Transfer learning on cat and dog image classification using VGG16

```
import tensorflow as tf
from tensorflow.keras import Sequential,Model
from tensorflow.keras.layers import Dense,Dropout,Flatten,BatchNormalization
from tensorflow.keras.applications.vgg16 import VGG16,preprocess_input
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
# Drive mounting
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
img_width=50
img_height=50
batch_size=500
```

```
# Initializing training dataset
train_data_dir="/content/drive/My Drive/train"
datagen = ImageDataGenerator(rescale=1./255)
train_generator = datagen.flow_from_directory(directory=train_data_dir,
                                              target_size = (img_width, img_height),
                                              classes=['cat','dog'],
                                              class_mode = 'binary',
                                              batch_size=batch_size)
```

```
Found 300 images belonging to 2 classes.
```

```
# Initializing validation dataset
validation_dir="/content/drive/My Drive/validation"
val_generator=datagen.flow_from_directory(validation_dir,target_size=(img_width,img_height),classes=['cat','dog'],batch_size
```

```
Found 40 images belonging to 2 classes.
```

```
# importing VGG16 model
vgg_arch=VGG16(input_shape=(img_width,img_height,3),weights="imagenet",include_top=False) #include_top= False represents tha
```

```
# This says that not to use dense values mentioned in VGG (Not to use VGG layers)
for layers in vgg_arch.layers:
  layers.trainable=False
```

```
# Training the model
model=Sequential()
model.add(vgg_arch)
model.add(Flatten())
model.add(Dense(128,activation='relu',))
model.add(Dropout(0.5))
model.add(BatchNormalization())
model.add(Dense(1,activation="sigmoid"))
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 1, 1, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               65664
```

```
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 128)               512       
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 129       
=================================================================
Total params: 14,780,993
Trainable params: 66,049
Non-trainable params: 14,714,944
_________________________________________________________________
```

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

```
history=model.fit_generator(generator=train_generator, steps_per_epoch=len(train_generator), epochs = 10,
                            validation_data=val_generator, validation_steps=len(val_generator)
                            , verbose = 1)
```

```
Epoch 1/10
1/1 [==============================] - 8s 8s/step - loss: 0.8413 - accuracy: 0.4733 - val_loss: 0.6797 - val_accuracy:
Epoch 2/10
1/1 [==============================] - 1s 1s/step - loss: 0.7687 - accuracy: 0.5233 - val_loss: 0.6572 - val_accuracy:
Epoch 3/10
1/1 [==============================] - 1s 1s/step - loss: 0.7080 - accuracy: 0.5733 - val_loss: 0.6340 - val_accuracy:
Epoch 4/10
1/1 [==============================] - 1s 1s/step - loss: 0.6931 - accuracy: 0.5967 - val_loss: 0.6149 - val_accuracy:
Epoch 5/10
1/1 [==============================] - 1s 1s/step - loss: 0.6180 - accuracy: 0.6600 - val_loss: 0.5996 - val_accuracy:
Epoch 6/10
1/1 [==============================] - 1s 1s/step - loss: 0.6092 - accuracy: 0.7033 - val_loss: 0.5872 - val_accuracy:
Epoch 7/10
1/1 [==============================] - 1s 1s/step - loss: 0.5350 - accuracy: 0.7367 - val_loss: 0.5764 - val_accuracy:
Epoch 8/10
1/1 [==============================] - 1s 1s/step - loss: 0.5980 - accuracy: 0.6767 - val_loss: 0.5669 - val_accuracy:
Epoch 9/10
1/1 [==============================] - 1s 1s/step - loss: 0.5360 - accuracy: 0.7567 - val_loss: 0.5585 - val_accuracy:
Epoch 10/10
1/1 [==============================] - 1s 1s/step - loss: 0.5101 - accuracy: 0.7367 - val_loss: 0.5510 - val_accuracy:
```

Predicting the model output

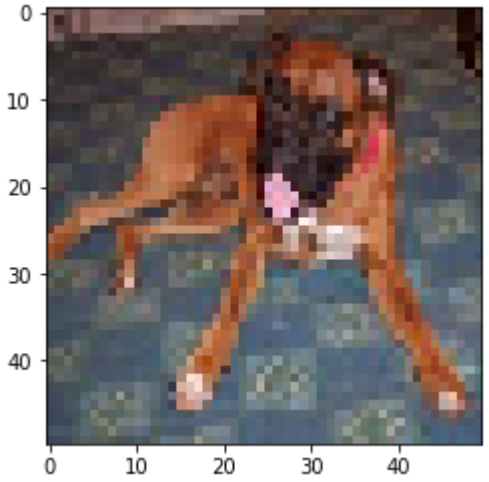
```
from tensorflow.keras.preprocessing import image
```

```
img=image.load_img("/content/dog.6990.jpg",target_size=(img_width,img_height))
plt.imshow(img)
img=image.img_to_array(img)
img=img/255.0
img = np.expand_dims(img, axis=0)
img_class = np.argmax(model.predict(img),axis=1)

if(model.predict(img)<=0.5):
  print('cat')

else:
  print('dog')
```

```
dog
```



```
img=image.load_img("/content/cat.2863.jpg",target_size=(img_width,img_height))
plt.imshow(img)
img=image.img_to_array(img)
img=img/255.0
img = np.expand_dims(img, axis=0)

if(model.predict(img)<=0.5):
  print('cat')

else:
  print('dog')
```

```
cat
```