VGG16 study notes

2018-07-26 |

2,486 | 11

summary

This article briefly introduces the classic deep learning model VGG16 in image classification tasks, analyzes its structure, and discusses its advantages and disadvantages. The existing VGG16 model in Keras is called to test its classification performance. The results show that VGG16 can correctly classify all three test images.

Preface

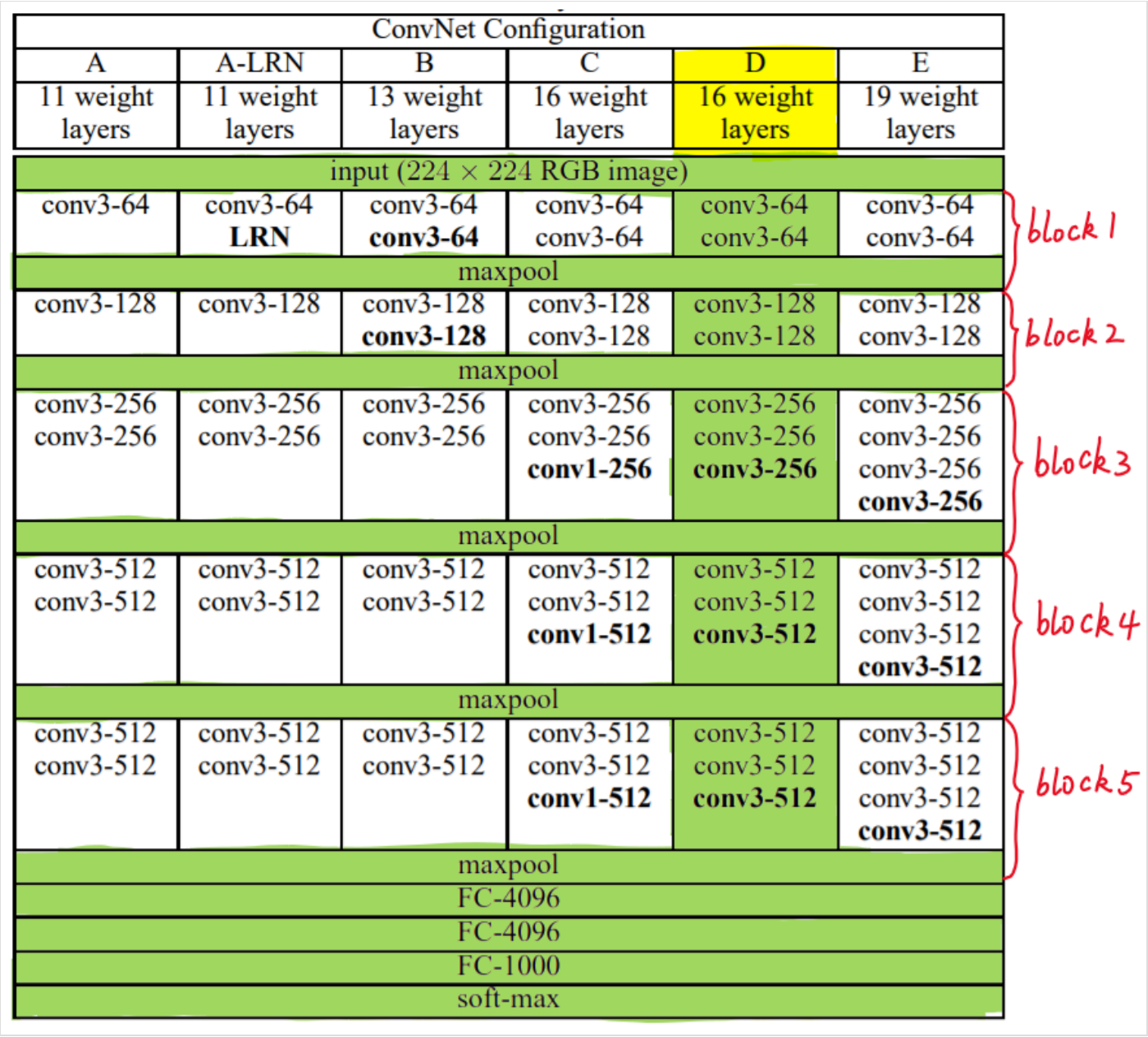
VGG is a convolutional neural network model proposed by Simonyan and Zisserman in the paper " *Very Deep Convolutional Networks for Large Scale Image Recognition* ". Its name comes from the abbreviation of the Visual Geometry Group at the University of Oxford where the authors are located.

This model participated in the 2014 ImageNet image classification and localization challenge and achieved excellent results: ranking second in the classification task and first in the localization task.

structure

In VGG, according to the **size of the convolution kernel** and the number **of convolution layers** , it can be divided `A` into 6 configurations (ConvNet Configuration), of which two configurations are more commonly used, called and respectively . `A-LRN B C D E D E VGG16 VGG19`

The following figure shows six structural configurations of VGG:



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 1 VGG model configuration

In the figure above, each column corresponds to an architectural configuration. For example, the green part in the figure indicates the architecture used by VGG16.

We conducted a specific analysis of VGG16 and found that `VGG16` it contains:

- 13 convolutional layers, represented by conv3–XXX
- 3 fully connected layers, represented by FC–XXXX
- 5 pooling layers, represented by maxpool

Among them, the convolutional layer and the fully connected layer have weight coefficients, so they are also called 权重层 , and the total number is 13+3=16, which is the source of 16 in VGG16. (The pooling layer does not involve weights, so it does not belong to the weight layer and is not counted).

Features

The outstanding feature of VGG16 is **its simplicity** , which is reflected in:

1. The convolution layers all use the same convolution kernel parameters

   The convolutional layers are represented as `conv3-XXX` , which `conv3` indicates that the kernel size used in the convolutional layer is 3, that is, the width and height are both 3, 3∗3 which is **a very small kernel size** . Combined with other parameters (stride=1, padding=same), each convolutional layer (tensor) can maintain the same width and height as the previous layer (tensor). `XXX` represents the number of channels of the convolutional layer.

2. The pooling layers all use the same pooling kernel parameters

   The parameters of the pooling layer are 2$\times$2, stride=2, and max pooling mode, so that the width and height of each pooling layer (tensor) are $\dfrac{1}{2}$ of the previous layer (tensor).

3. The model is composed of several stacked convolutional layers and pooling layers, which makes it easier to form a deeper network structure (in 2014, 16 layers were considered very deep).

Based on the above analysis, the advantages of VGG can be summarized as: **Small filters, Deeper networks**

The figure below shows the specific structure diagram of VGG16:

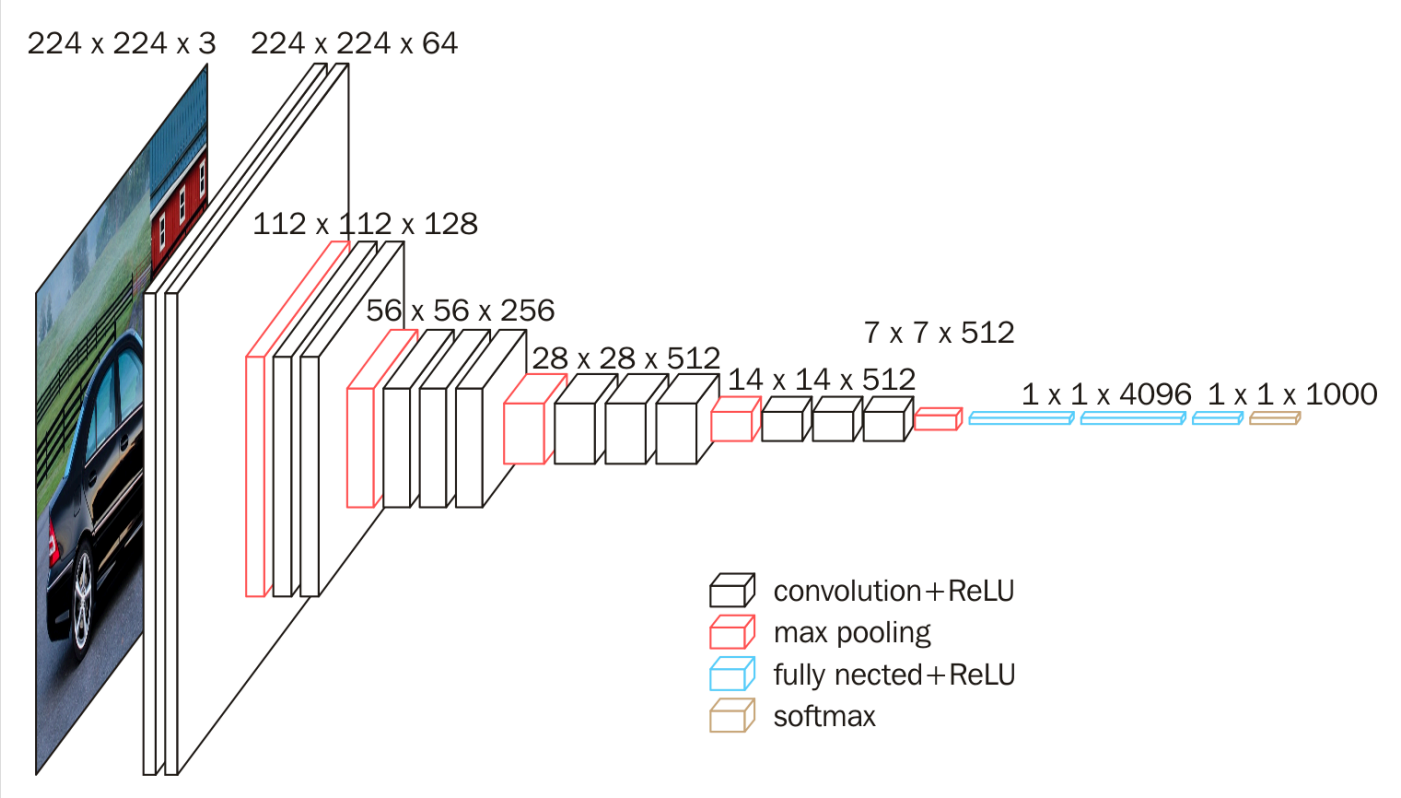


Figure 2 VGG16 structure diagram

Block structure

Note that on the right side of Figure 1, the convolutional layers and pooling layers of VGG16 can be divided into different blocks, numbered from front to back as Block1 to Block5. Each block contains **several convolutional layers** and **one pooling layer** . For example `Block4` :

- 3 convolutional layers, conv3–512
- 1 pooling layer, maxpool

And within the same block, **the number of channels** of the convolutional layer is the same, for example:

- `block2` There are 2 convolutional layers in it, each of which is `conv3–128` represented by , that is, the convolution kernel is: 3×3, and the number of channels is 128
- `block3` There are 3 convolutional layers in it, each of which is `conv3–512` represented by , that is, the convolution kernel is: $3\times 3 $, and the number of channels is 256

The following is a block–based structure diagram of VGG16, which can be understood in conjunction with Figure 2:

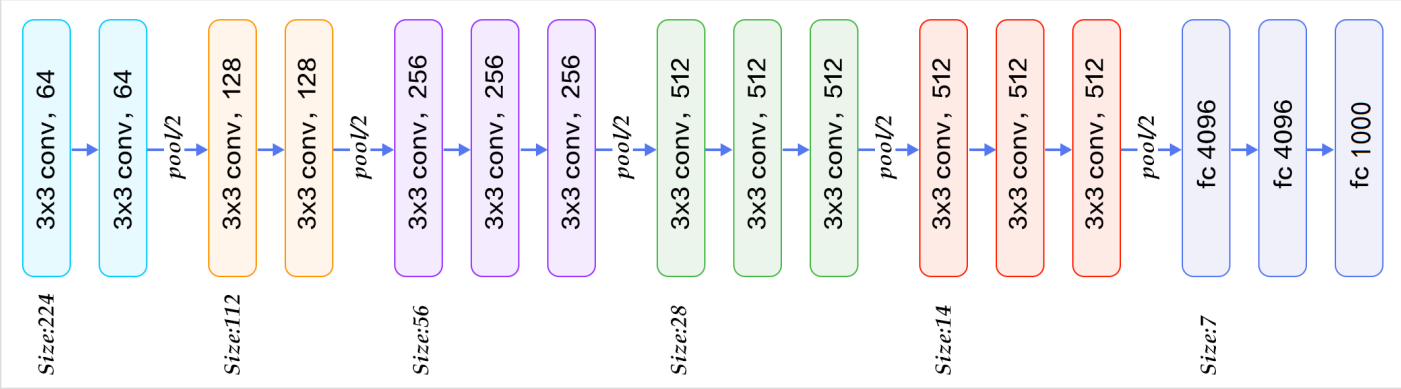


Figure 3 VGG16 structure divided by blocks

The input image of VGG is a 224$\times$224$\times$3 image tensor. As the number of layers increases, the tensor in the latter block is compared to the tensor in the previous block:

- The number of channels doubles from 64 to 128, then to 256, and finally to 512, remaining unchanged and no longer doubling.
- The height and width are halved, from 224$\rightarrow$ 112$\rightarrow$ 56$\rightarrow$28$\rightarrow$ 14$\rightarrow$ 7

Weight parameters

Although VGG has a simple structure, the number of weights it contains is very large, reaching an astonishing 138,357,544 parameters. These parameters include **convolution kernel weights** and **fully connected layer weights** .

- For example, for the first layer of convolution, since the number of channels of the input image is 3, the network must learn a convolution kernel of size 3$\times$3 and a channel number of 3. There are 64 such convolution kernels, so there are a total of (3$\times$3 $\times$ 3)$\times$ 64 = 1728 parameters.

- The method for calculating the number of weight parameters of the fully connected layer is:

  .

  因此，全连接层的参数分别为：

  - 7 $\times$ 7 $\times$ 512 $\times$ 4096 = 1027,645,444
  - 4096 $\times$ 4096 = 16,781,312
  - 4096 $\times$ 1000 = 4,097 000

FeiFei Li在CS231的课件中给出了整个网络的全部参数的计算过程（不考虑偏置），如下图所示：

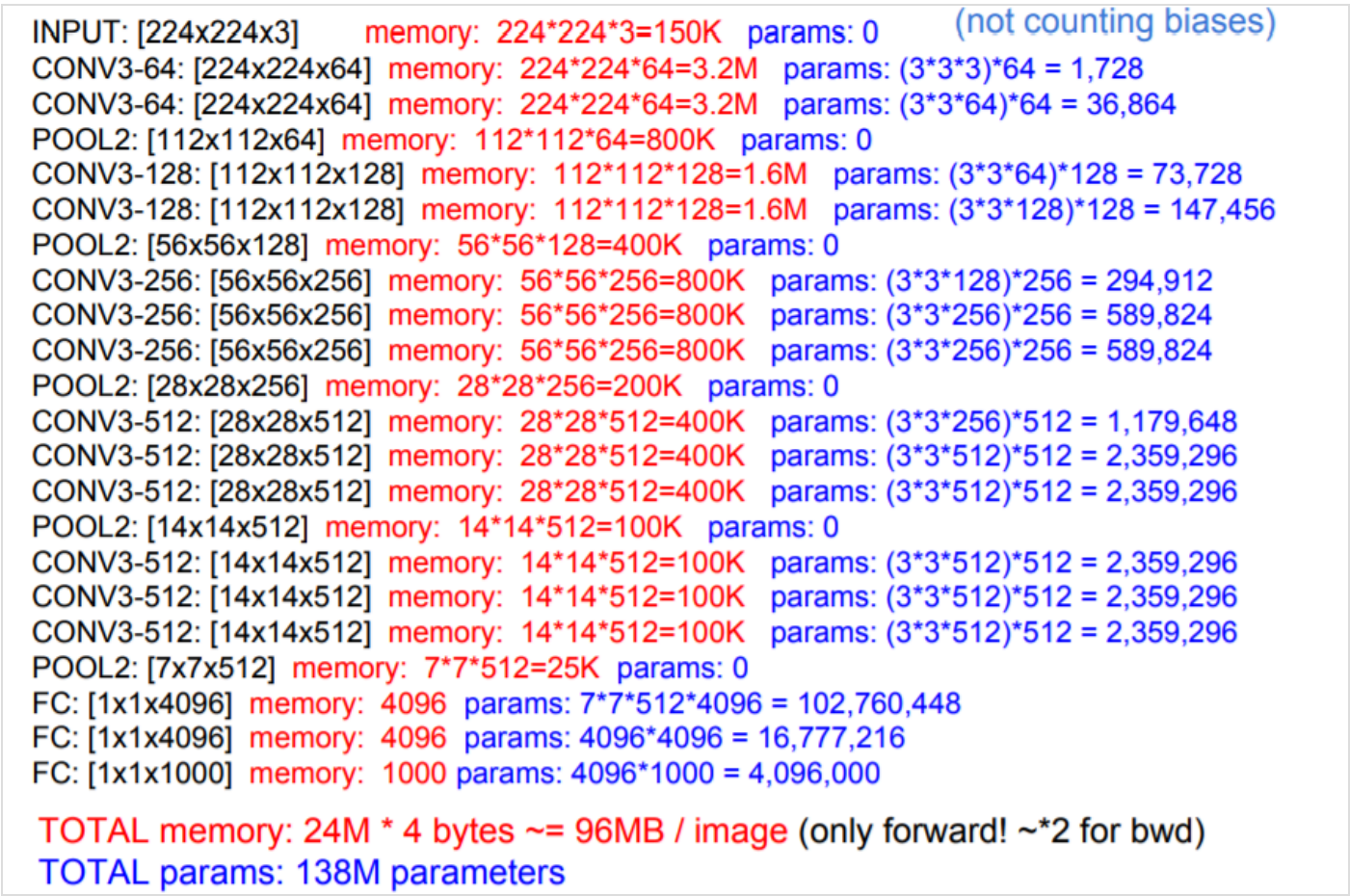


图4 训练参数计算

图中蓝色是计算权重参数数量的部分；红色是计算所需存储容量的部分。

VGG16具有如此之大的参数数目，可以预期它具有很高的拟合能力；但同时缺点也很明显：

- 即训练时间过长，调参难度大。
- 需要的存储容量大，不利于部署。例如存储VGG16权重值文件的大小为500多MB，不利于安装到嵌入式系统中。

实践

下面，我们应用Keras对VGG16的图像分类能力进行试验。

Keras是一个高层神经网络API,*Keras*由纯Python编写 ， 是tensorflow和Theano等底层深度学习库的高级封装 。使用Keras时，我们不需要直接调用底层API构建深度学习网络，仅调用keras已经封装好的函数即可。

本次试验平台：python 3.6 + tensorflow 1.8 + keras 2.2

源代码如下：

```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""
import matplotlib.pyplot as plt

from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input, decode_predictions
import numpy as np

def percent(value):
    return '%.2f%%' % (value * 100)

# include_top=True，表示會載入完整的 VGG16 模型，包括加在最後3層的卷積層
# include_top=False，表示會載入 VGG16 的模型，不包括加在最後3層的卷積層，通常是取得 Features
# 若下載失敗，請先刪除 c:\<使用者>\.keras\models\vgg16_weights_tf_dim_ordering_tf_kernels
model = VGG16(weights='imagenet', include_top=True)


# Input：要辨識的影像
img_path = 'frog.jpg'

#img_path = 'tiger.jpg' 并转化为224*224的标准尺寸
img = image.load_img(img_path, target_size=(224, 224))


x = image.img_to_array(img) #转化为浮点型
x = np.expand_dims(x, axis=0)#转化为张量size为(1, 224, 224, 3)
x = preprocess_input(x)

# 預測，取得features，維度為 (1,1000)
features = model.predict(x)

# 取得前五個最可能的類別及機率
pred=decode_predictions(features, top=5)[0]


#整理预测结果,value
values = []
bar_label = []
for element in pred:
```

```
45      values.append(element[2])
46      bar_label.append(element[1])
47
48  #绘图并保存
49  fig=plt.figure(u"Top-5 预测结果")
50  ax = fig.add_subplot(111)
51  ax.bar(range(len(values)), values, tick_label=bar_label, width=0.5, fc='g')
52  ax.set_ylabel(u'probability')
53  ax.set_title(u'Top-5')
54  for a,b in zip(range(len(values)), values):
55      ax.text(a, b+0.0005, percent(b), ha='center', va = 'bottom', fontsize=7)
56
57  fig = plt.gcf()
58  plt.show()
59
60  name=img_path[0:-4]+'_pred'
61  fig.savefig(name, dpi=200)
```

上述程序的基本流程是：

1. 载入相关模块，keras ， matplotlib，numpy
2. 下载**已经训练好的**模型文件：
3. 导入测试图像
4. 应用模型文件对图像分类

需要额外说明的是：

- 程序运行过程中，语句 `model = VGG16(weights='imagenet', include_top=True)` 会下载已经训练好的文件到 `c:\<使用者>\.keras\models` 文件夹下，模型的文件名为 `vgg16_weights_tf_dim_ordering_tf_kernels.h5` ,大小为527MB

- 语句 `pred=decode_predictions(features, top=5)[0]` 会下载分类信息文件到 `c:\<使用者>\.keras\models` 文件夹下，模型的文件名为 `imagenet_class_index.json` ,该文件指明了ImageNet大赛所用的1000个图像类的信息。(由于下载地址在aws上，需要科学上网，梯子请自备)

- 程序运行结束，会在工作目录下生成测试图片的预测图，给出了最有可能的前5个类列。名称为： 测试文件名`_pred.png`

- 在程序中还可以查看模型的结构，语句为： `model.summary()` ,命令行输出模型的结构配置为：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_16 (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
```

可以看到总的训练参数为138,357,544。

代码及图片文件全部放在我的github

结果

分别对 虎（tiger），贝果（bagel），树蛙（tree frog） 三张图片进行分类：

图5 测试图(虎)

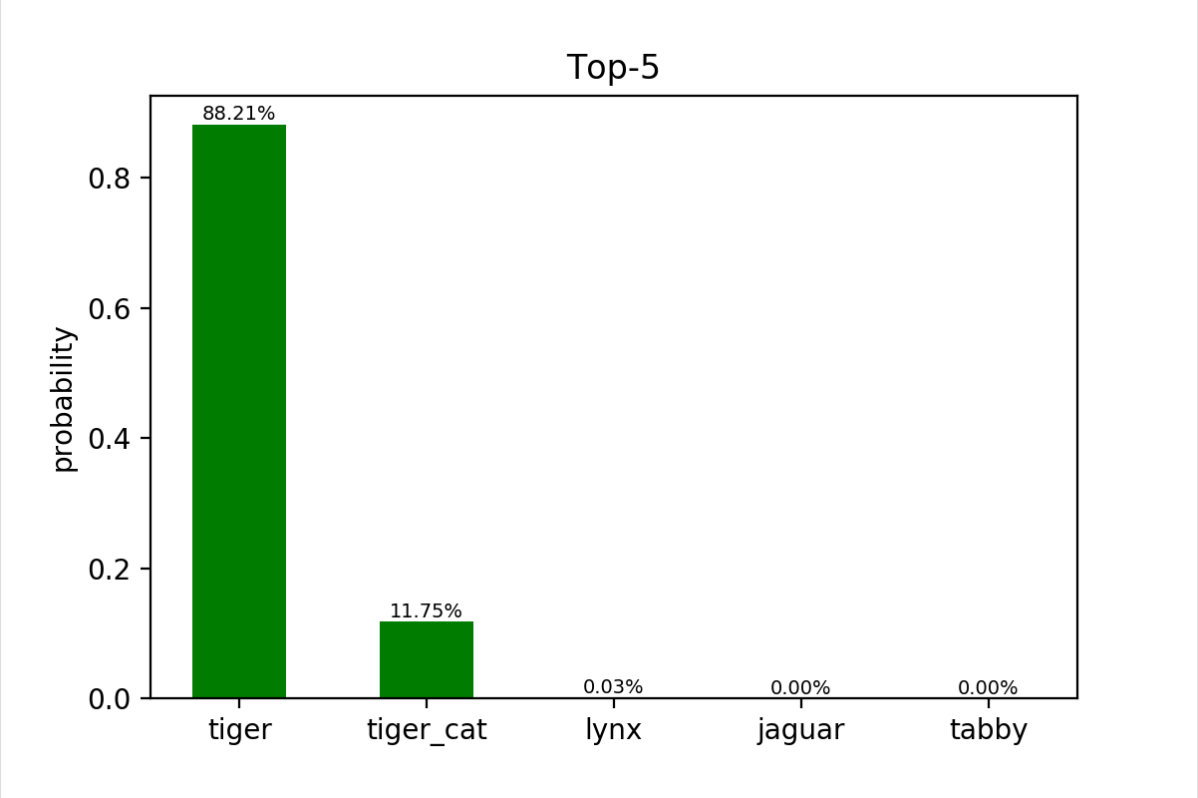


图6 预测分类(虎)

图7 测试图(树蛙)

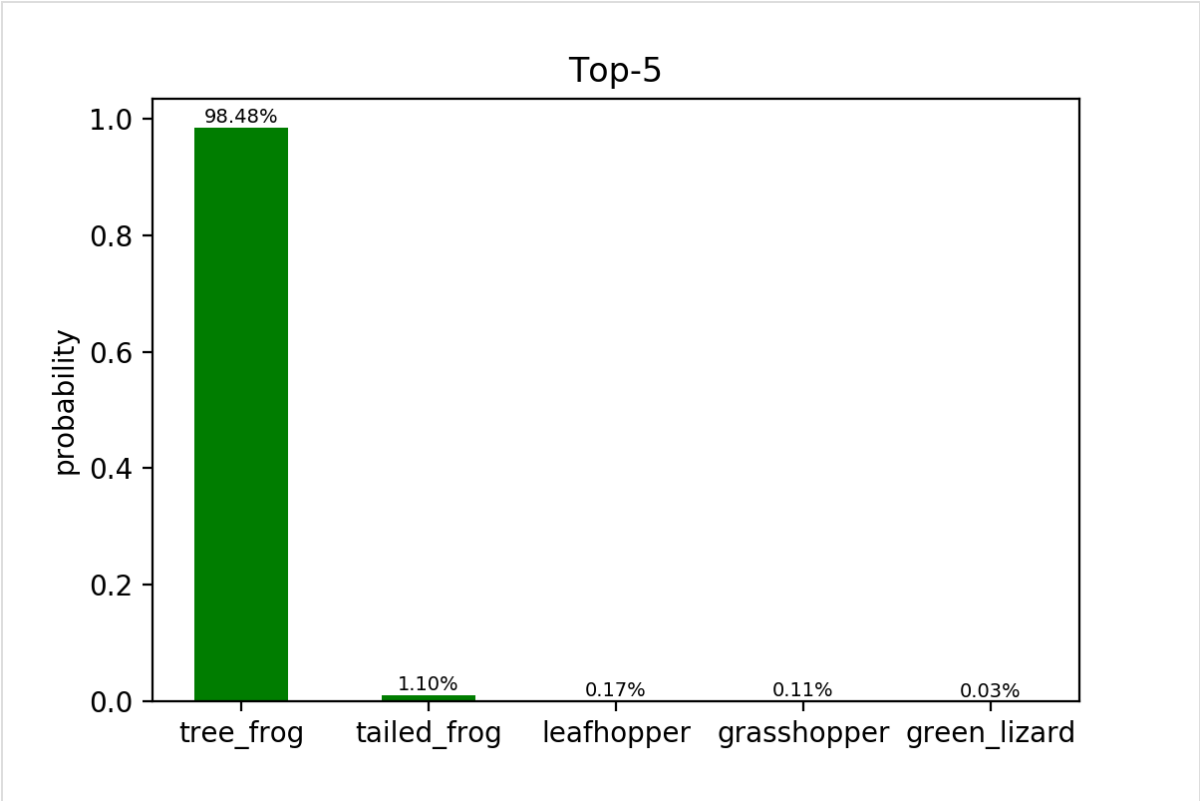


图8 预测分类(树蛙)

图9 测试图(贝果)

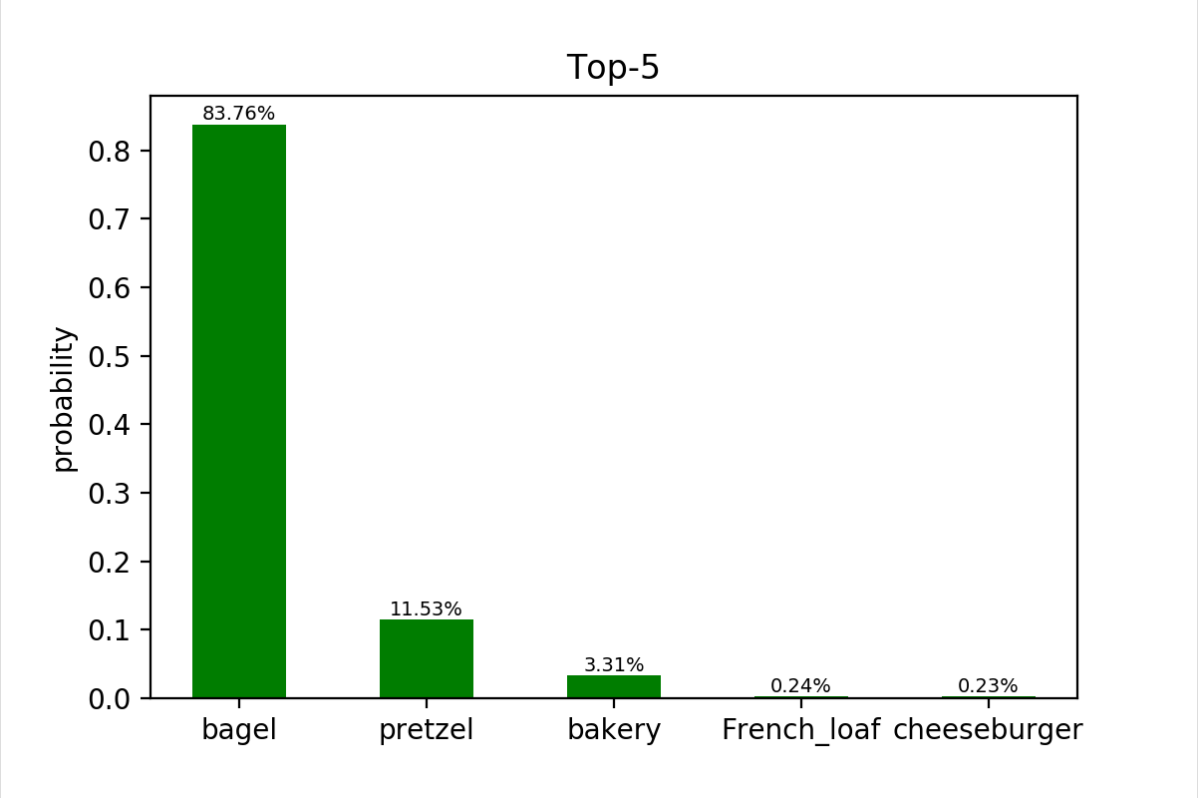


图10 预测分类(贝果)

可以看到，VGG16对这三幅图片均分类正确，且具有较高的准确率。

结论

This article introduces the structure of the VGG16 deep learning model, describes its structural characteristics, and analyzes its advantages and disadvantages. By calling the existing model in Keras to test three test pictures, the excellent performance of VGG16 in the image classification task is verified.

References

1. Deep Learning and TensorFlow: Reproducing the VGG Paper
2. Deep Learning with TensorFlow: Implementing Convolutional Neural Networks
3. ImageNet: VGGNet, ResNet, Inception, and Xception with Keras
4. VGG in TensorFlow
5. Day 09: Application of CNN classic model
6. Lecture 9: CNN Architectures

#VGG16 #keras

‹ Creating a virtual Python environment in Anaconda

Image segmentation application: object color highlighting ›

在 **DEANHAN** 上还有

梦缱绻

5 年前 · 2条评论

似为伊人念 醉梦续前缘 纵有凌云志 不忍负红颜

6 年前 · 12条评论

6 年前 · 2条评论

6 年前 · 2

1 0 回复

Wei Liu
5 年前
您好，您的参考文献1,2的链接访问地址好像不正确

0 0 回复

Veloc Fudarks
5 年前
666

0 0 回复

订阅 隐私 不要出售我的数据