library

1. tensorflow: fast numerical computing
2. os: functions for interacting with the operating system
3. NumPy is a Python library used for working with arrays

In [19]:
```python
import tensorflow as tf
import os
import numpy as np
```

Database file path

In [21]:
```python
base_dir=r"C:\Users\Subrata Ghose\Desktop\Cat vs Dog\PetImages"
```

1. tf.keras.preprocessing.image.ImageDataGenerator:augment images in real-time while model is still training
2. Flip Horizontally command reverses the active layer horizontally.That is from left to right.
3. validation_split allows users to split their data into training and testing sets.

In [22]:
```python
IMAGE_SIZE=224
BATCH_SIZE=64

train_datagen=tf.keras.preprocessing.image.ImageDataGenerator(


    rescale=1./255,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.1)

validation_datagen=tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,

    validation_split=0.1
)
```

1. For training there are 902 images
2. For validation there are 100 images
3. classes

In [23]:
```python
train_genarator=train_datagen.flow_from_directory(
    base_dir,
    target_size=(IMAGE_SIZE,IMAGE_SIZE),
    batch_size=BATCH_SIZE,
    subset='training'


)

validation_generator=validation_datagen.flow_from_directory(
    base_dir,
    target_size=(IMAGE_SIZE,IMAGE_SIZE),
    batch_size=BATCH_SIZE,
    subset='validation'


)
```

```
Found 902 images belonging to 2 classes.
Found 100 images belonging to 2 classes.
```

library

1. Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor.
2. glob module is used to retrieve pathnames matching a specified pattern.
3. Flatten:This function converts the multi-dimensional arrays into flattened one-dimensional arrays or single-dimensional arrays.

In [24]:
```python
from tensorflow.keras.layers import Input,Flatten,Dense
from tensorflow.keras.models import Model
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.models import Sequential
from glob import glob
```

Imagenet:labeling and categorizing images

In [25]:
```python
IMAGE_SIZE=[224,224]
vgg=VGG16(input_shape=IMAGE_SIZE+[3],weights='imagenet',include_top=False)
vgg.output
```

Out[25]:
```
<KerasTensor: shape=(None, 7, 7, 512) dtype=float32 (created by layer 'block5_pool')>
```

In [26]:
```python
for layer in vgg.layers:
    layer.trainable=False
```

In [27]:
```python
folders=glob(r"C:\Users\Subrata Ghose\Desktop\Cat vs Dog\PetImages\*")
print(len(folders))
```

```
2
```

1. Dense Layer is used to classify image based on output from convolutional layers.
2. Softmax is a mathematical function that converts a vector of numbers into a vector of probabilities.

In [42]:
```python
x=Flatten()(vgg.output)
prediction=Dense(len(folders),activation='softmax')(x)
model=Model(inputs=vgg.input,outputs=prediction)
model.summary()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten_2 (Flatten)         (None, 25088)             0

 dense_2 (Dense)             (None, 2)                 50178

=================================================================
Total params: 14,764,866
Trainable params: 50,178
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [29]:
```python
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [30]:
```python
epoch=50

history=model.fit(train_genarator,
                  steps_per_epoch=len(train_genarator),
                  epochs=epoch,
                  validation_data=validation_generator,
                  validation_steps=len(validation_generator)
                 )
```

```
Epoch 1/50
15/15 [==============================] - 317s 22s/step - loss: 0.6415 - accuracy: 0.6896 - val_loss: 0.3175 - val_accuracy: 0.9200
Epoch 2/50
15/15 [==============================] - 333s 22s/step - loss: 0.3504 - accuracy: 0.8614 - val_loss: 0.1881 - val_accuracy: 0.9500
Epoch 3/50
15/15 [==============================] - 305s 21s/step - loss: 0.2710 - accuracy: 0.8969 - val_loss: 0.1664 - val_accuracy: 0.9700
Epoch 4/50
15/15 [==============================] - 334s 22s/step - loss: 0.2270 - accuracy: 0.9146 - val_loss: 0.1383 - val_accuracy: 0.9800
Epoch 5/50
15/15 [==============================] - 301s 20s/step - loss: 0.2207 - accuracy: 0.9213 - val_loss: 0.2088 - val_accuracy: 0.9000
Epoch 6/50
15/15 [==============================] - 295s 20s/step - loss: 0.1910 - accuracy: 0.9224 - val_loss: 0.1215 - val_accuracy: 0.9800
Epoch 7/50
15/15 [==============================] - 292s 21s/step - loss: 0.1911 - accuracy: 0.9346 - val_loss: 0.1155 - val_accuracy: 0.9600
Epoch 8/50
15/15 [==============================] - 291s 19s/step - loss: 0.1687 - accuracy: 0.9401 - val_loss: 0.1176 - val_accuracy: 0.9700
Epoch 9/50
15/15 [==============================] - 289s 19s/step - loss: 0.1506 - accuracy: 0.9523 - val_loss: 0.1147 - val_accuracy: 0.9600
Epoch 10/50
15/15 [==============================] - 295s 19s/step - loss: 0.1302 - accuracy: 0.9590 - val_loss: 0.1051 - val_accuracy: 0.9800
Epoch 11/50
15/15 [==============================] - 287s 20s/step - loss: 0.1209 - accuracy: 0.9601 - val_loss: 0.1213 - val_accuracy: 0.9500
Epoch 12/50
15/15 [==============================] - 285s 19s/step - loss: 0.1238 - accuracy: 0.9656 - val_loss: 0.1175 - val_accuracy: 0.9500
Epoch 13/50
15/15 [==============================] - 314s 21s/step - loss: 0.1197 - accuracy: 0.9667 - val_loss: 0.0902 - val_accuracy: 0.9900
Epoch 14/50
15/15 [==============================] - 328s 22s/step - loss: 0.1018 - accuracy: 0.9745 - val_loss: 0.0878 - val_accuracy: 0.9900
Epoch 15/50
15/15 [==============================] - 353s 23s/step - loss: 0.1005 - accuracy: 0.9778 - val_loss: 0.0993 - val_accuracy: 0.9700
Epoch 16/50
15/15 [==============================] - 320s 21s/step - loss: 0.0922 - accuracy: 0.9834 - val_loss: 0.0755 - val_accuracy: 0.9900
Epoch 17/50
15/15 [==============================] - 304s 21s/step - loss: 0.0928 - accuracy: 0.9745 - val_loss: 0.1221 - val_accuracy: 0.9400
Epoch 18/50
15/15 [==============================] - 316s 21s/step - loss: 0.1011 - accuracy: 0.9656 - val_loss: 0.0781 - val_accuracy: 0.9700
Epoch 19/50
15/15 [==============================] - 303s 20s/step - loss: 0.0821 - accuracy: 0.9856 - val_loss: 0.0923 - val_accuracy: 0.9700
Epoch 20/50
15/15 [==============================] - 330s 22s/step - loss: 0.0788 - accuracy: 0.9845 - val_loss: 0.0723 - val_accuracy: 0.9700
Epoch 21/50
15/15 [==============================] - 322s 22s/step - loss: 0.0692 - accuracy: 0.9856 - val_loss: 0.0631 - val_accuracy: 0.9900
Epoch 22/50
15/15 [==============================] - 308s 21s/step - loss: 0.0725 - accuracy: 0.9823 - val_loss: 0.0647 - val_accuracy: 0.9800
Epoch 23/50
15/15 [==============================] - 355s 24s/step - loss: 0.0751 - accuracy: 0.9834 - val_loss: 0.0938 - val_accuracy: 0.9600
Epoch 24/50
15/15 [==============================] - 329s 22s/step - loss: 0.0658 - accuracy: 0.9856 - val_loss: 0.0939 - val_accuracy: 0.9600
Epoch 25/50
15/15 [==============================] - 339s 23s/step - loss: 0.0682 - accuracy: 0.9867 - val_loss: 0.0609 - val_accuracy: 0.9800
Epoch 26/50
15/15 [==============================] - 226s 15s/step - loss: 0.0673 - accuracy: 0.9856 - val_loss: 0.0665 - val_accuracy: 0.9800
Epoch 27/50
15/15 [==============================] - 188s 13s/step - loss: 0.0650 - accuracy: 0.9856 - val_loss: 0.1044 - val_accuracy: 0.9600
Epoch 28/50
15/15 [==============================] - 209s 14s/step - loss: 0.0605 - accuracy: 0.9878 - val_loss: 0.1112 - val_accuracy: 0.9500
Epoch 29/50
15/15 [==============================] - 294s 20s/step - loss: 0.0579 - accuracy: 0.9900 - val_loss: 0.0647 - val_accuracy: 0.9700
Epoch 30/50
15/15 [==============================] - 284s 19s/step - loss: 0.0524 - accuracy: 0.9922 - val_loss: 0.0837 - val_accuracy: 0.9700
Epoch 31/50
15/15 [==============================] - 269s 18s/step - loss: 0.0465 - accuracy: 0.9967 - val_loss: 0.0613 - val_accuracy: 0.9800
Epoch 32/50
15/15 [==============================] - 267s 18s/step - loss: 0.0548 - accuracy: 0.9889 - val_loss: 0.0910 - val_accuracy: 0.9600
Epoch 33/50
15/15 [==============================] - 267s 18s/step - loss: 0.0500 - accuracy: 0.9933 - val_loss: 0.0506 - val_accuracy: 1.0000
Epoch 34/50
15/15 [==============================] - 269s 18s/step - loss: 0.0486 - accuracy: 0.9911 - val_loss: 0.0612 - val_accuracy: 0.9800
Epoch 35/50
15/15 [==============================] - 268s 18s/step - loss: 0.0461 - accuracy: 0.9945 - val_loss: 0.0676 - val_accuracy: 0.9800
Epoch 36/50
15/15 [==============================] - 268s 18s/step - loss: 0.0396 - accuracy: 0.9956 - val_loss: 0.0511 - val_accuracy: 1.0000
Epoch 37/50
15/15 [==============================] - 271s 18s/step - loss: 0.0435 - accuracy: 0.9945 - val_loss: 0.0756 - val_accuracy: 0.9600
Epoch 38/50
15/15 [==============================] - 270s 18s/step - loss: 0.0367 - accuracy: 0.9978 - val_loss: 0.0819 - val_accuracy: 0.9600
Epoch 39/50
15/15 [==============================] - 270s 18s/step - loss: 0.0393 - accuracy: 0.9933 - val_loss: 0.0706 - val_accuracy: 0.9700
Epoch 40/50
15/15 [==============================] - 271s 18s/step - loss: 0.0392 - accuracy: 0.9967 - val_loss: 0.0778 - val_accuracy: 0.9600
Epoch 41/50
15/15 [==============================] - 270s 18s/step - loss: 0.0318 - accuracy: 1.0000 - val_loss: 0.0721 - val_accuracy: 0.9700
Epoch 42/50
15/15 [==============================] - 269s 19s/step - loss: 0.0342 - accuracy: 0.9956 - val_loss: 0.0725 - val_accuracy: 0.9700
Epoch 43/50
15/15 [==============================] - 270s 18s/step - loss: 0.0358 - accuracy: 0.9945 - val_loss: 0.0607 - val_accuracy: 0.9700
Epoch 44/50
15/15 [==============================] - 271s 18s/step - loss: 0.0325 - accuracy: 0.9978 - val_loss: 0.0557 - val_accuracy: 0.9700
Epoch 45/50
15/15 [==============================] - 269s 18s/step - loss: 0.0285 - accuracy: 0.9989 - val_loss: 0.0426 - val_accuracy: 1.0000
Epoch 46/50
15/15 [==============================] - 268s 18s/step - loss: 0.0379 - accuracy: 0.9922 - val_loss: 0.0450 - val_accuracy: 0.9900
Epoch 47/50
15/15 [==============================] - 269s 18s/step - loss: 0.0314 - accuracy: 0.9989 - val_loss: 0.0432 - val_accuracy: 1.0000
Epoch 48/50
15/15 [==============================] - 271s 18s/step - loss: 0.0302 - accuracy: 0.9978 - val_loss: 0.0510 - val_accuracy: 0.9800
Epoch 49/50
15/15 [==============================] - 270s 18s/step - loss: 0.0295 - accuracy: 0.9989 - val_loss: 0.0510 - val_accuracy: 0.9900
Epoch 50/50
15/15 [==============================] - 267s 19s/step - loss: 0.0413 - accuracy: 0.9922 - val_loss: 0.0503 - val_accuracy: 0.9800
```

library

1. Converts a PIL Image instance to a Numpy array.
2. nInsert a new axis that will appear at the axis position in the expanded array shape

In [41]:
```python
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
import numpy as np

img_pred=image.load_img(r"C:\Users\Subrata Ghose\Desktop\Cat vs Dog\Dog_test\586.JPG",target_size=(224,224))

img_pred=image.img_to_array(img_pred)
img_pred=np.expand_dims(img_pred, axis=0)


rslt= model.predict(img_pred)

print(rslt)
if rslt[0][0]>rslt[0][1]:
    prediction="Cat"
else:
    prediction="Dog"
print(prediction)
```

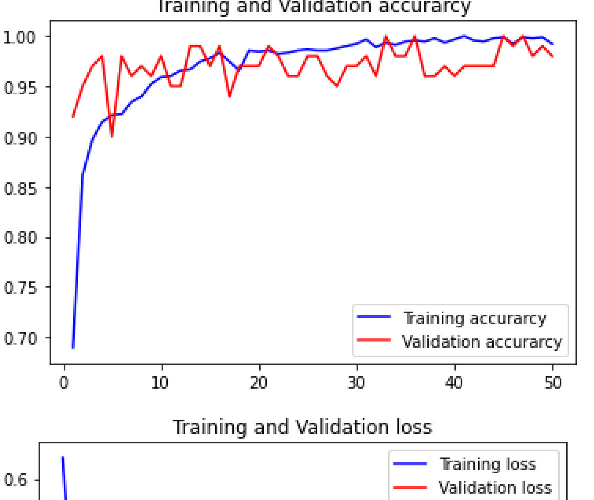
```
1/1 [==============================] - 0s 134ms/step
[[0. 1.]]
Dog
```

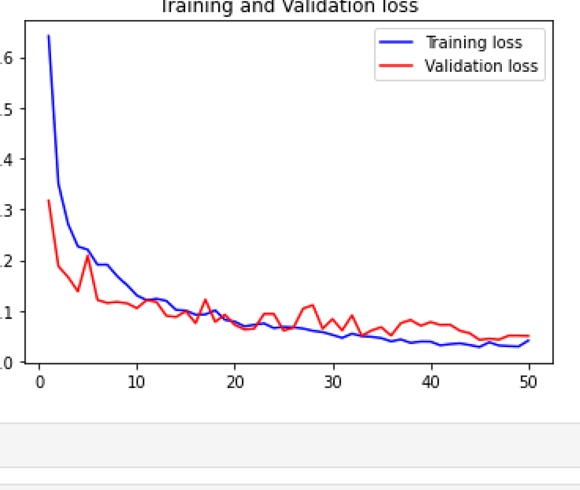
In [ ]:

In [ ]:

In [31]:
```python
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
#Train and validation accuracy
plt.plot(epochs, acc, 'b', label='Training accurarcy')
plt.plot(epochs, val_acc, 'r', label='Validation accurarcy')
plt.title('Training and Validation accurarcy')
plt.legend()

plt.figure()
#Train and validation loss
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()
plt.show()
```





In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: