# 经典神经网络 | VGGNet 论文解析及代码实现

潘庆发 223040410

**摘要**：深度卷积神经网络在图像识别方面已取得了重大突破。然而，现有的网络结构在深度和复杂度方面仍然有限制。这篇文章提出了一种非常深的卷积神经网络，即VGG网络，用于大规模图像识别任务。通过对网络结构进行系统性的分析和实验，发现在相对较小的卷积核（3x3）和较小的步长（1）下，可以构建更深的网络。在ImageNet数据集上的实验结果表明，VGG网络在Top-1和Top-5准确率方面都超过了先前的最佳结果。还展示了通过使用多尺度和多裁剪测试来进一步提高准确率的方法。

## 1 简介

1方法：  
 提出的VGG网络由16层或19层卷积层和全连接层组成。使用相对较小的卷积核（3x3）和较小的步长（1）来构建更深的网络。还使用ReLU非线性函数来代替Sigmoid函数，以加速训练过程。在训练过程中，使用了数据增强技术来扩展训练集，并使用Dropout技术来减少过拟合。在测试过程中，使用多尺度和多裁剪测试来进一步提高准确率。

2结果：  
 我们在ImageNet数据集上进行了实验，使用了不同的网络结构和测试方法。我们的VGG网络在Top-1和Top-5准确率方面都超过了先前的最佳结果。我们的16层VGG网络在Top-5准确率方面达到了92.0％，而19层VGG网络在Top-5准确率方面达到了92.7％。我们还发现，使用多尺度和多裁剪测试可以进一步提高准确率。我们的最佳结果在Top-5准确率方面达到了94.0％。

3结论：

我们提出了一种非常深的卷积神经网络，即VGG网络，用于大规模图像识别任务。我们的实验结果表明，在相对较小的卷积核和步长下，可以构建更深的网络，从而获得更好的识别性能。我们还展示了通过使用多尺度和多裁剪测试来进一步提高准确率的方法。我们的VGG网络在ImageNet数据集上取得了最佳结果。

贡献：

该论文的主要贡献在于，提出了一个非常深的卷积神经网络模型，该模型在ImageNet数据集上的表现超过了之前所有的模型，并且取得了当时最好的结果。论文还探讨了dropout对网络性能的影响，发现dropout能够提高网络的性能，特别是在网络非常深的情况下。  
 该论文的研究方法是，通过增加网络的深度来提高性能。作者使用了非常深的网络结构，包括19层和26层的网络，以及一个更深的网络结构，有152层。这些网络结构中包含了大量的卷积层和池化层，以及全连接层。  
 作者还使用了一些技巧来减少过拟合的影响，包括dropout和数据增强。dropout是一种在训练过程中随机删除一些神经元的技术，可以减少过拟合的风险。数据增强是一种通过对训练数据进行随机变换来扩增数据集的方法，可以提高模型的泛化能力。  
 在ImageNet数据集上的实验结果表明，该模型在Top-5准确率方面取得了新的最好成绩，达到了92.7%的准确率。这个结果比之前所有的模型都要好，证明了该模型的有效性。  
 总之，该论文提出了一个非常深的卷积神经网络模型，通过增加网络的深度来提高性能，并使用了一些技巧来减少过拟合的影响。该模型在ImageNet数据集上取得了非常好的表现，证明了深度卷积神经网络在图像识别领域的重要性和有效性。

## 2关键细节

关键：

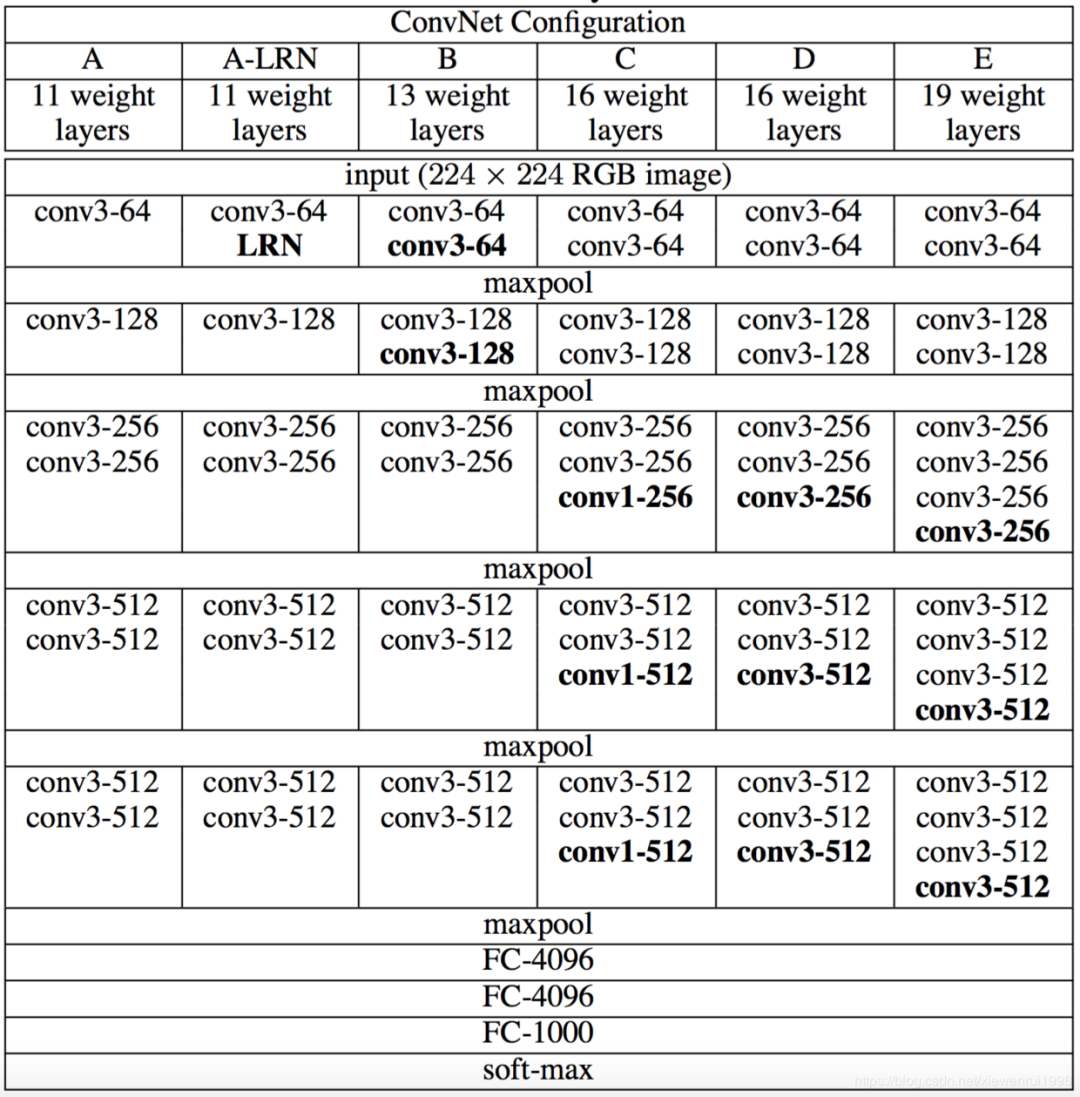


图 1 网络总览

VGGnet是一种经典的卷积神经网络结构，其主要特点是使用非常小的卷积核（3x3），多次堆叠卷积层和池化层，以达到更深的网络结构。VGGnet的网络结构分为5个卷积块和3个全连接层，具体结构如下：

（

在VGG网络中，A代表网络的一个版本，11表示该版本网络有11层，其中包括卷积层、池化层和全连接层等。11层中的权重层（Weight Layers）是指卷积层和全连接层中的参数，这些参数需要通过训练来学习，以使网络能够对输入进行准确的分类。这些权重层通常是由卷积核和偏置项组成的。在训练过程中，网络通过反向传播算法来更新这些权重层，以最小化损失函数，从而提高网络的准确性。

LRN是局部响应归一化（Local Response Normalization）的缩写，是一种用于提高深度神经网络性能的技术。它通常被用于卷积神经网络中的某些层，例如AlexNet、GoogLeNet和VGGNet等。

局部响应归一化是一种数据归一化的方法，它可以使每个神经元的输出对其相邻的神经元的输出产生抑制作用，以增强网络的泛化能力。具体来说，它会对每个神经元的输出进行归一化，使得每个神经元的输出与其相邻的神经元的输出的差异更加明显。这种差异可以使得网络更加鲁棒，从而提高其泛化能力。

在卷积神经网络中，局部响应归一化通常被用于卷积层之后，以增强网络的表征能力。它可以使得网络更加稳定，从而提高其性能。

conv3-64是卷积神经网络中的一种层，其中3表示卷积核的大小为3x3，64表示该层有64个滤波器。这意味着该层将输入数据的每个3x3区域与64个不同的滤波器进行卷积操作，生成64个输出特征图。这种层通常用于图像分类、目标检测等计算机视觉任务中。

在卷积神经网络中，滤波器也被称为卷积核或权重，是一组小型的矩阵，用于对输入数据进行卷积操作。滤波器的作用是提取输入数据中的特征信息，通过不同的滤波器组合，可以从输入数据中提取出不同的特征，例如边缘、纹理、颜色等。在卷积神经网络中，滤波器的权重是通过反向传播算法进行训练得到的，以最大化网络输出的正确率。

）

- 输入层：224x224x3的图像

- 卷积块1：两个卷积层（64个3x3的卷积核）、一个2x2的最大池化层

- 卷积块2：两个卷积层（128个3x3的卷积核）、一个2x2的最大池化层

- 卷积块3：三个卷积层（256个3x3的卷积核）、一个2x2的最大池化层

- 卷积块4：三个卷积层（512个3x3的卷积核）、一个2x2的最大池化层

- 卷积块5：三个卷积层（512个3x3的卷积核）、一个2x2的最大池化层

- 全连接层1：4096个神经元

- 全连接层2：4096个神经元

- 输出层：1000个神经元，对应1000个类别的概率输出

VGGnet的优点是结构简单、易于理解和实现，同时在多个计算机视觉任务中表现优异。缺点是网络结构比较深，训练和推理速度较慢，同时需要较大的存储空间。

卷积神经网络的输入是一个固定大小的224×224 RGB图像。做的唯一预处理是从每个像素中减去在训练集上计算的RGB平均值。图像通过一堆卷积(conv.)层传递，我们使用带有非常小的接受域的过滤器:3×3(这是捕捉左/右、上/下、中间概念的最小大小)。在其中一种配置中，还使用了1×1的卷积滤波器，它可以看作是输入通道的线性变换(其次是非线性)。卷积步幅固定为1像素;凹凸层输入的空间填充是卷积后保持空间分辨率，即3×3凹凸层的填充为1像素。空间池化由五个最大池化层执行，它们遵循一些对流层(不是所有对流层都遵循最大池化)。最大池是在一个2×2像素的窗口上执行的，步长为2。

  如上图所示，vggnet不单单的使用卷积层，而是组合成了“卷积组”，即一个卷积组包括2-4个3x3卷积层（a stack of 3x3 conv），有的层也有1x1卷积层，因此网络更深，网络使用2x2的max pooling，在full-image测试时候把最后的全连接层（fully-connected）改为全卷积层（fully-convolutional net），重用训练时的参数，使得测试得到的全卷积网络因为没有全连接的限制，因而可以接收任意宽或高为的输入，另外VGGNet卷积层有一个显著的特点：特征图的空间分辨率单调递减，特征图的通道数单调递增，这是为了更好地将HxWx3(1)的图像转换为1x1xC的输出，之后的GoogLeNet与Resnet都是如此。另外上图后面4个VGG训练时参数都是通过pre-trained 网络A进行初始赋值。

VGGNet由5个卷积层和3个全连接层构成。卷积层一般是3x3的卷积，结果表明比1x1卷积效果要好。

VGGNet三个全连接层为：

FC4096-ReLU6-Drop0.5，FC为高斯分布初始化（std=0.005），bias常数初始化（0.1）

FC4096-ReLU7-Drop0.5，FC为高斯分布初始化（std=0.005），bias常数初始化（0.1）

FC1000（最后接SoftMax1000分类），FC为高斯分布初始化（std=0.005），bias常数初始化（0.1）

（

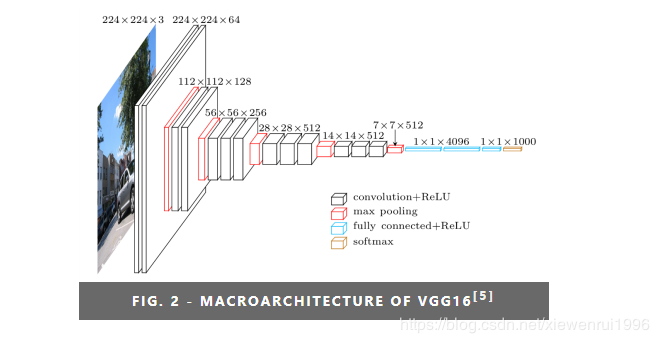
FC4096-ReLU6-Drop0.5是一个神经网络的结构表示，其中：

- FC4096表示一个全连接层，有4096个神经元。

- ReLU6表示使用ReLU激活函数，但是限制输出在0到6之间。

- Drop0.5表示在训练过程中，以50%的概率对该层的输出进行dropout操作，即随机将 一些神经元的输出设置为0，以减少过拟合的风险。

）



训练细节：

batch\_size=256，momentum=0.9，权值衰减使用L2正则化，L2=0.0005。第一、二层的全连接层droupout=0.5.lr初始化为0.01，后面改为0.001。我们从均值为0和方差为的正态分布中采样权重。偏置初始化为零。

  为了获得固定大小的224×224 ConvNet输入图像，它们从归一化的训练图像中被随机裁剪（每个图像每次SGD迭代进行一次裁剪）。为了进一步增强训练集，裁剪图像经过了随机水平翻转和随机RGB颜色偏移。S=256或384，对图片进行尺度抖动比增强数据集有助于捕获多尺度图像统计信息，提高检测正确率。

测试：

  最后三个全连接层首先被转换成卷积层（第一FC层转换到7×7卷积层，最后两个FC层转换到1×1卷积层）。然后将所得到的全卷积网络应用于整个（未裁剪）图像上。结果是类得分图的通道数等于类别的数量，以及取决于输入图像大小的可变空间分辨率。最后，为了获得图像的类别分数的固定大小的向量，类得分图在空间上平均（和池化）。我们还通过水平翻转图像来增强测试集；将原始图像和翻转图像的soft-max类后验进行平均，以获得图像的最终分数。

结论：

作者评估了用于大规模图像分类的非常深的卷积网络（多达19个权重层）。结果表明，随着深度的增加有利于分类精度的提高，并且在ImageNet挑战数据集上的最新性能可以使用传统的ConvNet架构实现。

Vgg19架构：

import keras

import numpy as np

from keras.datasets import cifar10

from keras.preprocessing.image import ImageDataGenerator

from keras.models import Sequential

from keras.layers import Dense, Dropout, Activation, Flatten

from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D, AveragePooling2D

from keras.initializers import he\_normal

from keras import optimizers

from keras.callbacks import LearningRateScheduler, TensorBoard

from keras.layers.normalization import BatchNormalization

from keras.utils.data\_utils import get\_file

num\_classes = 10

batch\_size = 128

epochs = 200

iterations = 391

dropout = 0.5

weight\_decay = 0.0001

log\_filepath = r'./vgg19\_retrain\_logs/'

from keras import backend as K

if('tensorflow' == K.backend()):

import tensorflow as tf

from keras.backend.tensorflow\_backend import set\_session

config = tf.ConfigProto()

config.gpu\_options.allow\_growth = True

sess = tf.Session(config=config)

def scheduler(epoch):

if epoch < 80:

return 0.1

if epoch < 160:

return 0.01

return 0.001

WEIGHTS\_PATH = 'https://github.com/fchollet/deep-learning-models/releases/download/v0.1/vgg19\_weights\_tf\_dim\_ordering\_tf\_kernels.h5'

filepath = get\_file('vgg19\_weights\_tf\_dim\_ordering\_tf\_kernels.h5', WEIGHTS\_PATH, cache\_subdir='models')

# data loading

(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()

y\_train = keras.utils.to\_categorical(y\_train, num\_classes)

y\_test = keras.utils.to\_categorical(y\_test, num\_classes)

x\_train = x\_train.astype('float32')

x\_test = x\_test.astype('float32')

# data preprocessing

x\_train[:,:,:,0] = (x\_train[:,:,:,0]-123.680)

x\_train[:,:,:,1] = (x\_train[:,:,:,1]-116.779)

x\_train[:,:,:,2] = (x\_train[:,:,:,2]-103.939)

x\_test[:,:,:,0] = (x\_test[:,:,:,0]-123.680)

x\_test[:,:,:,1] = (x\_test[:,:,:,1]-116.779)

x\_test[:,:,:,2] = (x\_test[:,:,:,2]-103.939)

# build model

model = Sequential()

# Block 1

model.add(Conv2D(64, (3, 3), padding='same', kernel\_regularizer=keras.regularizers.l2(weight\_decay), kernel\_initializer=he\_normal(), name='block1\_conv1', input\_shape=x\_train.shape[1:]))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(Conv2D(64, (3, 3), padding='same', kernel\_regularizer=keras.regularizers.l2(weight\_decay), kernel\_initializer=he\_normal(), name='block1\_conv2'))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(MaxPooling2D((2, 2), strides=(2, 2), name='block1\_pool'))

# Block 2

model.add(Conv2D(128, (3, 3), padding='same', kernel\_regularizer=keras.regularizers.l2(weight\_decay), kernel\_initializer=he\_normal(), name='block2\_conv1'))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(Conv2D(128, (3, 3), padding='same', kernel\_regularizer=keras.regularizers.l2(weight\_decay), kernel\_initializer=he\_normal(), name='block2\_conv2'))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(MaxPooling2D((2, 2), strides=(2, 2), name='block2\_pool'))

# Block 3

model.add(Conv2D(256, (3, 3), padding='same', kernel\_regularizer=keras.regularizers.l2(weight\_decay), kernel\_initializer=he\_normal(), name='block3\_conv1'))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(Conv2D(256, (3, 3), padding='same', kernel\_regularizer=keras.regularizers.l2(weight\_decay), kernel\_initializer=he\_normal(), name='block3\_conv2'))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(Conv2D(256, (3, 3), padding='same', kernel\_regularizer=keras.regularizers.l2(weight\_decay), kernel\_initializer=he\_normal(), name='block3\_conv3'))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(Conv2D(256, (3, 3), padding='same', kernel\_regularizer=keras.regularizers.l2(weight\_decay), kernel\_initializer=he\_normal(), name='block3\_conv4'))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(MaxPooling2D((2, 2), strides=(2, 2), name='block3\_pool'))

# Block 4

model.add(Conv2D(512, (3, 3), padding='same', kernel\_regularizer=keras.regularizers.l2(weight\_decay), kernel\_initializer=he\_normal(), name='block4\_conv1'))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(Conv2D(512, (3, 3), padding='same', kernel\_regularizer=keras.regularizers.l2(weight\_decay), kernel\_initializer=he\_normal(), name='block4\_conv2'))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(Conv2D(512, (3, 3), padding='same', kernel\_regularizer=keras.regularizers.l2(weight\_decay), kernel\_initializer=he\_normal(), name='block4\_conv3'))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(Conv2D(512, (3, 3), padding='same', kernel\_regularizer=keras.regularizers.l2(weight\_decay), kernel\_initializer=he\_normal(), name='block4\_conv4'))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(MaxPooling2D((2, 2), strides=(2, 2), name='block4\_pool'))

# Block 5

model.add(Conv2D(512, (3, 3), padding='same', kernel\_regularizer=keras.regularizers.l2(weight\_decay), kernel\_initializer=he\_normal(), name='block5\_conv1'))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(Conv2D(512, (3, 3), padding='same', kernel\_regularizer=keras.regularizers.l2(weight\_decay), kernel\_initializer=he\_normal(), name='block5\_conv2'))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(Conv2D(512, (3, 3), padding='same', kernel\_regularizer=keras.regularizers.l2(weight\_decay), kernel\_initializer=he\_normal(), name='block5\_conv3'))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(Conv2D(512, (3, 3), padding='same', kernel\_regularizer=keras.regularizers.l2(weight\_decay), kernel\_initializer=he\_normal(), name='block5\_conv4'))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(MaxPooling2D((2, 2), strides=(2, 2), name='block5\_pool'))

# model modification for cifar-10

model.add(Flatten(name='flatten'))

model.add(Dense(4096, use\_bias = True, kernel\_regularizer=keras.regularizers.l2(weight\_decay), kernel\_initializer=he\_normal(), name='fc\_cifa10'))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(Dropout(dropout))

model.add(Dense(4096, kernel\_regularizer=keras.regularizers.l2(weight\_decay), kernel\_initializer=he\_normal(), name='fc2'))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(Dropout(dropout))

model.add(Dense(10, kernel\_regularizer=keras.regularizers.l2(weight\_decay), kernel\_initializer=he\_normal(), name='predictions\_cifa10'))

model.add(BatchNormalization())

model.add(Activation('softmax'))

# load pretrained weight from VGG19 by name

model.load\_weights(filepath, by\_name=True)

# -------- optimizer setting -------- #

sgd = optimizers.SGD(lr=.1, momentum=0.9, nesterov=True)

model.compile(loss='categorical\_crossentropy', optimizer=sgd, metrics=['accuracy'])

tb\_cb = TensorBoard(log\_dir=log\_filepath, histogram\_freq=0)

change\_lr = LearningRateScheduler(scheduler)

cbks = [change\_lr,tb\_cb]

print('Using real-time data augmentation.')

datagen = ImageDataGenerator(horizontal\_flip=True,

width\_shift\_range=0.125,height\_shift\_range=0.125,fill\_mode='constant',cval=0.)

datagen.fit(x\_train) #将数据生成器用于某些示例数据。它基于一组样本数据，计算与数据转换相关的内部数据统计。

model.fit\_generator(datagen.flow(x\_train, y\_train,batch\_size=batch\_size), #生成增强后的数据

steps\_per\_epoch=iterations,

epochs=epochs,

callbacks=cbks,

validation\_data=(x\_test, y\_test))

model.save('retrain.h5')

Vgg19预测：

import keras

from keras.applications.vgg19 import VGG19

from keras.applications.vgg19 import preprocess\_input, decode\_predictions

from PIL import Image

import numpy as np

import os.path

# Using ImageNet pre\_trained weights to predict image's class(1000 class)

# ImageNet -- http://www.image-net.org/

# make sure your package pillow is the latest version

model = VGG19(weights='imagenet') # load keras ImageNet pre\_trained model

while True:

img\_path = input('Please input picture file to predict ( input Q to exit ): ')

if img\_path == 'Q':

break

if not os.path.exists(img\_path):

print("file not exist!")

continue

try:

img = Image.open(img\_path)

ori\_w,ori\_h = img.size

new\_w = 224.0;

new\_h = 224.0;

if ori\_w > ori\_h:

bs = 224.0 / ori\_h;

new\_w = ori\_w \* bs

weight = int(new\_w)

height = int(new\_h)

img = img.resize( (weight, height), Image.BILINEAR )

region = ( weight / 2 - 112, 0, weight / 2 + 112, height)

img = img.crop( region )

else:

bs = 224.0 / ori\_w;

new\_h = ori\_h \* bs

weight = int(new\_w)

height = int(new\_h)

img = img.resize( (weight, height), Image.BILINEAR )

region = ( 0, height / 2 - 112 , weight, height / 2 + 112 )

img = img.crop( region )

x = np.asarray( img, dtype = 'float32' )

x[:, :, 0] = x[:, :, 0] - 123.680

x[:, :, 1] = x[:, :, 1] - 116.779

x[:, :, 2] = x[:, :, 2] - 103.939

x = np.expand\_dims(x, axis=0)

results = model.predict(x)

print('Predicted:', decode\_predictions(results, top=5)[0])

except Exception as e:

Pass