Cat and Dog Image Classifier

Our Goal is to develop an image classification model to distinguish between images of cats and dogs using data science techniques in Python.

```
!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 'chmod 600 /root/.kaggle/kaggle.js
Downloading dogs-vs-cats.zip to /content
 98% 1.05G/1.06G [00:10<00:00, 160MB/s]
100% 1.06G/1.06G [00:10<00:00, 109MB/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
```

Dividing Data into small Batches

```
#generators -create batches
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.
```

Normalise The Data

```
#normalise
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)
```

```
#Creating a 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'))
```

Restructured CNN Models to increase Accuracy!!!

```
#Approach 1: Focus on activation functions and pooling:

model = Sequential()

model.add(Conv2D(32, kernel_size=(3, 3), padding='valid', activation='leaky_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='leaky_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='leaky_relu'))
model.add(BatchNormalization())
model.add(AveragePooling2D(pool_size=(2, 2), strides=2, padding='valid'))  # Change from MaxPooling to AveragePooling

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'))
```

```
#Approach 2: Add a residual connection and explore SE blocks (choose one SE block implementation):
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense

def se_block(x, channels):
  y = GlobalAveragePooling2D()(x)
  y = Dense(channels//2, activation='relu')(y)
  y = Dense(channels, activation='sigmoid')(y)
  return x * y

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'))

# Add residual connection
model.add(Conv2D(64, kernel_size=(3, 3), padding='valid', activation='relu'))
model.add(BatchNormalization())
# Add SE block with global average pooling
model.add(se_block(model.layers[-1].output, 64))
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"
_________________________________________________________________
 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)
_________________________________________________________________
```

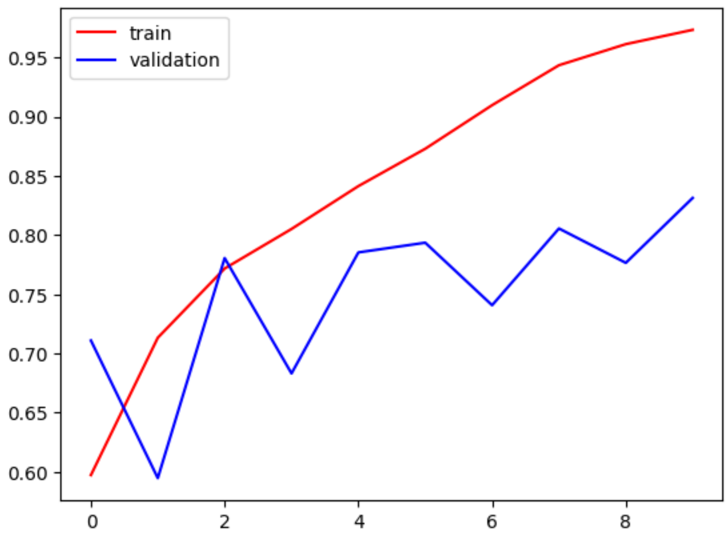
```
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

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

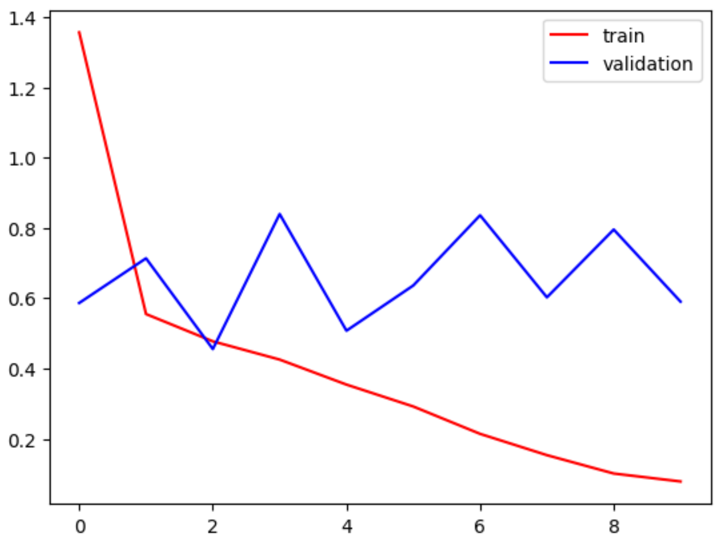
```
Epoch 1/10
625/625 [==============================] - 99s 132ms/step - loss: 1.3561 - accuracy: 0.5974 - val_loss: 0.5866 - val_accuracy: 0.71
Epoch 2/10
625/625 [==============================] - 84s 134ms/step - loss: 0.5551 - accuracy: 0.7133 - val_loss: 0.7136 - val_accuracy: 0.59
Epoch 3/10
625/625 [==============================] - 68s 108ms/step - loss: 0.4775 - accuracy: 0.7718 - val_loss: 0.4556 - val_accuracy: 0.78
Epoch 4/10
625/625 [==============================] - 69s 109ms/step - loss: 0.4256 - accuracy: 0.8052 - val_loss: 0.8395 - val_accuracy: 0.68
Epoch 5/10
625/625 [==============================] - 67s 106ms/step - loss: 0.3545 - accuracy: 0.8414 - val_loss: 0.5077 - val_accuracy: 0.78
Epoch 6/10
625/625 [==============================] - 67s 106ms/step - loss: 0.2922 - accuracy: 0.8730 - val_loss: 0.6364 - val_accuracy: 0.79
Epoch 7/10
625/625 [==============================] - 67s 106ms/step - loss: 0.2144 - accuracy: 0.9097 - val_loss: 0.8360 - val_accuracy: 0.74
Epoch 8/10
625/625 [==============================] - 67s 106ms/step - loss: 0.1538 - accuracy: 0.9435 - val_loss: 0.6026 - val_accuracy: 0.80
Epoch 9/10
625/625 [==============================] - 68s 109ms/step - loss: 0.1016 - accuracy: 0.9613 - val_loss: 0.7954 - val_accuracy: 0.77
Epoch 10/10
625/625 [==============================] - 66s 106ms/step - loss: 0.0789 - accuracy: 0.9734 - val_loss: 0.5904 - val_accuracy: 0.83
```

```
import matplotlib.pyplot as plt
```

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




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



Testing out with random image from the internet

```
import cv2
```

Cat Test Image

```
test_image = cv2.imread('/content/Cat_Test.jpg')
```

```
plt.imshow(test_image)
```

```
<matplotlib.image.AxesImage at 0x7f14dffbe650>
```


```
test_image.shape
```

```
(382, 480, 3)
```

```
test_image=cv2.resize(test_image,(256,256))
```

```
test_input = test_image.reshape(1,256,256,3)
```

```
model.predict(test_input)
```

```
1/1 [==============================] - 0s 272ms/step
array([[1.]], dtype=float32)
```

Dog Test Image

```
test_image = cv2.imread('/content/Dog_Test.jpg')
```

```
plt.imshow(test_image)
```

```
<matplotlib.image.AxesImage at 0x7f14b5b82770>
```