Assignments 3

NUID : 03137910 Nelson Lefebvre

First of all let's see how does the VGG16 predict the output of a panda image and let's see how the VGG16 is composed.

In [5]:

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.layers import Activation, Conv2D, Input, BatchNormalization, Reshape, GlobalAveragePoolin
from tensorflow.keras.activations import sigmoid, softmax, relu, tanh
from tensorflow.keras import Sequential
import keras
import tensorflow as tf
tf.compat.v1.disable_eager_execution()
import vis
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import cv2
from keras.preprocessing import image
from keras import backend as K
from keras.applications.vgg16 import preprocess_input, decode_predictions,VGG16, preprocess_input
from vis.utils import utils
from tensorflow.keras.preprocessing.image import load_img

model = VGG16(weights='imagenet')
model.summary()
img = load_img('Panda.jpg',target_size=(224,224))
x = tf.keras.utils.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)
print('The most accurate possibility is :', tf.keras.applications.vgg16.decode_predictions(model.predict(x), to
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (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 (Flatten)           (None, 25088)             0         
                                                                 
 fc1 (Dense)                 (None, 4096)              102764544 
                                                                 
 fc2 (Dense)                 (None, 4096)              16781312  
                                                                 
 predictions (Dense)         (None, 1000)              4097000   
                                                                 
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
_________________________________________________________________
C:\ProgramData\Anaconda3\lib\site-packages\keras\engine\training_v1.py:2356: UserWarning: `Model.state_updates` will be removed in a future version. This property should not be used in TensorFlow 2.0, as `updates` are applied automatically.
  updates=self.state_updates,
The most accurate possibility is : [('n02510455', 'giant_panda', 0.9979412), ('n02447366', 'badger', 0.0011150375), ('n02133161', 'American_black_bear', 0.0006159243)]
```

So we will make a VGG16 "by hand". For that I have retrieved the layers of the VGG16 available with the associate library and I have added different SENet. (I would explain why there are several)

In [36]:
```
# Squeeze and Excitation
def se_block(input, channels, r=16):
    # Squeeze
    x = GlobalAveragePooling2D()(input)
    # Excitation
    x = Dense(channels//r, activation="relu")(x)
    x = Dense(channels, activation="sigmoid")(x)
    return tf.keras.layers.Multiply()([input, x])



class SENET_Attn(Layer):
    """
    Channel Attention Block as reported in SENET
    """
    def __init__(self,out_dim, ratio, layer_name="SENET"):
        super(SENET_Attn, self).__init__()
        self.out_dim = out_dim
        self.ratio = ratio
        self.layer_name = layer_name
```

```
    def build(self, ratio, layer_name="SENET"):
        self.Global_Average_Pooling = GlobalAveragePooling2D(keepdims= True)
        self.Fully_connected_1_1 = Dense(units= self.out_dim/self.ratio, name=self.layer_name+'_fully_connected
                                         kernel_initializer="glorot_uniform")
        self.Relu = ReLU()
        self.Fully_connected_2 = Dense(units=self.out_dim, name=layer_name+'_fully_connected2', activation = "t
        self.Sigmoid = Activation("sigmoid")

    def call(self, inputs):
        inputs = tf.cast(inputs, dtype = "float32")
        squeeze = self.Global_Average_Pooling(inputs)
        excitation = self.Fully_connected_1_1(squeeze)
        excitation = self.Relu(excitation)
        excitation = self.Fully_connected_2(excitation)
        excitation =  self.Sigmoid(excitation)
        excitation = tf.reshape(excitation, [-1,1,1,self.out_dim])

        scale = inputs * excitation
        return scale

Vgg = VGG16(weights='imagenet', include_top=True)

# SENET-RATIO
ratio = 16

input_layer = tf.keras.Input(shape=(224,224,3))
out = Vgg.layers[0](input_layer)
out = Vgg.layers[1](out) # Block 1 of VGG16
out = Vgg.layers[2](out)
out = se_block(out,64)
# out = SENET_Attn(out.shape[-1], ratio, )(out) # SENET Attention
#tf.keras.applications.mobilenet.preprocess_input(out)
out = Vgg.layers[3](out)
out = Vgg.layers[4](out) # Block 2 of VGG16
out = Vgg.layers[5](out)
out = se_block(out,128)
# out = SENET_Attn(out.shape[-1], ratio, )(out) # SENET Attention
#tf.keras.applications.mobilenet.preprocess_input(out)
out = Vgg.layers[6](out)
out = Vgg.layers[7](out) # Block 3 of VGG16
out = Vgg.layers[8](out)
out = Vgg.layers[9](out)
out = se_block(out,256)
# out = SENET_Attn(out.shape[-1], ratio, )(out) # SENET Attention
#tf.keras.applications.mobilenet.preprocess_input(out)
out = Vgg.layers[10](out)
out = Vgg.layers[11](out) # Block 4 of VGG16
out = Vgg.layers[12](out)
out = Vgg.layers[13](out)
out = se_block(out,512)
# out = SENET_Attn(out.shape[-1], ratio, )(out) # SENET Attention
#tf.keras.applications.mobilenet.preprocess_input(out)
out = Vgg.layers[14](out)
out = Vgg.layers[15](out) #Block 5 of VGG16
out = Vgg.layers[16](out)
out = Vgg.layers[17](out)
out = se_block(out,512)
# out = SENET_Attn(out.shape[-1], ratio, )(out) # SENET Attention
#tf.keras.applications.mobilenet.preprocess_input(out)
out = Vgg.layers[18](out)
out = Vgg.layers[19](out)
out = Vgg.layers[20](out)
out = Vgg.layers[21](out)
out = Vgg.layers[22](out)

model = tf.keras.Model(inputs=input_layer, outputs= out)
model.compile(optimizer = "adam", loss= keras.losses.categorical_crossentropy, metrics = ['accuracy'])
model.summary()

model.fit()
```

```
Model: "model_7"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_19 (InputLayer)          [(None, 224, 224, 3  0           []
                                )]

 input_18 (InputLayer)          multiple             0           ['input_19[0][0]']

 block1_conv1 (Conv2D)          (None, 224, 224, 64  1792        ['input_18[1][0]']
                                )

 block1_conv2 (Conv2D)          (None, 224, 224, 64  36928       ['block1_conv1[1][0]']
                                )

 global_average_pooling2d_36 (G  (None, 64)          0           ['block1_conv2[1][0]']
 lobalAveragePooling2D)
```

```
 dense_72 (Dense)               (None, 4)            260         ['global_average_pooling2d_36[0][
                                                                  0]']

 dense_73 (Dense)               (None, 64)           320         ['dense_72[0][0]']

 multiply_36 (Multiply)         (None, 224, 224, 64  0           ['block1_conv2[1][0]',
                                )                                 'dense_73[0][0]']

 block1_pool (MaxPooling2D)     (None, 112, 112, 64  0           ['multiply_36[0][0]']
                                )

 block2_conv1 (Conv2D)          (None, 112, 112, 12  73856       ['block1_pool[1][0]']
                                8)

 block2_conv2 (Conv2D)          (None, 112, 112, 12  147584      ['block2_conv1[1][0]']
                                8)

 global_average_pooling2d_37 (G  (None, 128)         0           ['block2_conv2[1][0]']
 lobalAveragePooling2D)

 dense_74 (Dense)               (None, 8)            1032        ['global_average_pooling2d_37[0][
                                                                  0]']

 dense_75 (Dense)               (None, 128)          1152        ['dense_74[0][0]']

 multiply_37 (Multiply)         (None, 112, 112, 12  0           ['block2_conv2[1][0]',
                                8)                                'dense_75[0][0]']

 block2_pool (MaxPooling2D)     (None, 56, 56, 128)  0           ['multiply_37[0][0]']

 block3_conv1 (Conv2D)          (None, 56, 56, 256)  295168      ['block2_pool[1][0]']

 block3_conv2 (Conv2D)          (None, 56, 56, 256)  590080      ['block3_conv1[1][0]']

 block3_conv3 (Conv2D)          (None, 56, 56, 256)  590080      ['block3_conv2[1][0]']

 global_average_pooling2d_38 (G  (None, 256)         0           ['block3_conv3[1][0]']
 lobalAveragePooling2D)

 dense_76 (Dense)               (None, 16)           4112        ['global_average_pooling2d_38[0][
                                                                  0]']

 dense_77 (Dense)               (None, 256)          4352        ['dense_76[0][0]']

 multiply_38 (Multiply)         (None, 56, 56, 256)  0           ['block3_conv3[1][0]',
                                                                  'dense_77[0][0]']

 block3_pool (MaxPooling2D)     (None, 28, 28, 256)  0           ['multiply_38[0][0]']

 block4_conv1 (Conv2D)          (None, 28, 28, 512)  1180160     ['block3_pool[1][0]']

 block4_conv2 (Conv2D)          (None, 28, 28, 512)  2359808     ['block4_conv1[1][0]']

 block4_conv3 (Conv2D)          (None, 28, 28, 512)  2359808     ['block4_conv2[1][0]']

 global_average_pooling2d_39 (G  (None, 512)         0           ['block4_conv3[1][0]']
 lobalAveragePooling2D)

 dense_78 (Dense)               (None, 32)           16416       ['global_average_pooling2d_39[0][
                                                                  0]']

 dense_79 (Dense)               (None, 512)          16896       ['dense_78[0][0]']

 multiply_39 (Multiply)         (None, 28, 28, 512)  0           ['block4_conv3[1][0]',
                                                                  'dense_79[0][0]']

 block4_pool (MaxPooling2D)     (None, 14, 14, 512)  0           ['multiply_39[0][0]']

 block5_conv1 (Conv2D)          (None, 14, 14, 512)  2359808     ['block4_pool[1][0]']

 block5_conv2 (Conv2D)          (None, 14, 14, 512)  2359808     ['block5_conv1[1][0]']

 block5_conv3 (Conv2D)          (None, 14, 14, 512)  2359808     ['block5_conv2[1][0]']

 global_average_pooling2d_40 (G  (None, 512)         0           ['block5_conv3[1][0]']
 lobalAveragePooling2D)

 dense_80 (Dense)               (None, 32)           16416       ['global_average_pooling2d_40[0][
                                                                  0]']

 dense_81 (Dense)               (None, 512)          16896       ['dense_80[0][0]']

 multiply_40 (Multiply)         (None, 14, 14, 512)  0           ['block5_conv3[1][0]',
                                                                  'dense_81[0][0]']

 block5_pool (MaxPooling2D)     (None, 7, 7, 512)    0           ['multiply_40[0][0]']

 flatten (Flatten)              (None, 25088)        0           ['block5_pool[1][0]']
```

```
 fc1 (Dense)                  (None, 4096)         102764544   ['flatten[1][0]']

 fc2 (Dense)                  (None, 4096)         16781312    ['fc1[1][0]']

 predictions (Dense)          (None, 1000)         4097000     ['fc2[1][0]']

==================================================================================================
Total params: 138,435,396
Trainable params: 138,435,396
Non-trainable params: 0
__________________________________________________________________________________________________
```

In [40]:

```python
img = load_img('Panda.jpg',target_size=(224,224))
x = tf.keras.utils.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)
print('The most accurate possibility is :', tf.keras.applications.vgg16.decode_predictions(model.predict(x), to
```

```
The most accurate possibility is : [('n04209239', 'shower_curtain', 0.028491486), ('n03271574', 'electric_fan', 0.018216325), ('n04590129', 'window_shade', 0.016928265)]
```

In [26]:

```python
img = load_img('Fish.jpg',target_size=(224,224))
x = tf.keras.utils.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)
print('The most accurate possibility is :', tf.keras.applications.vgg16.decode_predictions(model.predict(x), to
```

```
The most accurate possibility is : [('n03788365', 'mosquito_net', 0.021503624), ('n03495258', 'harp', 0.0212835
9), ('n04589890', 'window_screen', 0.020757237)]
```

In [27]:

```python
img = load_img('human.jpg',target_size=(224,224))
x = tf.keras.utils.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)
print('The most accurate possibility is :', tf.keras.applications.vgg16.decode_predictions(model.predict(x), to
```

```
The most accurate possibility is : [('n03788365', 'mosquito_net', 0.08594845), ('n04589890', 'window_screen', 0
.032408703), ('n04590129', 'window_shade', 0.024134178)]
```

In [28]:

```python
img = load_img('Magpie.jpg',target_size=(224,224))
x = tf.keras.utils.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)
print('The most accurate possibility is :', tf.keras.applications.vgg16.decode_predictions(model.predict(x), to
```

```
The most accurate possibility is : [('n04589890', 'window_screen', 0.07303518), ('n03788365', 'mosquito_net', 0
.028470566), ('n03271574', 'electric_fan', 0.015849922)]
```

How is it working ?

SENet (squeeze-and-excitation blocks) is a block who is here to perform dynamic channel-wise feature recalibration. And he is composed of to feature, one normal block which is composed of Gloobal pooling, FC, ReLu, FC, Sigmoid. The we concatenate the Residual block that have (H x W x C size) with the block (who at the end had the 1x1xC size). The goal of using SENet is to increase the rate of prediction but..

Results

As we can see i implement the SENet with 2 algorithm, both of them seems to work, nevertheless the prediction that i receive is completly inacurrate. I fix the VGG16 but when i add the SENet block the prediction is not accurate. I also tried the CBAM and the STN but without success (same problem, but i add the source is used for implementation).

Sources :

1 Follow Mr.Zhong advice) https://www.tensorflow.org/api_docs/python/tf/keras/applications/vgg16/preprocess_input 2 I asked on tensorflow for the accuracy problem, and i fix half of the problem) https://stackoverflow.com/questions/74255600/tensorflow-vgg16-senet-implementation-prediction-problem?noredirect=1#comment131098835_74255600 3 VGG16 prediction) https://www.youtube.com/watch?v=YEkriuvrtG0&ab_channel=SaptarsiGoswami 4 CBAM) https://youtu.be/vRYM1KdFtnk 5 SENet) https://youtu.be/MTiqzPdNkFM