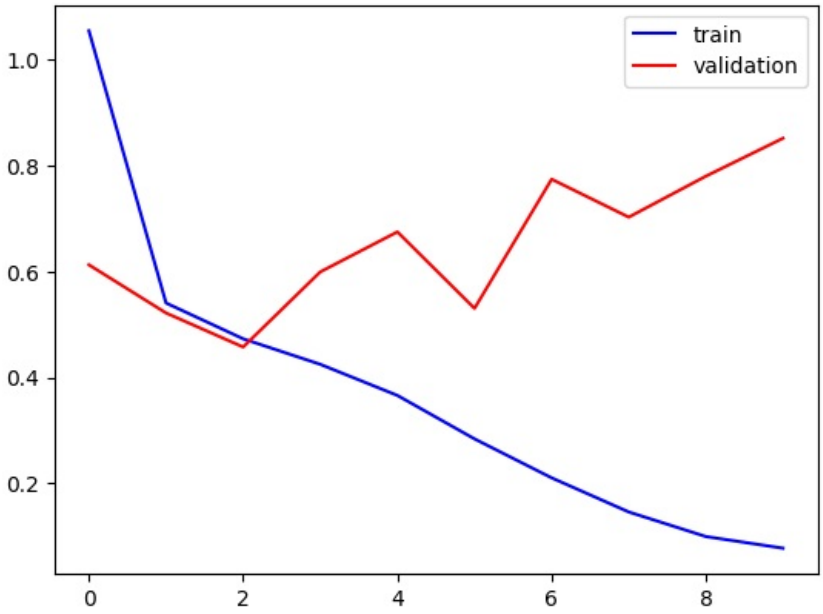
In [ ]:
```
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'],color='blue',label='train')
```

```
plt.plot(history.history['val_accuracy'],color='red',label='validation')
plt.legend()
plt.show()
```



In [ ]:
```
plt.plot(history.history['loss'],color='blue',label='train')
plt.plot(history.history['val_loss'],color='red',label='validation')
plt.legend()
plt.show()
```

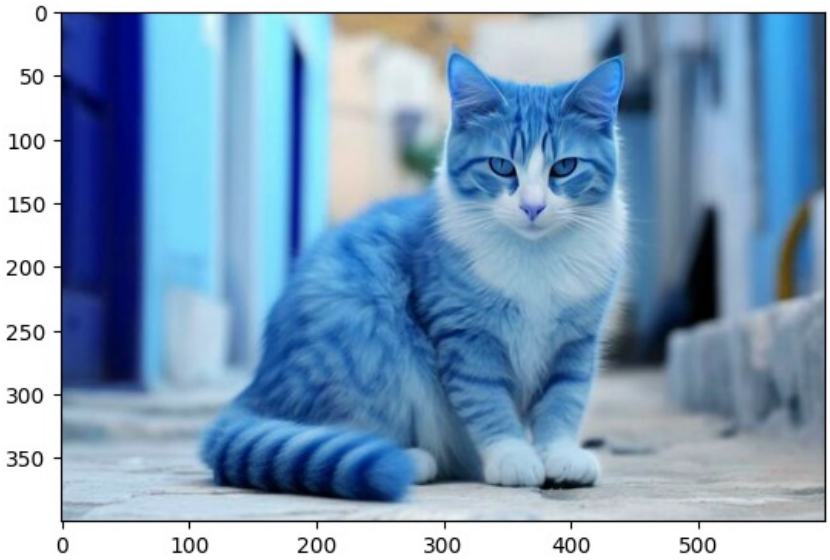


In [ ]:
```
import cv2
```

In [ ]:
```
test_img = cv2.imread('/content/cat.jpg')
```

In [ ]:
```
plt.imshow(test_img)
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7959f83b7550>
```



In [ ]:
```
test_img.shape
```

Out[ ]: (400, 600, 3)

In [ ]:
```
test_img = cv2.resize(test_img,(256,256))
```

In [ ]:
```
test_input = test_img.reshape((1,256,256,3))
```

In [ ]:
```
result = model.predict(test_input)
```

```
1/1 [==============================] - 0s 18ms/step
```

In [ ]:
```
if result == 0:
  print("This is the Cat Image")

else:
  print("This is the Dog Image")
```

```
This is the Cat Image
```