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

```
!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 'chmoc
Downloading dogs-vs-cats.zip to /content
 98% 1.05G/1.06G [00:08<00:00, 280MB/s]
100% 1.06G/1.06G [00:08<00:00, 139MB/s]
```

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

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

```
#generator : Divides the data into the batches at a time
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.
```

```
# 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)
```

```
# 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'))


model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 254, 254, 32)      896

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

 max_pooling2d_3 (MaxPoolin  (None, 127, 127, 32)      0
 g2D)

 conv2d_4 (Conv2D)           (None, 125, 125, 64)      18496

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

 max_pooling2d_4 (MaxPoolin  (None, 62, 62, 64)        0
 g2D)

 conv2d_5 (Conv2D)           (None, 60, 60, 128)       73856

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

 max_pooling2d_5 (MaxPoolin  (None, 30, 30, 128)       0
 g2D)

 flatten_1 (Flatten)         (None, 115200)            0

 dense_3 (Dense)             (None, 128)               14745728

 dropout (Dropout)           (None, 128)               0

 dense_4 (Dense)             (None, 64)                8256

 dropout_1 (Dropout)         (None, 64)                0

 dense_5 (Dense)             (None, 1)                 65

=================================================================
Total params: 14848193 (56.64 MB)
Trainable params: 14847745 (56.64 MB)
```

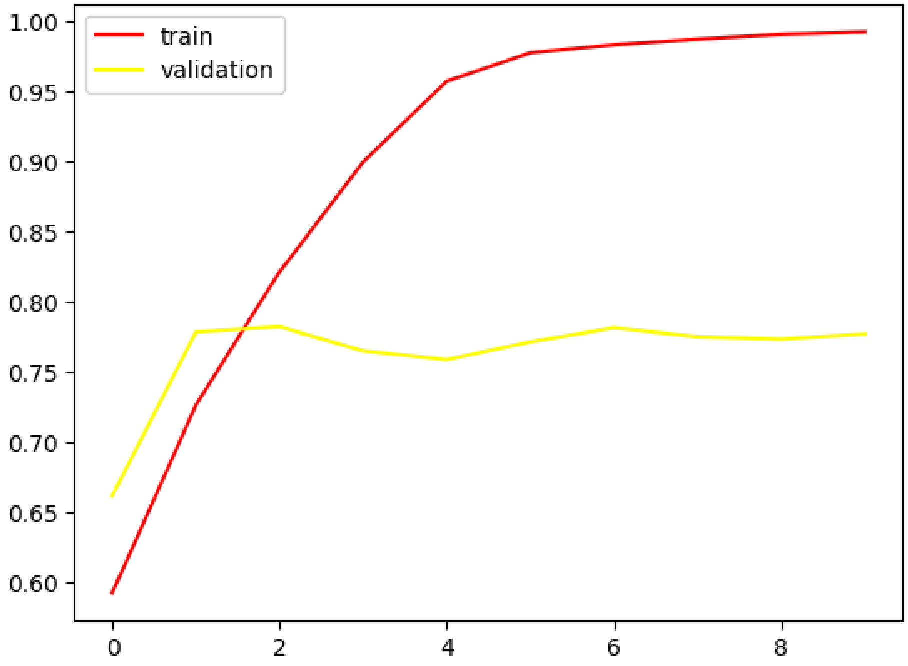
```
Non-trainable params: 448 (1.75 KB)
_________________________________________________________________
```

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

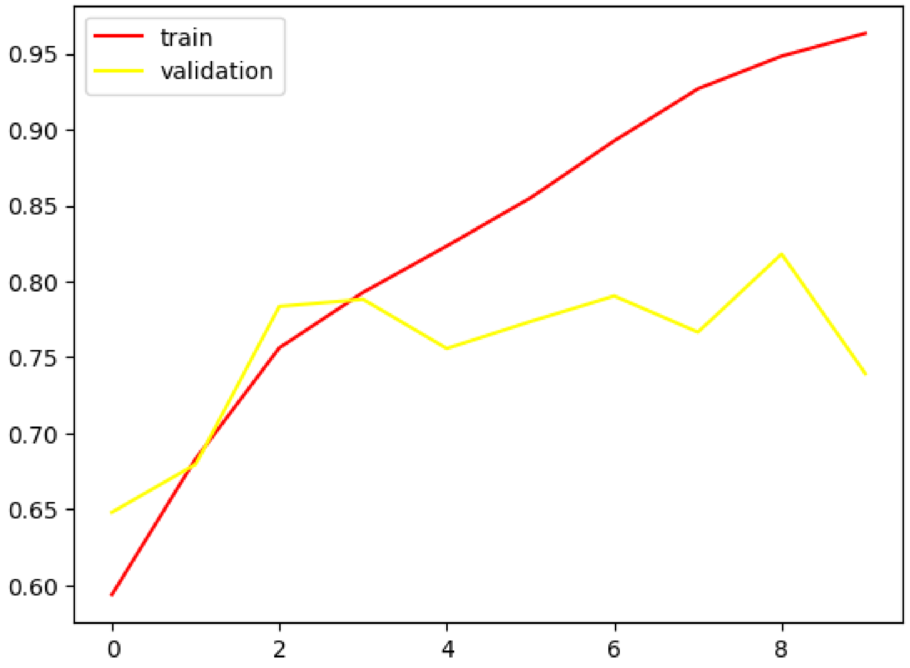
```
history = model.fit(train_ds, epochs=10, validation_data=validation_ds)
```

```
Epoch 1/10
625/625 [==============================] - 74s 110ms/step - loss: 1.3066 - accuracy: 0.5938 - val_loss:
Epoch 2/10
625/625 [==============================] - 65s 103ms/step - loss: 0.5973 - accuracy: 0.6832 - val_loss:
Epoch 3/10
625/625 [==============================] - 66s 105ms/step - loss: 0.4944 - accuracy: 0.7563 - val_loss:
Epoch 4/10
625/625 [==============================] - 66s 105ms/step - loss: 0.4597 - accuracy: 0.7928 - val_loss:
Epoch 5/10
625/625 [==============================] - 66s 105ms/step - loss: 0.3897 - accuracy: 0.8233 - val_loss:
Epoch 6/10
625/625 [==============================] - 78s 124ms/step - loss: 0.3317 - accuracy: 0.8550 - val_loss:
Epoch 7/10
625/625 [==============================] - 67s 107ms/step - loss: 0.2547 - accuracy: 0.8925 - val_loss:
Epoch 8/10
625/625 [==============================] - 65s 104ms/step - loss: 0.1857 - accuracy: 0.9268 - val_loss:
Epoch 9/10
625/625 [==============================] - 66s 105ms/step - loss: 0.1400 - accuracy: 0.9483 - val_loss:
Epoch 10/10
625/625 [==============================] - 65s 104ms/step - loss: 0.1010 - accuracy: 0.9633 - val_loss:
```

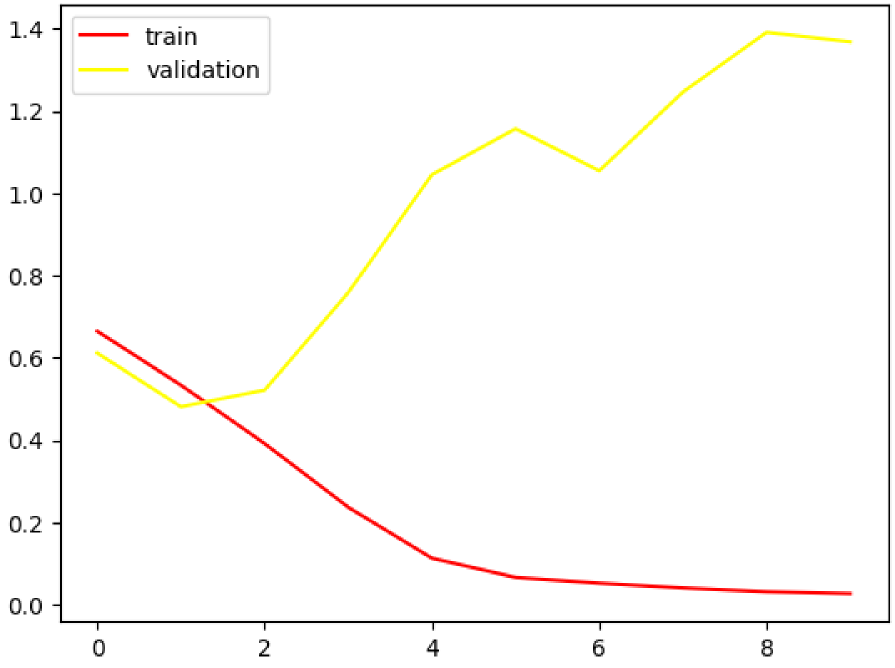
```
# Checking for training and validation accuracy
import matplotlib.pyplot as plt

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



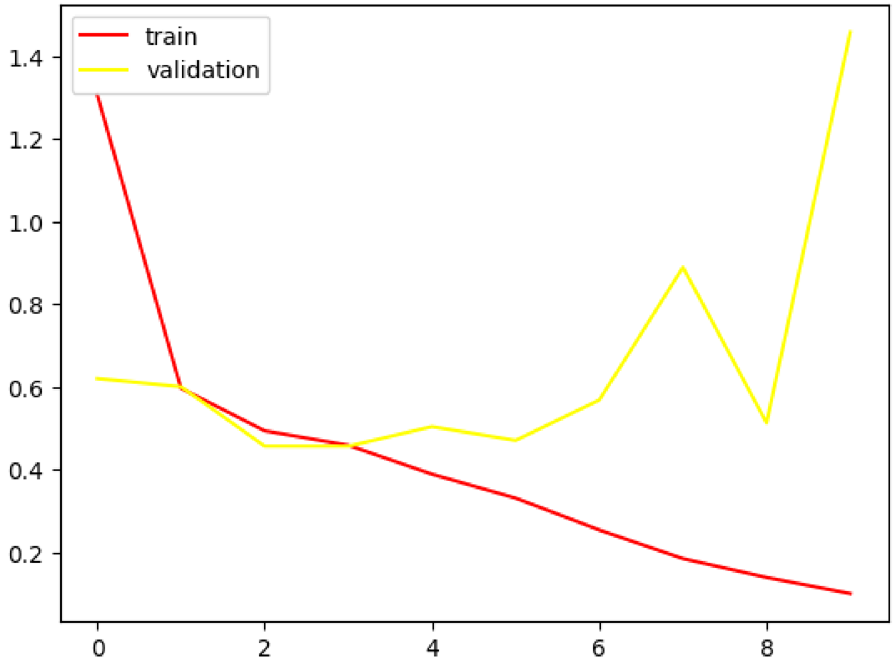
```
# After applying the batchNormalization
import matplotlib.pyplot as plt

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



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



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



```
# it is a clear indication of overfitting the model
```