華中科技大學

电子信息与通信学院
《数字图像处理》课程设计报告

小组成员 周宇轩 U201613409

李雪扬 U201614468

泥俊沛 U201614448

班级 种子 1601 班

时间 2019 年 5 月 19 日

一、课程设计目的

正确对 tiny_vid 数据集进行物体识别、检测鸟、车、狗、蜥蜴、乌龟五类物体，并且交并比 IOU 达到 0.5 以上。在完成课程设计的过程中加深对深度学习图像目标检测的理解。

二、算法原理

2.1 目标定位分类

目标定位分类是神经网络的一个重要方向，即“定位”加上“分类”。分类问题非常常见，如输入一张图片到多层卷积神经网络，输出一个特征向量，并且反馈给 softmax 单元来预测图片类型。要注意的是，假设在构造汽车自动驾驶系统，那么对象可能包括以下几类：行人、汽车、摩托车和背景，这意味着图片中不含有前三种对象，也就是说图片中没有行人、汽车和摩托车，输出结果会是背景对象，这四个分类就是 softmax 函数可能输出的结果。而如果同时还想要定位，就需要让神经网络再多输出 4 个数字，标记为 b_x,b_y,b_h 和 b_w，这四个数字是被检测对象的边界框的参数化表示。

要确定边界框的具体位置，需要指定红色方框的中心点，这个点表示为(b_x,b_y)，边界框的高度为 b_h，宽度为 b_w。因此训练集不仅包含神经网络要预测的对象分类标签，还要包含表示边界框的这四个数字，接着采用监督学习算法，输出一个分类标签，还有四个参数值，从而给出检测对象的边框位置。比如假设输出的是这样一个目标向量 y。

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

其中第一个组件 p_c 表示是否含有对象，如果对象属于前三类（行人、汽车、摩托车），则 p_c=1，如果是背景，则图片中没有要检测的对象，则 p_c=0。

我们可以这样理解 p_c，它表示被检测对象属于某一分类的概率，背景分类除外。

如果检测到对象，就输出被检测对象的边界框参数 b_x、b_y、b_h 和 b_w。最后，如果存在某个对象，那么 p_c=1，同时输出 c_1、c_2 和 c_3，表示该对象属

于 1-3 类中的哪一类，是行人，汽车还是摩托车。鉴于我们所要处理的问题，我们假设图片中只含有一个对象，所以针对这个分类定位问题，图片最多只会出现其中一个对象。

最后，这样的一个网络的损失函数是怎样的呢？如果采用平方误差策略，最后，我们介绍一下神经网络的损失函数，其参数为类别 y 和网络输出^y，如果采用平方误差策略，则 L 为如下，损失值等于每个元素相应差值的平方和。

$$L(\hat{y}, y) = (\hat{y_1} - y_1)^2 + (\hat{y_2} - y_2)^2 + \cdots (\hat{y_8} - y_8)^2$$

如果图片中存在定位对象，那么 y_1=1，所以 y_1=p_c，同样地，如果图片中存在定位对象，p_c=1，损失值就是不同元素的平方和。

另一种情况是，y_1=0，也就是 p_c=0，损失值是((y^_1)-y_1)^2，因为对于这种情况，我们不用考虑其它元素，只需要关注神经网络输出 p_c 的准确度。

如果输出的点更多，即有更多的特征点，我们就可以直接通过输出图片上的特征点坐标来实现对目标特征的识别，即特征点检测，如人脸识别和姿态识别。

2.2 目标检测

目标检测与目标定位分类的区别在于目标定位分类一般只有一个较大的对象，而目标检测可能不止一个对象，甚至单张图片中有不同分类的对象，要找出所有对象的位置和他们的分类。

滑动窗口，简单来说就是创建一个标签训练集，将数据图片裁剪成适当的小方块，其中应该同时有正样本和负样本（即存在对象和不存在对象的样本）输入到卷积网络中，使卷积网络输出 0 或 1，来辨别小格内是否有检测目标。训练完成后就可以用这个网络来进行窗口滑动了。即先判断图像的左上角小格子，再移动一步，继续移动，直到滑过整个图片，之后改变格子大小继续滑动。这样可以检测准确度很高，但是计算成本巨大。

滑动窗口对应的问题还有一个，就是不能输出比较精准的边界框，而 Bounding Box 预测可以解决这一问题，YOLO 就是体现这样一种预测的算法。

2.3 YOLO(You only look once)

YOLO 是这么做的，比如输入图像为 100*100 的，YOLO 会在图像上放一个网格，比如 3*3 的（实际应用会使用更精细一点的网格，比如 19*19）。基本思路是使用上面提到的图像分类和定位算法分别应用到 9 个格子上。

更具体一点，我们会如下图定义训练标签，对九个格子中的每一个指定一个标签 y。

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

p_c 等于 0 或 1 取决于这个绿色格子中是否有图像。

然后 b_x、b_y、b_h 和 b_w 作用就是，如果那个格子里有对象，那么就给出边界框坐标。

然后 c_1、c_2 和 c_3 就是你想要识别的三个类别，背景类别不算，所以你尝试在背景类别中识别行人、汽车和摩托车，那么 c_1、c_2 和 c_3 可以是行人、汽车和摩托车类别。这张图里有 9 个格子，所以对于每个格子都有这么一个向量。

值得一提的是，在 YOLO 中，对每个对象取一个中心点，将这个对象分配给包含这个中心点的方格来处理。对于其他方格，都“假装”格子中没有任何我们感兴趣的对象，即这些方格的标签向量的第一项为 0。

对于这里 9 个格子中的任何一个，会得到一个 8 维输出向量，因为这里是 3*3 的网络，所以总的输出尺寸为 3*3*8,。所以我们需要做的就是对于任何一张图片，即输入 x，通过一系列卷积层、最大池化层、激活层等等之后，映射成为一个 3*3*8 的向量。

2.4 NMS（非极大值抑制）

YOLO 可能出现的问题是，算法可能对同一个对象作出多次检测，所以算法可能会对面某个对象检测出多个结果。非极大值抑制这个方法可以确保算法对每一个算法只检测一次。

举个例子，在理想情况下，某个对象应该只有一个中心，因此应该只被分配到一个方格中。但实际上当我们运行对象分类和定位算法时，对于每一个格子都运行一次，所以格子 1 可能认为这个物体中点应该在自己内部，同时周围几个格子也都这么认为。

我们可以这么做，首先每次报告每个检测结果相关的概率 p_c。首先看概率最大的那个，找到 p_c 最高的，认为这是最可靠的检测。这么做之后，非极大值抑制就会逐一审视剩下的矩形，所有和这个最大的边框有很高交并比，高度重叠的其他边界框会被抑制。

所以这就是非极大值抑制，非最大值意味着你只输出概率最大的分类结果，但抑制很接近，但不是最大的其他预测结果，所以这方法叫做非极大值抑制。

2.5 Anchor boxex

anchor box 的思路是：预先定义多个不同形状的 anchor box，或者 anchor box 形状，你要做的是把预测结果和这两个 anchor box 关联起来。

如果对于某一个点有两个 anchor box，并且算法每次输出一个 8 元素向量，那么实际上对于这个有两个 anchor box 的点，输出的是一个 16 元素的向量

y=[p_c b_x b_y b_w b_h c1 c2 c3 p_c b_x b_y b_w b_h c1 c2 c3]

其中前八个元素对应第一个 anchor box，后八个对应另外一个。

总结一下，用 anchor box 之前，对于训练集图像中的每个对象，都根据那个对象中点位置分配到对应的格子中。用到 anchor box 这个概念后，每个对象都和之前一样分配到同一个格子中，但是会再次被分配到和对象形状交并比最高的 anchor box 中。

建立 anchor box 这个概念，是为了处理两个对象出现在同一个格子的情况，实践中这种情况很少发生，特别是如果你用的是 19×19 网格而不是 3×3 的网格，两个对象中点处于 361 个格子中同一个格子的概率很低，确实会出现，但出现频率不高。

一般手动选择 anchor box 形状，一般选择 5-10 个，但是也可以使用 k-means 算法，对多个形状聚类。

三、实验思路

通过课堂内容与线下自主学习，了解到物体识别定位的几种较为可靠的方案，Fast-RCNN, SSD, Yolo 等算法模型
，考虑到精度和训练速度等因素，我们选择尝试使用 yolo 算法来实现本次实验目的，并尝试分别使用 yolov1~yolov3 进行训练观察结果。

3.1 yolov1

3.1.1 YoloV1d 的提出

目标识别有两个方面的要求，一是确定位置，二是判别种类，在这方面现有算法做的比较好的是 R-CNN，但和人眼不同，R-CNN 的执行步骤是先选择性搜索确定候选边框，再对于框内物体进行分类，如果所要搜索的类就保留，否则丢弃，再对于每一个保留的框单独训练处一个回归模型来进行边框回归。这样的步骤是非常复杂和耗时的，而 Yolo 的执行是并行的，即所谓“You only look once”，快速准确的训练预测

3.1.2 yoloV1 的详细机制

模型训练

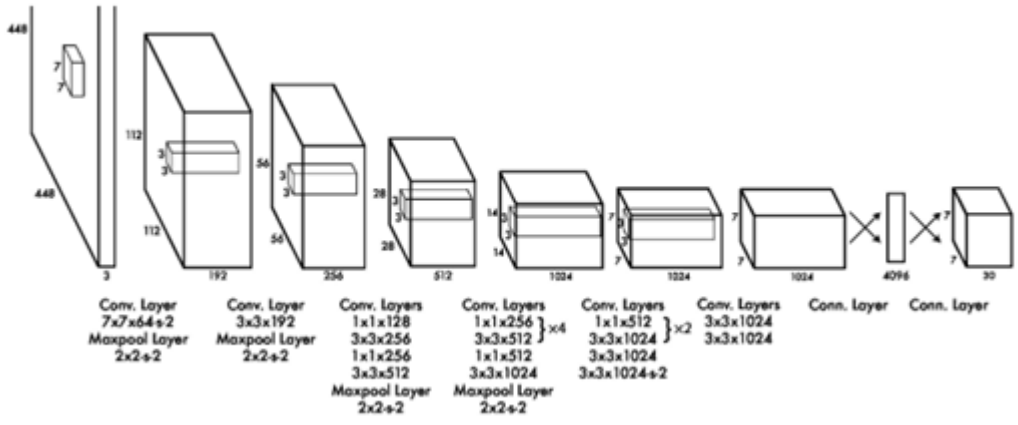
准备

Yolo 将所有组件组装在单一的神经网络里，每一个神经元单独负责其区域内的图像部分。首先在图像导入时进行采样，将图像大小确定为 448*448，同时通过卷积运算，将图像进行分块成 7*7 的网格，于是可以一次看到整个图像，变相实现了滑动窗口的作用，对于每一个区域是否存在检测对象进行判断。

检测

每一个单元格输出一个张量，维度为（B*5+C），B 为预测的边框的形状类型数、每一个边框具有中心点 x、y，宽高以及是否存在对象五个标签，C 为种类数量，每一个标签表示为该种类的可能性。

网络设计



基本设计如上图所示，最终输出维度为 7*7*（B*5+C）的张量。

损失函数

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

Yolo1 的不足

YOLO 对边界框预测强加空间约束，因为每个网格单元只能预测两个方框，并且只能有一个类。这种空间约束限制了模型可以预测的附近物体的数量。当预测的图片小，对象小时，没法准确的检测出物体，同时一个单元格内出现多物体的情况也无法被检测出

3.2.yolov2

3.2.1 Yolov2 的提出

Yolov2 的出现大大缓解了 Yolov1 版本的缺陷和不足，包括输入尺寸固定、目标较小时检测效果不佳、每个 bounding box 内最多只预测出一个物体等。

3.2.2 Yolov 的新机制

- 候选框anchor box

yolov2 引入了 faster RCNN 中的 anchor box 的思想，举个例子，就是对于图像的某一个位置都考虑多个面积、多种比例，这些不同大小的候选窗口就被称为 anchors，这样可以使模型更易学习。

- Darknet-19 network

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

网络总共包含 19 个卷积层加上 5 个最大池化层，同时在最后使用了 1*1 的卷积层来代替了 YOLOv1 的全连接层。相较 Yolov1, Yolov2 的网络全部使用 Batch 归一化，放弃了 dropout；使用了 1*1 卷积层来进行跨通道信息整合；同时在数据输入时使用了 SPP 空间金字塔池化，这样可以打破 Yolov1 中必须输入规定大小图片的桎梏。

• Multi-Scale Training

Yolov2 中可以通过改变 cfg 配置文件内 random 属性，配置是否每隔计策迭代后就微调网络的输入尺寸。训练时每迭代 10 次，就会随机选择新的输入图像尺寸。因为 YOLOv2 的网络使用的 downsamples 倍率为 32，所以使用 32 的倍数调整输入图像尺寸 {320,352，…，608}。训练使用的最小的图像尺寸为 320 x 320，最大的图像尺寸为 608 x 608。 这使得网络可以适应多种不同尺度的输入。

3.2.3 实验思路

参考已有的 darkflow 框架代码，使用 tiny_vid 数据集在 yolov2 网络上进行训练，得到一份结果，分析网络框架参数，进行调整，使得网络可以实现对 tiny_vid 数据集的对象检测定位。

3.3 Yolov3 网络框架训练

3.3.1 什么是 Yolov3？

• Darknet-53 network

整个网络从第 0 层到第 74 层，一共有 53 个卷积层，其余为 res 层，这就是 darknet-53 经典的卷积层，作为 yolov3 特征提取的主要网络结构，该结构使用一系列的 3*3 和 1*1 的卷积核的卷积层。

• 卷积层

layer filters size input output

0 conv 32 3 x 3 / 1 416 x 416 x 3 -> 416 x 416 x 32 0.299 BFLOPs

• 输入：像素为 416*416，通道数为 3 的的图片（开启 random 参数的话可以自适应以 32 为基础的变化，）

• BN 操作：对输入进行 BN 操作（这里未显示）。

• 卷积操作：32 层卷积核（filters），每个卷积核大小为 3*3，步伐为 1（每个卷积窗口逐步进行卷积）。

• 输出：32 个通道的 416*416 大小的 feature map

• Res 层（shortcut 操作）

layer filters size input output

4 res 1 208 x 208 x 64 -> 208 x 208 x 64

• 输入与输出：输入与输出一般保持一致，并且不进行其他操作，只是求差。

• 处理操作：res 层来源于 resnet，为了解决网络的梯度弥散或者梯度爆炸的现象，提出将深层神经网络的逐层训练改为逐阶段训练，将深层神经网络分为若干个子段，每个小段包含比较浅的网络层数，然后用 shortcut 的连接方式使得每个小段对于残差进行训练，每一个小段学习总差（总的损失）的一部分，最终达到总体较小的 loss，同时，很好的控制梯度的传播，避免出现梯度消失或者爆炸等不利于训练的情形。

• Yolo 层

从 75 到 105 层为 Yolo 网络的特征交互层，分为 3 个尺度，每个尺度内，通过卷积核的方式实现局部特征交互，作用类似于全连接层。

3.3.2 数据集划分

tiny_vid 数据集总共有五类对象，每一类下对应有效图片 180 张，划分每一类中 150 张为训练集，30 张为测试集。

3.3.3 思路

参考可用的 yolov3 框架代码，使用 tiny_vid 数据集进行训练，得到一份结果，检测此网络的可行性，如果可用，分析网络框架参数，进行调优工作，在有限时间内达到一个较好的训练结果。

四、实验方案与测试结果

4.1

1）数据预处理

标注格式：

image.jpg (center_X) (center_Y) (Width) (height) (class)

…….

将上述格式分为 train.txt 和 valid.txt

通过 Pytorch 中的 dataloader 载入内存

2）构建网络

使用 VGG16 构建网络

```
def vgg16(pretrained=False, **kwargs):
    """VGG 16-layer model (configuration "D")


    Args:
        pretrained (bool): If True, returns a model pre-trained on ImageNet
    """
    model = VGG(make_layers(cfg['D']), **kwargs)
    if pretrained:
        model.load_state_dict(model_zoo.load_url(model_urls['vgg16']))
    return model
```

3）训练

```
for epoch in range(num_epochs):
    net.train()
    # if epoch == 1:
    #     learning_rate = 0.0005
    # if epoch == 2:
    #     learning_rate = 0.00075
    # if epoch == 3:
    #     learning_rate = 0.001
    if epoch == 30:
        learning_rate=0.0001
    if epoch == 40:
        learning_rate=0.00001
    #                                  optimizer                                  =
torch.optim.SGD(net.parameters(),lr=learning_rate*0.1,momentum=0.9,weight_decay
=1e-4)
    for param_group in optimizer.param_groups:
        param_group['lr'] = learning_rate

    print('\n\nStarting epoch %d / %d' % (epoch + 1, num_epochs))
    print('Learning Rate for this epoch: {}'.format(learning_rate))

    total_loss = 0.

    for i,(images,target) in enumerate(train_loader):
        images = Variable(images)
        target = Variable(target)
        if use_gpu:
            images,target = images.cuda(),target.cuda()

        pred = net(images)
```

```
        loss = criterion(pred,target)
        total_loss += loss.item()

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if (i+1) % 5 == 0:
            print ('Epoch [%d/%d], Iter [%d/%d] Loss: %.4f, average_loss: %.4f'
            %(epoch+1,  num_epochs,  i+1,  len(train_loader),  loss.item(),
total_loss / (i+1)))
            num_iter += 1
            vis.plot_train_val(loss_train=total_loss/(i+1))
```

4) 验证

```
validation_loss = 0.0
    net.eval()
    for i,(images,target) in enumerate(test_loader):
        images = Variable(images,volatile=True)
        target = Variable(target,volatile=True)
        if use_gpu:
            images,target = images.cuda(),target.cuda()

        pred = net(images)
        loss = criterion(pred,target)
        validation_loss += loss.item()
    validation_loss /= len(test_loader)
    vis.plot_train_val(loss_val=validation_loss)

    if best_test_loss > validation_loss:
        best_test_loss = validation_loss
```

```
        print('get best test loss %.5f' % best_test_loss)
        torch.save(net.state_dict(),'best.pth')
    logfile.writelines(str(epoch) + '\t' + str(validation_loss) + '\n')
    logfile.flush()
    torch.save(net.state_dict(),'yolo.pth')
```

当 loss 更低时，则将其保存为新的最佳模型

5) 损失函数

```
    box_pred_response = box_pred[coo_response_mask].view(-1,5)
            box_target_response_iou = box_target_iou[coo_response_mask].view(-
1,5)
            box_target_response = box_target[coo_response_mask].view(-1,5)
            contain_loss =
F.mse_loss(box_pred_response[:,4],box_target_response_iou[:,4],size_average=Fal
se)
            loc_loss = F.mse_loss(box_pred_response[:, :2],
box_target_response[:, :2], size_average=False) + F.mse_loss(
                torch.sqrt(box_pred_response[:, 2:4]),
torch.sqrt(box_target_response[:, 2:4]), size_average=False)
            # 2.not response loss
            box_pred_not_response = box_pred[coo_not_response_mask].view(-1, 5)
            box_target_not_response = box_target[coo_not_response_mask].view(-1,
5)
            box_target_not_response[:, 4] = 0
            # not_contain_loss =
F.mse_loss(box_pred_response[:,4],box_target_response[:,4],size_average=False)

            # I believe this bug is simply a typo
            not_contain_loss = F.mse_loss(box_pred_not_response[:, 4],
box_target_not_response[:, 4], size_average=False)
```

```
            # 3.class loss
            class_loss = F.mse_loss(class_pred, class_target, size_average=False)

            return (
                        self.l_coord * loc_loss + 2 * contain_loss +
not_contain_loss + self.l_noobj * nooobj_loss + class_loss) / N
```

6) 训练结果

loss 约为 2.2 左右，随训练进行逐渐降低

7) 测试结果

由于预测函数存在部分问题，训练好的模型不能够顺利导入，暂未得到该模型的结果及正确率，但使用第三方已训练好的模型可以较为准确检测对象：

8) 结果分析

由于对于 Pytorch 框架的不熟悉，加上时间的仓促，实现的情况并不理想，在后续过程中，我会继续完成这次课设，同时尝试使用 darknet 框架实现 yoloV1 算法

4.2 Yolov2 网络框架训练 tiny_vid 数据集

参考代码：https://github.com/deep-diver/Object-Detection-YOLOv2-Darkflow/

4.2.1 训练步骤

1） 数据预处理

- 标注格式

darkflow 要求的输入格式为：

-Images
------0.jpg
------1.jpg

...
-Annotations
------0.xml
------1.xml
...

其中 xml 格式如下

```
<annotation>
<folder>captures_vlc</folder>
<filename>bird000002.JPEG</filename>
<size>
<width>128</width>
<height>128</height>
<depth>3</depth>
</size>
<object>
<name>bird</name>
<pose>Unspecified</pose>
<truncated>0</truncated>
<difficult>0</difficult>
<bndbox>
<xmin>24</xmin>
<ymin>33</ymin>
<xmax>64</xmax>
<ymax>66</ymax>
</bndbox>
</object>
</annotation>
```

需要将 txt 格式的 label 转化为 xml 格式。分别使用 shell 和 python 将 txt 转化为 xml 格式。首先将所有图片名输入到一个文件中，再将所有的 label.txt 输入到一个文件中，最后用 sublime 的列选择将文件名放到坐标前面，即如下格式

```
bird000001.JPEG 0 65 17 103 52
bird000002.JPEG 0 24 33 64 66
bird000003.JPEG 0 36 34 81 77
...
```

分别对应文件名、类别、x_min、y_min、x_max、y_max。

再使用如下程序转化为 xml。

```
# 修改为你自己的路径
template_file = './dataset/scene00601.xml'
target_dir = './dataset/Annotations/'
image_dir = './dataset/images/'    # 图片文件夹
train_file = './dataset/collectedLabelxxyy.txt'    # 存储了图片信息的 txt 文件

with open(train_file) as f:
trainfiles = f.readlines()    # 标注数据  格式(filename label x_min y_min x_max y_max)

file_names = []
tree = ElementTree()

for line in trainfiles:
trainFile = line.split()
file_name = trainFile[0]

# 如果没有重复，则顺利进行。这给的数据集一张图片的多个框没有写在一起。
if file_name not in file_names:
file_names.append(file_name)
lable = trainFile[1]
xmin = trainFile[2]
ymin = trainFile[3]
xmax = trainFile[4]
ymax = trainFile[5]
if lable =='0':
lable = 'bird'

elif lable =='1':
lable = 'car'

elif lable =='2':
lable = 'dog'

elif lable =='3':
lable = 'lizard'

elif lable =='4':
lable = 'turtle'

tree.parse(template_file)
root = tree.getroot()
```

```
root.find('filename').text = file_name

# size
sz = root.find('size')
im = cv2.imread(image_dir + file_name)#读取图片信息\


sz.find('height').text = str(im.shape[0])
sz.find('width').text = str(im.shape[1])
sz.find('depth').text = str(im.shape[2])

# object  因为我的数据集都只有一个框
obj = root.find('object')

obj.find('name').text = lable
bb = obj.find('bndbox')
bb.find('xmin').text = xmin
bb.find('ymin').text = ymin
bb.find('xmax').text = xmax
bb.find('ymax').text = ymax
#  如果重复，则需要添加 object 框
else:
lable = trainFile[1]
xmin = trainFile[2]
ymin = trainFile[3]
xmax = trainFile[4]
ymax = trainFile[5]

xml_file = file_name.replace('jpg', 'xml')
tree.parse(target_dir + xml_file)#如果已经重复
root = tree.getroot()

obj_ori = root.find('object')

obj = copy.deepcopy(obj_ori)    #  注意这里深拷贝

obj.find('name').text = lable
bb = obj.find('bndbox')
bb.find('xmin').text = xmin
bb.find('ymin').text = ymin
bb.find('xmax').text = xmax
bb.find('ymax').text = ymax
root.append(obj)

xml_file = file_name.replace('JPEG', 'xml')
```

```
print(xml_file)
tree.write(target_dir + xml_file, encoding='utf-8')
```

之后将各类的前 150 张图片作为训练集，将剩余的作为测试集。

2） 下载预训练权重与 cfg 配置文件

cfg 配置文件各参数意义如下：

```
[net]
# Testing
#batch=1
#subdivisions=1
# Training
batch=32 #每 batch 个样本更新一次参数
subdivisions=8    #如果内存不够大，通常将一个 batch 分为 subdivisions 份，在 darknet 中会将 batch/subdivisions 认为成 batch

width=128  #输入图片的宽度
height=128  #输入图片的高度
channels=3  #输入图片的通道
momentum=0.9  #梯度下降动量
decay=0.0005  #权重衰减正则化，为了防止过拟合
angle=0.5  #图片旋转的角度，来生成更多的训练样本
saturation = 1.5  #调整饱和度，来产生更多的训练样本
exposure = 1.5  #调整曝光度，来产生更多的训练样本
hue=.1  #调整色度

learning_rate=0.00001  #学习率，即训练的时候初始值
burn_in=100  #当训练次数达到 burn_in 后，学习率的更新开始使用 policy 策略
max_batches = 500  #训练的最高次数，即 batches 数量，到达该值停止训练
policy=steps  #学习率更新的方法(随步数进行衰减)
steps=1000,2000  #在 1000，2000 的时候进行调整学习率
scales=.1,.1  #每次更改的学习率为上一个学习率*scales，为当前步的学习率的 1/10

[convolutional]  #卷积层
batch_normalize=1  #在训练初始，进行 batch_normalize，其能够防止过拟合，规范化模型
filters=32  #卷积后输出的特征图个数
size=3  #卷积核的尺寸
stride=1  #卷积的步长
pad=1  #当 pad 为 0 时，padding 由 padding 的参数决定,反之，当 pad 为 1 时，padding 大小
```

为 size/2
activation=leaky　#使用 leaky RELU 激活函数

[maxpool]　#池化层
size=2　#池化的核的大小
stride=2　#池化的步长

...　//卷积层部分

[route]　#使用的是 Fine-Grained Features 细粒度特征
layers=-9　#将上个卷积层的输出导入到 network 中(将这一层之前第九层的特征图导入相加)

[convolutional]
batch_normalize=1
size=1
stride=1
pad=1
filters=64
activation=leaky

[reorg]　#将上个卷积层的输入与最后输出的 featuremap 做链接
stride=2

[route]　#进行将两层的 featuremap 做 concat
layers=-1,-4

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=1024
activation=leaky

[convolutional]
size=1
stride=1
pad=1
filters=425　#region 的前一个 filters 的数量是根据种类以及 anchor 来确定的：
filters=anchor_boxnum*(class+5)
activation=linear

```
[region]
anchors =  0.57273, 0.677385, 1.87446, 2.06253, 3.33843, 5.47434, 7.88282, 3.52778, 9.77052, 9.16828 #anchor boxes 的尺度比
bias_match=1
classes=80  #目标种类个数
coords=4  #每个 box 的坐标
num=5  #anchor 的数量
softmax=1  #使用 softmax 分类
jitter=.3  #通过抖动增加噪声，防止过拟合
ignore_thresh=.5  #当存在对象的概率大于 0.5 时认为有对象，否则认为没有
rescore=1  #非 0 时进行重新打分

object_scale=5
noobject_scale=1
class_scale=1
coord_scale=1

absolute=1
thresh = .1  #最后测试时置信度小于 0.1 的不会被输出
random=1  #训练时每隔一定输入随机调整网络输入，使网络可以适应各种大小的输入
```

这里需要说明的是为什么最后的[region]层的 classes 为 80 而不是 tiny_vid 的 5。因为 YOLO 是需要采用预训练的 weights 来进行配置的（尝试过不使用预训练的 weights 直接训练，结果是会将任何图都输出为没有我们感兴趣的对象），而我们想要使用 YOLO 官方给出的 weights 的时候，如果改动了 classes 属性就会导致 weights 参数与网络参数不一致，无法导入 weights。因此这里只能使用 80 这一数字。

80 是 coco 数据集的 label 数量，当使用 yolo.cfg 这个文件名时 darkflow 会自动读入 coco.names 这个文件里的内容作为标签。由于 tiny_vid 数据集中的 lizard 与 turtle 类别在 coco 中是不存在的，我们需要将 coco.names 中的两个替换为 lizard 和 turtle 再进行训练。

3）构建网络

使用 darkflow 的方法构建网络

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import copy
import cv2

from lxml.etree import Element, SubElement, tostring, ElementTree
from darkflow.net.build import TFNet

options = {"model": "cfg/yolo.cfg",
#"load": -1,
"load": "bin/yolo.weights",
"batch": 10,
"lr": 0.000001,
"epoch": 10,
"train": True,
"annotation": "./dataset/test/Annotations",
"dataset": "./dataset/test/images/",
}

tfnet = TFNet(options)
```

options 的含义是：使用 yolo.cfg 配置文件，载入 yolo.weights 预训练权重（load 设置为-1 时会载入最新的 ckpt 检查点模型继续训练），每个 batch 大小为 10，学习率为 1e-6，本次训练运行 10 个 epoch，训练模式，标签目录为./dataset/test/Annotations，图片目录为./dataset/test/images。

若载入预训练模型后直接进行测试，此时输出正确情况如下。

```
...
dog000173.JPEG    label: dog dog IOU= 0.3137142857142857
dog000177.JPEG    label: dog dog IOU= 0.26811594202898553
lizard000157.JPEG    label: bird lizard IOU= 0.23577235772357724
lizard000171.JPEG    label: bird lizard IOU= 0.3124567474048443
turtle000155.JPEG    label: bird turtle IOU= 0.22812812812812813
count =0 lose = 198
```

可见未训练时模型是无法正确检测对象的。

执行 tfnet = TFNet(options)后会输出构建好的网络

```
Building net ...
Source  Train?  Layer description                Output size
-------+--------+----------------------------------+---------------
input                                   (?, 128, 128, 3)
Init    Yep!    conv 3x3p1_1  +bnorm  leaky     (?, 128, 128, 32)
Load    Yep!    maxp 2x2p0_2                      (?, 64, 64, 32)
Init    Yep!    conv 3x3p1_1  +bnorm  leaky     (?, 64, 64, 64)
Load    Yep!    maxp 2x2p0_2                      (?, 32, 32, 64)
Init    Yep!    conv 3x3p1_1  +bnorm  leaky     (?, 32, 32, 128)
Init    Yep!    conv 1x1p0_1  +bnorm  leaky     (?, 32, 32, 64)
Init    Yep!    conv 3x3p1_1  +bnorm  leaky     (?, 32, 32, 128)
Load    Yep!    maxp 2x2p0_2                      (?, 16, 16, 128)
Init    Yep!    conv 3x3p1_1  +bnorm  leaky     (?, 16, 16, 256)
Init    Yep!    conv 1x1p0_1  +bnorm  leaky     (?, 16, 16, 128)
Init    Yep!    conv 3x3p1_1  +bnorm  leaky     (?, 16, 16, 256)
Load    Yep!    maxp 2x2p0_2                      (?, 8, 8, 256)
Init    Yep!    conv 3x3p1_1  +bnorm  leaky     (?, 8, 8, 512)
Init    Yep!    conv 1x1p0_1  +bnorm  leaky     (?, 8, 8, 256)
Init    Yep!    conv 3x3p1_1  +bnorm  leaky     (?, 8, 8, 512)
Init    Yep!    conv 1x1p0_1  +bnorm  leaky     (?, 8, 8, 256)
Init    Yep!    conv 3x3p1_1  +bnorm  leaky     (?, 8, 8, 512)
Load    Yep!    maxp 2x2p0_2                      (?, 4, 4, 512)
Init    Yep!    conv 3x3p1_1  +bnorm  leaky     (?, 4, 4, 1024)
Init    Yep!    conv 1x1p0_1  +bnorm  leaky     (?, 4, 4, 512)
Init    Yep!    conv 3x3p1_1  +bnorm  leaky     (?, 4, 4, 1024)
Init    Yep!    conv 1x1p0_1  +bnorm  leaky     (?, 4, 4, 512)
Init    Yep!    conv 3x3p1_1  +bnorm  leaky     (?, 4, 4, 1024)
Init    Yep!    conv 3x3p1_1  +bnorm  leaky     (?, 4, 4, 1024)
Init    Yep!    conv 3x3p1_1  +bnorm  leaky     (?, 4, 4, 1024)
Load    Yep!    concat [16]                       (?, 8, 8, 512)
Init    Yep!    conv 1x1p0_1  +bnorm  leaky     (?, 8, 8, 64)
Load    Yep!    local flatten 2x2                 (?, 4, 4, 256)
Load    Yep!    concat [27, 24]                   (?, 4, 4, 1280)
Init    Yep!    conv 3x3p1_1  +bnorm  leaky     (?, 4, 4, 1024)
Init    Yep!    conv 1x1p0_1  linear            (?, 4, 4, 425)
-------+--------+----------------------------------+---------------
Running entirely on CPU
cfg/yolo.cfg loss hyper-parameters:
H           = 4
W           = 4
box         = 5
classes = 80
```

scales   = [1.0, 5.0, 1.0, 1.0]

4）开始训练

tfnet.train()

会输出载入的数据集信息以及 Loss 更新状态。

Statistics:
bird: 150
car: 150
dog: 150
lizard: 150
turtle: 150
Dataset size: 750
Dataset of 750 instance(s)
Training statistics:
Learning rate : 1e-05
Batch size : 10
Epoch number : 2
Backup every : 2000
step 1 - loss 2.4945101737976074 - moving ave loss 2.4945101737976074
step 2 - loss 2.6167423725128174 - moving ave loss 2.5067333936691285
step 3 - loss 2.1875224113464355 - moving ave loss 2.4748122954368594
...

之后观察 loss，及时更新 lr 学习率以及注意使 load 参数为-1，接着上一次训练结果的模型继续训练，直到 loss 大致收敛。

5）测试

先构建用来测试的网络，即之前使用的 cfg 配置文件，与最新的检查点模型。

```
options = {"model": "cfg/yolo.cfg",
"load": -1,
"gpu": 1.0}

tfnet2 = TFNet(options)
```

为了进行正确的检测，需要先实现一些函数。

计算通过两个矩形的左上、右下两个点来计算 IOU。

```
import os
def calculateIOU(x1,y1,x2,y2,x3,y3,x4,y4):
w1 = x2 - x1
w2 = x4 - x3
h1 = y2 - y1
h2 = y4 - y3
IOU_W = min(x1,x2,x3,x4)+w1+w2-max(x1,x2,x3,x4)
IOU_H = min(y1,y2,y3,y4)+h1+h2-max(y1,y2,y3,y4)
IOU_S = IOU_W * IOU_H
if IOU_S<0:
return 0
whole_s = w1*h1 + w2*h2 - IOU_S
return IOU_S/whole_s
```

之后我们将测试集送到模型中，返回预测结果，与 xml 中的正确标签对比，计算出 IOU，计算正确的数量。需要另外说明的是，由于我们为了使用预训练权重我们载入了 80 个类，虽然是使用的 5 类进行训练，但是还是可能会输出这 5 类之外的类。因此为了不造成干扰，我们将判决的这 5 类之外的 label 进行剔除，由于测试集都为单对象图片，因此也只保留同类别里置信度最大的结果最为预测输出，用来和正确标签进行比对。

```
from lxml.etree import Element, SubElement, tostring, ElementTree

tree = ElementTree()

files = os.listdir('./dataset/test/Annotations')
lose = 0
count = 0
for fi in files:
tree.parse('./dataset/test/Annotations/'+fi)
root = tree.getroot()
filename = root.find('filename').text
obj = root.find('object')
label = obj.find('name').text
```

```
bb = obj.find('bndbox')
xmin = int(bb.find('xmin').text)
ymin = int(bb.find('ymin').text)
xmax = int(bb.find('xmax').text)
ymax = int(bb.find('ymax').text)

original_img = cv2.imread("./dataset/test/images/"+filename)
original_img = cv2.cvtColor(original_img, cv2.COLOR_BGR2RGB)
results = tfnet2.return_predict(original_img)

buf = list()
for i in range(len(results)):
if(results[i]['label']== 'car' or results[i]['label']== 'bird' or results[i]['label']== 'dog' or
results[i]['label']== 'lizard' or results[i]['label']== 'turtle'):
buf.insert(0,i)

if len(buf) == 0:
lose += 1
else:
if len(buf) == 1:
result = results[buf[0]]
#print(filename +'    '+ str(result) + '\n')

elif len(buf) > 1:
tmp = 0
index = 0
for j in range(len(buf)):
if results[buf[j]]['confidence'] > tmp:
tmp = results[buf[j]]['confidence']
index = buf[j]
result = results[index]

x1 = result['topleft']['x']
y1 = result['topleft']['y']
x2 = result['bottomright']['x']
y2 = result['bottomright']['y']
IOU = calculateIOU(x1,y1,x2,y2,xmin,ymin,xmax,ymax)
print(filename +'    label:', result['label'], label, 'IOU=', IOU,    '\n')
if result['label'] ==    label    and    IOU>0.5:
count+=1
else:
lose+=1


print('correct =' + str(count) + ' lose = '+ str(lose) )
```

6）结果可视化

使用 boxing 函数来将输出的方框在图中画出，并且标注上判断的 label。为了抑制一下出现的同对象多个框的现象，我简单实现了一下非极大值抑制（实际上应该是使用 IOU 来判断的）。

其中 ifNear()是用来进行非极大值抑制的。

```
def ifNear(x1,y1,x2,y2,width,height):
if abs(x1-x2) < 0.05*width:
if abs(y1-y2) < 0.05*width:
return True
return False

def boxing(original_img, predictions):
newImage = np.copy(original_img)

buf = list()
for i in range(len(results)):
if(results[i]['label'] != 'car' and results[i]['label'] != 'bird' and results[i]['label'] != 'dog' and results[i]['label'] != 'lizard' and results[i]['label'] != 'turtle'):
buf.insert(0,i)

for j in range(len(buf)):
predictions.remove(predictions[buf[j]])

height = original_img.shape[0]
width = original_img.shape[1]
buf2 = list()
for i in range(len(predictions)):
for j in range(len(predictions)):
if i != j and predictions[i]['label'] == predictions[j]['label']:
x1i = predictions[i]['topleft']['x']
y1i = predictions[i]['topleft']['y']
x2i = predictions[i]['bottomright']['x']
y2i = predictions[i]['bottomright']['y']
x1j = predictions[j]['topleft']['x']
y1j = predictions[j]['topleft']['y']
x2j = predictions[j]['bottomright']['x']
```

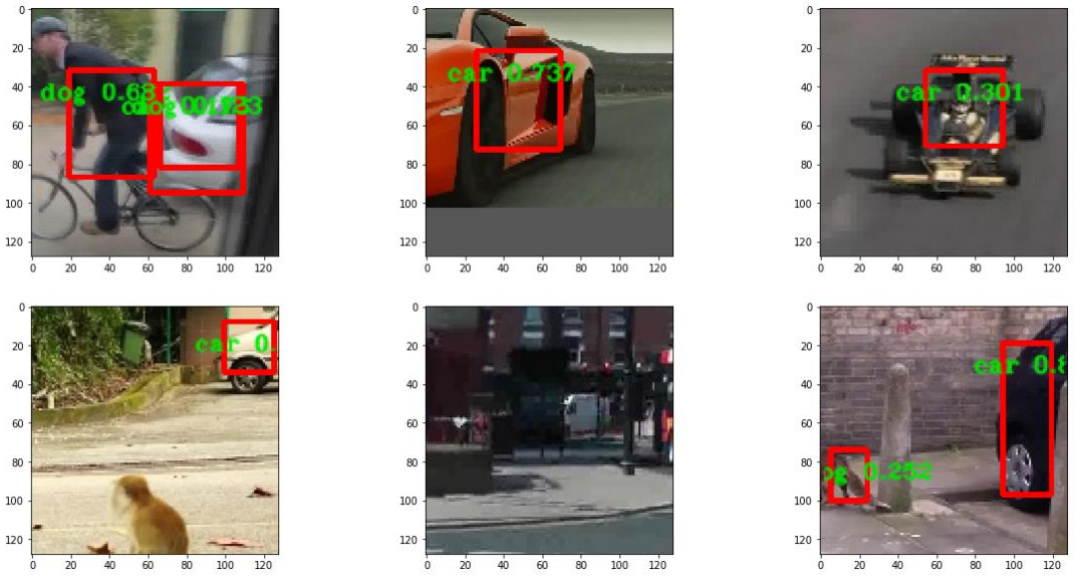
```
y2j = predictions[j]['bottomright']['y']
if ifNear(x1i,y1i,x1j,y1j,width,height) and ifNear(x2i,y2i,x2j,y2j,width,height):
if predictions[i]['confidence'] >= predictions[j]['confidence']:
buf2.insert(0,j)
else:
buf2.insert(0,i)

for j in range(len(buf2)):
predictions.remove(predictions[buf2[j]])

for result in predictions:
top_x = result['topleft']['x']
top_y = result['topleft']['y']

btm_x = result['bottomright']['x']
btm_y = result['bottomright']['y']

confidence = result['confidence']
label = result['label'] + " " + str(round(confidence, 3))

if confidence > 0.1:
if result['label'] == "car" or result['label'] == "bird" or result['label'] ==  "lizard"  or result['label']
== "dog" or result['label'] == "turtle":
newImage = cv2.rectangle(newImage, (top_x, top_y), (btm_x, btm_y), (255,0,0), 2)
newImage = cv2.putText(newImage, label, (top_x-15, top_y+15),
cv2.FONT_HERSHEY_COMPLEX_SMALL , 0.6, (0, 230, 0), 1, cv2.LINE_AA)

return newImage
```

4.2.2 实验结果

1）正确率

```
...
turtle000175.JPEG  label: turtle turtle IOU= 0.8301886792452831
turtle000176.JPEG  label: turtle turtle IOU= 0.27282453637660486
turtle000177.JPEG  label: dog turtle IOU= 0.7366430260047281
turtle000179.JPEG  label: bird turtle IOU= 0.1449562023170387
correct =79 lose = 119  accuracy = 0.398989898989899
```

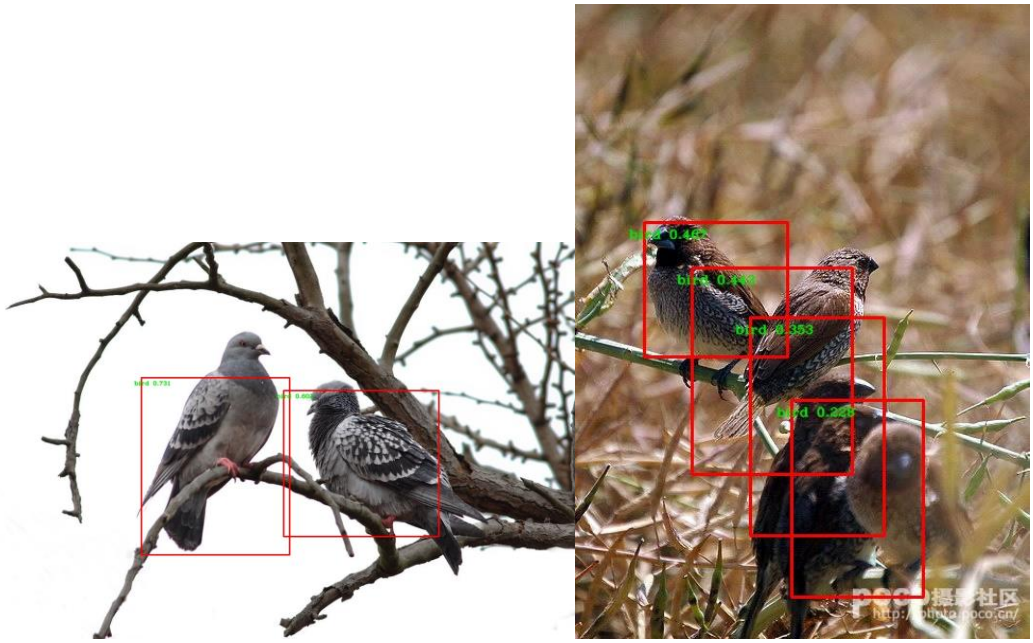
2）可视化结果

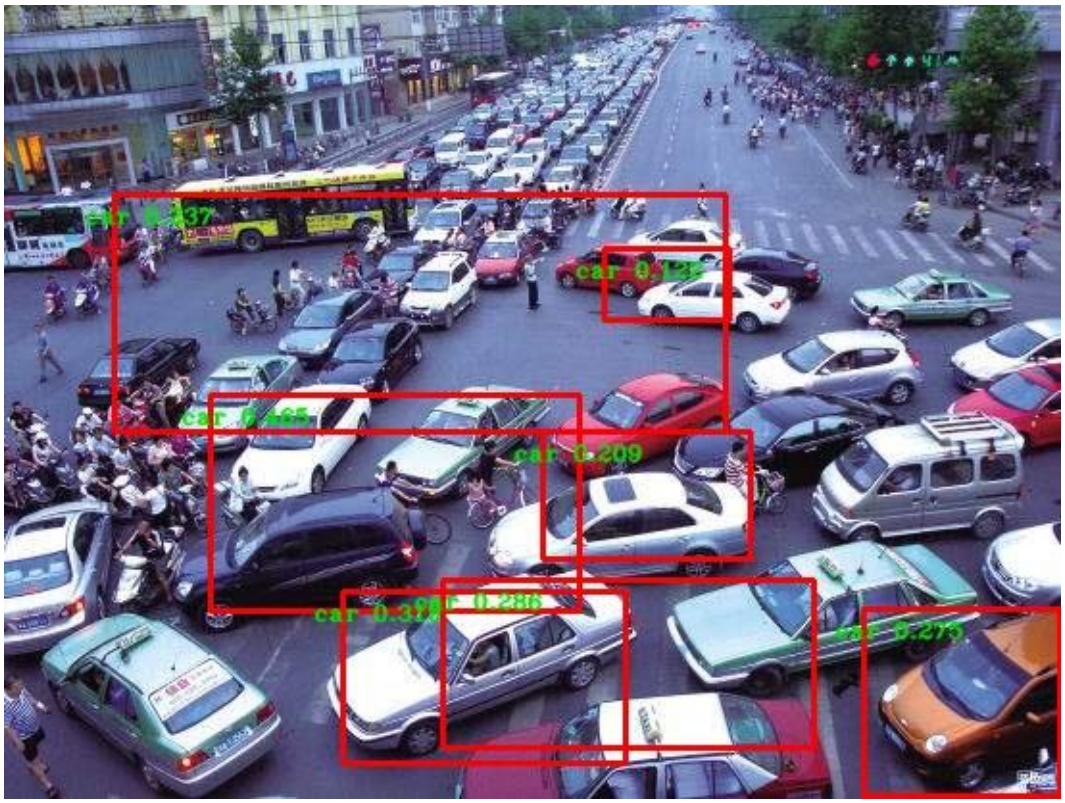


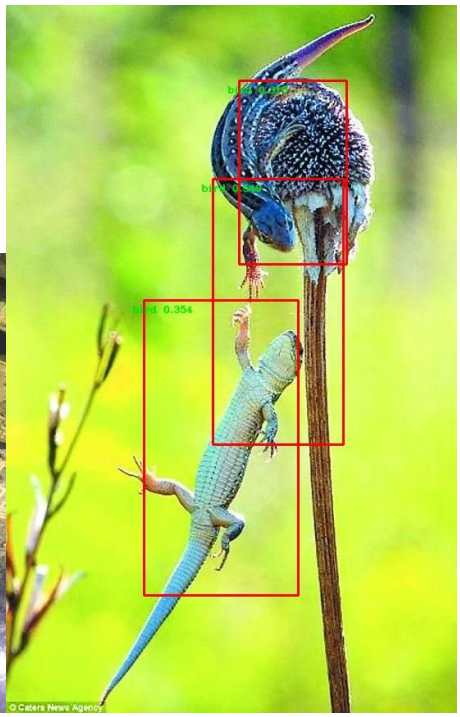
由于针对标签的单结果，我在计算正确率的时候只是取了置信度最高的一个。而画框的时候并没有这个限制，会把所有检测到的对象都框出来。

3） 多对象检测

我在百度中找了几个多对象的进行测试，发现我们的网络是可以检测到多个对象的。







4） 结果讨论

正确率不是很高，分析得出可能造成的原因有如下几条：

- 图片像素太小，细节不够清楚，导致判断的时候不能很好的提取特征从而很好的分类。同时从可视化结果中的车辆图中也能看出，由于对细节提取得不够导致图片中更多的较小的车没有被识别检测到。
- 数据集太小，仅 750 张图片不能很好地使网络找到鸟、车等五类目标的共性，因此在使用数据集之外的图片进行检测二点时候效果并不是很好。
- 预训练的影响，由于采用的预训练是在 coco 数据集上完成的，包含了我们的数据集的 bird、car、dog 三类，从可视化结果中也可以看出参与了预训练的三类的效果明显好于 lizard、turtle 两类。这两类不仅在定位上有问题，分类识别也不是很准。
- 可视化结果中也可以看出，yolov2 训练出的网络是可以检测多个对象的。

如果要进一步提高正确率，我认为可以从以下几个方面入手：增大数据集、增加较大分辨 率的数据、将网络的输入从 128*128 扩大、使用 k-means 求得更加符合我们数据集的 anchor boxex 比例、以及使用同时包含五类对象的预训练结果。

同时 darkflow 是 darknet 在 tensorflow 的移植版本，可能在性能、兼容上有一些差别，如果还有时间将会尝试使用 C 语言编写的 darknet 框架训练。

4.3 Yolov3 网络框架训练 tiny_vid 数据集

参考代码：https://github.com/ultralytics/yolov3

4.3.1 训练步骤

1）数据集处理

- 标注格式

参考模型需要的数据目录格式为：

```
-images
------0.jpg
------1.jpg
...
-labels
------0.txt
------1.txt
...
```

其中所需要的 label 格式为 Class x_center y_center Wigth Height（归一化后）

如：

```
# 归一化：x_center=x/img_width,Width=w/img_width...
0 0.656250 0.273438 0.296875 0.273438
```

所以需要将现有的 txt 文件进行修改，转化成网络支持的标注格式。并且在上层目录中有 tiny_vid_train.txt 和 tiny_vid_test.txt 来索引训练和测试数据，以训练索引为例，txt 文件内容格式为：

```
data/coco/images/tiny_vid/bird000001.jpg
data/coco/images/tiny_vid/bird000002.jpg
data/coco/images/tiny_vid/bird000003.jpg
...
```

- tiny_vid.data 和 tiny_vid.names 文件

除了标签的格式以及数据索引文件之外，还需要一个文件负责给网络传递参数，作为所有文件的索引，另外一个文件作为类别标签。

其中 tiny_vid.data 作为索引文件内容如下:

```
classes=5
train=data/tiny_vid_train.txt
valid=data/tiny_vid_test.txt
names=data/tiny_vid.names
```

```
backup=backup/
eval=coco
```

tiny_vid.names 作为标签名格式如下:

```
bird
car
dog
lizard
turtle
```

要与标签 0-4 顺序一致。

2）配置网络参数

yolov3 采用.cfg 文件来记录网络参数，在 cfg 目录下可以看到几个 cfg 文件，我选择 yolov3-spp.cfg 来作为实验的基础网络参数，内容如下：

```
[net]
# Testing
batch=64
subdivisions=16   # 表示 batch 划分，这里就表示一次训练 64/16
# Training
# batch=64
# subdivisions=16
width=608   # 输入大小，分辨率高可以得到更多细节
height=608
channels=3   # 通道数
momentum=0.9 # 动量，影响下降到梯度最优的速度
decay=0.0005 # 权重衰减正则项，防止过拟合，学习后参数固定比例下降
angle=0          # 通过旋转角度的到更多样本
saturation = 1.5 # 饱和度
exposure = 1.5     # 曝光度
hue=.1     # 色调

learning_rate=0.001   # 学习率
burn_in=1000 # 迭代次数大于 burnin 时，学习率使用 policy 方式更新
max_batches = 500200 # 训练达到该值结束
policy=steps # 学习率调整策略
steps=400000,450000   # 设置学习率的变化
```

```
scales=.1,.1
...
# 卷积 + res + yolo
[yolo]
mask = 6,7,8
anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,  156,198,  373,326
classes=5
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
```

tiny_vid 数据集的图片大小为 128*128，类别只有 5 类，所以需要修改网络中传递过程中 的类别数，以及 filter 的大小，修改部分如下：

```
[convolutional]
size=1
stride=1
pad=1
filters=30 # 修改处：15 + 3*classes
activation=linear


[yolo]
mask = 6,7,8
anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,  156,198,  373,326
classes=5 # 修改处：80->5
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
```

3）训练

要求的 python 版本至少为 3.7，pytorch>1.1.0，根据指示安装完成所有依赖包之后执行

```
python train.py --cfg=cfg/yolov3-spp.cfg        \
--data=data/tiny_vid.data      \
--batch-size=8          \
--num-workers=0             \
--epochs=50
```

4）输出训练结果

训练结果会追加的方式保存在 results.txt 中。

执行 utils 中的打印结果方法，即可获取统计后的训练结果。在当前目录生成 results.png。

训练过程中会保存每一轮 epoch 的 loss，但是实际上存储的是最优模型，所以即便是在训练到一定程度后，loss 出现波动或者上升，最后得到的模型不会是被破坏的模型。

```
python
>>> from utils import utils
>>> # 可以传入两个整数，表示起始行和结束行
>>> # 空参数表示默认输出全部 results.txt 结果到图中
>>> utils.plot_results()
```

5）测试

将要测试的图片放入 data/samples 中，执行：

```
python detect.py --cfg=cfg/yolov3-spp.cfg      \
--data=dara/tiny_vid.data    \
--weights=weights/best.pt
```

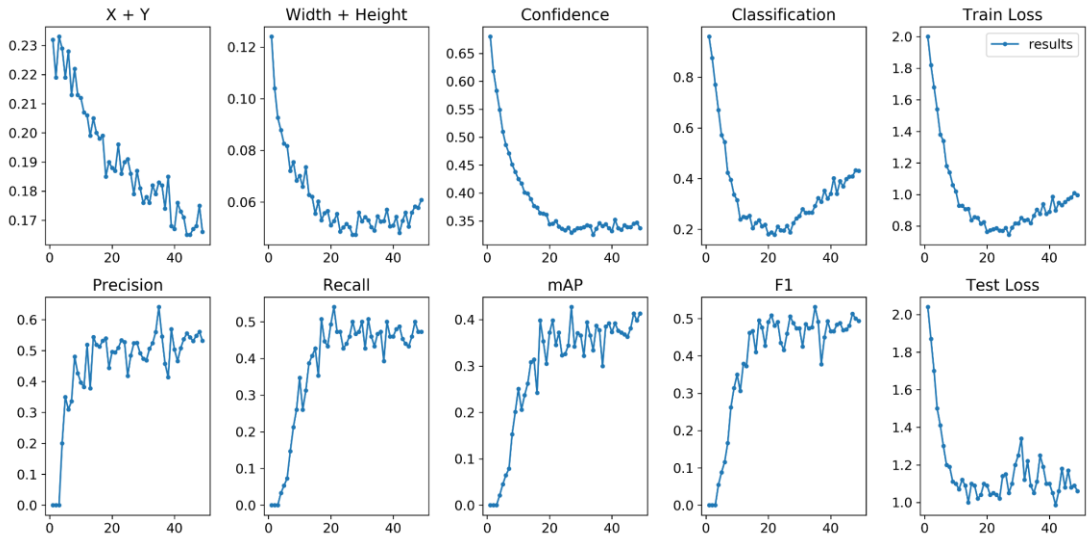
4.3.2 实验结果

1）参数选择

batch=8
width=416
height=416
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

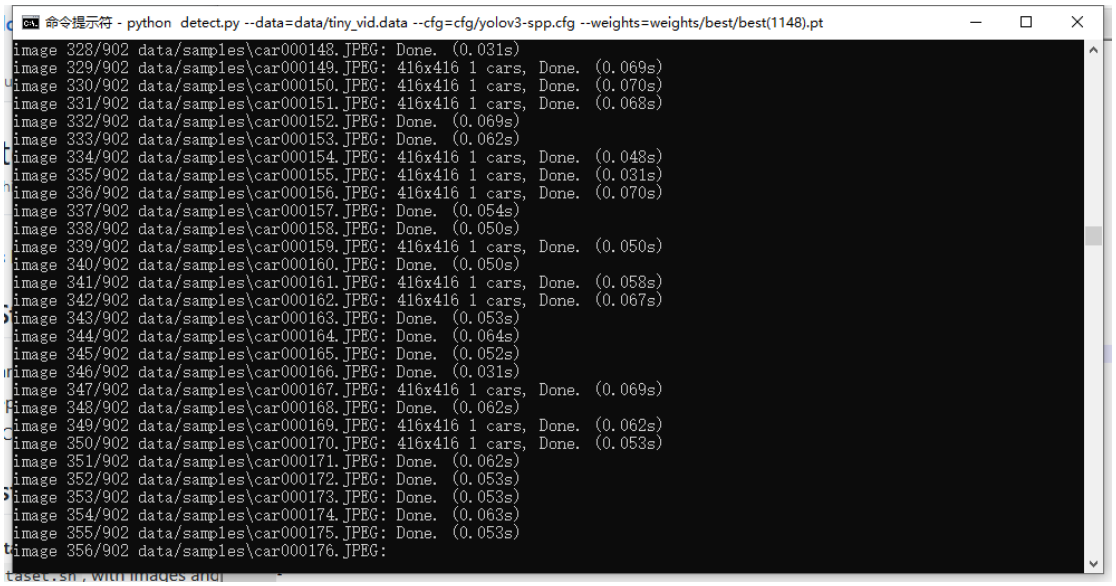
learning_rate=0.001
burn_in=1000
max_batches = 500200
policy=steps
steps=400000,450000
scales=.1,.1

2）Results

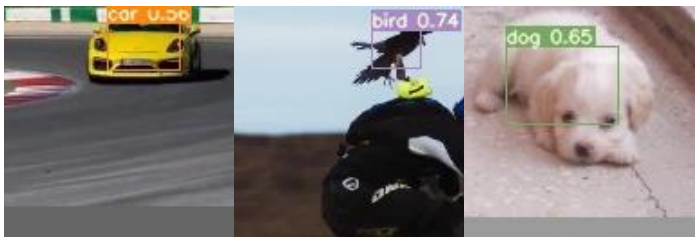


3）验证

detect 过程如下所示，原数据集上成功正确识别并定位对象的有 40%左右。



识别定位画框示例如下：



4）结果讨论

实验过程中，使用了多种参数组合，最终得到的最优结果就是上面的测试结果。可以发现结果并不理想，在测试集上验证只有不到一半的图片可以准确定位，很大一部分数据无法识别出对象，经过学习和思考，我认为可能的原因如下:

- 数据集太小，每一类别只有 150 张用来训练，数据集较小，泛化特性不足，比如鸟有多种种类（目前世界上约有 9000 钟），多种颜色，150 张图无法穷举种类更不用说同种鸟类中不同的个体差异。所以，数据集极大的限制了最终模型的结果。
- 图片质量较差，细节丢失严重，经过浏览可以发现数据集上存在很多人眼无法直观观

测出结果的图片，比如扬尘遮盖的车，全图不到 1/5 的篇幅是车体，我认为这种数据的存在对于特定场景需要的结果有负面影响，比如我需要看一棵树的样子，但是我学的确是树枝或者树叶的样子。

- 网络模型较为复杂，网络学习了太多针对于这份小数据集的特征的，丢失了一定的泛化能力，如前面所说，数据集质量较低，存在很多人眼无法观测的数据，并且又无法穷举对应类别下的对象，导致网络最终训练结果局限在当前数据集的特征无法接受差异较大的同类别物体，比如说假设数据集里面的鸟都是黑色的，那么我们拿这个数据集训练得到的模型取判决蓝色的鸟，大概率不会得到正确的结果。