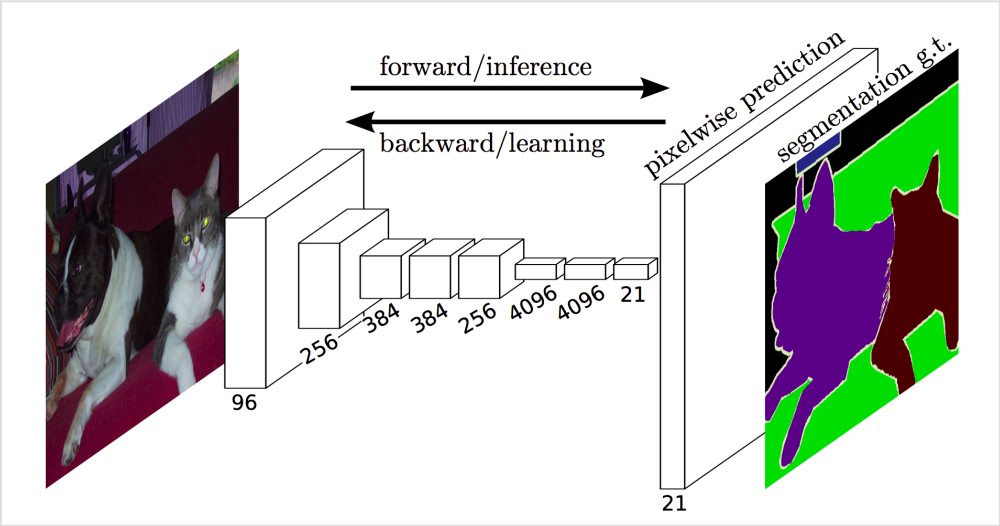
# FCN 图像语义分割实践

2017-06-23 | 181

FCN 是一种有效的对图像语义分割的方法, 方法十分简单, 分割效果非常好. 本文记录使用 FCN 做语义分割的过程. 本文记录的方法和 FCN 原始论文并不完全相同, 但是, 基本思想是一致的.

## 快速解释

图像语义分割的目的就是对于图像中每一个像素点, 给分配一个标签, 表明该像素点所属的目标. 例如下图中, 该图像经过语义分割之后, 猫的像素点和狗的像素点分别用不同的颜色表示, 表明模型完成了对该图片的语义分割.
分割是比检测跟进一步的任务, 检测只需要找到目标的 bounding box. 本文记录的分割更进一步的是 instance 的语义分割. 例如, 在一张合照中, 本文的分割方法会把所有的人标注为同一个类别”人”, 但是, 在 instance 的语义分割中, 会把每个人都认为是不同的目标实体, 分割结果中, 每个人的颜色都不一样.




我认为, FCN 的关键点有两个:

1. 去掉了 CNN 中通常会有的 full connect, 取而代之的是 conv
2. 使用 deconv(transpose conv) 来实现上采样, 从而使得缩小之后的 feature map 可以重建到原始输入图片的大小.

这两点也是实现 FCN 所必须的, 尤其是第二点, 目前除了 deconv 似乎没有更好的/更直观的方法. 而去掉 full connect 就对输入图片的图片就不需要是相同尺寸的, 甚至说可以是任意尺寸(这么说有一点点不严格, 要看整个网络中长宽最小的 feature map, 只要要求改 feature map 的长宽至少是 1 就可以了)

# 代码实现

这次没有使用 mxnet 实现而是使用了 TensorFlow.

首先是 load vgg19 的模型, 从该模型上 transfer. 由于 vgg19 是在 ImageNet 上面训练的, 而且是把输入图像 crop 到了 $224 \times 224$, 因此, 我也把输入图像 crop 到了 $224 \times 224$. 虽然下面的代码支持任意尺寸大于 $224 \times 224$ 的输入, 但是, 如果是完全任意尺寸的话, 基本上没办法进行矩阵话计算, 导致的结果就是每次只能输入一张图片训练,

速度会非常慢. 因此, 在训练阶段还是把所有的图像 resize 到相同的大小. 当然, predict 的输入图像只需要长宽都大于 224 就可以了.

解释一下为什么图片的尺寸要大于 $224 \times 224$. 从下面代码中可以看到, 在 conv 的操作中, feature map 的长宽都没有变化, 只有在 pooling 操作的时候发生变化, 每次减少 $\frac{1}{2}$, 共减少了 5 次, 这样, feature map 的大小变为 $\frac{224}{2^5} = 7$, 然后用了一个 $7 \times 7$ 的 global conv, 得到 $1 \times 1$ 的 feature map, 因此输入图像的长和宽都不能小于 224.

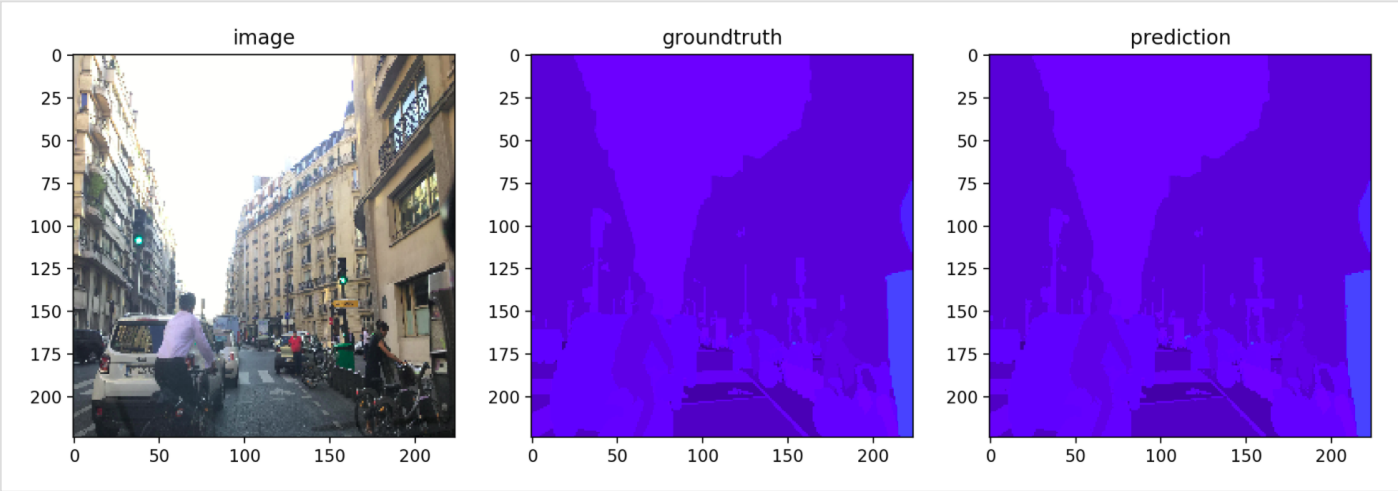
```
from __future__ import print_function, division
import tensorflow as tf
import numpy as np
import cv2
import scipy.io
import commentjson as json
conf = json.load(open("./config.json"))
logging.basicConfig(
    level=logging.DEBUG,
    format="%(asctime)s %(filename)s %(funcName)s(): %(lineno)i: %(levelname)s: %(message)s", )
logger = logging.getLogger(__name__)


def get_variable(weights, name=None):
    init = tf.constant_initializer(weights, dtype=tf.float32)
    return tf.get_variable(name=name, initializer=init, shape=weights.shape)


def weights_variable(shape, stddev=0.02, name=None):
    init = tf.truncated_normal(shape, stddev=stddev)
    return tf.Varialbe(init) if name is None else tf.get_variable(
        name, initializer=init)


```

```
25  def bias_variable(shape, name=None):
26      init = tf.constant(0.0, shape=shape)
27      return tf.Varialbe(init) if name is None else tf.get_variable(
28          name, initializer=init)
29
30
31  def conv2d_basic(x, W, bias):
32      conv = tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding="SAME")
33      return tf.nn.bias_add(conv, bias)
34
35
36  def conv2d_transpose_stride(x, W, b, output_shape=None, stride=2):
37      if output_shape is None:
38          output_shape = x.get_shape().as_list()
39          output_shape[1] *= 2
40          output_shape[2] *= 2
41          output_shape[3] *= W.get_shape().as_list()[2]
42      conv = tf.nn.conv2d_transpose(x, W, output_shape, strides=[1, stride, stride, 1], padding="SAME")
43      return tf.nn.bias_add(conv, b)
44
45
46  def avg_pool_2x2(x):
47      return tf.nn.avg_pool(x, ksize=(1, 2, 2, 1), strides=(1, 2, 2, 1), padding="SAME")
48
49
50  def max_pool_2x2(x):
51      return tf.nn.max_pool(x, ksize=(1, 2, 2, 1), strides=(1, 2, 2, 1), padding="SAME")
52
53
54  def vgg(weights, image):
55      layers = (
56      'conv1_1', 'relu1_1', 'conv1_2', 'relu1_2', 'pool1',
57      'conv2_1', 'relu2_1', 'conv2_2', 'relu2_2', 'pool2',
58      'conv3_1', 'relu3_1', 'conv3_2', 'relu3_2', 'conv3_3', 'relu3_3', 'conv3_4', 'relu3_4', 'pool3',
59      'conv4_1', 'relu4_1', 'conv4_2', 'relu4_2', 'conv4_3', 'relu4_3', 'conv4_4', 'relu4_4', 'pool4',
```

```
    'conv5_1', 'relu5_1', 'conv5_2', 'relu5_2', 'conv5_3', 'relu5_3', 'conv5_4', 'relu5_4', 'pool5',
    )
    net = {}
    current = image
    for idx, name in enumerate(layers):
        kind = name[:4]
        if kind == "conv":
            kernels, bias = weights[idx][0][0][0][0]
            kernels = get_variable(np.transpose(kernels, (1, 0, 2, 3)), name=name+'_w')
            bias = get_variable(bias.reshape(-1), name=name+'_b')
            current = conv2d_basic(current, kernels, bias)
        elif kind == 'relu':
            current = tf.nn.relu(current, name=name)
        elif kind == "pool":
            current = avg_pool_2x2(current)
        net[name] = current
    return net


def inference(image, keep_prob):
    image = image-tf.constant([123.68, 116.779, 103.939])
    modelpath = conf["vgg"]
    model_data = scipy.io.loadmat(modelpath)
    weights = np.squeeze(model_data['layers'])
    with tf.variable_scope("inference"):
        image_net = vgg(weights, image)
        conv_final_layer = image_net["conv5_3"]
        pool5 = max_pool_2x2(conv_final_layer)
        W6 = weights_variable([7, 7, 512, 4096], name="W6")
        b6 = bias_variable([4096], name="b6")
        conv6 = conv2d_basic(pool5, W6, b6)
        relu6 = tf.nn.relu(conv6, name="relu6")
        relu_dropout6 = tf.nn. dropout(relu6, keep_prob=keep_prob)
        W7 = weights_variable([1, 1, 4096, 4096], name="W7")
        b7 = bias_variable([4096], name="b7")
```

```
        conv7 = conv2d_basic(relu_dropout6, W7, b7)
        relu7 = tf.nn.relu(conv7, name="relu7")
        relu_dropout7 = tf.nn.dropout(relu7, keep_prob=keep_prob)
        W8 = weights_variable([1, 1, 4096, conf["num_of_classes"]], name="W8")
        b8 = bias_variable([conf["num_of_classes"]], name="b8")
        conv8 = conv2d_basic(relu_dropout7, W8, b8)

        # upscale
        deconv_shape1 = image_net["pool4"].get_shape()
        W_t1 = weights_variable([4, 4, deconv_shape1[3].value, conf["num_of_classes"]], name="W_t1")
        b_t1 = bias_variable([deconv_shape1[3].value], name="b_t1")
        conv_t1 = conv2d_transpose_stride(conv8, W_t1, b_t1, output_shape=tf.shape(image_net["pool4"]))
        fuse_1 = tf.add(conv_t1, image_net["pool4"], name="fuse_1")

        deconv_shape2 = image_net["pool3"].get_shape()
        W_t2 = weights_variable([4, 4, deconv_shape2[3].value, deconv_shape1[3].value], name="W_t2")
        b_t2 = bias_variable([deconv_shape2[3].value], name="b_t2")
        conv_t2 = conv2d_transpose_stride(fuse_1, W_t2, b_t2, output_shape=tf.shape(image_net["pool3"]))
        fuse_2 = tf.add(conv_t2, image_net["pool3"], name="fuse_2")

        shape = tf.shape(image)
        deconv_shape3 = tf.stack([shape[0], shape[1], shape[2], conf["num_of_classes"]])
        W_t3 = weights_variable([16, 16, conf["num_of_classes"], deconv_shape2[3].value], name="W_t3")
        b_t3 = bias_variable([conf["num_of_classes"]], name="b_t3")
        conv_t3 = conv2d_transpose_stride(fuse_2, W_t3, b_t3, output_shape=deconv_shape3, stride=8)

        pred = tf.argmax(conv_t3, dimension=3, name="prediction")
    return tf.expand_dims(pred, dim=3), conv_t3
```

结果如下:




预测结果已经很准了, 但是, 仔细观察还是有一些差别, 尤其是骑车后面骑自行的人, 仔细观察会有一些噪点和毛刺.

# Deep Learning # CNN # FCN # Segmentation

‹ MXNet 源码分析--Bind

怎样理解 Cross Entropy ›