[基于YOLOv2算法的深度学习目标识别 3](#_Toc482732370)

[摘要 3](#_Toc482732371)

[Abstract 3](#_Toc482732372)

[介绍 4](#_Toc482732373)

[YOLOv2使用卷积层产生候选框 5](#_Toc482732374)

[YOLOv2维度聚类 6](#_Toc482732375)

[YOLO更快速 6](#_Toc482732376)

[Darknet-19 7](#_Toc482732377)

[YOLOv2更强大 8](#_Toc482732378)

[层次式分类 9](#_Toc482732379)

[创建层次树： 10](#_Toc482732380)

[使用wordTree聚合数据 11](#_Toc482732381)

[联合训练分类和检测 12](#_Toc482732382)

[改进YOLOv2网络结构： 13](#_Toc482732383)

[卷积神经网络 14](#_Toc482732384)

[卷积层 14](#_Toc482732385)

[池化层 15](#_Toc482732386)

[训练框架CAFFE： 16](#_Toc482732387)

[表达式架构 16](#_Toc482732388)

[可扩展代码 16](#_Toc482732389)

[速度 16](#_Toc482732390)

[训练 17](#_Toc482732391)

[YOLOv2训练 17](#_Toc482732392)

[损失函数设计 17](#_Toc482732393)

[改进网络的训练： 19](#_Toc482732394)

[数据集： 19](#_Toc482732395)

[JPEGImages： 19](#_Toc482732396)

[Annotations： 20](#_Toc482732397)

[ImageSets: 20](#_Toc482732398)

[测试图片： 21](#_Toc482732399)

[测试 21](#_Toc482732400)

[测试代码 21](#_Toc482732401)

[总结： 24](#_Toc482732402)

[参考文献： 24](#_Toc482732403)

[附录A: 25](#_Toc482732404)

基于YOLOv2算法的深度学习目标识别

# 摘要

我们基于YOLOv2算法进行目标识别，YOLOv2是YOLO的改进版。YOLO是一种把目标识别问题转化为回归问题的深度学习方法，有别于rcnn系列目标识别算法，也有别于基于DPM的传统目标识别算法。rcnn系列算法采取proposal+分类的思路，proposal提供位置信息，分类对proposal进行分类。DPM是一种基于提取HOG特征+使用窗口滑动HOG特征+分类窗口 的目标识别算法。YOLO使用卷积神经网络进行图片的特征提取，在输出层回归bounding box的位置和box的类别。YOLOv2在YOLO的基础上，提出了一系列改进的方法，在处理速度不变的情况下，提高了检测的精度。我们主要改进了YOLOv2的网络结构，去掉一些冗余的中间层，虽然精度有些损失，但加快识别的速度，更加满足实时性检测的要求。

# Abstract

We are based on YOLOv2 algorithm for target recognition, YOLOv2 is YOLO improved version. YOLO is a kind of deep learning method which transforms the target recognition problem into the regression problem, which is different from the rcnn series target recognition algorithm and is different from the traditional target recognition algorithm based on DPM. Rcnn series of algorithms to take the idea of proposal + classification, proposal to provide location information, classification of the proposal to classify. DPM is a target recognition algorithm based on the extracted HOG feature + using the window sliding HOG feature + classification window. YOLO uses the convolution neural network to extract the features of the picture, and returns the bounding box position and the box's class at the output layer. On the basis of YOLO, YOLOv2 proposed a series of improved methods to improve the accuracy of the detection in the case of constant processing speed. We mainly improved the network structure of YOLOv2, remove some of the redundant middle layer, although the accuracy of some loss, but to speed up the identification of the speed, more to meet the requirements of real-time detection.

# 介绍

人类只需要看一眼，便可以立马识别出图像中的物体是什么，并且知道他们在哪里。人类的视力系统识别效率如此的高和快速，可以让我们很快识别出复杂的物体。快速，高效的目标检测算法可以让计算机实时识别出视频中各种物体，如车辆，行人，可以作为了机器人的视觉系统。

现在目标识别系统，都是基于分类来做的。为了识别目标，需要使用固定大小的窗口滑动整个图片，使用分类模型判断每个窗口所属类别的自信度，DPM目标识别模型便是依据这样的思路。rcnn系列模型先产生很多后选框，分类每个后选框，最后通过预测的张量调整每个后选框的位置来得出最终的目标位置。

YOLO将目标识别任务作为回归问题来处理，直接根据图像的像素值来预测目标区域的位置以及所属类别的概率。只需要扫描一次图片便可以识别出目标。检测过程如图1

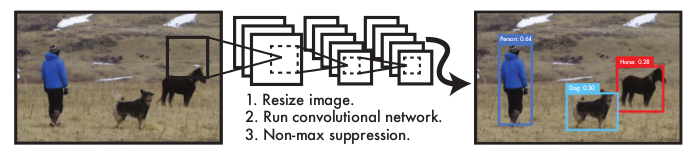


图1： 目标识别过程： 1）将图片调整为448\*448， 2） 使用卷积网络提取特征 3）通过模型得出的自信度来识别目标。

相比较其他目标识别方法，YOLO具有以下优点：

1）检测速度快，可以实时检测目标，在Titan X GPU上可以跑45帧每秒，并且识别效果是其他实时识别的两倍。

2）滑动窗口和列举后选框都是基于局部信息来进行分类，经常会把背景识别为目标，YOLO通过全局图像信息来识别目标，可以减少背景的错误识别率。

3）YOLO泛化能力强，当预测没有训练过的目标时，效果会好些。

YOLOv2在YOLO的基础上，提出了一系列方法改进了，在保持原有的速度的同时，提高了识别精度。主要在一下方面做了改进：

　1）把图片输入分辨率改为416 \* 416，目的是让后面产生的卷积特征图宽高都为奇数，这样就可以产生一个center cell。统计发现大物体通常占据了图像的中间位置，可以只用一个中心的cell来预测这些物体的位置，否则就要用中间的4个cell来进行预测，这个技巧可稍稍提升效率。

2）在每个的卷基层后面添加了batch normalization层，来归一化中间数据，使得输入数据，中间数据分布大致相同。通过添加batch normalization层，极大的加快了收敛速度。

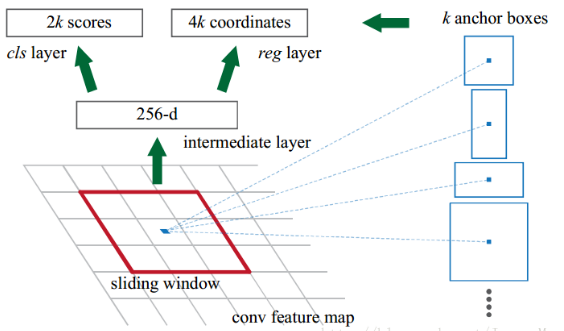
3） 提高了输入图片的分辨率，AlexNet输入图片会被resize到不足256 \* 256，这导致分辨率不够高，给检测带来困难。YOLO使用分辨率为448\*448的图片作为输入。

4）借鉴了Faster R-CNN中的anchor思想，在卷积特征图上进行滑窗采样，每个中心预测5种不同大小和比例的后选框。由于都是卷积不需要reshape，很好的保留的空间信息，最终特征图的每个特征点和原图的每个cell一一对应。而且用预测相对偏移（offset）取代直接预测坐标简化了问题，方便网络学习。

我们改进了网络结构，去除一些冗余的层，主要去掉了几个卷基层。我门在YOLOv2的基础上进行了微调，取得的效果基本和YOLOv2保持一致，但是速度更快，权重模型从200M降到160M，检测速度更快。

# YOLOv2使用卷积层产生候选框

YOLOv1使用全连接层的输出进行bounding box的预测，即要把1470\*1的全链接层reshape为7\*7\*30特征，这样导致丢失了很多空间信息，定位不准确。YOLOv2借鉴了Faster R-CNN中的anchor思想： 简单理解为 在卷积特征图上进行滑窗采样，每个中心预测9种不同大小和比例的候选框。由于卷基层是直接预测，不需要reshape，很好的保留了空间信息，最终特征图的每个特征点和原图的每个cell一一对应。如图2



# YOLOv2维度聚类

YOLOv2借鉴了anchor的思想，Faster-RCNN中anchor boxes的个数和宽高维度是手动精选的先验框。为了能够选择了更好的、更有代表性的boxes维度，YOLOv2使用K-means聚类方法，通过对数据集中的ground true box做聚类，找到ground true box的统计规律。以聚类个数k为anchor boxs个数，以k个聚类中心box的宽高维度为anchor box的维度。聚类的真正想要的是产生好的IOU得分的boxes。因此采用了如下距离度量：

　　　 d(box; centroid) = 1 - IOU(box; centroid)

聚类结果如图3：

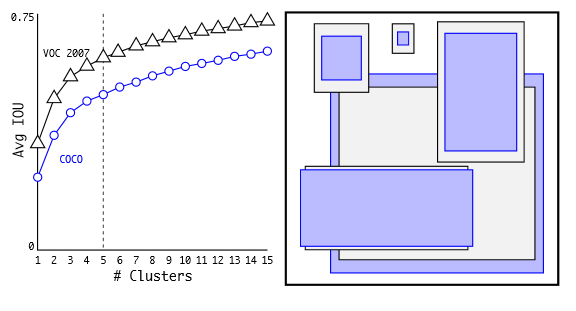


图3： 随着k的增大，IOU也在增大，但是复杂度也在增加。所以平衡复杂度和IOU之后，最终得到k值为5。

# YOLO更快速

大多数检测网络依赖于VGG-16作为特征提取网络，VGG-16是一个强大而准确的分类网络，但是确过于复杂，计算量太大，速度慢。224 \* 224的图片进行一次前向传播，其卷积层就需要多达306.9亿次浮点数运算。

YOLO使用的是基于Googlenet的自定制网络，比VGG-16更快，一次前向传播仅需85.2亿次运算，不过它的精度要略低于VGG-16。224 \* 224图片取 single-crop, top-5 accuracy，YOLO的定制网络得到88%（VGG-16得到90%）。

# Darknet-19

YOLOv2使用了一个新的分类网络作为特征提取部分，参考了前人的工作经验。类似于VGG，网络使用了较多的3 \* 3卷积核，在每一次池化操作后把通道数翻倍。借鉴了network in network的思想，网络使用了全局平均池化（global average pooling）做预测，把1 \* 1的卷积核置于3 \* 3的卷积核之间，用来压缩特征。使用batch normalization稳定模型训练，加速收敛，正则化模型。

最终得出的基础模型就是Darknet-19，包含19个卷积层、5个最大值池化层（max pooling layers ）。Darknet-19处理一张照片需要55.8亿次运算，imagenet的top-1准确率为72.9%，top-5准确率为91.2%。网络结构如图4

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

# YOLOv2更强大

YOLOv2提出了一种在分类数据集和检测数据集上联合训练的机制。使用检测数据集的图片去学习检测相关的信息，例如bounding box 坐标预测，是否包含物体以及属于各个物体的概率。使用仅有类别标签的分类数据集图片去扩展可以检测的种类。

训练过程中把监测数据和分类数据混合在一起。当网络遇到一张属于检测数据集的图片就基于YOLOv2的全部损失函数（包含分类部分和检测部分）做反向传播。当网络遇到一张属于分类数据集的图片就仅基于分类部分的损失函数做反向传播。

这种方法有一些难点需要解决。检测数据集只有常见物体和抽象标签，例如 “狗”，“船”。分类数据集拥有广而深的标签范围（例如ImageNet就有一百多类狗的品种，包括 “Norfolk terrier”, “Yorkshire terrier”, and “Bedlington terrier”等. ）。必须按照某种一致的方式来整合两类标签。

大多数分类的方法采用softmax层，考虑所有可能的种类计算最终的概率分布。但是softmax假设类别之间互不包含，但是整合之后的数据是类别是有包含关系的，例如“Norfolk terrier”和“dog”。所以整合数据集没法使用这种方式（softmax 模型），作者最后采用一种不要求互不包含的多标签模型（multi-label model）来整合数据集。这种方法忽略了数据集的结构（例如 COCO数据集的所有类别之间是互不包含的）

## 层次式分类

ImageNet的标签参考WordNet（一种结构化概念及概念之间关系的语言数据库）。例如：

Canine

Dog

Hunting dog

Terrier

Norfolk terrier Yorkshire terrier

很多分类数据集采用扁平化的标签。而整合数据集则需要结构化标签。

WordNet是一个有向图结构，因为语言是复杂的（例如“dog”既是“canine”又是“domestic animal”），为了简化问题，作者从ImageNet的概念中构建了一个层次树结构来代替图结构方案。

## 创建层次树：

遍历ImageNet的所有视觉名词

对每一个名词，在WordNet上找到从它所在位置到根节点的路径。许多同义词集只有一条路径。所以先把这些路径加入层次树结构。

然后迭代检查剩下的名词，得到路径，逐个加入到层次树。路径选择办法是：如果一个名词有两条路径到根节点，其中一条需要添加3个边到层次树，另一条仅需添加一条边，那么就选择添加边数少的那条路径。

最终结果是一颗 WordTree （视觉名词组成的层次结构模型）。用WordTree执行分类时，预测每个节点的条件概率。例如： 在“terrier”节点会预测：

Pr(Norfolk terrierjterrier)

Pr(Yorkshire terrierjterrier)

Pr(Bedlington terrierjterrier)

如果想求得特定节点的绝对概率，只需要沿着路径做连续乘积。例如 如果想知道一张图片是不是“Norfolk terrier ”需要计算：

Pr(Norfolk terrier) = Pr(Norfolk terrierjterrier)

*∗Pr*(terrier*j*hunting dog)

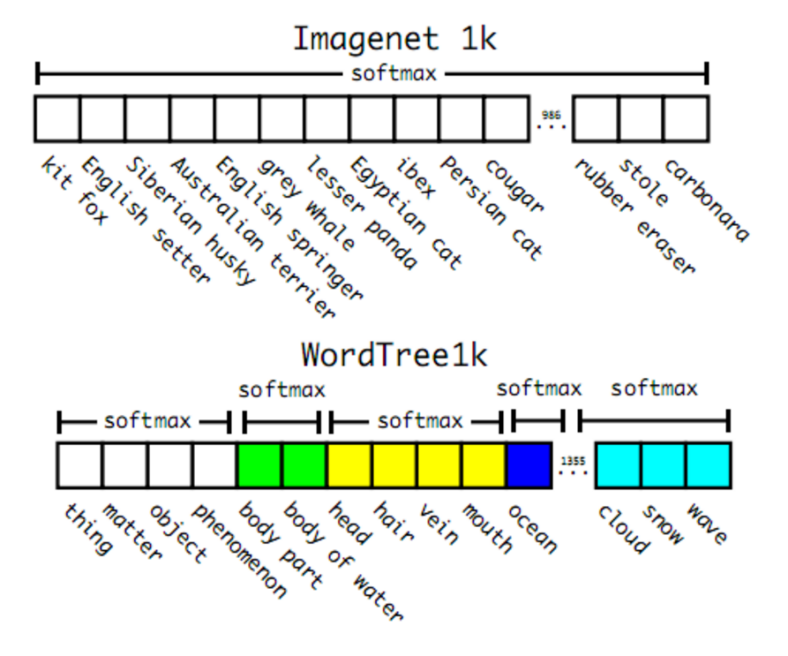
*∗ …..∗*

*∗Pr*(mammal*jPr*(animal)

*∗Pr*(animal*j*physical object)

分类时假设 图片包含物体：Pr(physical object) = 1.

为了验证这种方法作者在WordTree（用1000类别的ImageNet创建）上训练了Darknet-19模型。为了创建WordTree1k作者天添加了很多中间节点，把标签由1000扩展到1369。训练过程中ground truth标签要顺着向根节点的路径传播：例如 如果一张图片被标记为“Norfolk terrier”它也被标记为“dog” 和“mammal”等。为了计算条件概率，模型预测了一个包含1369个元素的向量，而且基于所有“同义词集”计算softmax，其中“同义词集”是同一概念的下位词。

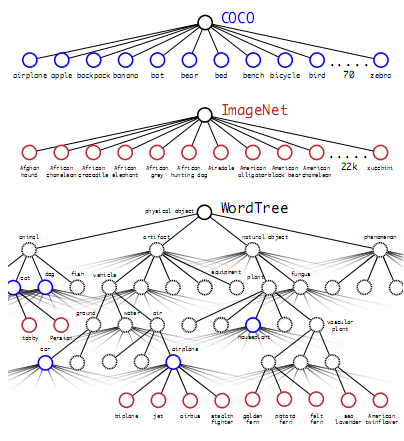


使用相同的训练参数，层次式Darknet-19获得71.9%的top-1精度和90.4%top-5精度。尽管添加了369个额外概念，且让网络去预测树形结构，精度只有略微降低。按照这种方式执行分类有一些好处，当遇到新的或未知物体类别，预测精确度降低的很温和（没有突然巨幅下降）。例如：如果网络看到一张狗的图片，但是不确定狗的类别，网络预测为狗的置信度依然很高，但是，狗的下位词（“xx狗”）的置信度就比较低。

这个策略野同样可用于检测。不在假设每一张图片都包含物体，取而代之使用YOLOv2的物体预测器（objectness predictor）得到Pr(physical object)的值。检测器预测一个bounding box和概率树（WordTree）。沿着根节点向下每次都走置信度最高的分支直到达到某个阈值，最终预测物体的类别为最后的节点类别。

## 使用wordTree聚合数据

可以使用WordTree把多个数据集整合在一起。只需要把数据集中的类别映射到树结构中的同义词集合（synsets）。使用WordTree整合ImageNet和COCO的标签如下图：



## 联合训练分类和检测

使用WordTree整合了数据集之后就可以在数据集（分类-检测数据）上训练联合模型。YOLOv2作者想要训练一个检测类别很大的检测器所以使用COCO检测数据集和全部ImageNet的前9000类创造一个联合数据集。为了评估使用的方法，也从ImageNet detection challenge 中向整合数据集添加一些还没有存在于整合数据集的类别。相应的WordTree有9418个类别。由于ImageNet是一个非常大的数据集，所以通过oversampling COCO数据集来保持平衡，使ImageNet：COCO = 4：1。

使用上面的数据集训练YOLO9000。采用基本YOLOv2的结构，anchor box数量由5调整为3用以限制输出大小。

当网络遇到一张检测图片就正常反向传播。其中对于分类损失只在当前及其路径以上对应的节点类别上进行反向传播。

当网络遇到一张分类图片仅反向传播分类损失。在该类别对应的所有bounding box中找到一个置信度最高的（作为预测坐标），同样只反向传播该类及其路径以上对应节点的类别损失。反向传播objectness损失基于如下假设：预测box与ground truth box的重叠度至少0.31IOU。

采用这种联合训练，YOLO9000从COCO检测数据集中学习如何在图片中寻找物体，从ImageNet数据集中学习更广泛的物体分类。

作者在ImageNet detection task上评估YOLO9000。ImageNet detection task和COCO有44个物体类别是相同的。这意味着YOLO9000只从大多数测试数据集中看到过分类数据而非检测数据。最终整体精度为19.7mAP，在从未见过的156个物体检测数据类别上精度为16.0mAP。这个结果高于DPM，但是YOLO9000是在不同数据集上进行半监督训练。而且YOLO9000可以同时实时检测9000多种其它物体类别。

作者也分析了YOLO9000在ImageNet上的性能，发现可以学习新的动物表现很好，但是学习衣服和设备这类物体则不行。因为从COCO数据集上动物类别那里学习到的物体预测泛化性很好。但是COCO数据集并没有任何衣服类别的标签数据（只有"人"类别），所以YOLO9000很难对“太阳镜”，“游泳裤”这些类别建模。

# 改进YOLOv2网络结构：

我们以Darknet-19网络结构为基础，对它进行了改进，去掉了一些中间层，虽然识别效果有点受影响，但是处理速度更快，更加满足实时性的要求。

改进后的网络结构，包含13个卷积层、5个最大值池化层（max pooling layers ）。改进后的网络模型的权重有160M，Darknet-19网络的模型权重200M，网络结构更小， 处理速度更快，可以更好的满足实时性要求。网络结构如下：

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Filters | Size/Stride | Output |
| Convolutional  Maxpool  Convolutional  Maxpool  Convolutional  Convolutional  Maxpool  Convolutional  Convolutional  Maxpool  Convolutional  Convolutional  Convolutional  Maxpool  Convolutional  Convolutional  Convolutional  Convolutional  Convolutional  Convolutional | 32  64  128  128  256  256  512  512  512  1024  512  1024  1024  1024  1024 | 3 × 3  2 × 2 / 2  3 × 3  2 × 2 / 2  3 × 3  3 × 3  2 × 2 / 2  3 × 3  3 × 3  2 × 2 / 2  3 × 3  1 × 1  3 × 3  2 × 2 / 2  3 × 1  1 × 1  3 × 1  3 × 1  3 × 1  3 × 1 | 416 × 416  208 × 208  208 × 208  104 × 104  104 × 104  104 × 104  52 × 52  52 × 52  52 × 52  26 × 26  26 × 26  26 × 26  26 × 26  13 × 13  13 × 13  13 × 13  13 × 13  13 × 13  13 × 13  13 × 13 |
| RegionLoss |  |  |  |

# 卷积神经网络

卷积神经网络与普通神经网络非常相似：它们由具有学习权重和偏差的神经元组成。 每个神经元接收一些输入，执行点积，并且可选择地在后面加入非线性操作。整个网络仍然表现出单一的可微分评分功能：从一端的原始像素图像的输入到另一段每个类别的自信度。

卷积神经网络是由一系列层组成，卷积神经网络的每一层都通过可微分的函数将一个激活的值转换为另一个。 我们使用三种主要类型的层来构建网络架构：卷积层，池层和完全连接层。目标识别的网络结构便是堆叠这些层，形成一个完整的识别架构。

## 卷积层

卷积层的参数由一组可学习的过滤器组成。每个过滤器在空间上很小（只沿着宽度和高度），但延伸到体积加上深度时，就变大了。例如，一个典型的卷积神经网络，第一层上的滤波器可能具有5×5×3的尺寸（即5像素宽度和高度，3是因为图像具有3个颜色通道rgb）。在前向传播期间，我们将滑动每个滤波器在输入图片上，并计算滤波器和输入图片任何位置之间的点积（更准确地说是卷积）。当我们将滤波器滑过输入图片的宽度和高度时，我们将产生一个二维特征图，给出该滤波器在每个空间位置的响应。直观来说，网络将学习他们看到某种类型的视觉特征（例如第一层上的一些方向的边缘或一些颜色的斑点），最终在网络的较高层上的看到整个蜂窝状或轮状图案特征。现在，我们在每个卷积层中添加一整套滤波器（例如12个滤波器），并且每个滤波器将产生一个单独的二维激活图。我们将沿深度维度堆叠这些激活图，并产生特征图。卷积层具有局部区域感知，权重共享等特征。

综述：

* 接受输入大小W1×H1×D1
* 需要四个超参数：

滤波器数K，

滤波器大小F，

步幅S，

零填充P的量。

* 输出W2×H2×D2的特征图。

## 池化层

在卷积网络体系结构中，通常在连续卷积层之间定期插入一个池化层。 其功能是逐步减少特征图的大小，以减少网络中的参数和计算量，从而也可以控制过拟合。 池化层在输入的每个深度层上独立运行，并使用MAX操作在空间上调整其大小。 最常见的形式是一个大小为2x2的过滤器的池化层，每个深度切片在2个沿宽度和高度两个下采样，每个深度切片采用2个步长，丢弃75％的激活。 在这种情况下，每个MAX操作将占用超过4个数字（某些深度片段中的2×2个区域）。 深度维度保持不变。 更一般来说，池化层：

* 接受输入大小W1×H1×D1
* 需要两个超参数：

滤波器大小F，

步幅S。

* 输出W2×H2×D2大小的特征图

# 训练框架CAFFE：

Caffe，全称Convolutional Architecture for Fast Feature Embedding，是一个深刻的学习框架，由伯克利AI研究（BAIR）和社区贡献者开发，贾杨清在加州大学伯克利分校的博士期间创建了该项目。代替了之前的decaf工具。Caffe是用C++和Python实现的，并提供了C++、Python、Matlab的接口，目前有Linux 和Windows版。它是一个清晰、高效的深度学习框架，它是开源的，核心语言是C++，既可以在CPU上运行也可以在GPU上运行。它的license是BSD 2-Clause。

## 表达式架构

方面应用和创新。模型和优化由配置文件定定义，不需要编码。训练时，在CPU和GPU之间切换，只需要通过设置单个标志即可，然后部署到集群或移动设备，。

## 可扩展代码

促进了Caffe积极的发展。在Caffe的第一年，它已被1,000多名开发商分拆，并有许多重大变化。这些贡献者使得Caffe拥有最先进的模型和技术。

## 速度

使Caffe成为研究实验和行业部署的完美选择。 Caffe可以使用单个NVIDIA K40 GPU 每天处理超过60M图像。这是1 ms /图像的推理和4 ms /图像学习，更新的库版本和硬件更快。

# 训练

## YOLOv2训练

一幅图片送入卷积神经网络，进行特征学习，学习到13\*13的特征图，在特征图上的每个特征点，产生5个不同候选框，候选框负责预测物体的位置。

最后一层输出为 （13\*13）\*5\*25的维度。每个 1\*1\*5的维度表示每个特征点产生5个候选框，1\*1\*5\*25中含有类别预测和bbox坐标预测。总得来讲就是让网络直接学习类别信息，bounding box坐标信息。具体如下：

在特征图上，每个特征点（13\*13）产生5个候选框，基于每个候选框的类别信息和位置信息。要预测这5个候选框的坐标（，，w，h） ，其中：中心坐标的 ， 相对于对应的特征图归一化到0-1之间，w，h也归一化到0-1之间。 每个候选框除了要回归自身的位置之外，还要附带预测一个confidence值。 这个confidence代表了所预测的候选框中含有object的置信度和这个候选框预测的有多准两重信息：confidence = Pr(Object) \ast IOU^{truth}_{pred}。其中如果有ground true box(人工标记的物体)落在一个候选框里，第一项取1，否则取0。 第二项是预测的候选框和实际的ground truth box之间的IOU值。即：每个候选框要预测 x_{center},y_{center},w,h,confidence,共5个值 ，1个候选框共5个值，对应 1\*1\*5\*25维度特征中的前5个值。

每个特征图中的特征点还要预测类别信息，论文中有20类。13\*13\*5的特征点，每个特征点要预测1个 bounding box 和 20个类别概率，输出就是 13\*13\*5\*(5 + 20) 。

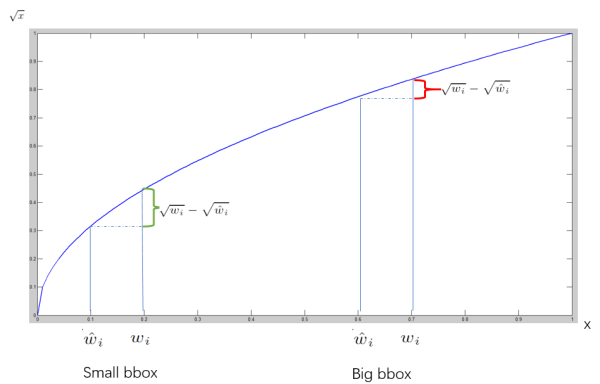
## 损失函数设计

损失函数的设计目标就是让坐标（x,y,w,h），confidence，classification 这个三个方面达到很好的平衡。简单的全部采用了sum-squared error loss来做这件事会有以下不足：

a) 4维的localization error和20维的classification error同等重要显然是不合理的；

b) 如果一个特征点中没有目标（一幅图中这种特征点很多），那么就会将这些特征点中的bbox的confidence 降到0，相比于较少的有目标的特征点，这种做法是overpowering的，这会导致网络不稳定甚至发散。 解决方案如下：

* 更重视4维的坐标预测，给这些损失前面赋予更大的loss weight, 记为 \lambda_{coord} ,在训练中取5。
* 对没有目标的bbox的confidence loss，赋予小的loss weight，记为 \lambda_{noobj} ，在pascal VOC训练中取0.5。
* 有目标的bbox的confidence loss和类别的loss的loss weight正常取1。
* 对不同大小的bbox预测中，相比于大bbox预测偏一点，小bbox预测偏一点更不能忍受。而sum-square error loss中对同样的偏移loss是一样。 为了缓和这个问题，作者用了一个比较取巧的办法，就是将bbox的width和height取平方根代替原本的height和width。 如下图：small bbox的横轴值较小，发生偏移时，反应到y轴上的loss（下图绿色）比big box(下图红色)要大。



* 一个特征点预测多个候选框，在训练时我们希望每个目标（ground true box）只有一个候选框专门负责（一个目标 一个候选框）。具体做法是与ground true box（目标）的IOU最大的候选框 负责该ground true box(目标)的预测。

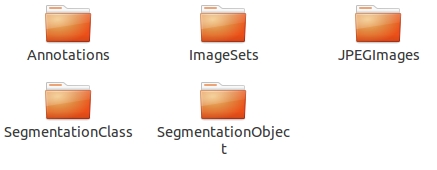
## 改进网络的训练：

我们没有使用YOLOv2那种复杂的训练数据集，我们直接使用Voc2012 voc2007作为训练的数据集。我们直接在voc数据集上，进行目标识别的训练。我们使用YOLOv2训练好的权重模型就行微调。

我们使用Caffe框架作为我们的训练平台。使用多阶段学习率方式进行训练，学习率在100内为0.0001，23000内为0.001，35000内为 0.0001，45000内为0.00001，设置平均损失为25，设置batch\_size为64，并使用GPU来加速训练。

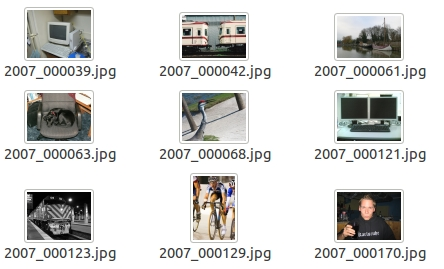
# 数据集：

PASCAL VOC为图像识别和分类提供了一整套标准化的优秀的数据集，从2005年到2012年每年都会举行一场图像识别challenge。我们采用PASCAL VOC2012作为例子进行讲解。下载VOC2012完之后解压，可以在VOCdevkit目录下的VOC2012中看到如下的文件夹：



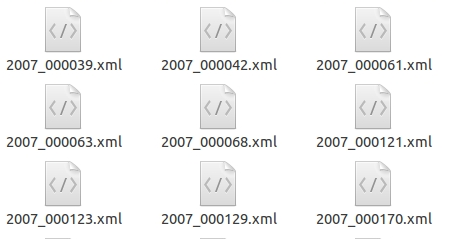
其中在图像物体识别上着重需要了解的是Annotations、ImageSets和JPEGImages。

JPEGImages：JPEGImages文件夹中包含了PASCAL VOC所提供的所有的图片信息，包括了训练图片和测试图片。



这些图像都是以“年份\_编号.jpg”格式命名的。图片的像素尺寸大小不一，但是横向图的尺寸大约在500\*375左右，纵向图的尺寸大约在375\*500左右，基本不会偏差超过100。在之后的训练中，第一步就是将这些图片都resize到416\*416，所有原始图片不能离这个标准过远。这些图像就是用来进行训练和测试验证的图像数据。

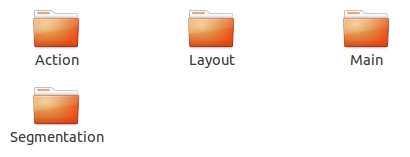
Annotations：Annotations文件夹中存放的是xml格式的标签文件，每一个xml文件都对应于JPEGImages文件夹中的一张图片。



Xml文件会包含 所对应的图片名字，图形尺寸，类别等其他属性信息。

## ImageSets:

ImageSets存放的是每一种类型的challenge对应的图像数据。在ImageSets下有四个文件夹:



其中Action下存放的是人的动作（例如running、jumping等等，这也是VOC challenge的一部分）Layout下存放的是具有人体部位的数据（人的head、hand、feet等等，这也是VOC challenge的一部分）

Main下存放的是图像物体识别的数据，总共分为20类。

Segmentation下存放的是可用于分割的数据。因为与目标识别无关，这里便不再介绍。

# 测试图片：

## 测试

测试的时候，每个候选框预测的class信息( Pr(Class_i | Object) )和候选框预测的confidence信息( Pr(Object) \ast IOU^{truth}_{pred} ) 相乘，就得到每个候选框的真实类别概率。

Pr（）\* Pr(Object) \* = Pr() \*

等式左边第一项就是每个候选框预测的类别信息，第二三项就是每个候选框预测的confidence。这个乘积即encode了预测的候选框属于某一类的概率，也有该候选框准确度的信息。

得到每个候选框的真实类别概率以后，设置阈值，滤掉得分低的候选框，对保留的候选框进行NMS处理，就得到最终的检测结果

## 测试代码

测试代码使用Python语言实现，Python调用Caffe的Python接口，完整可执行代码看附录A。

导入Caffe模块：

**import** **sys**

caffe\_root = '../../'

sys.path.insert(0, caffe\_root + 'python')

**import** **caffe**

设置caffe为cpu模式

caffe.set\_mode\_cpu()

定义模型文件和权重文件:*：*

model\_def = './good/good\_deploy.prototxt'

model\_weights = './good/change\_iter\_45000.caffemodel'

初始化网络：

net = caffe.Net(model\_def,

model\_weights,

caffe.TEST)

定义 均值大小：

mu = np.array([105, 117, 123])

调整输入数据大小：

net.blobs['data'].reshape(1, *# batch size*

3, *# 通道数*

416, 416) *#图像大小*

加载图片:

image = caffe.io.load\_image('darknet\_v3/dog.jpg')

处理图片:

transformed\_image = transformer.preprocess('data', image)

设置输入数据:

net.blobs['data'].data[...] = transformed\_image

执行forward操作，前向网络传播:

output = net.forward()

获取网络输出:

res = output['conv\_reg'][0]

biases = [1.08,1.19,3.42,4.41,6.63,11.38,9.42,5.11,16.62,10.52]

处理:

boxes = list()

**for** h **in** range(13):

**for** w **in** range(13):

**for** n **in** range(5):

box = list();

cls = list();

s = 0;

x = (w + sigmoid(swap[h\*13+w][n][0])) / 13.0; *#center x*

y = (h + sigmoid(swap[h\*13+w][n][1])) / 13.0; *#center y*

ww = (math.exp(swap[h\*13+w][n][2])\*biases[2\*n]) / 13.0; *#w*

hh = (math.exp(swap[h\*13+w][n][3])\*biases[2\*n+1]) / 13.0; *#h*

obj\_score = sigmoid(swap[h\*13+w][n][4]);

**for** p **in** range(20):

cls.append(swap[h\*13+w][n][5+p]);

large = max(cls);

**for** i **in** range(len(cls)):

cls[i] = math.exp(cls[i] - large);

s = sum(cls);

**for** i **in** range(len(cls)):

cls[i] = cls[i] \* 1.0 / s;

box.append(x); *#0*

box.append(y); *#1*

box.append(ww); *#2*

box.append(hh); *#3*

box.append(cls.index(max(cls))+1) *#4*

box.append(obj\_score); *#5*

box.append(max(cls)); *#6*

box.append(obj\_score \* max(cls))

**if** box[5] \* box[6] > 0.8:

boxes.append(box);

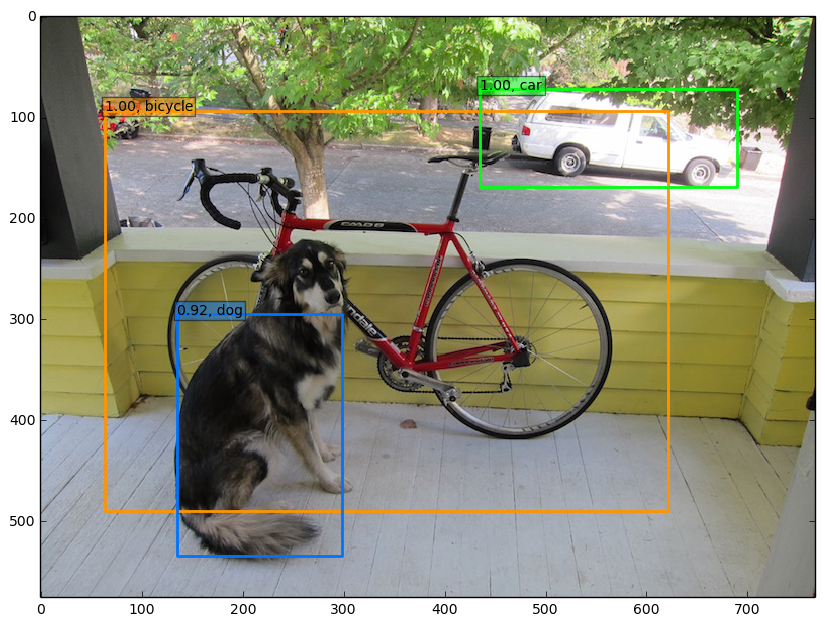
使用nms去重:

res = apply\_nms(boxes, 0.5)

显示图片:

show\_boxes(res)

显示结果:



# 总结：

目标识别从传统的人为组织特征，到现在的通过大量的数据来让机器学习最佳的特征，准确率也在大幅度的提高，特征的标示性，抽象性比人为的组织的特征更精确。目标识别从滑动窗口的分类，到学习候选框的位置然后进行分类，目标识别的精确度越来越高，效果更好。在和训练数据集相似的图片上，目标识别的效果基本可以达到人类的视觉识别的高度，但是相比于人类的视觉系统，泛化能力太弱了，对光线也太敏感了。YOLOv2算法虽然在实时识别方面，可以满足要求，我们通过简化网络结构，将模型的权重参数降到原来的一半，更能满足实时性的要求。

# 参考文献：

YOLO一二

Redmon J, Divvala S, Girshick R, et al. You Only Look Once: Unified, Real-Time Object Detection[J]. 2015:779-788.

Redmon J, Farhadi A. YOLO9000: Better, Faster, Stronger[J]. 2016.

rcnn系列：

Ren S, He K, Girshick R, et al. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks[J]. IEEE Transactions on Pattern Analysis & Machine Intelligence, 2015, PP(99):1-1.

Girshick R. Fast R-CNN[J]. Computer Science, 2015.

Girshick R, Donahue J, Darrell T, et al. Rich feature hierarchies for accurate object detection and semantic segmentation[J]. Computer Science, 2014:580-587.

Dpm

Forsyth D. Object Detection with Discriminatively Trained Part-Based Models[J]. IEEE Transactions on Pattern Analysis & Machine Intelligence, 2010, 32(9):1627-1645.

google net

Szegedy C, Liu W, Jia Y, et al. Going deeper with convolutions[J]. 2014:1-9.

alxnet

Krizhevsky A, Sutskever I, Hinton G E. ImageNet classification with deep convolutional neural networks[C]// International Conference on Neural Information Processing Systems. Curran Associates Inc. 2012:1097-1105.

Voc数据集

Everingham M, Winn J. The PASCAL Visual Object Classes Challenge 2010 (VOC2010) Development Kit Contents[C]// International Conference on Machine Learning Challenges: Evaluating Predictive Uncertainty Visual Object Classification. Springer-Verlag, 2011:117-176.

batch normalization：

Ioffe S, Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift[J]. Computer Science, 2015.

Vgg：

Simonyan K, Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition[J]. Computer Science, 2014.

network in network:

Lin M, Chen Q, Yan S. Network In Network[J]. Computer Science, 2013.

Caffe

Jia, Yangqing, Shelhamer, et al. Caffe: Convolutional Architecture for Fast Feature Embedding[J]. Eprint Arxiv, 2014:675-678.

# 附录A:

*# coding: utf-8*

*#设置Python环境*

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

*# 在notebook上 显示图片*

get\_ipython().magic(u'matplotlib inline')

**import** **matplotlib.patches** **as** **patches**

*# set display defaults*

plt.rcParams['figure.figsize'] = (10, 10) *# large images*

plt.rcParams['image.interpolation'] = 'nearest' *# don't interpolate: show square pixels*

plt.rcParams['image.cmap'] = 'gray' *# 使用灰度图*

*# 添加caffe Python接口路径*

**import** **sys**

caffe\_root = '../../'

sys.path.insert(0, caffe\_root + 'python')

*#导入caffe模块*

**import** **caffe**

**import** **math**

*#定义sigmoid函数*

**def** sigmoid(p):

**return** 1.0 / (1 + math.exp(-p \* 1.0))

*#定义 overlap 函数*

**def** overlap(x1, w1, x2, w2): *#x1 ,x2 are two box center x*

left = max(x1 - w1 / 2.0, x2 - w2 / 2.0)

right = min(x1 + w1 / 2.0, x2 + w2 / 2.0)

**return** right - left

*#定义 iou 函数*

**def** cal\_iou(box, truth):

w = overlap(box[0], box[2], truth[0], truth[2])

h = overlap(box[1], box[3], truth[1], truth[3])

**if** w < 0 **or** h < 0:

**return** 0

inter\_area = w \* h

union\_area = box[2] \* box[3] + truth[2] \* truth[3] - inter\_area

**return** inter\_area \* 1.0 / union\_area

*#定义 nms函数 去重复*

**def** apply\_nms(boxes, thres):

sorted\_boxes = sorted(boxes,key=**lambda** d: d[7])[::-1]

p = dict()

**for** i **in** range(len(sorted\_boxes)):

**if** i **in** p:

**continue**

truth = sorted\_boxes[i]

**for** j **in** range(i+1, len(sorted\_boxes)):

**if** j **in** p:

**continue**

box = sorted\_boxes[j]

iou = cal\_iou(box, truth)

**if** iou >= thres:

p[j] = 1

res = list()

**for** i **in** range(len(sorted\_boxes)):

**if** i **not** **in** p:

res.append(sorted\_boxes[i])

**return** res

*#定义 显示函数*

**def** show\_boxes(boxes):

label\_name = {0: "bg", 1: "aeroplane", 2: "bicycle", 3: "bird", 4: "boat", 5: "bottle", 6: "bus", 7: "car", 8: "cat", 9: "chair", 10: "cow", 11: "diningtable", 12: "dog", 13: "horse", 14: "motorbike", 15: "person", 16: "pottedplant", 17: "sheep", 18: "sofa", 19: "train", 20: "tvmonitor"}

w = image.shape[1]

h = image.shape[0]

plt.imshow(image)

currentAxis = plt.gca()

colors = plt.cm.hsv(np.linspace(0, 1, 21)).tolist()

**for** box **in** boxes:

x\_min = int(round((box[0]-box[2]/2.0) \* w))

x\_max = int(round((box[0]+box[2]/2.0) \* w))

y\_min = int(round((box[1]-box[3]/2.0) \* h))

y\_max = int(round((box[1]+box[3]/2.0) \* h))

**if** x\_min < 0:

x\_min = 0

**if** x\_max > w:

x\_max = w

**if** y\_min < 0:

y\_min = 0

**if** y\_max > h:

y\_max = h

display\_txt = '{:0.2f}, {}'.format(box[7], label\_name[box[4]])

coords = (x\_min, y\_min), x\_max-x\_min+1, y\_max-y\_min+1

color = colors[box[4]]

currentAxis.add\_patch(plt.Rectangle(\*coords, fill=False, edgecolor=color, linewidth=2))

currentAxis.text(x\_min, y\_min, display\_txt, bbox={'facecolor':color, 'alpha':0.5})

plt.show()

*#设置caffe为cpu模式*

caffe.set\_mode\_cpu()

*#定义模型文件和权重文件*

model\_def = './good/good\_deploy.prototxt'

model\_weights = './good/change\_iter\_45000.caffemodel'

*#初始化网络*

net = caffe.Net(model\_def,

model\_weights,

caffe.TEST)

*#定义 均值大小*

mu = np.array([105, 117, 123])

*# 为输入数据做变换 做准备*

transformer = caffe.io.Transformer({'data': net.blobs['data'].data.shape})

transformer.set\_transpose('data', (2,0,1)) *# 改变图像的维度 改为 通道，高度，宽度*

transformer.set\_mean('data', mu) *# 设置均值*

transformer.set\_raw\_scale('data', 255) *# 归一化*

transformer.set\_channel\_swap('data', (2,1,0)) *# 将通道RGB改为BGR*

net.blobs['data'].reshape(1, *# batch size*

3, *# 通道数*

416, 416) *#图像大小*

image = caffe.io.load\_image('darknet\_v3/dog.jpg')*#加载图片*

transformed\_image = transformer.preprocess('data', image) *#处理图片*

net.blobs['data'].data[...] = transformed\_image *#设置输入数据*

*### 执行forward操作*

output = net.forward()

res = output['conv\_reg'][0] *#获取网络输出*

swap = np.zeros((13\*13,5,25))

*#改变数据结构*

index = 0

**for** h **in** range(13):

**for** w **in** range(13):

**for** c **in** range(125):

swap[h\*13+w][c/25][c%25] = res[c][h][w]

biases = [1.08,1.19,3.42,4.41,6.63,11.38,9.42,5.11,16.62,10.52]

*#处理*

boxes = list()

**for** h **in** range(13):

**for** w **in** range(13):

**for** n **in** range(5):

box = list();

cls = list();

s = 0;

x = (w + sigmoid(swap[h\*13+w][n][0])) / 13.0; *#center x*

y = (h + sigmoid(swap[h\*13+w][n][1])) / 13.0; *#center y*

ww = (math.exp(swap[h\*13+w][n][2])\*biases[2\*n]) / 13.0; *#w*

hh = (math.exp(swap[h\*13+w][n][3])\*biases[2\*n+1]) / 13.0; *#h*

obj\_score = sigmoid(swap[h\*13+w][n][4]);

**for** p **in** range(20):

cls.append(swap[h\*13+w][n][5+p]);

large = max(cls);

**for** i **in** range(len(cls)):

cls[i] = math.exp(cls[i] - large);

s = sum(cls);

**for** i **in** range(len(cls)):

cls[i] = cls[i] \* 1.0 / s;

box.append(x); *#0*

box.append(y); *#1*

box.append(ww); *#2*

box.append(hh); *#3*

box.append(cls.index(max(cls))+1) *#4*

box.append(obj\_score); *#5*

box.append(max(cls)); *#6*

box.append(obj\_score \* max(cls))

**if** box[5] \* box[6] > 0.8:

boxes.append(box);

*#使用nms去重*

res = apply\_nms(boxes, 0.5)

*#显示图片*

show\_boxes(res)