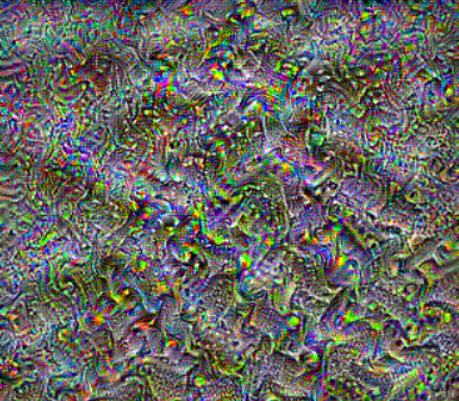
实验结果：

1. 噪音图像



训练结果



1. 风景图



训练结果：



源代码：

from \_\_future\_\_ import print\_function

from io import BytesIO

import numpy as np

from functools import partial

import PIL.Image

import scipy.misc

import tensorflow as tf

tf.compat.v1.disable\_eager\_execution()

import os

os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '2'#CPU 支持AVX2 FMA（加速CPU计算）

import tensorboard as tb

tf.compat.v1.io.gfile = tb.compat.tensorflow\_stub.io.gfile

class Deepdream:

    def \_\_init\_\_(self):

        self.name = "mixed4c"

        self.model\_fn = r'D:\\python\\DeepLearning\\imges\\photo.pb'

        self.img0 = PIL.Image.open("D:\\python\\DeepLearning\\imges\\photo.jpg")

        self.img0 = np.float32(self.img0)

    #导入Inception模型

    def Inception(self):

        #创建图和会话

        self.graph = tf.compat.v1.Graph()

        self.sess = tf.compat.v1.InteractiveSession(graph=self.graph)

        #导入Inception网络

        with tf.compat.v1.io.gfile.FastGFile(self.model\_fn,"rb") as f:

            graph\_def = tf.compat.v1.GraphDef()

            graph\_def.ParseFromString(f.read())

        #定义输入图像的占位符

        self.t\_input = tf.compat.v1.placeholder(np.float32,name="input")

        #图像的预处理--减均值

        imagenet\_mean = 117.0

        #图像的预处理—增加维度

        t\_preprocessed = tf.expand\_dims(self.t\_input-imagenet\_mean,0)

        #导入模型并将经处理的图像送入网络中

        tf.compat.v1.import\_graph\_def(graph\_def,{"input":t\_preprocessed})

        self.layer\_output = self.graph.get\_tensor\_by\_name("import/%s:0" %self.name)

    #图像的拉普拉斯金字塔分解

    def render\_deepdream(self,t\_obj,img0,iter\_n=10,step=1.5,octave\_n=4,octave\_scale=1.4):

        t\_score = tf.compat.v1.reduce\_mean(t\_obj)

        t\_grad = tf.compat.v1.gradients(t\_score,self.t\_input)[0]

        img = img0.copy()

        #将图像进行金字塔分解

        #从而分为高频、低频部分

        octaves = []

        for i in range(octave\_n-1):

            hw = img.shape[:2]

            lo = self.resize(img,np.int32(np.float32(hw)/octave\_scale))

            hi = img - self.resize(lo,hw)

            img = lo

            octaves.append(hi)

        #首先生成低频的图像，再依次放大并加上高频

        for octave in range(octave\_n):

                if octave > 0:

                    hi = octaves[-octave]

                    img = self.resize(img,hi.shape[:2]) + hi

                for i in range(iter\_n):

                    g = self.calc\_grad\_tiled(img,t\_grad)

                    img += g\*(step/(np.abs(g).mean()+1e-7))

        img = img.clip(0,255)

        self.savearray(img,r'D:\\python\\DeepLearning\\imges\\photo1.jpg')

        im = PIL.Image.open(r'D:\\python\\DeepLearning\\imges\\photo1.jpg').show()  #保存并显示图片

    def savearray(self,img\_arry,img\_name):

        scipy.misc.toimage(img\_arry).save(img\_name)

        print("img saved:%s" %img\_name)

    #生成更大尺寸的图像

    #原始图像尺寸可能很大，从而导致内存耗尽问题

    def calc\_grad\_tiled(self,img,t\_grad,tile\_size=512):

        sz = tile\_size

        h,w = img.shape[:2]

        sx,sy = np.random.randint(sz,size=2)

        #先在行作整体运动，再在列作整体运动

        img\_shift = np.roll(np.roll(img,sx,1),sy,0)

        grad = np.zeros\_like(img)

        for y in range(0,max(h-sz//2,sz),sz):

            for x in range(0,max(w-sz//2,sz),sz):

                sub = img\_shift[y:y+sz,x:x+sz]

                g = self.sess.run(t\_grad,{self.t\_input:sub})

                grad[y:y+sz,x:x+sz] = g

        return np.roll(np.roll(grad,-sx,1),-sy,0)

    #将图像放大ratio倍

    def resize\_ratio(self,img,ratio):

        min = img.min()

        max = img.max()

        img = (img - min)/(max - min)\*255

        img = np.float32(scipy.misc.imresize(img,ratio))

        img = img/255\*(max-min)+min

        return img

    #调整图像尺寸

    def resize(self,img,hw):

        min = img.min()

        max = img.max()

        img = (img - min)/(max - min)\*255

        img = np.float32(scipy.misc.imresize(img,hw))

        #img = np.float32(np.array(PIL.Image.fromarray(img,mode="RGB").resize(hw)))

        img = img/255\*(max-min) + min

        return img

if \_\_name\_\_ == "\_\_main\_\_":

    D = Deepdream() #生成Deepdream对象

    D.Inception()  #导入Inception模型

    D.render\_deepdream(tf.square(D.layer\_output),D.img0)