華中科技大學

电子信息与通信学院
《数字图像处理课设》
目标检测课程设计论文

小组成员 刘静雯、李泽霖、谢怡荣

班　　级 种子1601班

时　　间 2019年5月19日

目录

基于 YOLOv3 的目标检测研究

1. 实验目的

1.1 实验要求

本次实验我们将基于 YOLO 目标检测理论对 tiny_vid 数据集进行分类和检测，我们采用的是 YOLO 的最近更新版本 YOLOv3 版本，模型基于 pytorch 实现。

选择 YOLO 的主要原因是我们在调查的过程中发现，YOLO 具有速度快，精度高的特点，且随着其版本的更新，其综合性能不断提升，尤其是最近的 YOLOv3 版本，从作者论文中的实验结果来看，该模型效果已经非常不错。于是我们选择了 YOLOv3，我们希望能够复现论文中展示的其在目标检测上的优良效果，并且能够通过我们的研究，进一步提升模型效果。

具体过程：

1. 使用 CNN 网络构建物体边缘检测器
   (1) 深入理解神经网络。
   (2) 开始动手编写和了解卷积神经网络。
   (3) 利用卷积神经网络和图像特征识别二维物体边缘。
   (4) 深入了解残差块，上采样，边界框回归，非最大值抑制等算法。
2. 使用新的算法和框架改进实验结果
   （1） 熟悉使用 pytorch 构建深度工程
   （2） 了解和尝试 yolo.v1-yolo.v3
   （3） 比较 yolo 算法和 ssd 算法的优点缺点

1.2 实验环境

本次实验的环境为：python >= 3.6，pytorch >= 1

1.3 实验准备

1. 学习目标检测的基础原理，CNN

共同阅读了 YOLO、YOLO9000、YOLOV3 的论文，学习 YOLO 相关

2. 阅读 pytorch 中文文档，了解关于使用框架进行自动求导。了解 nn.Module, nn.Sequential 和 torch.nn.prameter 类，并使用他们进行前向传播求导和后向传播更新梯度。

3.观看网易云课堂上的 CS231n，CNN 课程，深入了解池化，卷积等各种概念。

2. 实验原理

2.1 YOLO 目标检测原理

进行 yolo 图像检测的时候，首先需要了解如下概念：

（1）残差块和 Skip Connection：

Residual Network 通过引入 Skip Connection 到 CNN 网络结构中，使得网络深度达到了千层的规模，并且其对于 CNN 的性能有明显的提升。

我们知道，在计算机视觉里，特征的“等级”随增网络深度的加深而变高，研究表明，网络的深度是实现好的效果的重要因素。然而梯度弥散（后向传播时候，神经网络层数越多，数量级下降的越快，导致浅层网络的权重参数得不到很好的训练）/爆炸成为训练深层次的网络的障碍，导致无法收敛。

关于梯度弥散和梯度爆炸：

一个激活函数的偏导数在链式求导中累乘导致越来越小，第二个就是极小梯度在传递链上的连续出现。这两个现象共同引起了所谓的梯度消失。实际上，如果出现连续的极大梯度，则梯度爆炸就发生了。

有一些方法可以弥补，如归一初始化，各层输入归一化，使得可以收敛的网络的深度提升为原来的十倍。然而，虽然收敛了，但网络却开始退化了，即增加网络层数却导致更大的误差。

但是单纯使用恒等映射函数 y=x，非线性网络无法在深层网络逼近恒等映射。所以我们必须使用残差块，定义新的映射函数，使得目标逼近恒等映射 y=x 而不是恒等映射 y=0。

通过加入残差块结构，可以解决深层网络的退化问题，残差块结构如下：

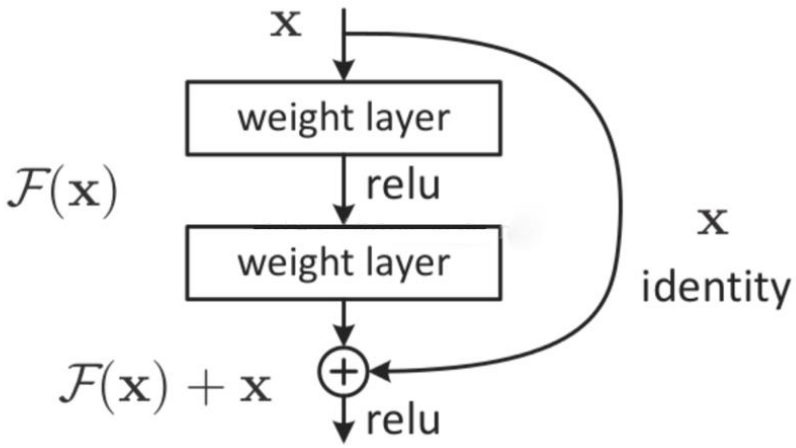


图 2.1 残差结构示意图

上述结构中，经过试验，这个残差块往往需要两层以上，单单一层的残差块(y=W1x+x)并不能起到提升作用。

（2）上采样：

最相近补充：Nearest Neighbor。即进行上采样的时候，对框内的补充是直接扩充，取得值就是原始值，复制补充。

相近补零： Bed of Nails。顾名思义，就是扩充之后相近补零。

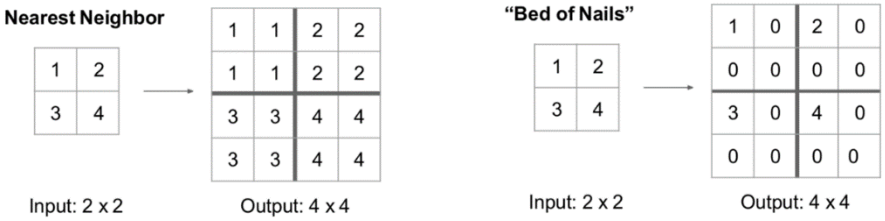


图 2.2 上采样不同方案示意图

最大值池化：取最大值池化，在卷积网络中，在池化的时候去池化方框中的最大值，这样就能达到提取特征的目的。

最大值上池化：Max Upooling 是 Max pooling 之后恢复图像。

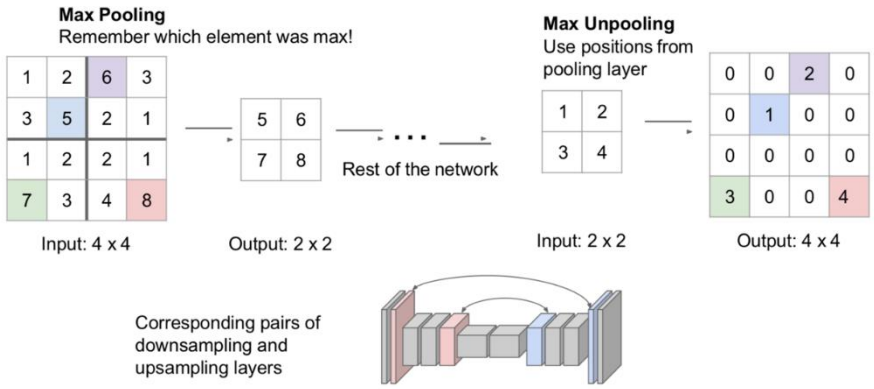


图 2.3 池化过程示意图

（3）IOU（交并比）：

目标检测中使用的一个概念，是产生的候选框（candidate bound）与原标记框（ground truth bound）的交叠率，即它们的交集与并集的比值。最理想情况是完全重叠，即比值为 1。

（4）边界框回归：

对于图 2，红色的框 P 代表原始的 Proposal，绿色的框 G 代表目标的 Ground Truth，我们的目标是寻找一种关系使得输入原始的窗口 P 经过映射得到一个跟真实窗口 G 更接近的回归窗口 G_。

当输入的 Proposal 与 Ground Truth 相差较小时(RCNN 设置的是 IoU>0.6)，可以认为这种变换是一种线性变换，那么我们就可以用线性回归来建模对窗口进行微调，否则会导致训练的回归模型不 work（当 Proposal 跟 GT 离得较远，就是复杂的非线性问题了，此时用线性回归建模显然不合理）。

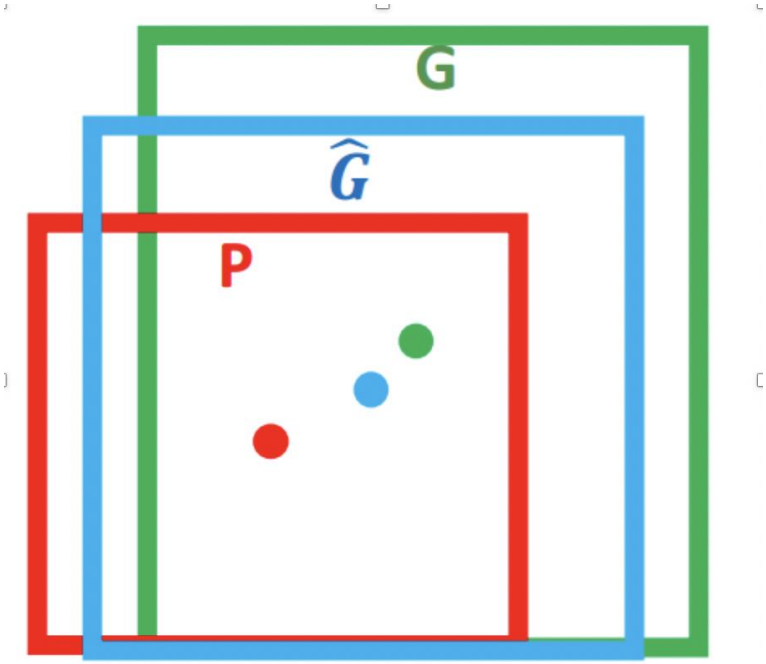


图 2.4 边界框回归的过程

线性回归就是给定输入的特征向量 X，学习一组参数 W，使得经过线性回归后的值跟真实值 Y(Ground Truth)非常接近. 即 Y≈WX。就是让结果边框能自动的进行适应（经过线性回归），平移和大小变换之后就能在数据集基础上更加接近真实的边框。

（5）非极大值抑制(NON-MAX suppression algorithm)

目前为止，目标检测中的一个问题是，你的算法可能对同一个对象作出多次检测，非极大值抑制这个方法可以确保你的算法对每个对象只检测一次。

举个例子，假如你需要在这张图片里检测行人和汽车，你可能会在上面放 19x19 网格，理论上这辆车只有一个中点，所以它应该只被分配到一个格子里，而实践中当你跑目标分类和定位算法时，对每个格子都跑一次，可能会有多个格子认为对象的中心点在其自己的格子内。

因为你要在 361 格子上都跑一次，图像检测和定位算法，那么可能很多格子都会说我这个格子里有车的概率很高，所以当你跑算法的时候，最后可能会对同一个对象做出多次检测，如下图所示。因此非极大值抑制做的就是清理这些检测结果，这样一辆车只检测一次，而不是每辆车都出发多次检测。

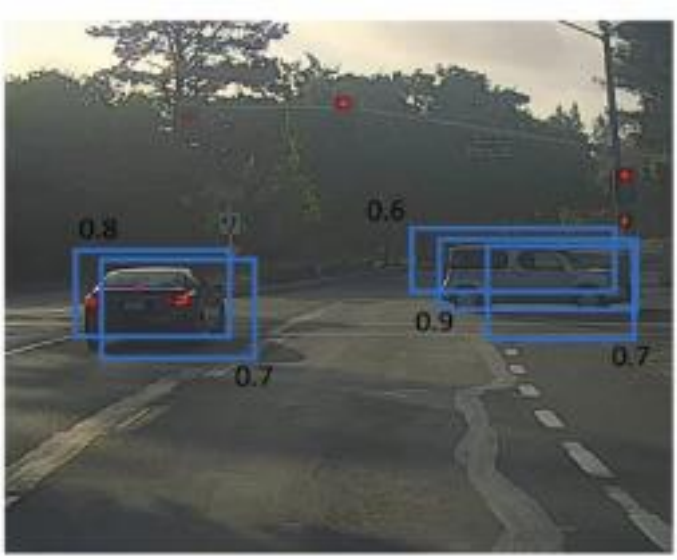


图 2.5 最大值抑制图片解释

在交并比高（重叠度高）的框中，对比相关概率 Pc(置信度)，选取置信度最高的框图。这样非极大值抑制算法就会去掉其他 IoU 值很高的矩形。最后剩下的矩形框就是最终结果。

（6）锚框（Anchor Box）

预测边界框的宽度和高度可能是有意义的，但实际上，这会导致训练期间的梯度变得不稳定。因此，现在大多数目标检测器预测对数空间变换，或简单地预测与预定义的默认边界框（称作锚）之间的偏移。

然后，对锚框进行这些变换以获得预测。 YOLO v3 有三个锚，可以为每个单元预测三个边界框。负责检测狗的边界框将是这样的边界框——它的锚与真实标签框有最高的 IoU 值。

网络的输出是通过中心锚框来描述的。而结果是通过 sigmoid 函数来预测中心坐标，它迫使输出的值压缩在 0 和 1 之间。这是为了使得我们预测目标中心能必须保持在当前框格中，而不是位于红色单元旁。

（7）激活函数的设置

在边缘检测的深度学习网络构建中，不同于分类模型。我们需要各个类别之间不是互斥的，而是都有可能性那种。

不再使用 softmax，使用 sigmoid，因为 softmax 会意味着类别之间是相互排斥的。

2.2 YOLO 到 YOLOv3

2.2.1 YOLO.v1

将一幅图像分成 SxS 个网格（grid cell），如果某个 object 的中心落在这个网格中，则这个网格就负责预测这个 object。

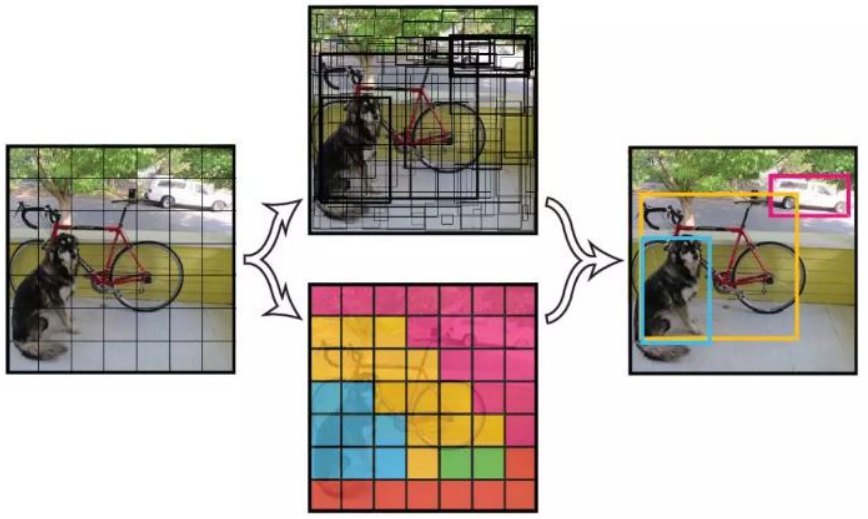


图 2.6 yolo1 基本思想

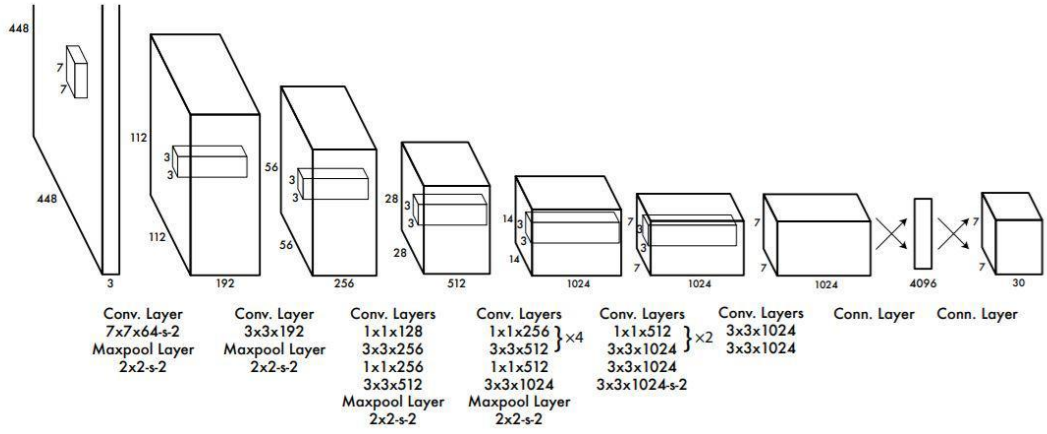


图 2.7 YOLOv1 结构图

每个网格要预测 B 个 bounding box，每个 bounding box 除了要回归自身的位置之外，还要附带预测一个 confidence 值,代表了所预测的 box 中含有 object 的置信度和这个 box 预测的有多准。

每个 bounding box 要预测 (x, y, w, h) 和 confidence 共 5 个值，每个网格还要预测一个类别信息，记为 C 类。则 SxS 个 网格，每个网格要预测 B 个 bounding box 还要预测 C 个 categories。输出就是 S x S x (5*B+C) 的一个 tensor。

得到每个 box 的分类置信度得分以后，设置阈值，滤掉得分低的 boxes，对保留的 boxes 进行 NMS（非极大值抑制） 处理，就得到最终的检测结果。

相对于 Fast R-CNN 没有候选区的步骤，直接回归完成位置的检测和类别的判定。

2.2.2 YOLO.v2

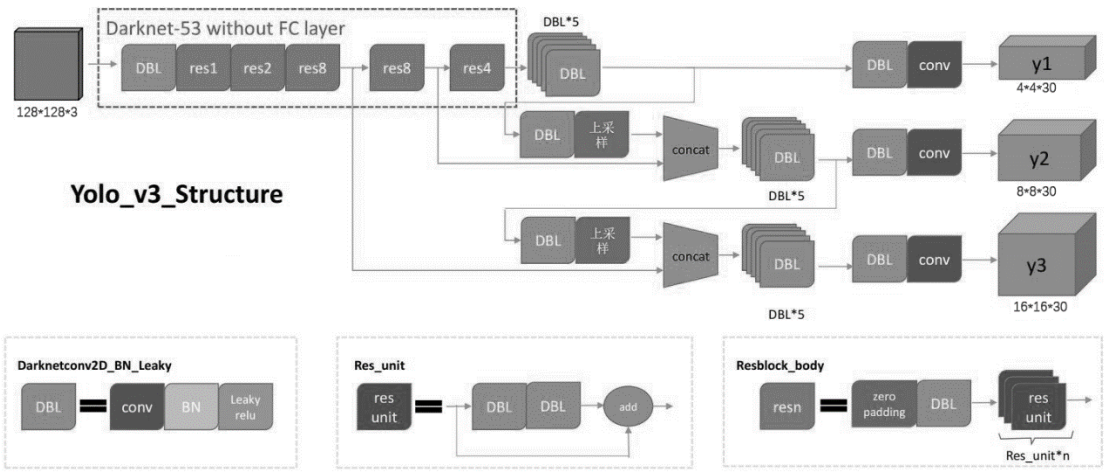
YOLOv2 是在 YOLOv1 的基础上的优化，速度优于 Fast R-CNN、ResNet、SSD，

用 batch normalization 作为正则化，使得模型的收敛速度显著提升，不必使用其他形式的正则化，避免过拟合。尝试加入 anchor 机制，通过全连接层直接预测 Bounding Box 的坐标值。

增加跳层，使用多种分辨率的图片进行训练

YOLOv2 使用 darknet-19 作为 backbone 网络，联合训练结合 wordtree 等，使得 YOLOv2 的检测种类得到扩充

2.2.3 YOLO.v3



相对于 YOLOv2 在 YOLOv3 中没有池化层和全连接层，YOLOv2 中的下采样使用最大池化进行，而 YOLOv3 使用增大卷积核的步长实现。YOLOv3 使用 darknet-53 作为 backbone 网络，和 v2 一样会将输出特征图缩小到输入的 1/32。

相对于前两个 YOLO 的另一个改进是 YOLOv3 输出 3 个不同维度的 feature map，采用多维度来对不同大小的目标来进行检测，越精细的 grid cell 产生 anchor box 将更更有利于检测小目标物体。BOXES 产生 **x_center,y_center,width,hight ,confidence** 五个基本参数，每个 grid cell 有 3 个 anchor box,判断 5 个分类则 3*（5+5）=30。

我们也有可能在其他地方看到一个通用公式：G*G*B*（5+C）。其中 G 是维度，分隔的网格数（神经元维度），B 是每一个单元格中的基本框，锚框就是根据 B 来产生偏差的。

2.3 模型优化

3. 实验过程

3.1 模型实现

神经网络结构：

将特征图分割成 19x19 个方格，网络步幅是 2。特征图中的输出是 19x19x(Bx(5+C))。其中 B 是每个方格可以预测的边框的数量，5 是 4 个位置变量和 1 个是否存在的变量，C 是需要预测的种类。

而在 YOLO.v3 中，我们使用多个卷积神经网络进行训练， 从下面的框图中可以看出，每个图片经过一开始的（共同点）残差块处理之后，后面会经过不同的卷积网络，并输出不同维度的结果。可以看出这些用不同维度（不同数量的单元格分隔）的情况，相差还是比较大的。我们接下来也会对 YOLO.V3 中的这三层网络进行分析。

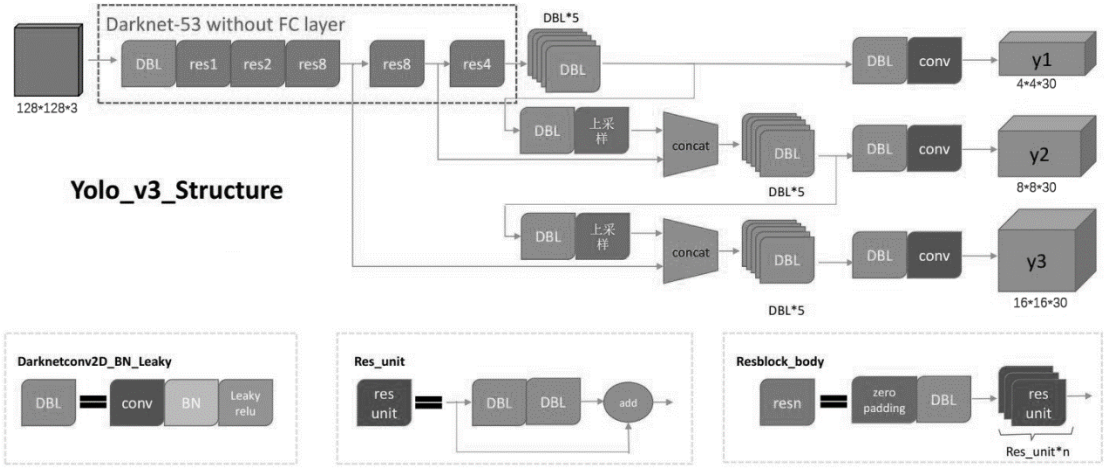


图 3.1 yolo 网络框架

在 YOLO.v3 中的 DARKNET 53 中的配置，我们配置文件如下：

Layer0:

```
[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
```

Layer 1:

```
[yolo]
mask = 3,4,5
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
```

Layer2：

```
[yolo]
mask = 0,1,2
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
```

在不同维度上进行预测：

YOLO v3 可以进行 3 种不同维度的预测。检测层在分别具有步幅 32,16,8 的三种不同尺寸的特征图上进行检测。这意味着，在输入 128*128 的情况下，我们在尺寸为 13 x 13,26 x 26 和 52 x 52 上进行检测

网络对输入图像进行下采样直到第一个检测层，它使用具有步幅为 32 的层的特征图进行检测。然后，将层上采样 2 倍，并与具有相同特征图尺寸的前一层的特征图连接。现在在具有步幅 16 的层上进行另一次检测，重复相同的上采样过程。并且在步幅 8 的层处进行最终检测。

在每个维度上，每个单元使用 3 个锚来预测 3 个边界框，锚的总数为 9（在不同维度上的锚是不同的）。

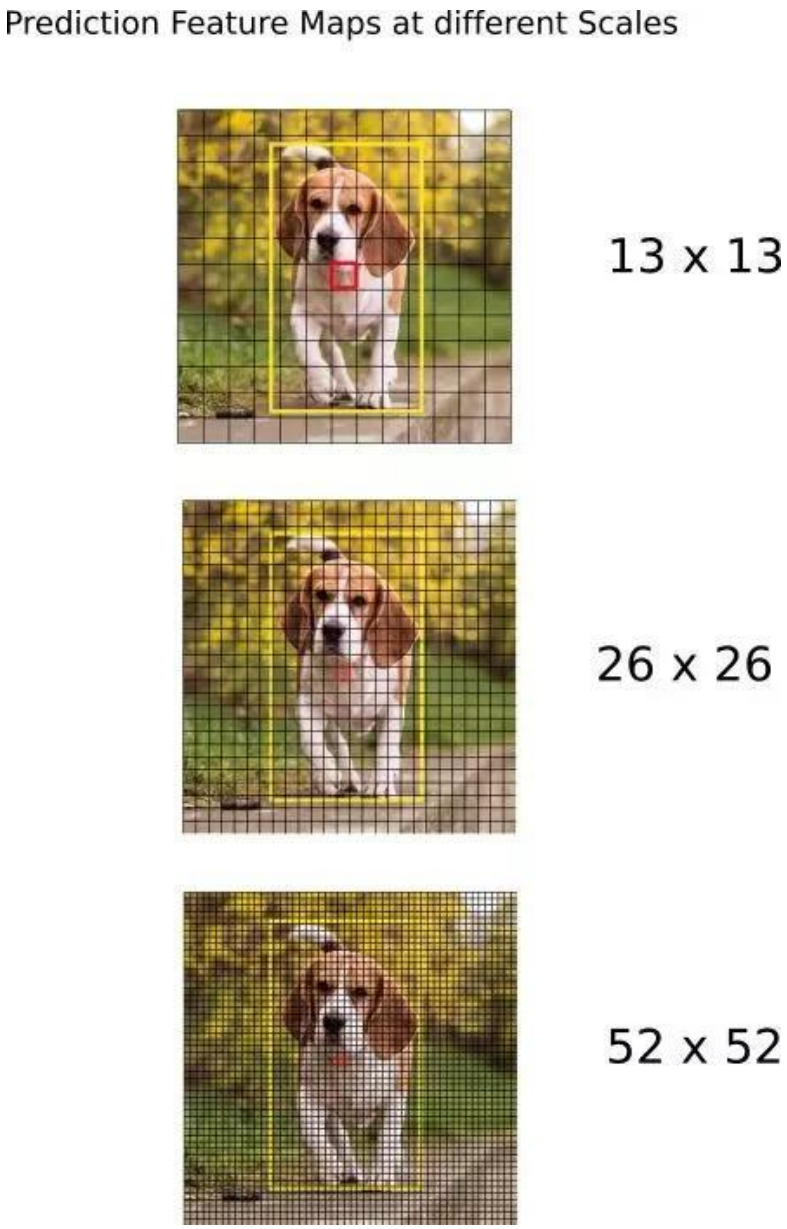


图 3.2 不同维度的分隔和预测

单个神经元卷积处理：

接下来，我们从前面的 2 个层中取得特征图，并将其上采样 2 倍。我们还从网络中的较前的层中获取特征图，并使用按元素相加的方式将其与我们的上采样特征图进行合并。这种方法使我们能够从上采样的特征图中获得更有意义的语义信息，同时可以从更前的层中获取更细粒度的信息。然后，我们再添加几个卷积层来处理这个组合的特征图，并最终预测出一个类似的张量，虽然其尺寸是之前的两倍。

我们再次使用相同的设计来预测最终尺寸的边界框。因此，第三个尺寸的预测将既能从所有先前的计算中减少计算，又能从网络前面的层中的深层次的特征中获益。

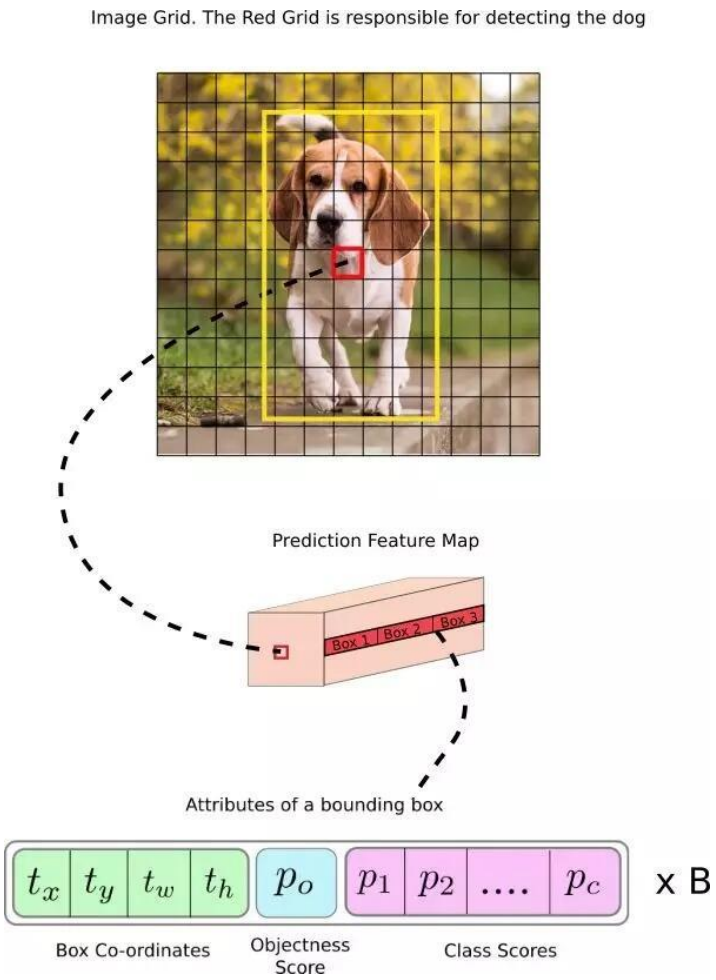


图 3.3 单个神经元特征提取

对于尺寸为 128×128 的图像，YOLO 预测（（52×52）+（26×26）+ 13×13））×3 = 10647 个边界框。 但是，在我们的图像中，只有一个物体——一只狗。 我们如何将检测结果从 10647 减少到 1？

输出处理，进行非最大值抑制和目标置信度的阈值处理：

NMS 的作用是解决同一图像的的多重检测问题。 例如，红色网格单元的 3 个边界框可能检测到同一个框，或者相邻单元可能检测到相同的目标。

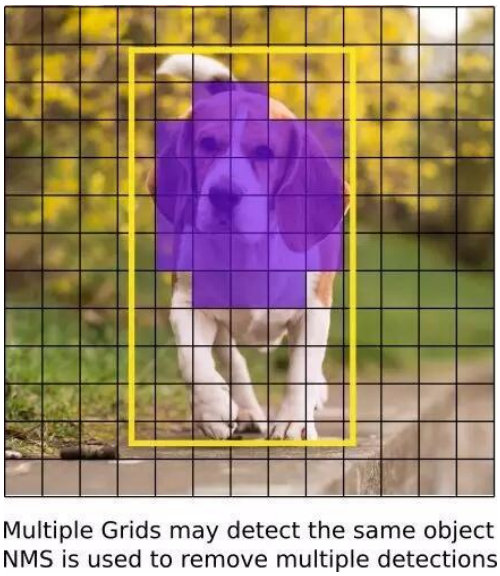


图 3.4 利用非极大值抑制去除多余的框

3.2 数据预处理

针对模型需要的数据格式进行预处理,将 label 数据文本的"image_index, xmin, ymin, xmax, ymax”格式转换为 “class_index,x_center,ycenter,width,hight”格式，并生成对应的训练集和测试集图片索引路径文件。

部分代码如下：

```
def data_proc(label, data_pre):
    img_size = 128
    line=0
    for line in  range(0,data_pre.shape[0]) :
        x_center = (data_pre[line][1] + data_pre[line][3])/2
        y_center = (data_pre[line][2] + data_pre[line][4])/2
        length = data_pre[line][3] - data_pre[line][1]
        hight = data_pre[line][4] - data_pre[line][2]
#         print(x_center,y_center,length,hight)
        data_pre[line][0] = label
        data_pre[line][1] = x_center / img_size
        data_pre[line][2] = y_center / img_size
        data_pre[line][3] = length / img_size
        data_pre[line][4] = hight / img_size

        if line < 150 :
            head="train_"
        elif line<180 :
            head="valid_"

        if label == 0:
            label_na="bird_"
        elif label == 1 :
            label_na="car_"
        elif label == 2:
            label_na="dog_"
        elif label == 3:
            label_na="lizard_"
        elif label == 4:
            label_na="turtle_"

        tem='%d' %(line+1)
        s = tem.zfill(6)
        name = head+label_na+s+".txt"
        data_save = data_pre[line].reshape(1,-1)
        np.savetxt(name,data_save, fmt="%f")
```

图 3.5 把数据读取格式转换

4. 结果分析

4.1 问题与解决

首先是数据遇到的问题，github 上大多数代码是基于 coco 数据集所做的实现，而 tiny_vid 数据集无论是规模上，还是数据标注都与 coco 不同，一方面我们需要考虑如何使 yolo 在较小数据集和较小图片上表现不错，一方面我们需要先将数据集转化为相同的格式。

然后是代码本身的问题，完成数据转换之后，我们发现代码依旧无妨成功运行，研究发现，在我们的参考代码里面，默认的图片格式没有 JPEG 格式图片，我们于是进行了修改。之后又遇到代码 pytorch 的 CUDA 调用部分存在问题，多次尝试依旧无法修复，最后我们决定这部分代码重新实现。

在代码原作者的训练代码部分，为了给出相应输出，调用了一部分 test 的代码，经过多次调试，最终发现这部分代码的张量处理代码存在越界隐患且未做处理，于是我们

在训练部分启用了这部分代码，并对可能存在错误的代码进行了修改。

经过多方参考，我们终于把代码跑通。

但依旧存在准确率较低的问题，基础准确率只有 0.20 左右。我们认为可能有一下两个原因：

1. 数据集太小。

2. 官方给出的预训练权重是基于 80 个 coco 数据集的类别，其中并不包含我们的目标 turtle 和 lizard，对模型的训练造成的一定影响。

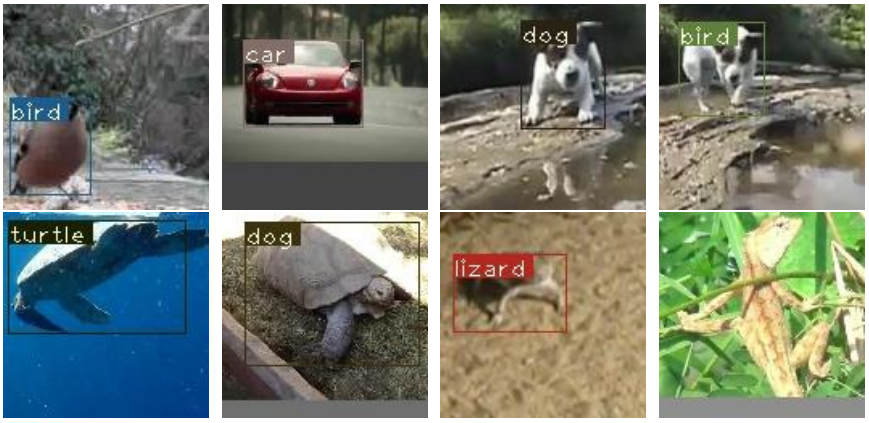
3. 图片较小，anchor box 的设置上需要不断尝试。

4. Iou 阈值的选取，不合适的选择会导致偏差较大。

经过调试，最终能够将准确率提高到 0.375—0.5。

除此以外还有目标位置检测的问题，一开始我们用预训练的权重完全无法识别正确的物体位置，分类准确率也很低。经过调试，分类的准确率有所提升，但是依旧无法找到正确的目标位置。

最终，我们重新实现了 detect 网络。并能够一定程度正确识别目标，但仍旧会存在一些错误，其在 lizard 和 turtle 数据集上表现很差，在 bird、car 和 dog 上的表现较好，检测效果优于分类效果，一下是各分类效果的简单展示：



4.2 方案与尝试

我们尝试通过修改 yolo 的网络设计来适应我们自己的数据集：

1. 修改 anchor box 的尺寸，策略是将原本的 anchor 尺寸修改为四分之一，鉴于当前数据集的图片的长宽是原本的四分之一，但保持原本的分布，实验效果无差异；

2. 尝试对 turtle 和 lizard 进行专门的训练，并将得到的权重作为预训练权重，但导致了整体准确率的下降。

5. 实验总结

5.1 未完成的工作

1. 没有将 YOLOV1-YOLO9000-YOLOV3 三个检测器都尝试一遍。

2. 实验的最终结果不够理想，但没有时间进行更多尝试。

5.2 心得与体会

机器学习是最近几年被打得火热的研究学习领域,我们现在能够看到和学习的知识其实都是基于前人无数的研究的积累和经验。

对于我们原来不是电信学院的同学来说，若不是来到种子班，恐怕很难能这样接触学习和实践机器学习的课程，结合团队"干中学"宗旨在其中学习到了许多。

目标检测是一个有难度也有深度的领域,在许多研究者已经取得了不错的研究成果的基础上想要进行更多的改进是有一定难度的。需要我们具有较为充分的目标分类和检测基础，且对基础的卷积神经网络有一定了解。

在一开始了解卷积神经网络的时候，我们通过观看 cs231n 在网易云课堂上的慕课，知道了 CNN 在很多方面都有使用。在图像识别这种特征提取的领域是由巨大提升的，同时因为 CNN 有着缩减问题规模，处理大数据集的功能，它在文本分析、视频、语音领域也有着非常广泛的应用。

根据斯坦福的课程思路，结合实践作业，和小伙伴一起完成 project,之前没接触过 python,pytorch 也学会的 python,pytorch 的一些基本用法和思想，使用 KNN,CNN 做图像的分类，搭建基本的网络结构，一次一次地试错，虽然过程会比较艰难，内容也相对抽象，比如一个问题可能抓耳挠腮搞个大半天，一个模型调个好几天才能跑，跑出来结果还不一定理想，人肉调参就更是崩溃，各种各样的参数，玄学调参。对于几个门外汉来说，就比如文本分类，调个大半天各种各样的参数，想着怎么控制，结果呢，都是玄学微调，还不如一个 feature 数来得直接。最后的 detect 其实还是非常有意思的，能够框出物体我觉得就已经非常"智能"了，太神奇了。尽管有的同学说本来对这门课挺期待的，学了之后发现机器学习并不是想象中的那么回事，反而没那么感兴趣了。不过我们觉得，同样一个东西是没有好坏之分的，在真正的体会之后只有合适之分，机器学习是朝阳产业，今后的几十年更会有大发展，但更适合数学理论扎实，概率模型优化的研究型同学学习。

我们在总结的时候认为选择 YOLOv3 也许不是一个好的选择，尽管从广泛的评价而言，其在目标检测上整体效果是不错的，但是对于我们的小数据集而言，YOLOv3 的网络过于庞大，且对于机器学习处于入门水平的学习者而言，debug 的成本会更高，也失去了一步步实现一个基础 CNN 网络的实现体验。

但是，我们仍旧收获了许多，无论是卷积神经网络的基础知识和应用，还是各种网络相结合的复杂设计，包括 pytorch 这个十分流行的机器学习框架的使用。以及从一个机器学习工程师的角度去看待数据和实验结果，并做较为深入的理解和分析。与此同时，我们也在共同努力实现代码的过程中体会到了合作开发的苦与乐，收获了共同的成长。