深度学习之计算机视觉
基于YOLO算法的物体识别
Deep Learning in the Field of Computer Version
Object Detection Based on YOLO Algorithm

计算机学院

智能161班

贾舒越

20163030130118

Tel: +(86)13844602327

Email: shuyuej@ieee.org

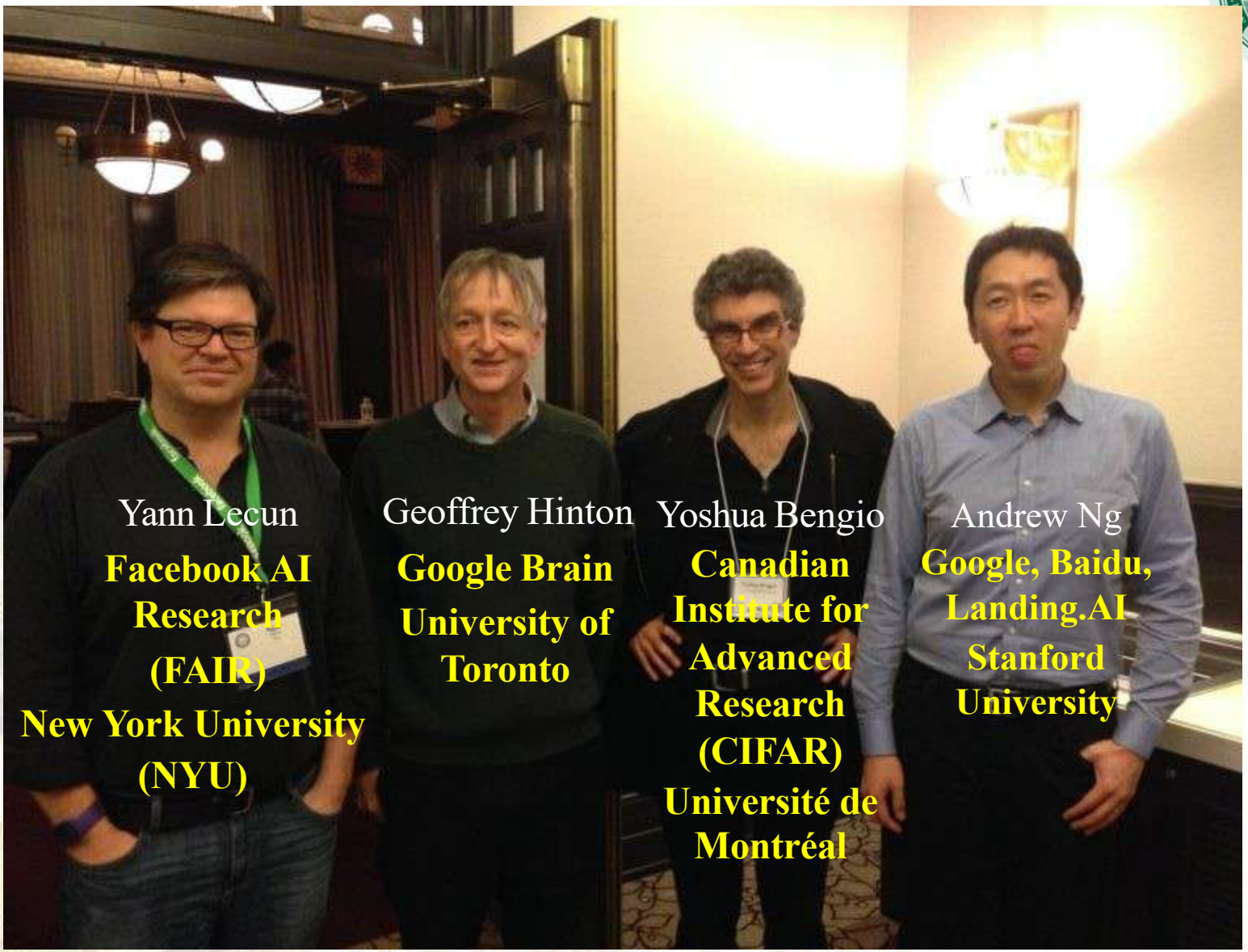
感谢 University of Washington 的 Joseph Redmon 与 Ali Farhadi 教授为我们带来的YOLO！

目录
CONTENTS

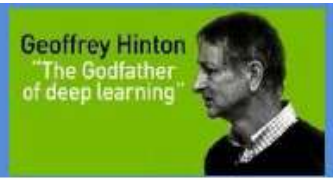
01

Deep Learning 大咖简介

- 深度学习三巨头
- 大咖们的师生关系
- 深度学习发展不平坦之路
- 其余大牛介绍


Yann Lecun
Facebook AI Research (FAIR)
New York University (NYU)
Geoffrey Hinton
Google Brain
University of Toronto
Yoshua Bengio
Canadian Institute for Advanced Research (CIFAR)
Université de Montréal
Andrew Ng
Google, Baidu, Landing.AI
Stanford University

Geoffrey Hinton & NNs



- 1970年，当神经网络研究的第一个寒冬降临时，在英国的爱丁堡大学，一位23岁的年轻人 Geoffrey Hinton，刚刚获得心理学的学士学位.
- Hinton 六十年代还是中学生时就对脑科学着迷。当时一个同学给他介绍关于大脑记忆的理论是：大脑对于事物和概念的记忆，不是存储在某个单一的地点，而是像全息照片一样，分布式地存在于一个巨大的神经元的网络里.
- 分布式表征 (Distributed Representation)和传统的局部表征 (Localized Rep.) 相比
  - 存储效率高很多：线性增加的神经元数目，可以表达指数级增加的大量不同概念
  - 鲁棒性好：即使局部出现硬件故障，信息的表达不会受到根本性的破坏
- 这个理念让 Hinton 顿悟，使他40多年来一直在神经网络研究的领域里坚持
  - 本科毕业后，Hinton 选择继续在爱丁堡大学读研，把人工智能作为自己的博士研究方向
  - 1978 年，Hinton在爱丁堡获得博士学位后，来到美国继续他的研究工作

Hinton & Deep Learning

- 2003年, Geoffrey Hinton, 还在多伦多大学, 在神经网络的领域苦苦坚守
- 2003年在温哥华大都会酒店, 以Hinton 为首的十五名来自各地的不同专业的科学家, 和加拿大先进研究院 (Canadian Institute of Advanced Research, CIFAR) 的基金管理负责人 Melvin Silverman 交谈
  - Silverman 问大家, 为什么 CIFAR 要支持他们的研究项目
  - 计算神经科学研究者, Sebastian Sung (现为普林斯顿大学教授) 回答道： "喔, 因为我们有点古怪. 如果CIFAR 要跳出自己的舒适区, 寻找一个高风险, 极具探索性的团体, 就应当资助我们了!"
  - 最终 CIFAR 同意从2004年开始资助这个团体十年, 总额一千万加元. CIFAR 成为当时世界上唯一支持神经网络研究的机构
- Hinton 拿到资金支持不久, 做的第一件事, 就是把"神经网络" 改名换姓为 "深度学习"
- 此后，Hinton 的同事不时会听到他突然在办公室大叫： "我知道人脑是如何工作的了!".

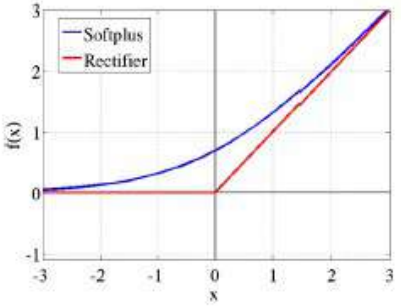
Yann Lecun & CNN

- Yann Lecun于1960年出生于巴黎
- 1987年在法国获得博士学位后，他曾追随 Hinton 教授到多伦多大学做了一年博士后的工作，随后搬到新泽西州的 Bell Lab 继续研究工作
- 在 Bell Lab，Lecun1989年发表了论文，“反向传播算法在手写邮政编码上的应用”，他用美国邮政系统提供的近万个手写数字的样本来训练神经网络系统，训练好的系统在独立的测试样本中，错误率只有5%
- Lecun进一步运用一种叫做“卷积神经网络” (Convolutional Neural Networks) 的技术，开发出商业软件，用于读取银行支票上的手写数字，这个支票识别系统在九十年代末占据了美国接近20%的市场
- 此时就在Bell Lab，Yann Lecun临近办公室的一个同事Vladmir Vapnik的工作，又把神经网络的研究带入第二个寒冬！

MNIST数据集介绍及其下载地址 http://yann.lecun.com/exdb/mnist/

Yoshua Bengio & RELU

- 2011 年, 加拿大的蒙特利尔大学学者 Xavier Glorot 和 Yoshua Bengio 发表论文：Deep Sparse Rectifier Neural Networks
- 论文的算法中使用一种称为"修正线性单元" (REctified Linear Unit, RELU) 的激励函数. 对于特定的输入, 统计上有一半神经元是没有反应, 保持沉默
- 和使用别的激励函数的模型相比, RELU 不仅识别错误率普遍更低, 而且其有效性, 对于神经网络是否进行"预先训练"过并不敏感

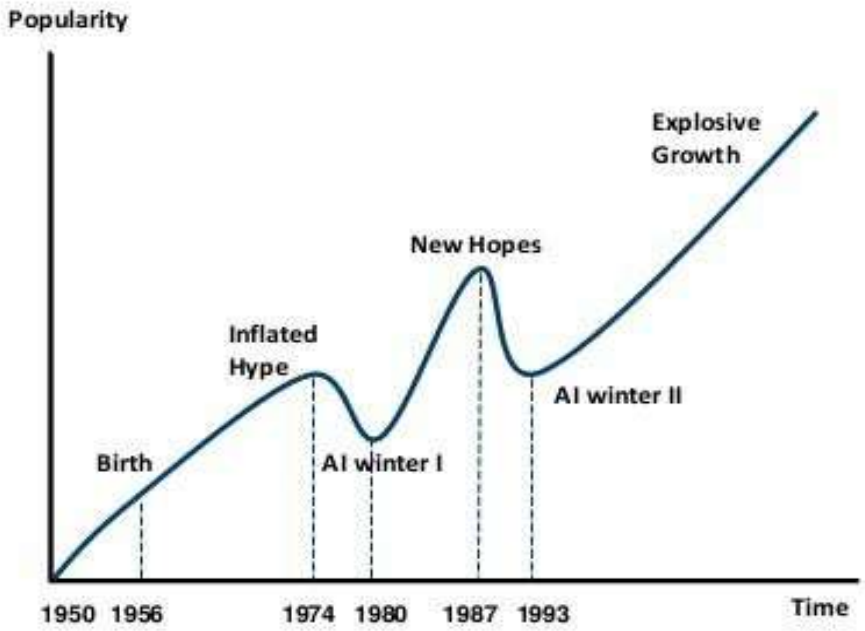


  - 传统的激励函数, 计算时要用指数或者三角函数, 计算量要比简单的RELU至少高两个数量级
  - RELU的导数是常数, 非零即一, 不存在传统激励函数在反向传播计算中的"梯度消失问题"
  - 由于统计上约一半的神经元在计算过程中输出为零, 使用 RELU 的模型计算效率更高, 而且自然而然的形成了所谓 "稀疏表征" (sparse representation), 用少量的神经元可以高效, 灵活, 稳健地表达抽象复杂的概念

深度学习发展 – 不平坦之路

Artificial Intelligence

AI HAS A LONG HISTORY OF BEING "THE NEXT BIG THING"...



Timeline of AI Development

- **1950s-1960s**: First AI boom - the age of reasoning, prototype AI developed
- **1970s**: AI winter I
- **1980s-1990s**: Second AI boom: the age of Knowledge representation (appearance of expert systems capable of reproducing human decision-making)
- **1990s**: AI winter II
- **1997**: Deep Blue beats Gary Kasparov
- **2006**: University of Toronto develops Deep Learning
- **2011**: IBM's Watson won Jeopardy
- **2016**: Go software based on Deep Learning beats world's champions

Source: Literature review, Gao Feng analysis

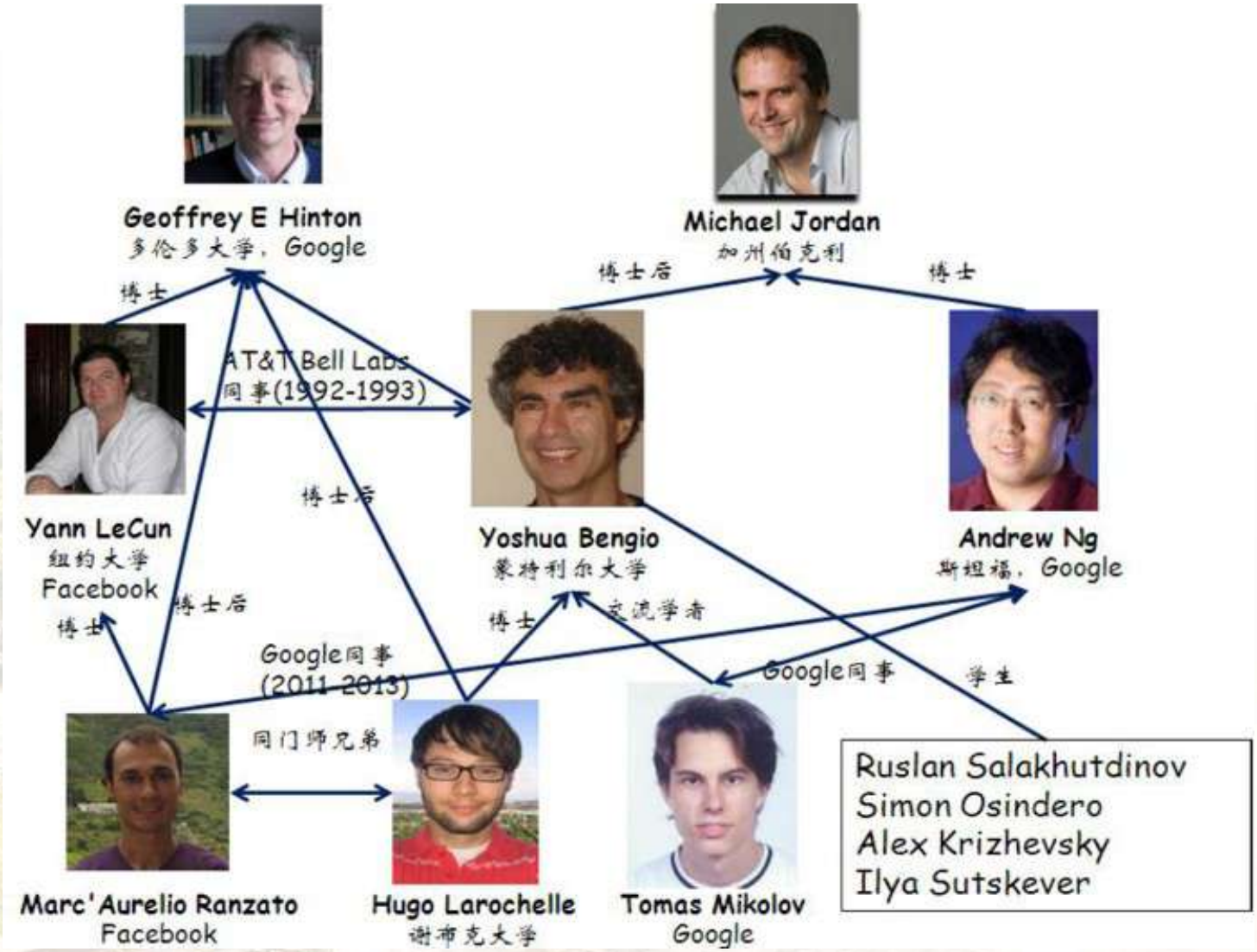
7

Team Finland Future Watch Report, August 2017

team FINLAND

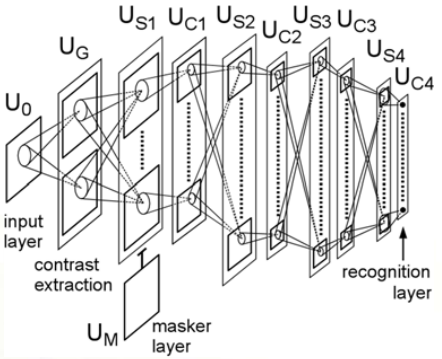
图灵奖获得者、人工智能之父Marvin Minsky对Frank Rosenblatt提出的人工神经网络模型单层感知机无法表示异或的批评最终导致1970年的人工智能冬天。而这个人工神经网络模型60年后的今天掀起了智能革命。

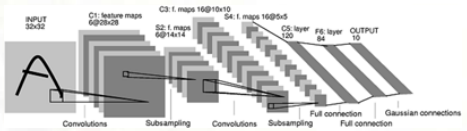
深度学习传说中的最牛逼的那帮人的师生关系

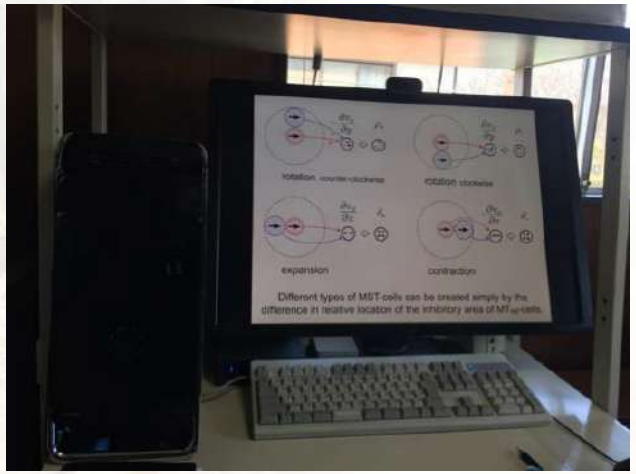


Kunihiko Fukushima & CNN

日本学者福島邦彦（Kunihiko Fukushima）40年前提出深度学习原型，在几乎容不下两个人的小屋内潜心研究50年。

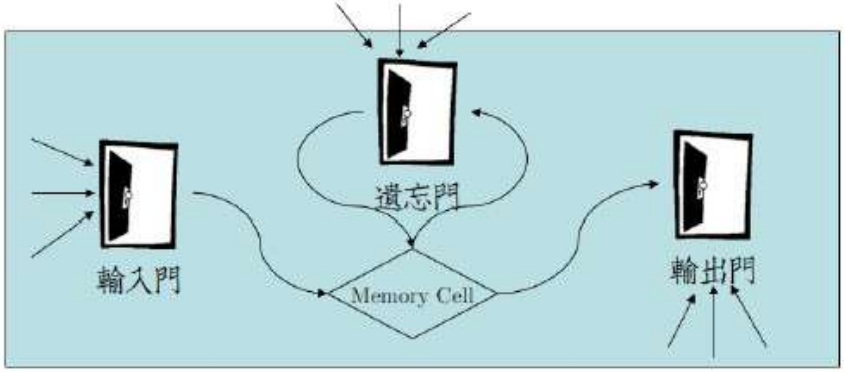






Schmidhuber & LSTM

- 1997年瑞士 Lugano 大学的 Schmidhuber 和他的学生 Sepp Hochreiter 合作, 提出了长短期记忆 (LSTM, Long Short-Term Memory) 的计算模型
- LSTM：背后要解决的问题, 是如何将有效信息, 在多层循环神经网络传递之后, 仍能输送到需要的地方去
- LSTM 模块, 是通过内在参数的设定 (如图, input gate, output gate, forget gate), 决定某个输入信息在很久以后是否还值得记住, 何时取出使用, 何时废弃不用



Andrew Y. Ng & GPU

- 2007年之前，用GPU编程缺乏一个简单的软件接口，编程繁琐，Debug困难
  2007年 Nvidia 推出 CUDA 的GPU 软件接口后才真正改善
- 2009年6月，斯坦福大学的 Rajat Raina 和吴恩达合作发表论文：Large-scale Deep Unsupervised Learning using Graphic Processors (ICML09)；论文采用DBNs模型和稀疏编码(Sparse Coding)，模型参数达到一亿（与Hinton模型参数的对比见下表）
- 论文结果显示：使用GPU运行速度和用传统双核CPU相比，最快时要快近70倍，在一个四层，一亿个参数的深信度网络上使用GPU把程序运行时间从几周降到一天

Published source	Application	Params
Hinton et al., 2006	Digit images	1.6mn
Hinton & Salakhutdinov	Face images	3.8mn
Salakhutdinov & Hinton	Sem. hashing	2.6mn
Ranzato & Szummer	Text	3mn
Our model		100mn

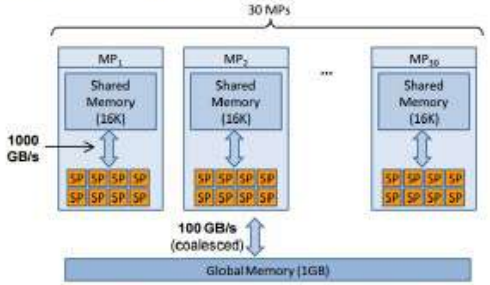


Figure 1. Simplified schematic for the Nvidia GeForce GTX 280 graphics card, with 240 total cores (30 multiprocessors with 8 stream processors each).

Jen-Hsun Huang & GPU

- 黄仁勋, 1963年出生于台湾
- 1993年从斯坦福大学硕士毕业后不久创立了 Nvidia
- Nvidia 起家时做的是图像处理的芯片, 主要面对电脑游戏市场. 1999 年Nvidia推销自己的 Geforce 256 芯片时, 发明了 GPU (Graphics Processing Unit)这个名词
- GPU 的主要任务, 是要在最短时间内显示上百万、千万甚至更多的像素. 这在电脑游戏中是最核心的需求. 这个计算工作的核心特点, 是要同时并行处理海量的数据
- 传统的 CPU 芯片架构, 关注点不在并行处理, 一次只能同时做一两个加减法运算. 而 GPU 在最底层的算术逻辑单元 (ALU, Arithmetic Logic Unit), 是基于所谓的 Single Instruction Multiple Data (单指令多数据流)的架构, 擅长对于大批量数据并行处理
- 一个 GPU, 往往包含几百个 ALU, 并行计算能力极高. 所以尽管 GPU 内核的时钟速度往往比 CPU的还要慢, 但对大规模并行处理的计算工作, 速度比 CPU 快许多
- 神经网络的计算工作, 本质上就是大量的矩阵计算的操作, 因此特别适合于使用 GPU

Generative Adversarial Networks (GANs)

Ian Goodfellow, OpenAI Research Scientist

NIPS 2016 tutorial

Barcelona, 2016-12-4

OpenAI

02

计算机视觉之物体识别

- 近5年内物体识别发展史
- YOLO算法下的识别效果
- YOLO算法简单介绍
- YOLO算法的优缺点

识别 = 定位+分类

Object Detection = Localization + Classification

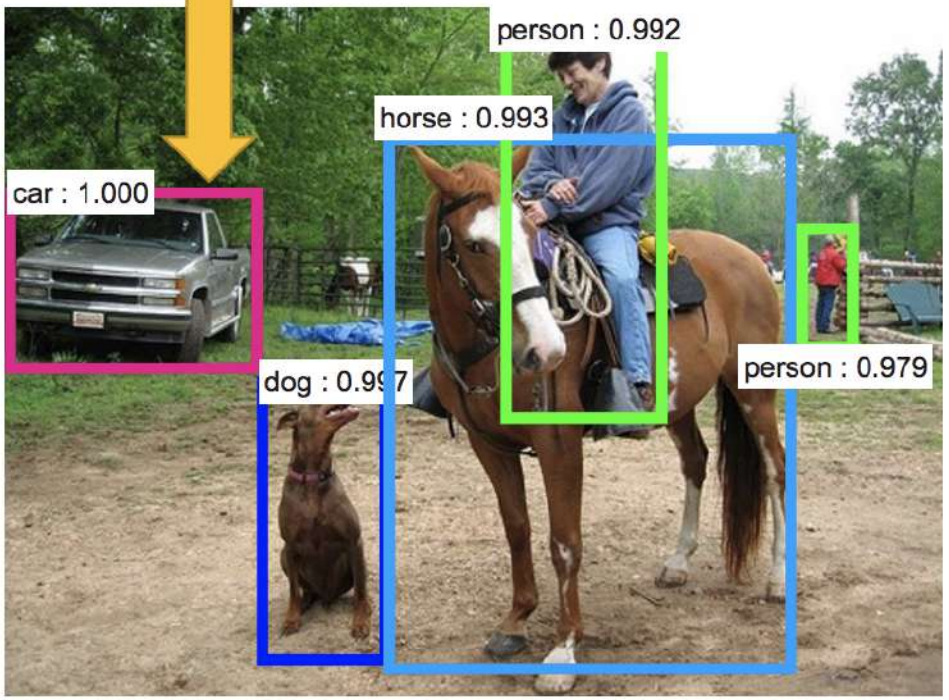
Object Detection = What, and Where

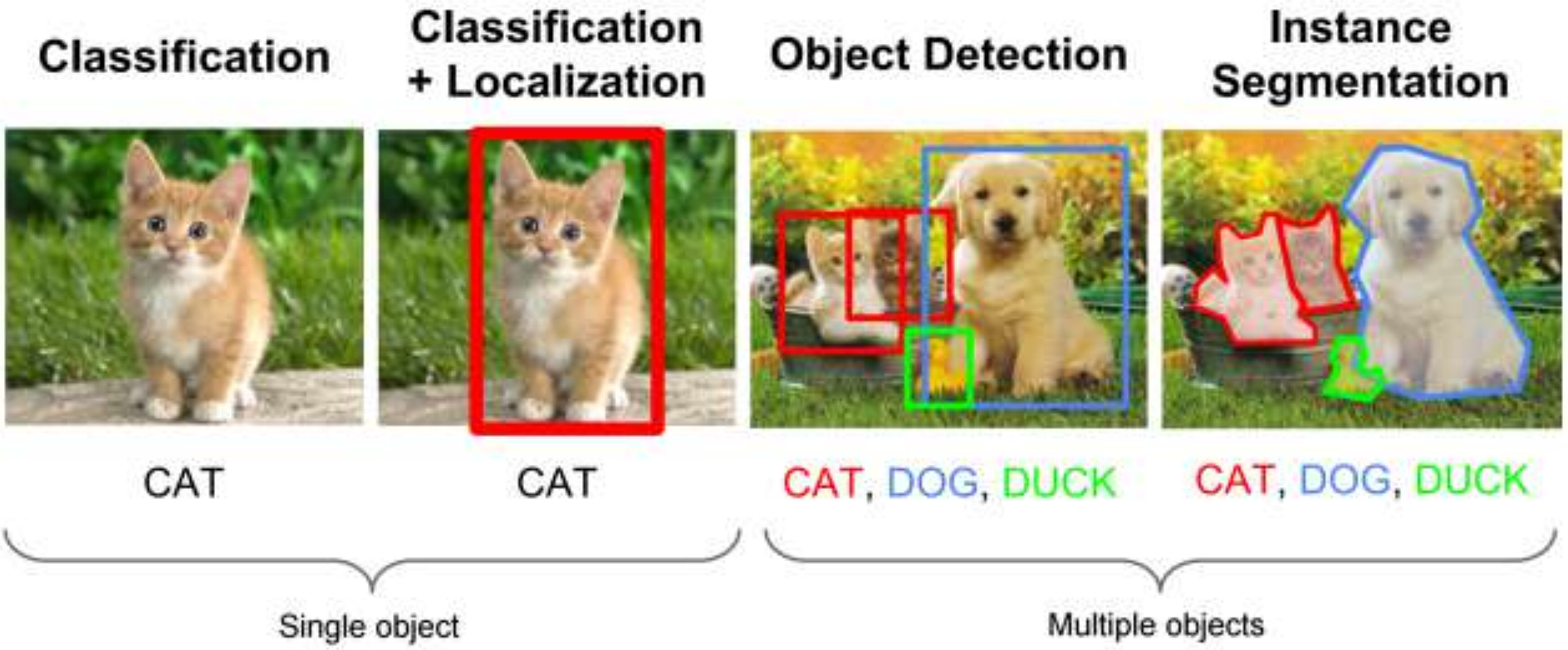
Localization

Where?

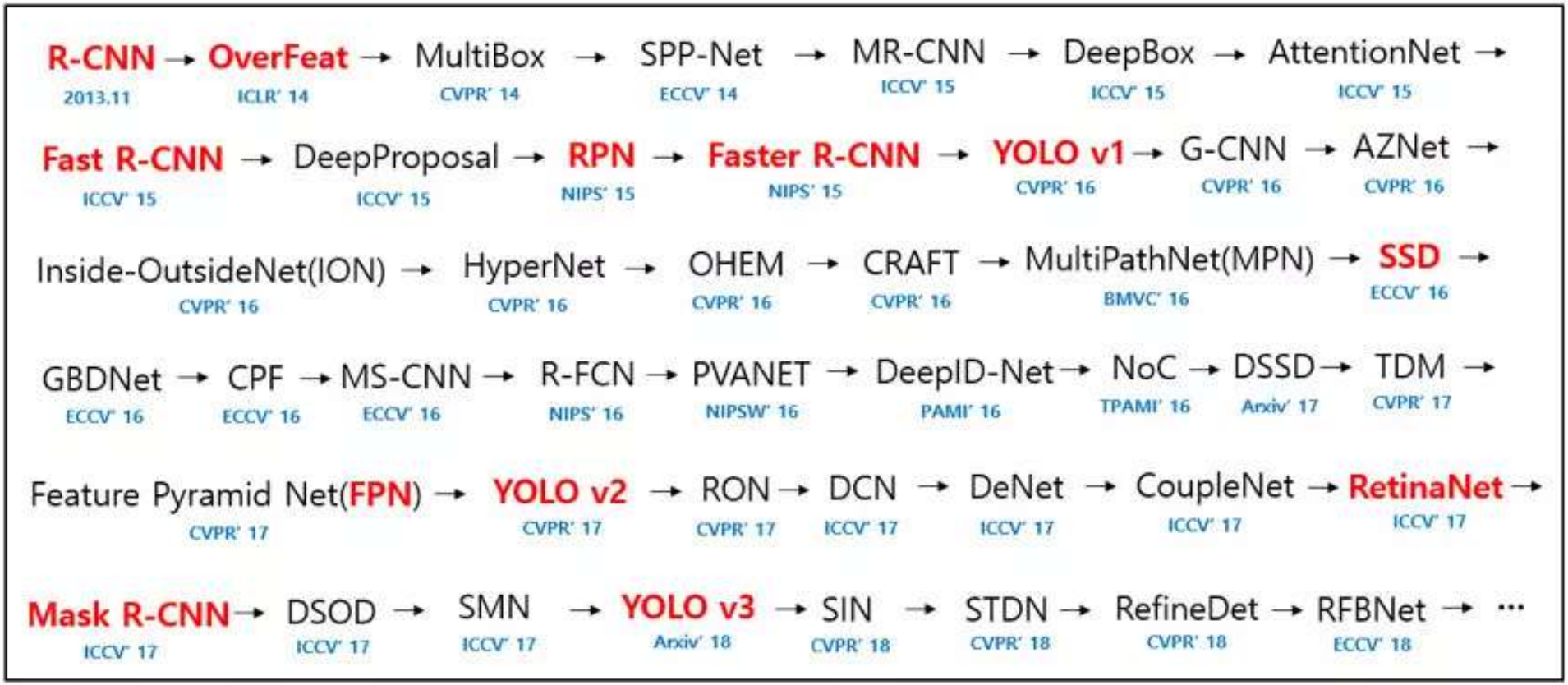
Recognition

What?

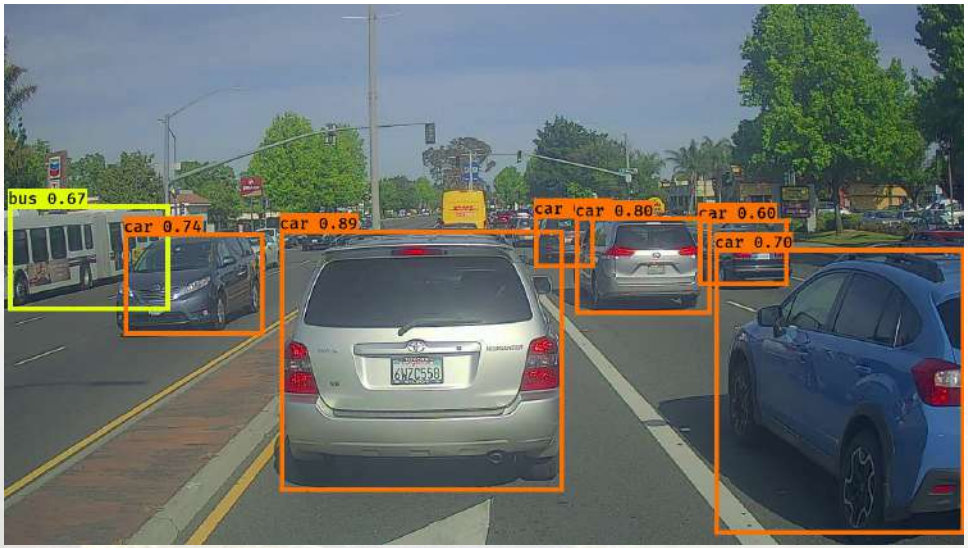


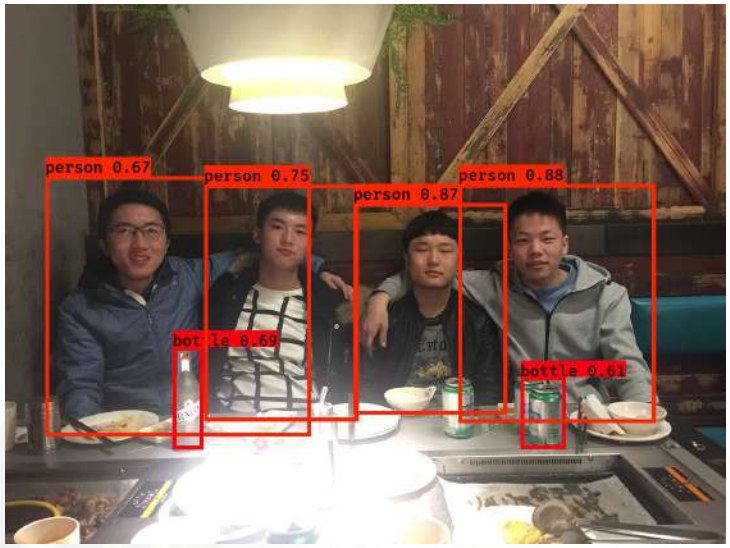

Classification
Classification + Localization
Object Detection
Instance Segmentation
CAT
CAT
CAT, DOG, DUCK
CAT, DOG, DUCK
Single object
Multiple objects

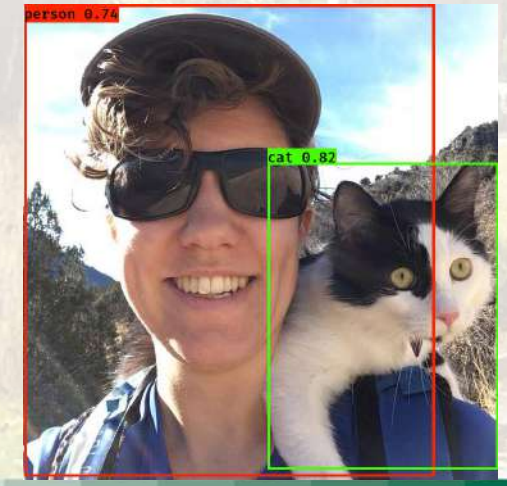
物体识别算法在近5年内的发展史

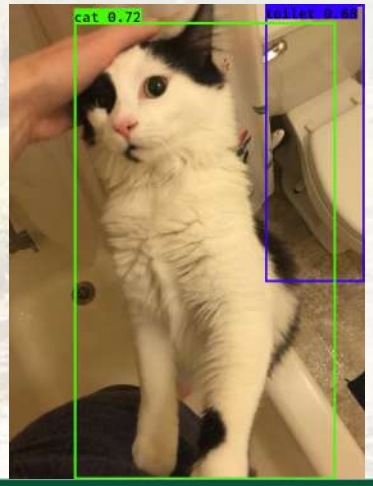


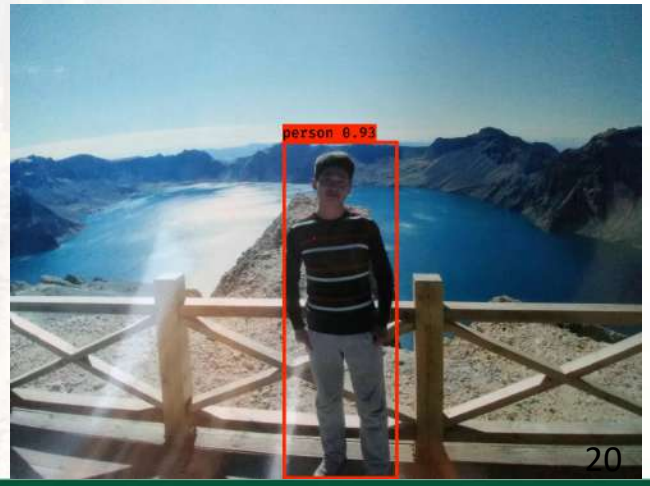
YOLO算法下物体识别效果

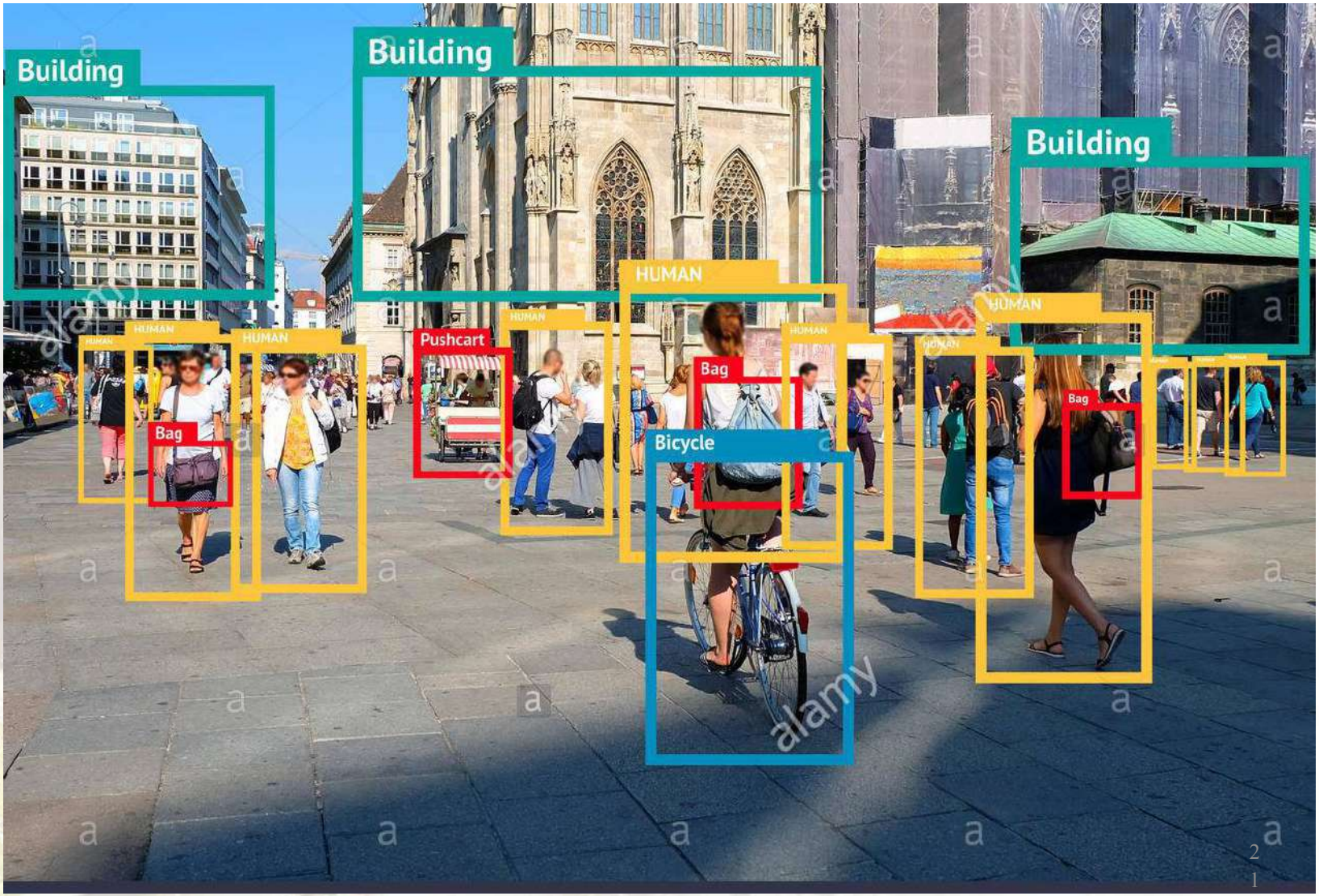




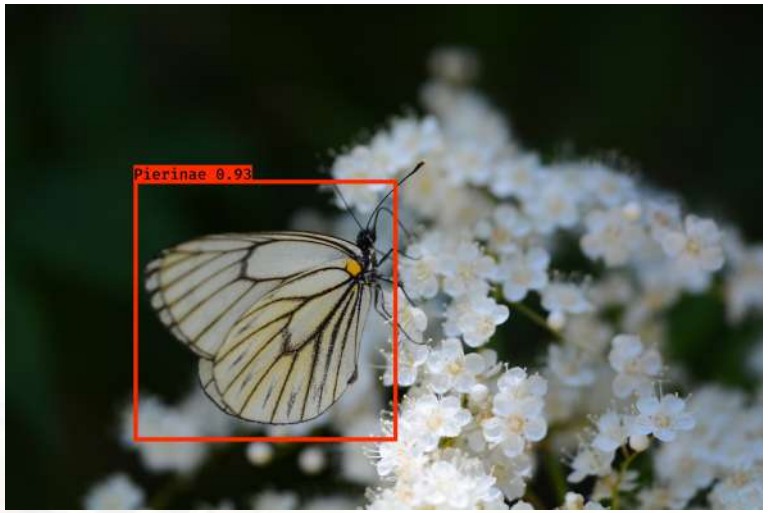


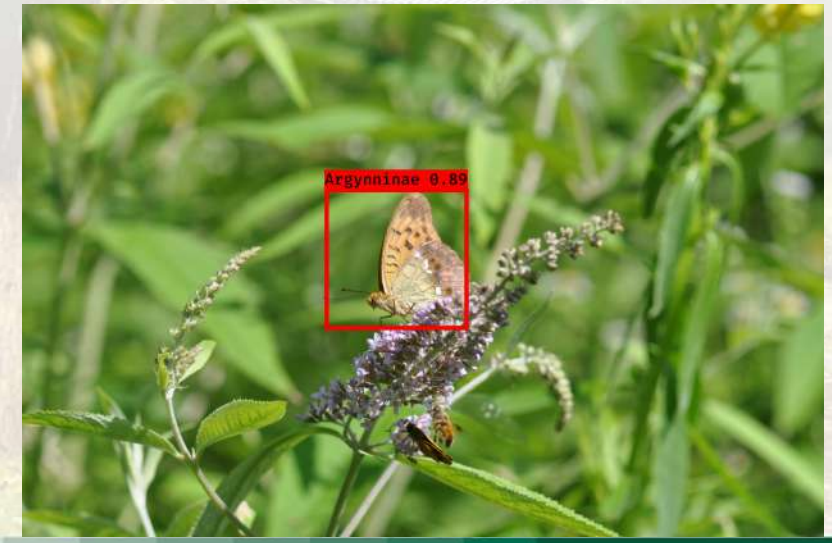


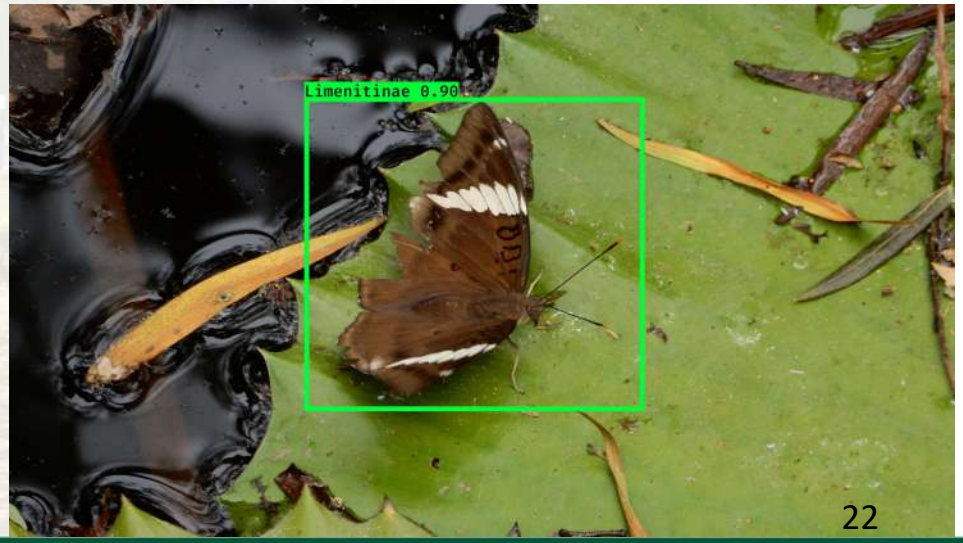



Building
Building
Building
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
Pushcart
Bag
Bag
Bag
Bicycle

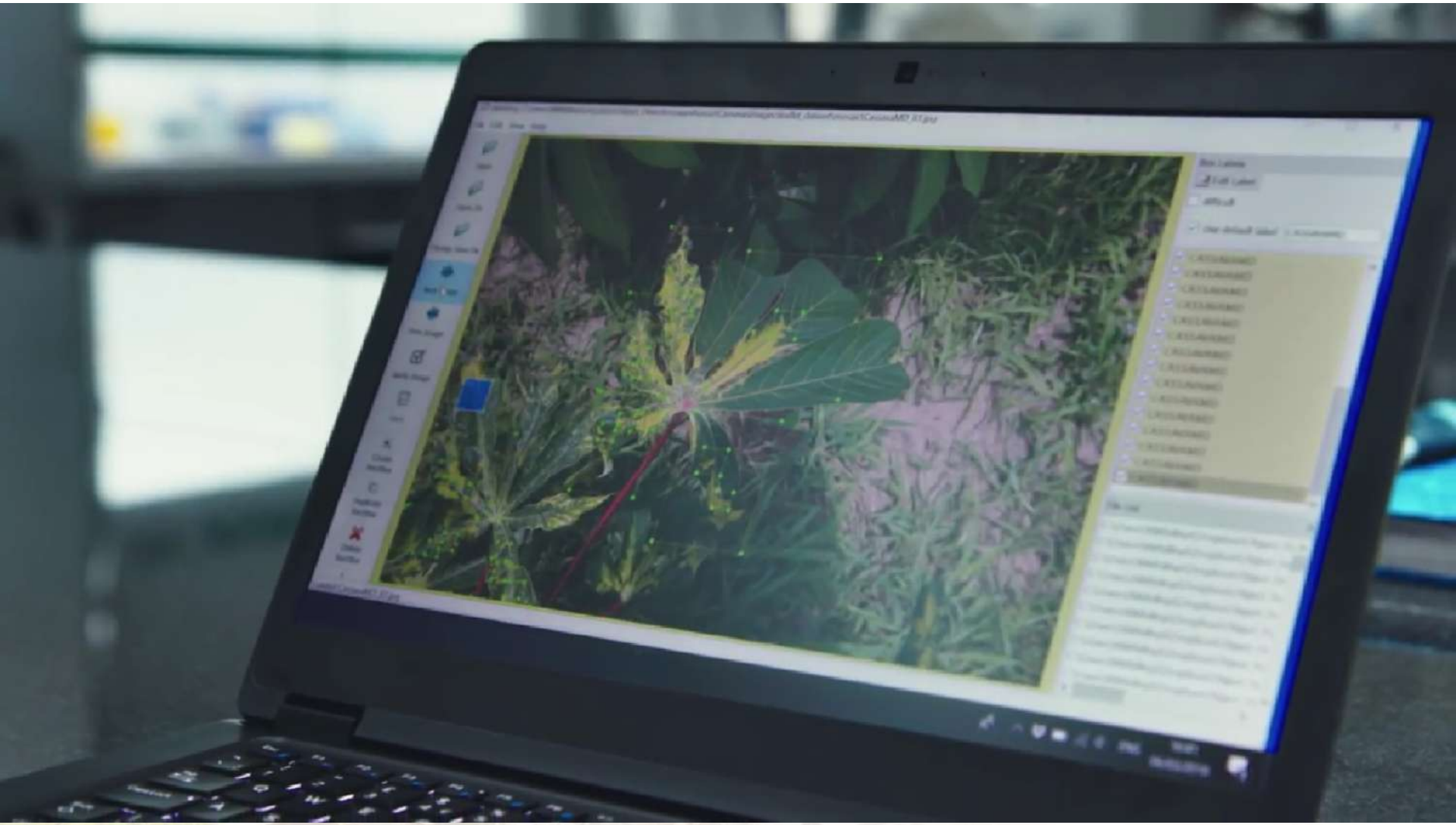
YOLO算法下训练自己的数据集

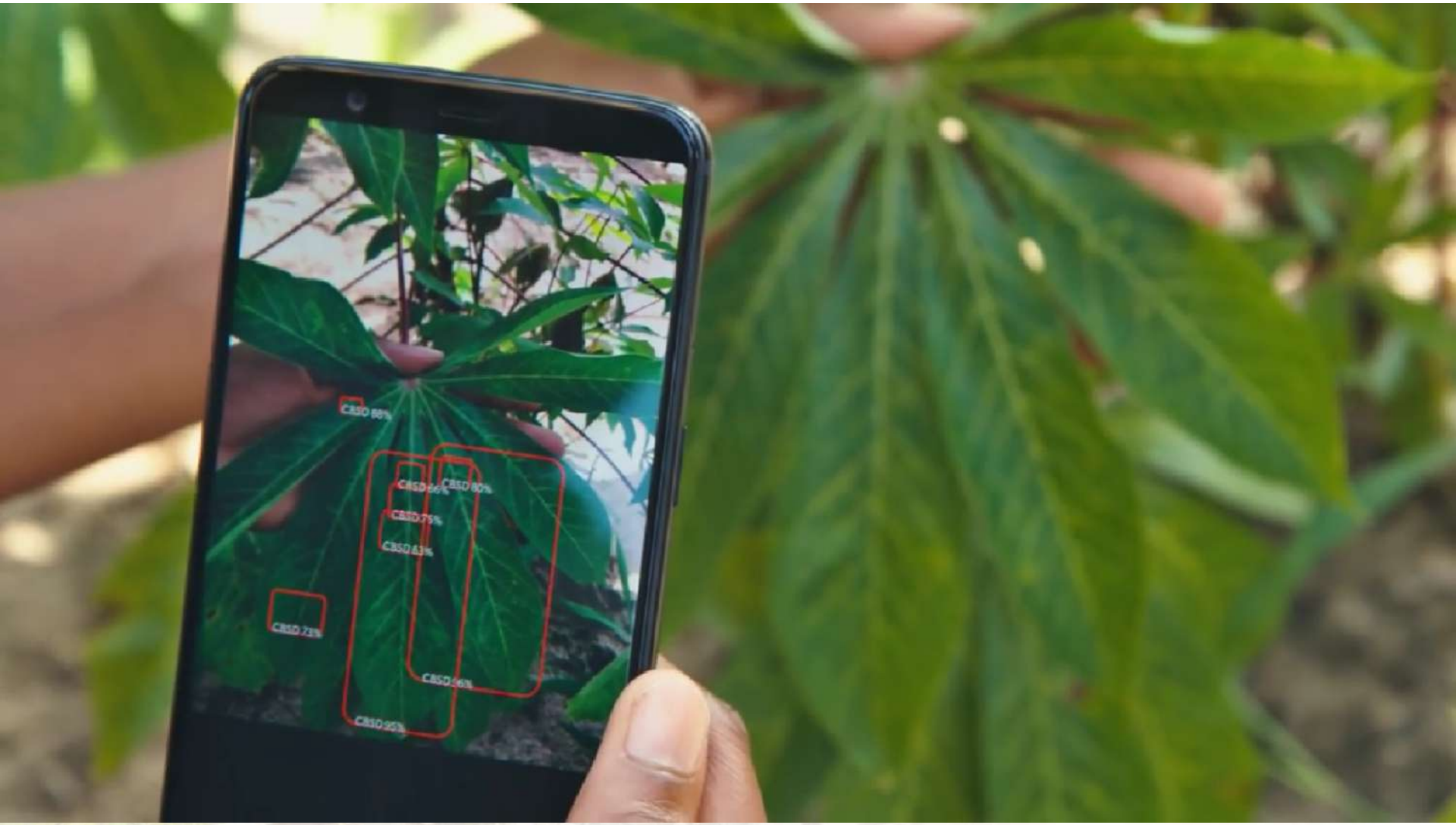


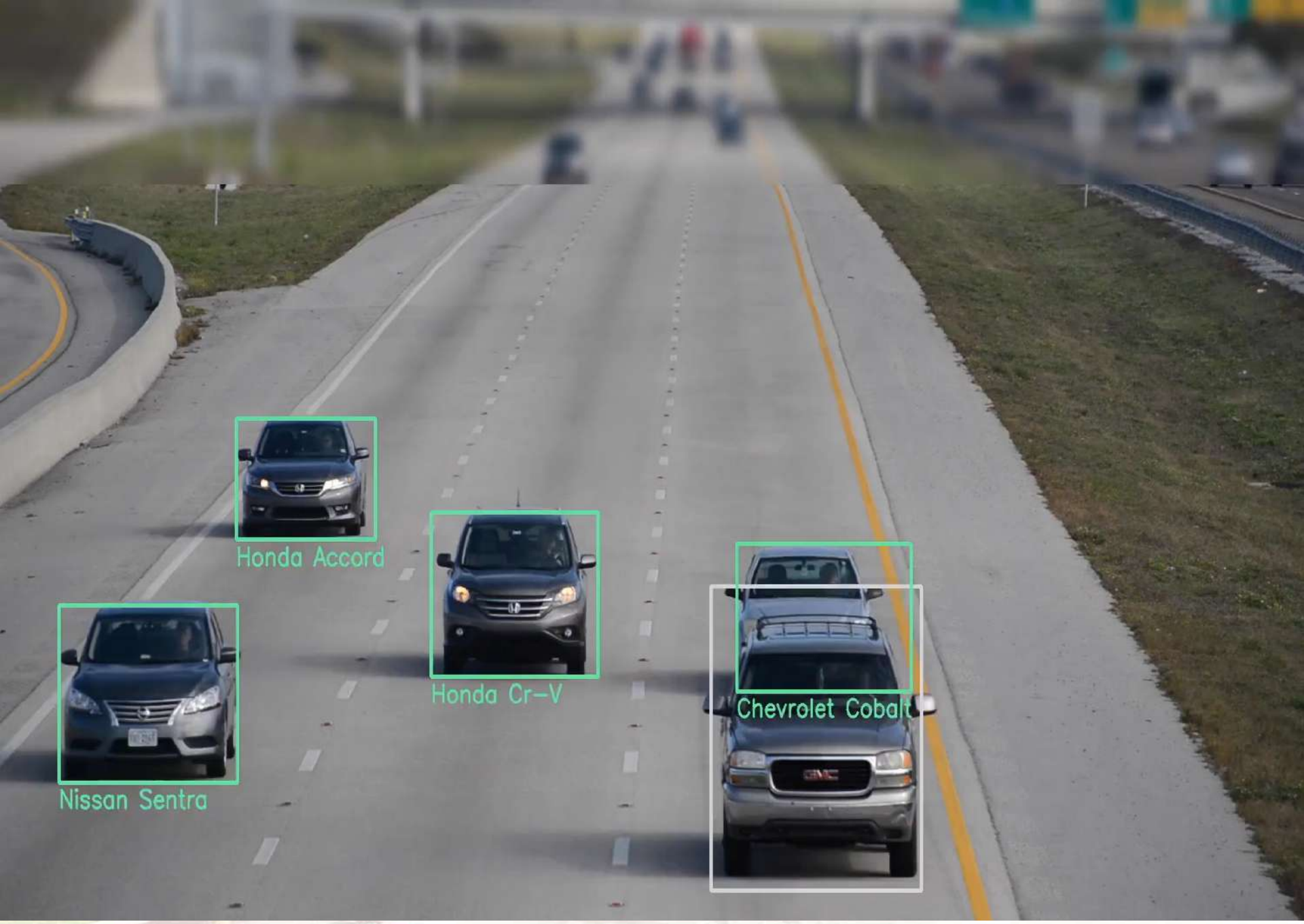




除了这些，物体识别还可以干什么




CBSD 80%
CBSD 80%
CBSD 75%
CBSD 63%
CBSD 73%

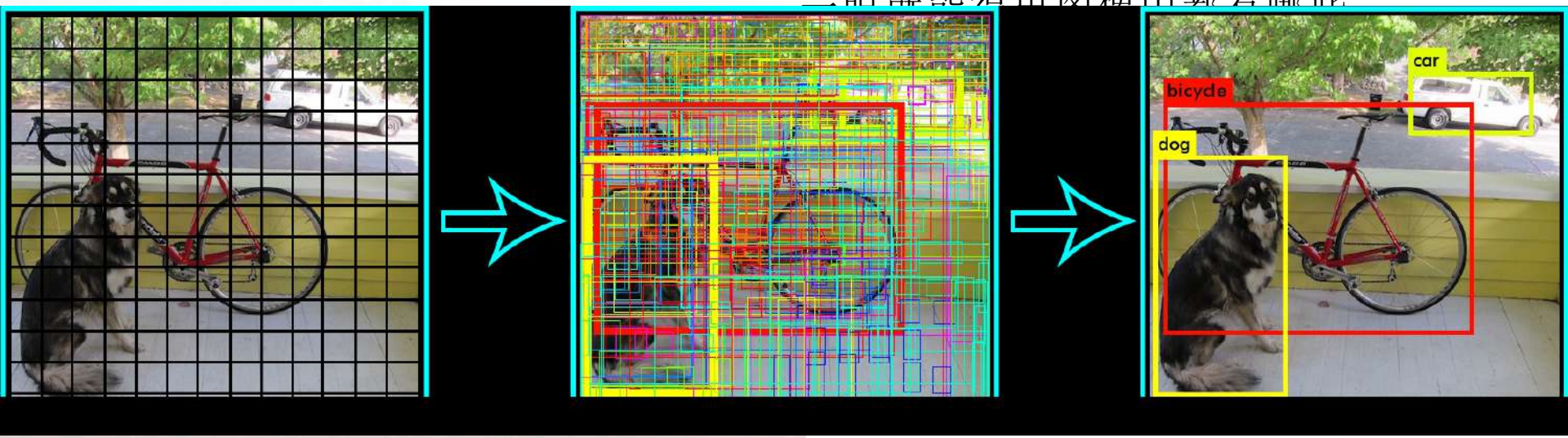


https://www.sighthound.com/technology/

YOLO：You Only Look Once

YOLO是目前比较流行的

通过YOLO，每张图像只需要看



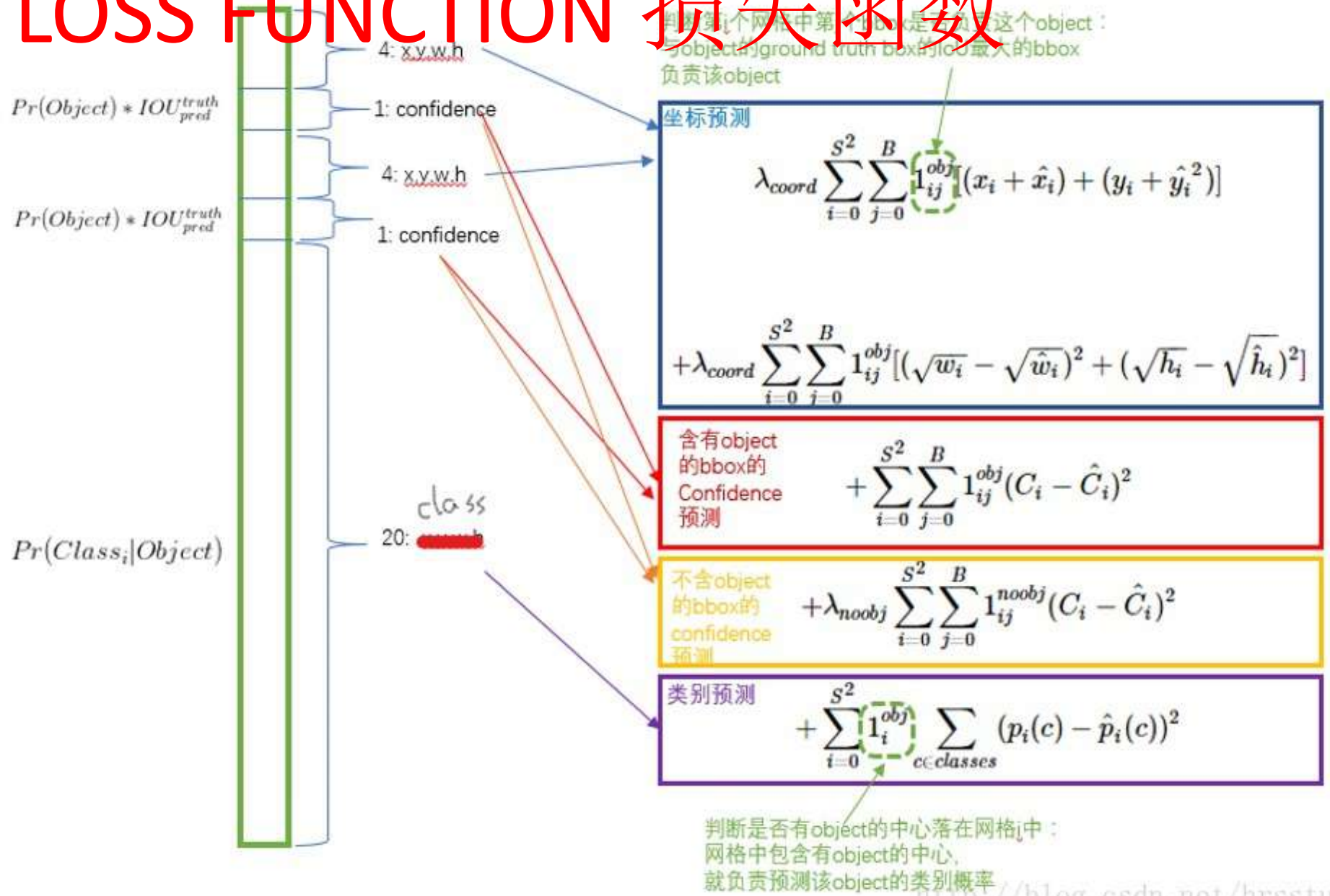
probabilities。

Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

YOLO官网：https://pjreddie.com/darknet/yolo/

论文链接：https://pjreddie.com/media/files/papers/YOLOv3.pdf

LOSS FUNCTION 损失函数



http://blog.csdn.net/hrsstudy

为什么要用YOLO做物体识别呢？

1. 代码(Windows & Linux版)开源：https://github.com/AlexeyAB/darknet
   代码(TensorFlow & Caffe & PyTorch & Keras)开源（基于Python）：
   TensorFlow：https://github.com/hizhangp/yolo_tensorflow
   Caffe ：https://www.yuthon.com/2016/11/26/Train-Caffe-YOLO-on-our-own-dataset/
   PyTorch ：https://github.com/longcw/yolo2-pytorch
   Keras ：https://blog.csdn.net/Hansry/article/details/78691303
2. YOLO预测速度非常快。训练自己的数据集后预测一张照片0. 05秒。
3. YOLO是基于图像的全局信息进行预测的。这一点和基于sliding window以及region proposal等检测算法不一样。与Fast R-CNN相比，YOLO在误检测（将背景检测为物体）方面的错误率能降低一半多。
4. YOLO可以学到物体的generalizable representations。可以理解为泛化能力强。
5. 准确率较高。

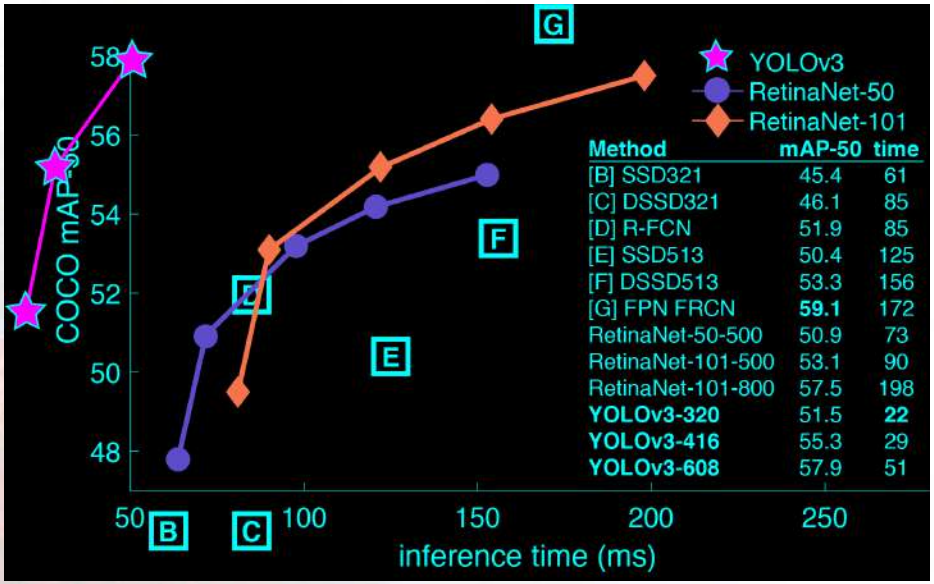
YOLO缺点

1. YOLO-V1与V2的物体检测精度低于其他state-of-the-art的物体检测系统。V3准确率与训练速度有较大提升！！！

Detection Frameworks	Train	mAP	FPS
Fast R-CNN [5]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[15]	2007+2012	73.2	7
Faster R-CNN ResNet[6]	2007+2012	76.4	5
YOLO [14]	2007+2012	63.4	45
SSD300 [11]	2007+2012	74.3	46
SSD500 [11]	2007+2012	76.8	19
YOLOv2 288 × 288	2007+2012	69.0	91
YOLOv2 352 × 352	2007+2012	73.7	81
YOLOv2 416 × 416	2007+2012	76.8	67
YOLOv2 480 × 480	2007+2012	77.8	59
YOLOv2 544 × 544	2007+2012	**78.6**	40

Table 3: Detection frameworks on PASCAL VOC 2007. YOLOv2 is faster and more accurate than prior detection methods. It can also run at different resolutions for an easy tradeoff between speed and accuracy. Each YOLOv2 entry is actually the same trained model with the same weights, just evaluated at a different size. All timing information is on a Geforce GTX Titan X (original, not Pascal model).

http://blog.csdn.net/Jesse_Mx



2. YOLO容易产生物体的定位错误。

3. YOLO对小物体的检测效果不好

（尤其是密集的小物体， YOLO一个栅格只能预测1个物体）

03

YOLO-V1算法

YOLO-V1

1. 卷积神经网络层设置(Backbone)介绍
2. 全连接层 → 输出定位
   2.1 Bounding Box等各参数含义
   2.2 非极大值抑制算法(Non-max Suppression)
3. 输出维度与注意事项

YOLO-V1

1. 网络：1)YOLO检测网络包括24个卷积层(24个卷积层 + 5个最大池化层Max Pooling)和2个全连接层(Global Average Pooling + Softmax)

   2)卷积层用来提取图像特征(分类)

   全连接层用来预测图像位置和类别概率值(定位)

   3)使用1x1卷积层(此处1x1卷积层的存在是为了跨通道信息整合)+3x3卷积层简单替代

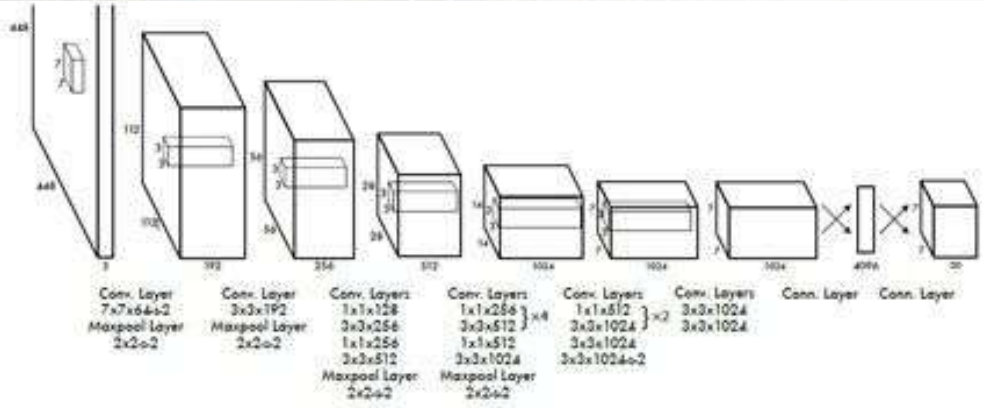


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

论文地址：https://arxiv.org/abs/1506.02640

代码地址：https://pjreddie.com/darknet/yolov1/

2. YOLO全连接输出层：YOLO将输入图像分成S×S个格子，每个格子负责检测‘落入’该格子的物体。若某个物体的中心位置的坐标落入到某个格子，那么这个格子就负责检测出这个物体。

Bounding box信息包含5个数据值，分别是x, y, w, h和confidence。

1)x, y是指当前格子预测得到的物体的bounding box的中心位置的坐标

w, h是bounding box的宽度和高度

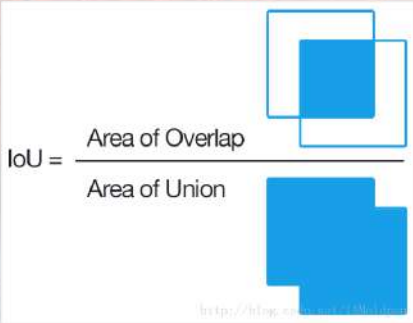
注意：实际训练过程中，w和h的值使用图像的宽度和高度进行归一化到[0,1]区间内；x, y是bounding box中心位置相对于当前格子位置的偏移值，并且被归一化到[0,1]。

2)confidence(置信度)反映当前bounding box是否包含物体以及物体位置的准确性。

$$confidence = P(object) \times IOU$$

»»若bounding box包含物体，则P(object) = 1；否则P(object) = 0。

»» IOU(intersection over union)为预测bounding box与物体真实区域的交集面积(以像素为单位，用真实区域的像素面积归一化到[0,1]区间)。

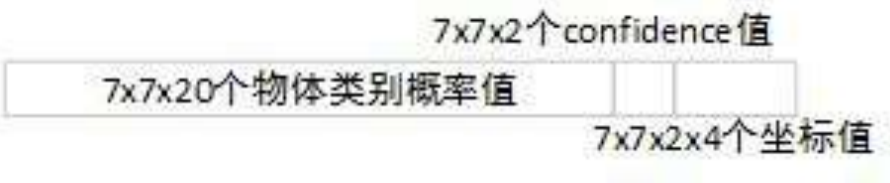


YOLO-V1网络最终的全连接层的输出维度是：S×S× (B×5+C)

S: 图片分为的S×S个小格子

B: 一个小格子里有B个预测框(YOLO-V1中B=2)

C: 共预测的种类数C



YOLO-V1论文中，作者训练采用的输入图像分辨率是448x448，S=7，B=2；采用VOC 20类标注物体作为训练数据，C=20。因此输出向量为7×7× (20 + 2×5)=1470维。

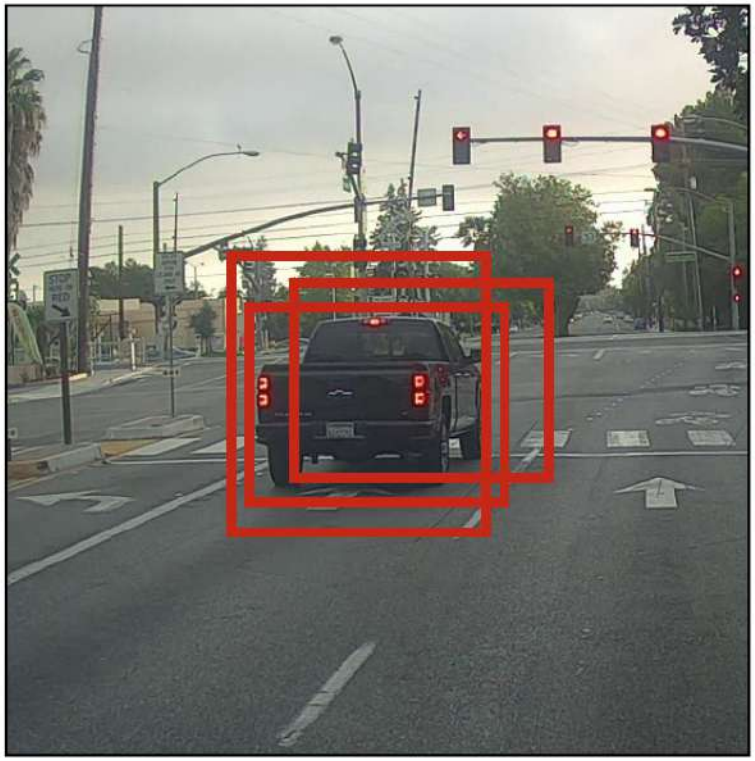
注意：

1. 由于输出层为全连接层，因此在检测(测试)时，YOLO-V1训练模型只支持与训练图像相同的输入分辨率(可以通过reshape的方法把你的照片压缩或扩张成YOLO要求的尺寸)。

2. 虽然每个格子可以预测B个bounding box，但是最终只选择只选择IOU最高的bounding box作为物体检测输出，即每个格子最多只预测出一个物体。当物体占画面比例较小，如图像中包含畜群或鸟群时，每个格子包含多个物体，但却只能检测出其中一个。这是YOLO方法的一个缺陷。

其实我们想要的结果是概率最大的那个预测框！

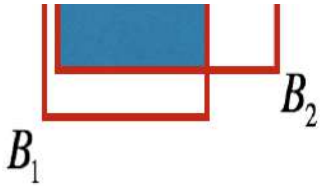
Before non-max suppression



Non-Max Suppression

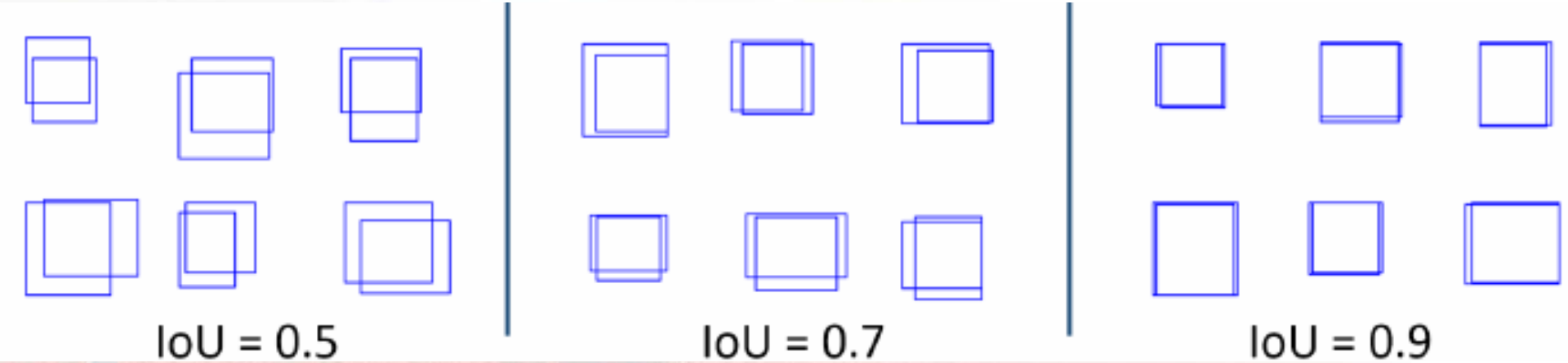
After non-max suppression

http://blog.csdn.net/Hansry

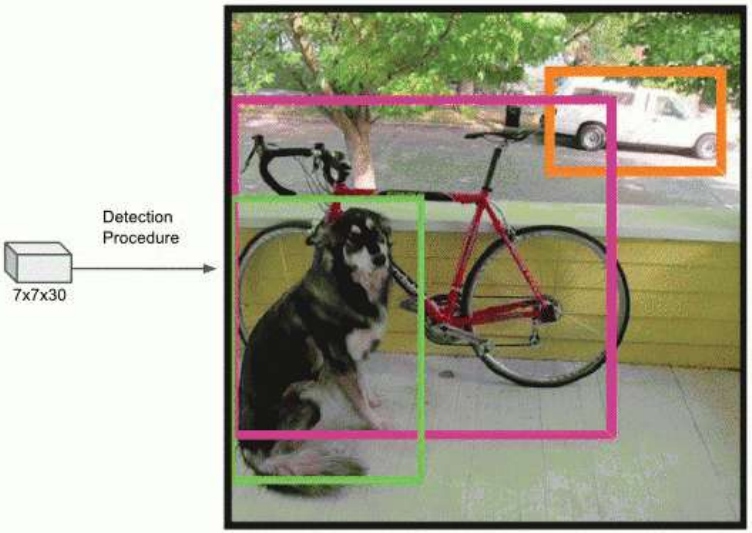


$B_1 \cup B_2$

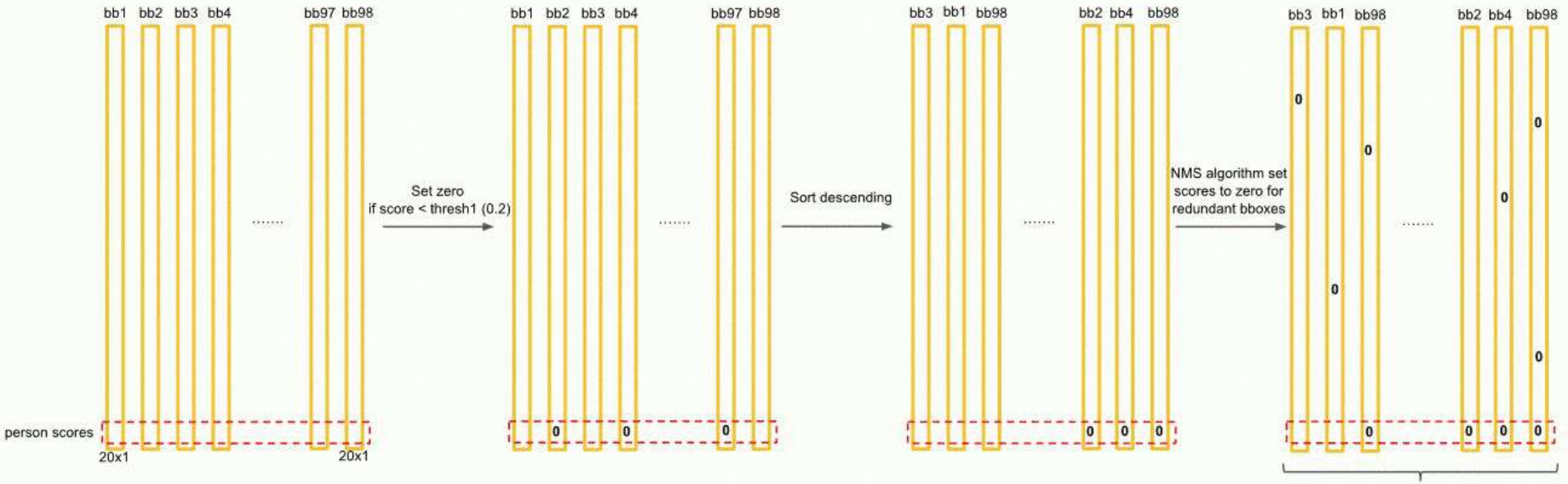
http://blog.csdn.net/Hansry


IoU = 0.5
IoU = 0.7
IoU = 0.9

Look at detection procedure



http://blog.csdn.net/hrsstudy



Select bboxes to draw by class score values

http://blog.csdn.net/hrsstudy

04

YOLO-V2算法

YOLO-V2对V1的改进

1. 批正则化(Batch Normalization)
2. 高分辨率分类器(High Resolution Classifier)
3. 使用预设框(Convolutional With Anchor Boxes)
4. 维度聚类(Dimension Clusters)
5. 直接位置预测(Direct Location Prediction)
6. 细粒度特征(Fine-Grained Features)
7. 多尺度训练(Multi-Scale Training)

论文地址：https://arxiv.org/abs/1612.08242
代码地址：https://pjreddie.com/darknet/yolov2/
算法介绍：https://xmfbit.github.io/2017/02/04/yolo-paper/

正则化(Batch Normalization)

1. 原因：神经网络学习过程本质就是为了学习数据分布。

I. 一旦训练数据与测试数据的分布不同, 那么网络的泛化能力也大大降低;

II. 一旦每批训练数据的分布各不相同(batch梯度下降), 那么网络就要在每次迭代都去学习适应不同的分布, 这样将会大大降低网络的训练速度。

2. 方法：对神经网络的每一个卷积层输出结果进行一下归一化，而不是在池化与激活函数之后。作者又给batch normalization层进行一些限制的放松，给它增加两个可学习的参数 β 和 γ ，对数据进行缩放和平移，平移参数 β 和缩放参数 γ 是学习出来的。

3. 效果：通过这一方法，mAP获得了2%的提升。正则化也有助于规范化模型，可以在舍弃dropout优化后依然不会过拟合。

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

http://blog.csdn.net/gzs0927

高分辨率分类器(High Resolution Classifier)

所有state-of-the-art的检测方法基本上都会使用ImageNet预训练过的模型(classifier)来提取特征。例如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-V1分辨率为448×448。这样做不仅切换为检测算法也改变了分辨率。所以作者想能不能在预训练的时候就把分辨率提高了，训练的时候只是由分类算法切换为检测算法。

YOLO-V2首先修改预训练分类网络的分辨率为448×448，在ImageNet数据集上训练10轮(10 epochs)。这个过程让网络有足够的时间调整filter去适应高分辨率的输入。然后fine tune为检测网络。mAP获得了4%的提升。

ImageNet是一个计算机视觉系统识别项目名称， 是目前世界上图像识别最大的数据库。是美国斯坦福的计算机科学家李飞飞与普林斯顿大学教授李佳，模拟人类的识别系统建立的。

ImageNet数据集有1400多万幅图片，涵盖2万多个类别；其中有超过百万的图片有明确的类别标注和图像中物体位置的标注，具体信息如下：

1)Total number of non-empty synsets: 21841
2)Total number of images: 14,197,122
3)Number of images with bounding box annotations: 1,034,908
4)Number of synsets with SIFT features: 1000
5)Number of images with SIFT features: 1.2 million

Big Data: ImageNet

- 2009年, 一群在普林斯顿大学计算机系的华人学者 (第一作者为 Jia Deng)发表了论文：ImageNet: A large scale hierarchical image database, 宣布建立了第一个超大型图像数据库供计算机视觉研究者使用
- 数据库建立之初,包含了320万个图像. 它的目的, 是要把英文里的8万个名词,每个词收集5百到1千个高清图片,存放到数据库里. 最终达到5千万以上的图像
- 2010 年, 以 ImageNet 为基础的大型图像识别竞赛, ImageNet Large Scale Visual Recognition Challenge 2010 (ILSVRC2010) 第一次举办

- 竞赛最初的规则：以数据库内120万个图像为训练样本. 这些图像从属于1千多个不同的类别, 都被手工标记。
  经过训练的程序, 再用于5万个测试图像评估分类准确率

Image Classification : ILSVRC竞赛

- Top Five Category：计算机会对图像的分类，答出最有可能的头五个类别，如果正确答案都不在里面即为错误
- 2010年冠军：NEC 和伊利诺伊大学香槟分校的联合团队，用支持向量机 (SVM) 的技术，识别分类的错误率为 28%
- 2011年冠军：用 Fisher Vector 的计算方法 (类似SVM)，将错误率降到了 25.7%
- 2012年冠军：Hinton 和两个研究生 Alex Krizhevsky，Illya Sutskever ，利用CNN+Dropout算法 + RELU激励函数，用了 两个 Nvidia 的 GTX 580 CPU (内存 3GB，计算速度 1.6 TFLOPS)，花了接近六天时间，错误率只有 15.3%
- 2012年10月13日，当竞赛结果公布后，学术界沸腾了 这是神经网络二十多年来，第一次在图像识别领域，毫无疑义的，大幅度挫败了别的技术
- 这也许是人工智能技术突破的一个转折点

YOLO-V2选取预选框：维度聚类
Dimension Clusters

使用anchor时，作者发现Faster-RCNN中anchor boxes的个数和宽高维度往往是手动精选的先验框(hand-picked priors)，设想能否一开始就选择了更好的、更有代表性的先验boxes维度，那么网络就应该更容易学到准确的预测位置。

解决办法就是统计学习中的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 boxes个数，以k个聚类中心box的宽高维度为anchor box的维度。

如果按照标准k-means使用欧式距离函数，大boxes比小boxes产生更多error。但是，我们真正想要的是产生好的IOU得分的boxes(与box的大小无关)。因此采用了如下距离度量：

distance(box,centroid) = 1-IOU(box,centroid)

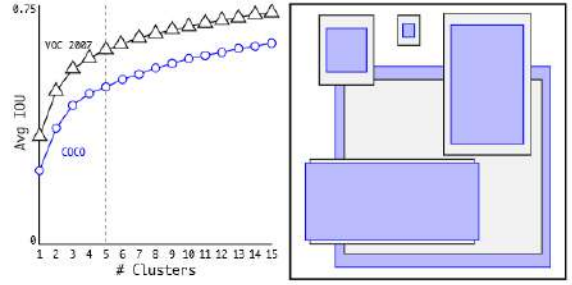


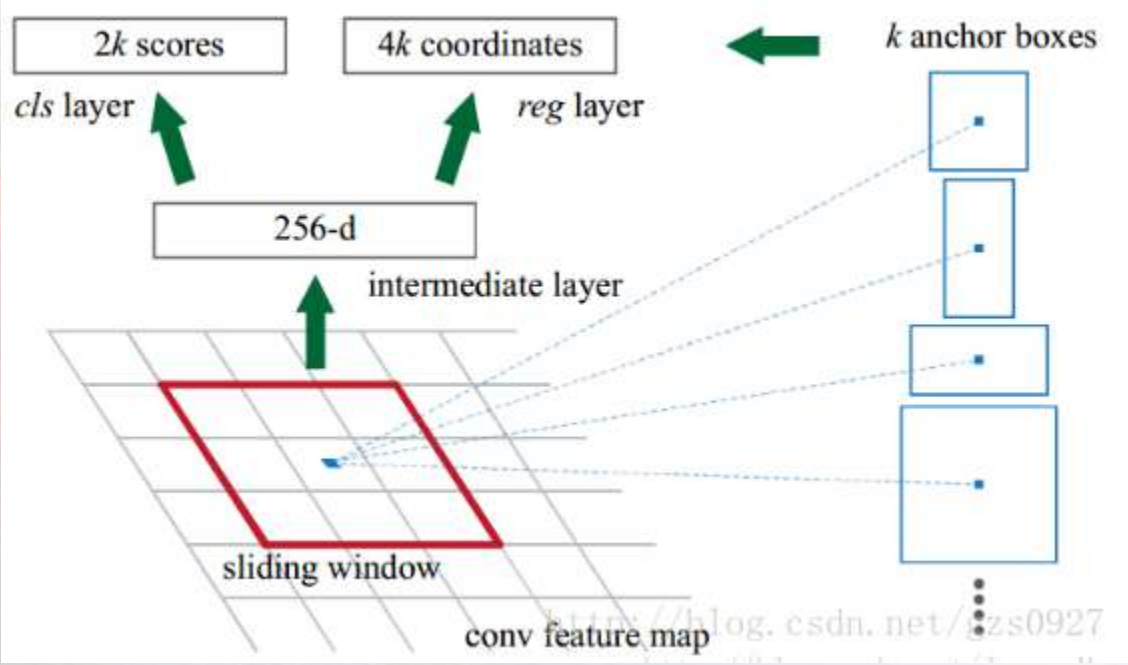
Figure 2: Clustering box dimensions on VOC and COCO. We run k-means clustering on the dimensions of bounding boxes to get good priors for our model. The left image shows the average IOU we get with various choices for k. We find that $k = 5$ gives a good tradeoff for recall vs. complexity of the model. The right image shows the relative centroids for VOC and COCO. Both sets of priors favor thinner, taller boxes while COCO has greater variation in size than VOC.

http://blog.csdn.net/gzs0927

YOLO-V2使用预选框
Convolutional With Anchor Boxes

YOLO-V1使用全连接层数据进行bounding box预测(要把1470×1的全连接层reshape为7×7×30的最终特征)，这会丢失较多的空间信息定位不准。

YOLO-V2借鉴了Faster R-CNN中的anchor思想：简单理解为卷积特征图上进行**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。



直接位置预测(Direct Location Prediction)

把对box位置的预测不再是基于全图，box的中心规定在cell之中。从公式可以看出，box的中心点的选择范围被Sigmoid函数限制在相应的小格子内了。约束了位置预测的范围，参数就更容易学习，模型就更稳定。

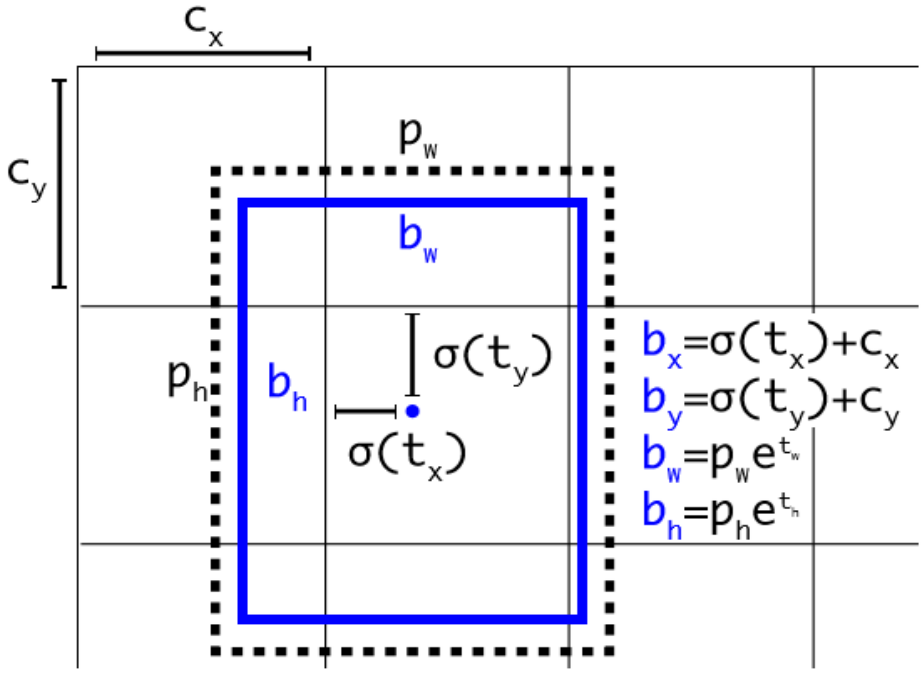


Figure 3: Bounding boxes with dimension priors and location prediction. We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function.

http://blog.csdn.net/gzs0927

在output的feature map上，对于每个cell(共计13×13个)，给出对应每个bounding box的输出t_x, t_y, t_w, t_h。
每个cell共计k=5个bounding box。

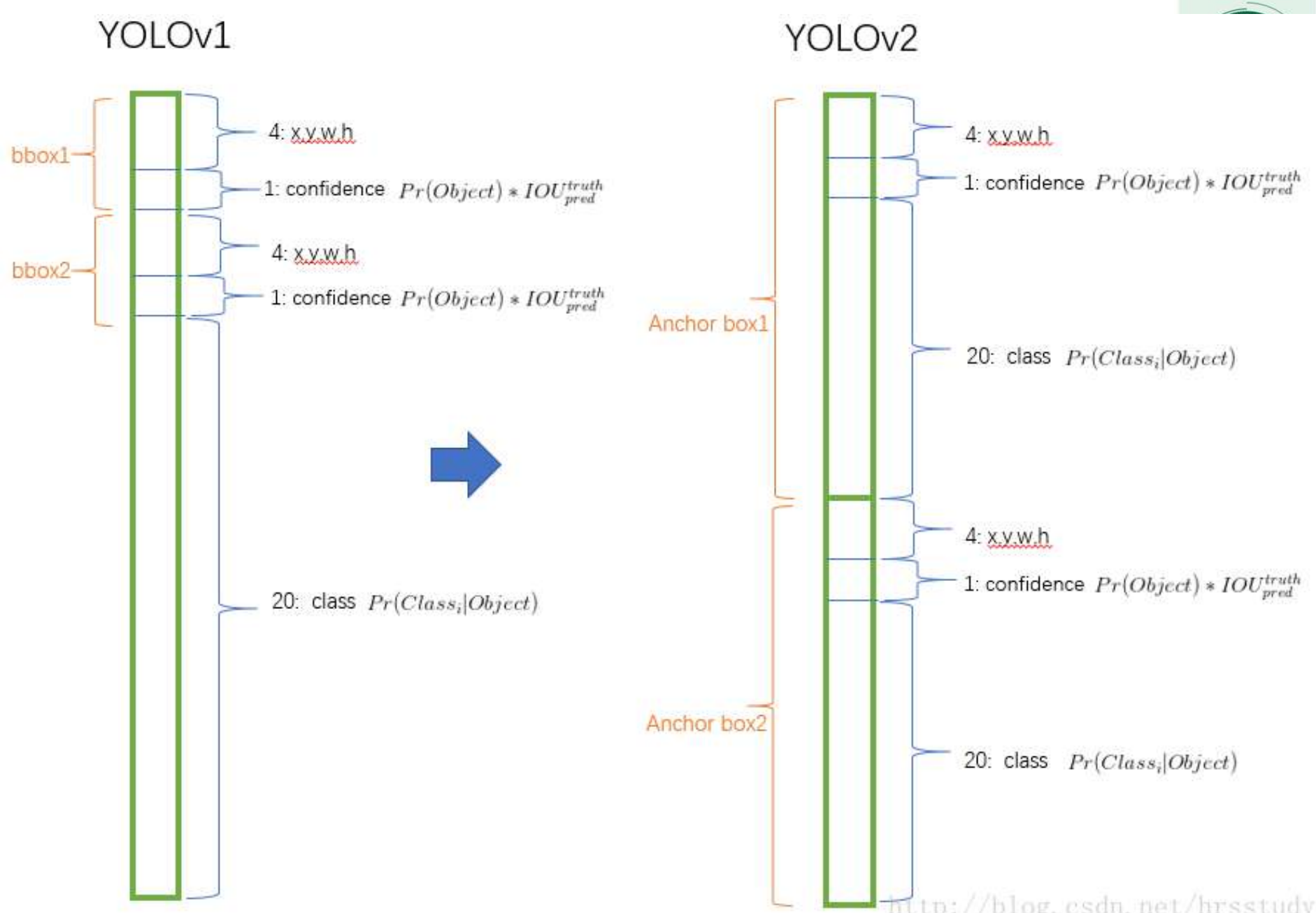
$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$
$$Pr(\text{object}) * IOU(b, \text{object}) = \sigma(t_o)$$

YOLO-V2输出维度

输出维度：在YOLO-V1中输出的维度为S×S×(B×5+C)，而YOLO-V2为S×S×(B×(5+C))

结果：加入了anchor boxes后，可以预料到的结果是召回率(Recall)上升，准确率(Precision)下降。我们来计算一下，假设每个cell预测9个建议框，那么总共会预测13×13×9=1521个boxes，而之前的网络仅仅预测7×7×2=98个boxes。

具体数据为：没有anchor boxes，模型recall为81%，mAP为69. 5%；加入anchor boxes，模型recall为88%，mAP为69. 2%。这样看来，准确率只有小幅度的下降，而召回率则提升了7%，说明可以通过进一步的工作来加强准确率，的确有改进空间。

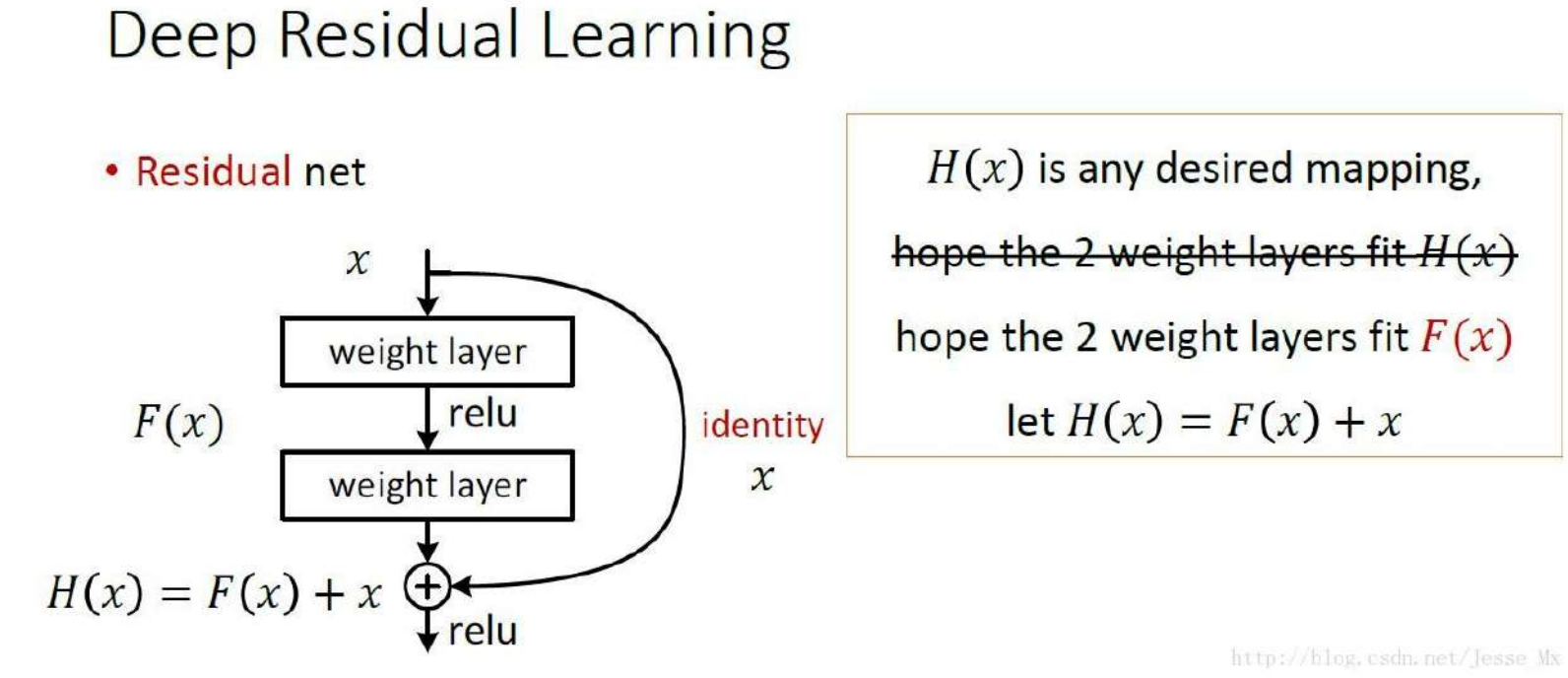


http://blog.csdn.net/hrsstudy

细粒度特征(Fine-Grained Features)

上述网络上的修改使YOLO最终在13×13的特征图上进行预测，虽然这足以胜任大尺度物体的检测，但是用上细粒度特征的话，这可能对小尺度的物体检测有帮助。

这里使用了一种不同的方法，简单添加了一个转移层(passthrough layer)，这一层要把浅层特征图(分辨率为26×26，是底层分辨率4倍)连接到深层特征图。



细粒度特征(Fine-Grained Features)

这个转移层也就是把高低两种分辨率的特征图做了一次连结，连接方式是叠加特征到不同的通道而不是空间位置，类似于Resnet中的identity mappings。这个方法把26×26×512的特征图连接到了13×13×2048的特征图，这个特征图与原来的特征相连接。YOLO的检测器使用的就是经过扩张的特征图，它可以拥有更好的细粒度特征，使得模型的性能获得了1%的提升。

补充：关于passthrough layer，具体来说就是特征重排(不涉及到参数学习)，前面26×26×512的特征图使用按行和按列隔行采样的方法，就可以得到4个新的特征图，维度都是13×13×512，然后做concat操作，得到13×13×2048的特征图，将其拼接到后面相当于做了一次特征融合，有利于检测小目标。

细粒度图像分类：细粒度图像识别是现在图像分类中一个颇具挑战性的任务，它的目标是在一个大类中的数百数千个子类中正确识别目标。
细粒度图像分类的关键点在寻找一些存在细微差别的局部区域。

细粒度分类：https://blog.csdn.net/huang2818138/article/details/78154396

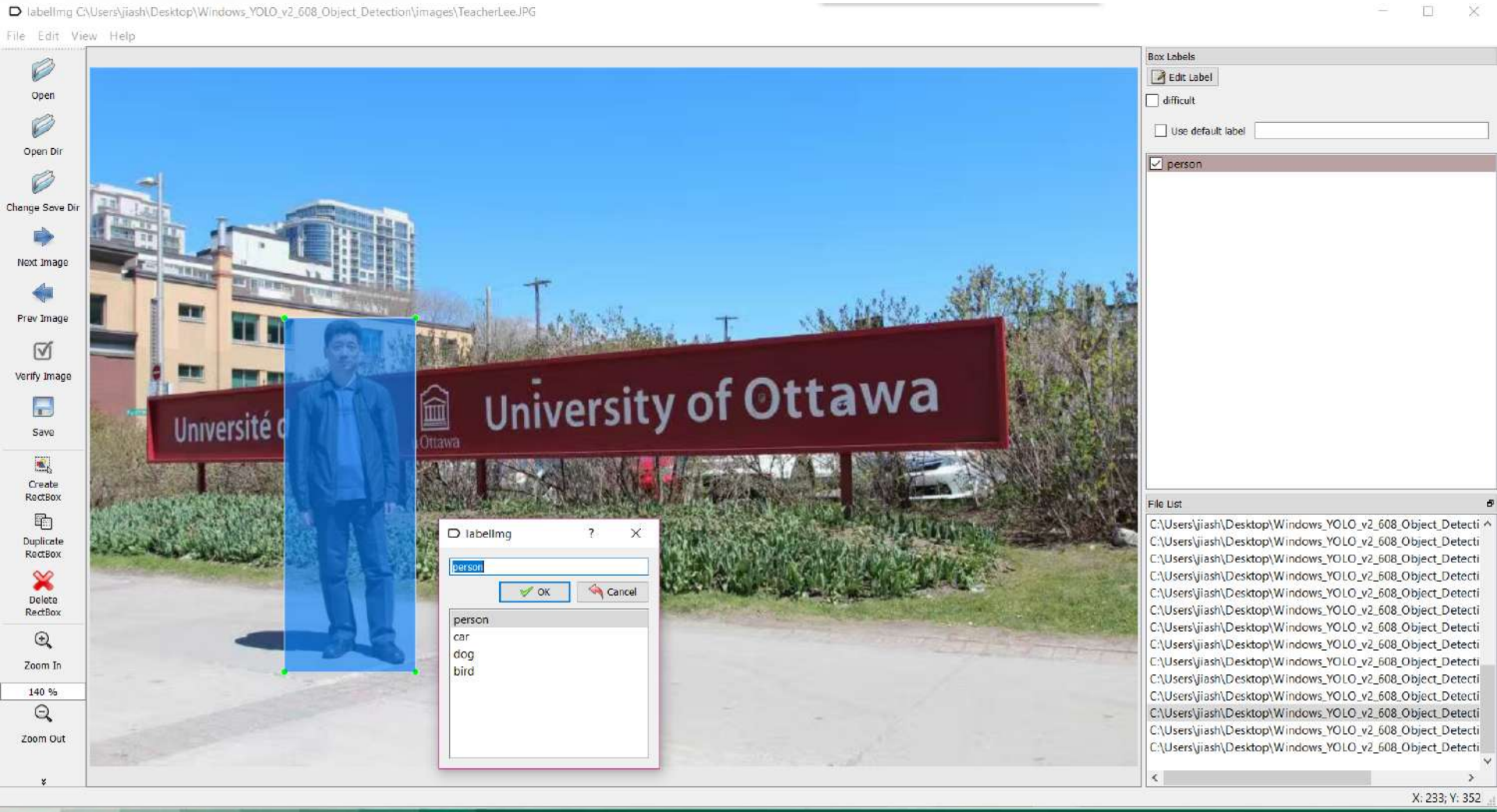
多尺度训练(Multi-Scale Training)

- 原来的YOLO-V1网络使用固定的448×448的图片作为输入。
- 不同于固定输入网络的图片尺寸的方法，作者在几次迭代后就会微调网络。没经过10次训练(10 epoch)，就会随机选择新的图片尺寸。YOLO网络使用的降采样参数为32，那么就使用32的倍数进行尺度池化{320,352，…，608}。最终最小的尺寸为320×320，最大的尺寸为608×608。接着按照输入尺寸调整网络进行训练。
- 这种机制使得网络可以更好地预测不同尺寸的图片，意味着同一个网络可以进行不同分辨率的检测任务，在小尺寸图片上YOLO-V2运行更快，在速度和精度上达到了平衡。
- 在小尺寸图片检测中，YOLO-V2成绩很好，输入为228×228的时候，帧率达到90FPS，mAP几乎和Faster R-CNN的水准相同。使得其在低性能GPU、高帧率视频、多路视频场景中更加适用。

05

算法·技术实现细节

1. 收集数据(Data)并做标记(label):在这里我们的label是Bounding Box



标注软件下载地址： https://blog.csdn.net/jesse_mx/article/details/53606897

究竟标注后生产了什么信息？

K:\机器学习\第三届中国数据挖掘大赛数据集\IMG_001426.xml - Notepad++

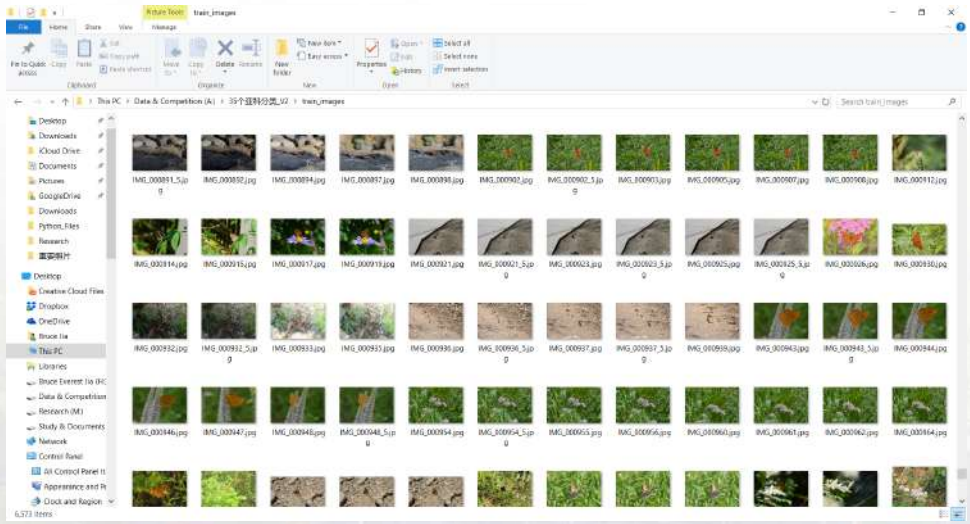
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

IMG_001426.xml

```xml
<annotation>
    <folder>hudie</folder>
    <filename>IMG_001426.jpg</filename>
    <path>G:\hudie\IMG_001426.jpg</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>4608</width>
        <height>2592</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>雅弄蝶</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>2202</xmin>
            <ymin>979</ymin>
            <xmax>2756</xmax>
            <ymax>1538</ymax>
        </bndbox>
    </object>
</annotation>
```

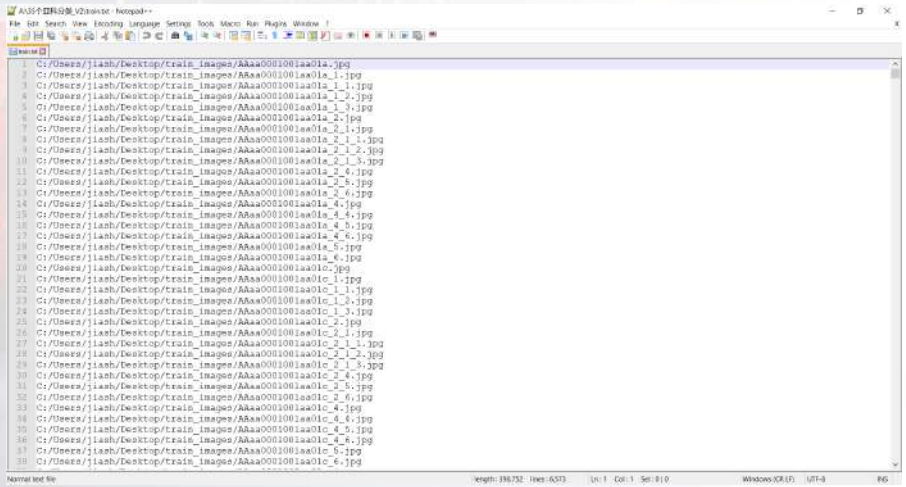
eXtensible Markup Language file | length : 518 lines : 26 | Ln : 14 Col : 13 Sel : 0 | 0 | Unix (LF) | UTF-8 | INS

制作训练集&测试集及其路径

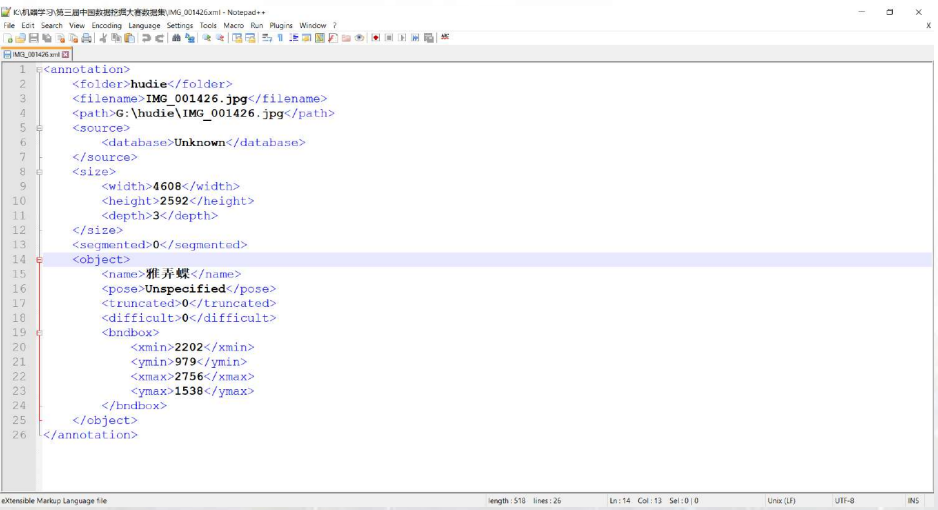


扩充数据集的方法

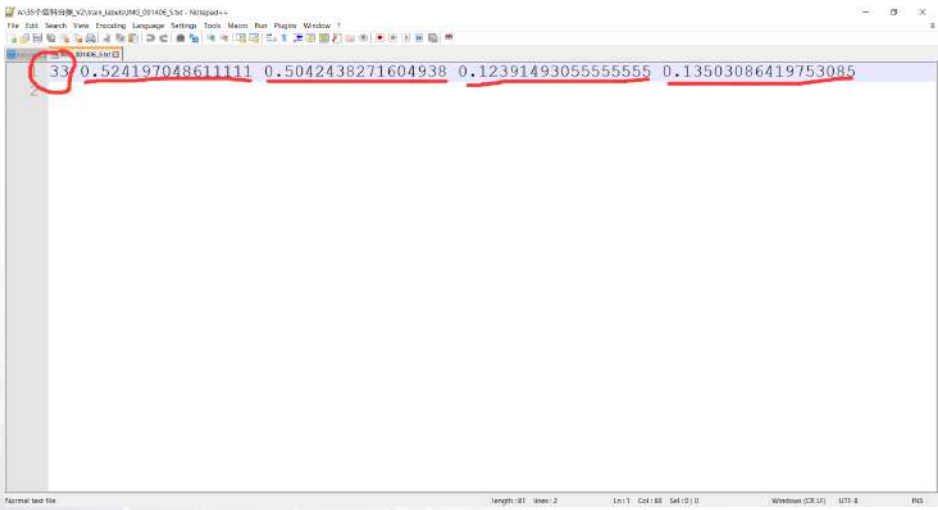
1. 仿射变换
2. 加噪
3. 旋转
4. 提高对比度



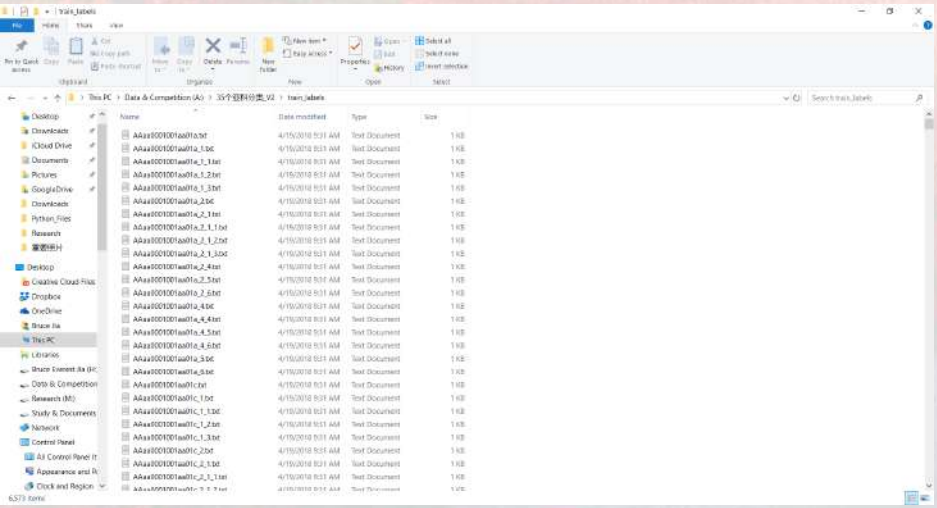
制作训练集&测试集标注(label)及其路径

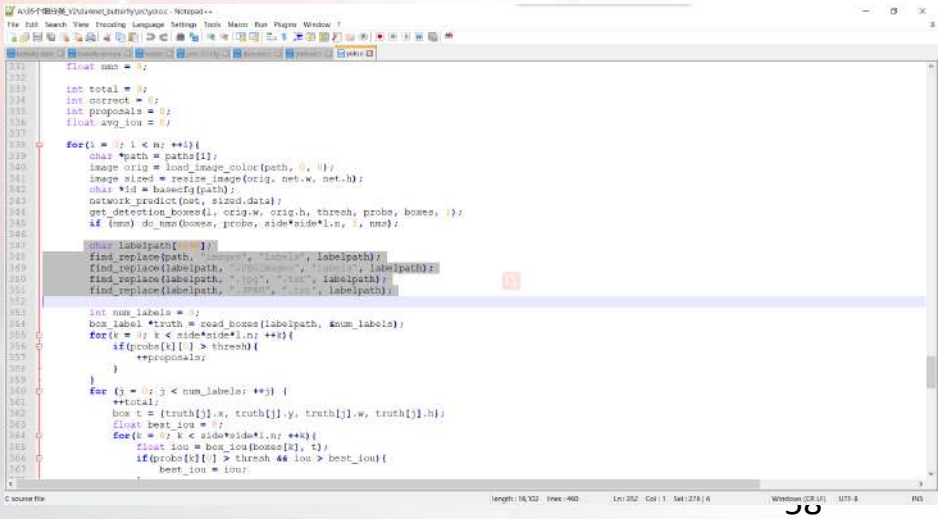


```
<annotation>
    <folder>hudie</folder>
    <filename>IMG_001426.jpg</filename>
    <path>G:\hudie\IMG_001426.jpg</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>4608</width>
        <height>2592</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>雅弄蝶</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>2202</xmin>
            <ymin>979</ymin>
            <xmax>2756</xmax>
            <ymax>1538</ymax>
        </bndbox>
    </object>
</annotation>
```



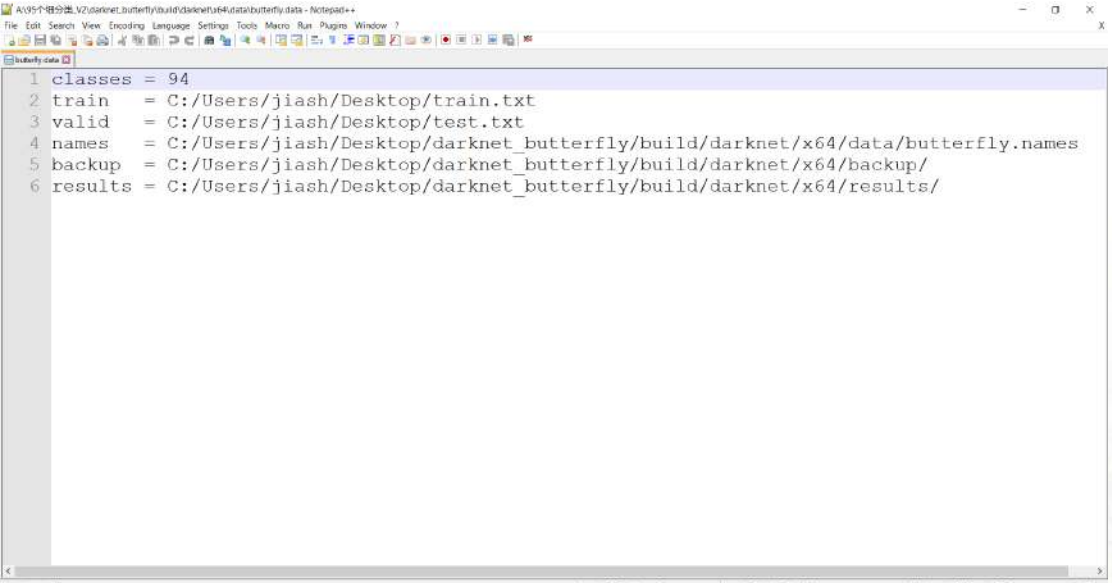
```
33 0.5241970486111111 0.5042438271604938 0.12391493055555555 0.13503086419753085
```



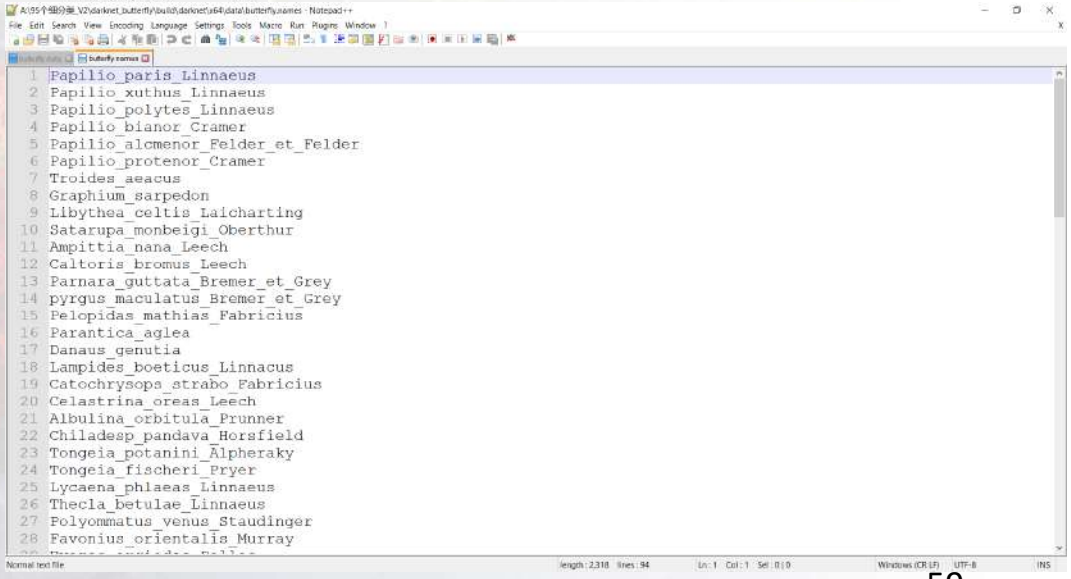


```
    float nms = 0;

    int total = 0;
    int correct = 0;
    int proposals = 0;
    float avg_iou = 0;

    for(i = 0; i < m; ++i){
        char *path = paths[i];
        image orig = load_image_color(path, 0, 0);
        image sized = resize_image(orig, net.w, net.h);
        char *id = basecfg(path);
        network_predict(net, sized.data);
        get_detection_boxes(l, orig.w, orig.h, thresh, probs, boxes, 1);
        if (nms) do_nms(boxes, probs, side*side*l.n, 1, nms);

        char labelpath[4096];
        find_replace(path, "images", "labels", labelpath);
        find_replace(labelpath, "JPEGImages", "labels", labelpath);
        find_replace(labelpath, ".jpg", ".txt", labelpath);
        find_replace(labelpath, ".JPEG", ".txt", labelpath);

        int num_labels = 0;
        box_label *truth = read_boxes(labelpath, &num_labels);
        for(k = 0; k < side*side*l.n; ++k){
            if(probs[k][0] > thresh){
                ++proposals;
            }
        }
        for (j = 0; j < num_labels; ++j) {
            ++total;
            box t = {truth[j].x, truth[j].y, truth[j].w, truth[j].h};
            float best_iou = 0;
            for(k = 0; k < side*side*l.n; ++k){
                float iou = box_iou(boxes[k], t);
                if(probs[k][0] > thresh && iou > best_iou){
                    best_iou = iou;
```

写明 分类个数、训练集与测试集路径、保存模型的文件路径

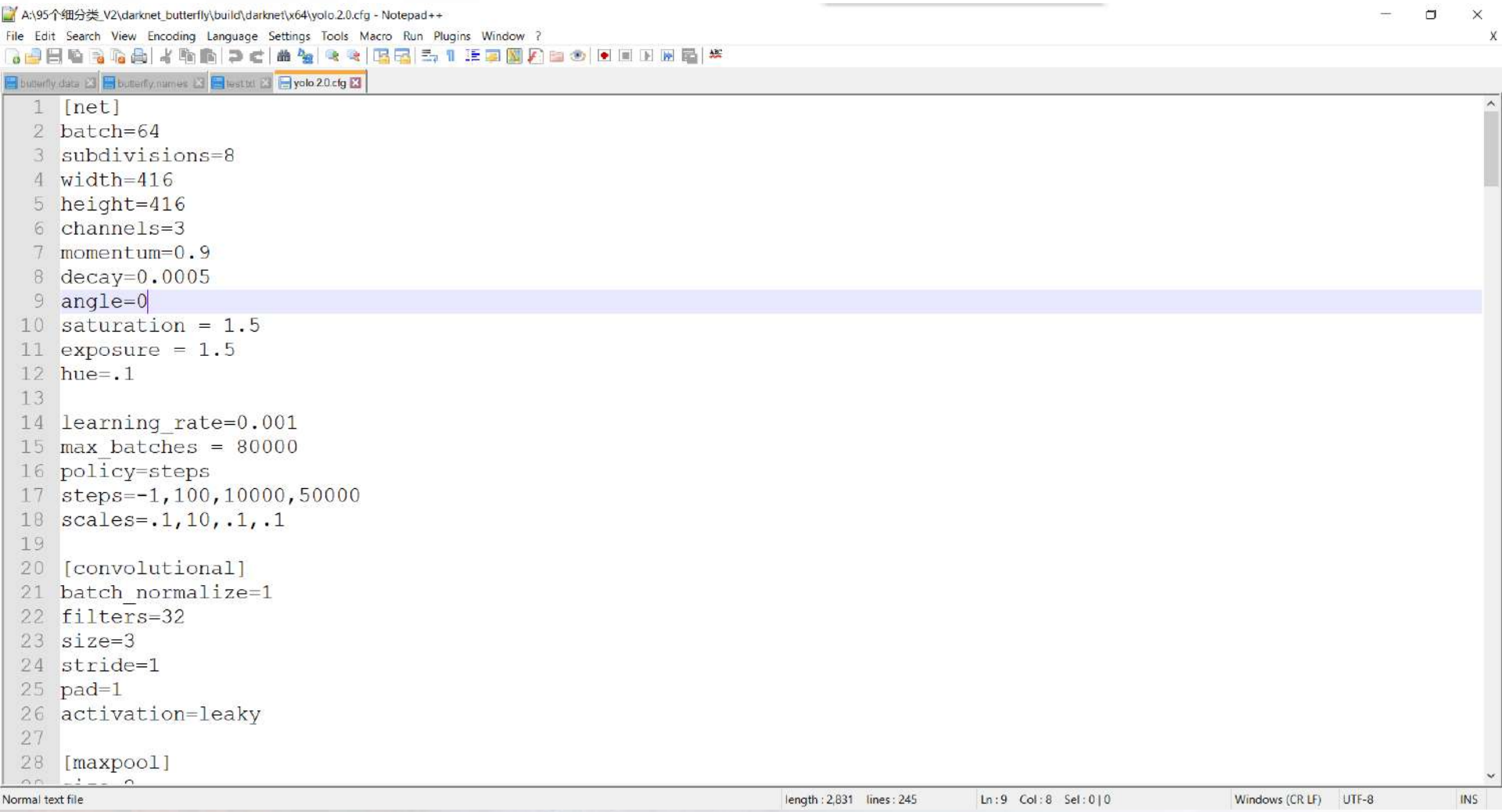


```
classes = 94
train   = C:/Users/jiash/Desktop/train.txt
valid   = C:/Users/jiash/Desktop/test.txt
names   = C:/Users/jiash/Desktop/darknet_butterfly/build/darknet/x64/data/butterfly.names
backup  = C:/Users/jiash/Desktop/darknet_butterfly/build/darknet/x64/backup/
results = C:/Users/jiash/Desktop/darknet_butterfly/build/darknet/x64/results/
```

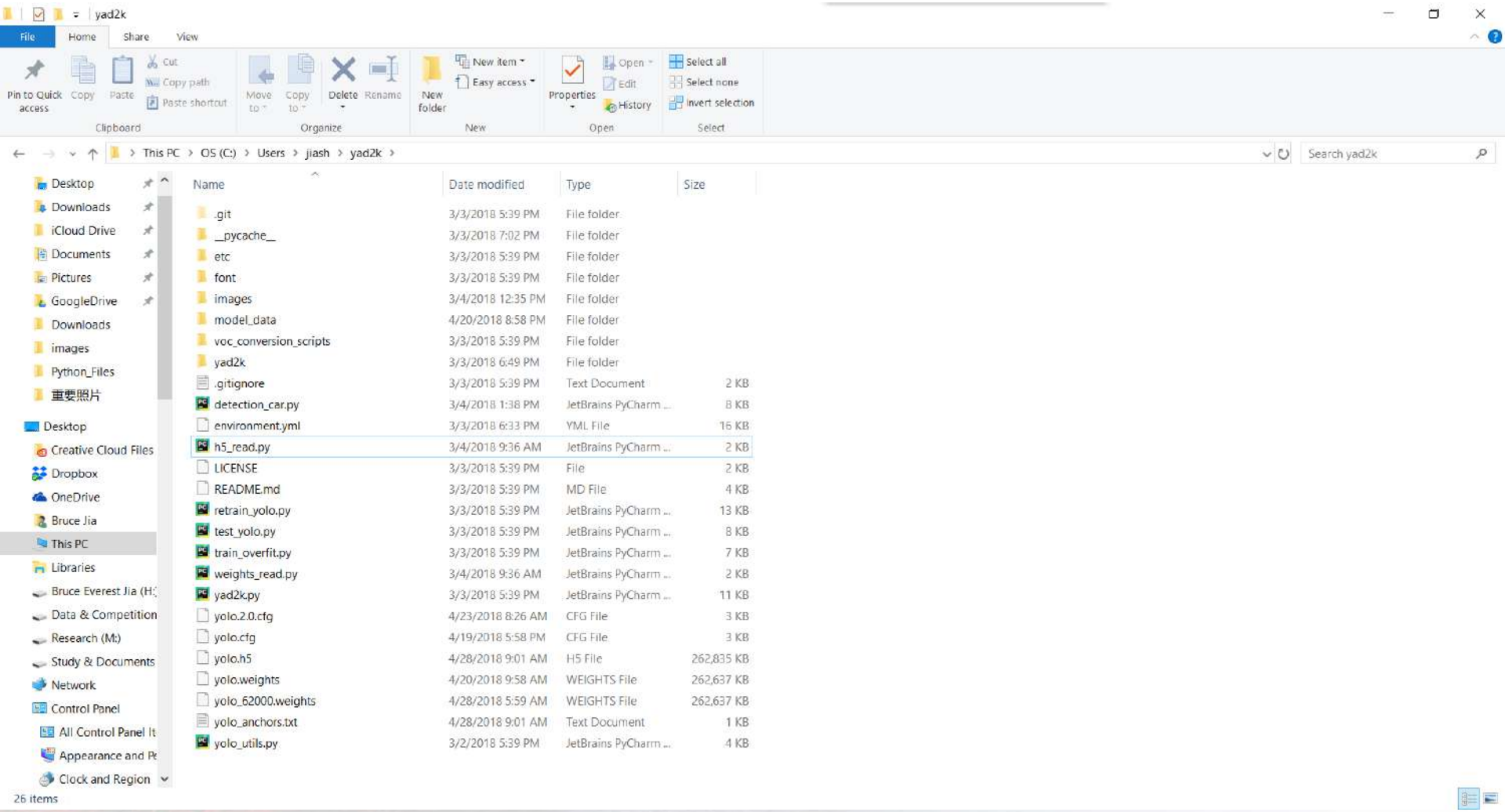


```
Papilio_paris_Linnaeus
Papilio_xuthus_Linnaeus
Papilio_polytes_Linnaeus
Papilio_bianor_Cramer
Papilio_alcmenor_Felder_et_Felder
Papilio_protenor_Cramer
Troides_aeacus
Graphium_sarpedon
Libythea_celtis_Laicharting
Satarupa_monbeigi_Oberthur
Ampittia_nana_Leech
Caltoris_bromus_Leech
Parnara_guttata_Bremer_et_Grey
pyrgus_maculatus_Bremer_et_Grey
Pelopidas_mathias_Fabricius
Parantica_aglea
Danaus_genutia
Lampides_boeticus_Linnacus
Catochrysops_strabo_Fabricius
Celastrina_oreas_Leech
Albulina_orbitula_Prunner
Chiladesp_pandava_Horsfield
Tongeia_potanini_Alpheraky
Tongeia_fischeri_Pryer
Lycaena_phlaeas_Linnaeus
Thecla_betulae_Linnaeus
Polyommatus_venus_Staudinger
Favonius_orientalis_Murray
```

设计网络结构：.cfg文件



.cfg + .weights ➔ .h5



.h5已训练模型的的导入问题 http://blog.csdn.net/overfit/article/details/78744947

Spyder (Python 3.5)

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\jiash\Desktop\Windows_YOLO_v2_608_Object_Detection

Editor - C:\Users\jiash\Desktop\Windows_YOLO_v2_608_Object_Detection\detection_car.py

MIND_Draft_1.py | MIND_draft3.py | MIND_draft4.py | MIND_Draft5.py | MIND_Draft7.py | MIND_Draft8.py | detection_car.py*

```
136     Runs the graph stored in "sess" to predict boxes for "image_file". Prints and plots the preditions.
137
138     Arguments:
139     sess -- your tensorflow/Keras session containing the YOLO graph
140     image_file -- name of an image stored in the "images" folder.
141
142     Returns:
143     out_scores -- tensor of shape (None, ), scores of the predicted boxes
144     out_boxes -- tensor of shape (None, 4), coordinates of the predicted boxes
145     out_classes -- tensor of shape (None, ), class index of the predicted boxes
146
147     Note: "None" actually represents the number of predicted boxes, it varies between 0 and max_boxes.
148     """
149
150     out_scores, out_boxes, out_classes = sess.run([scores, boxes, classes],feed_dict={yolo_model.input: image_data,K.lear
151     print('Found {} boxes for {}'.format(len(out_boxes), image_file))
152     colors = generate_colors(class_names)
153     draw_boxes(image, out_scores, out_boxes, out_classes, class_names, colors)
154     image.save(os.path.join("out", image_file), quality=90)
155     output_image = scipy.misc.imread(os.path.join("out", image_file))
156     imshow(output_image)
157
158     return out_scores, out_boxes, out_classes
159
160 sess = K.get_session()
161 class_names = read_classes("model_data\\coco_classes.txt")
162 anchors = read_anchors("model_data\\yolo_anchors.txt")
163
164 #%%
165 image, image_data, image_file, image_input = input_photo_shape(sess, "dog.jpg")
166 image_shape = np.array(image_input.shape[:2]).astype(np.float32)
167
168 yolo_model = load_model("model_data\\yolo.h5")
169 #yolo_model.summary()
170
171 yolo_outputs = yolo_head(yolo_model.output, anchors, len(class_names))
172 scores, boxes, classes = yolo_eval(yolo_outputs, image_shape)
173
174 out_scores, out_boxes, out_classes = predict(sess, image_file)
175
176
```

Help

Source Console | Object

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

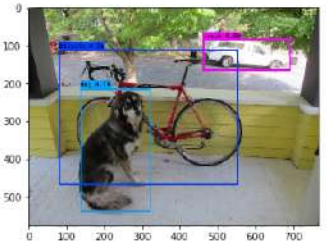
Help can also be shown automatically after

Variable explorer | File explorer | Help

IPython console

Console 8/A

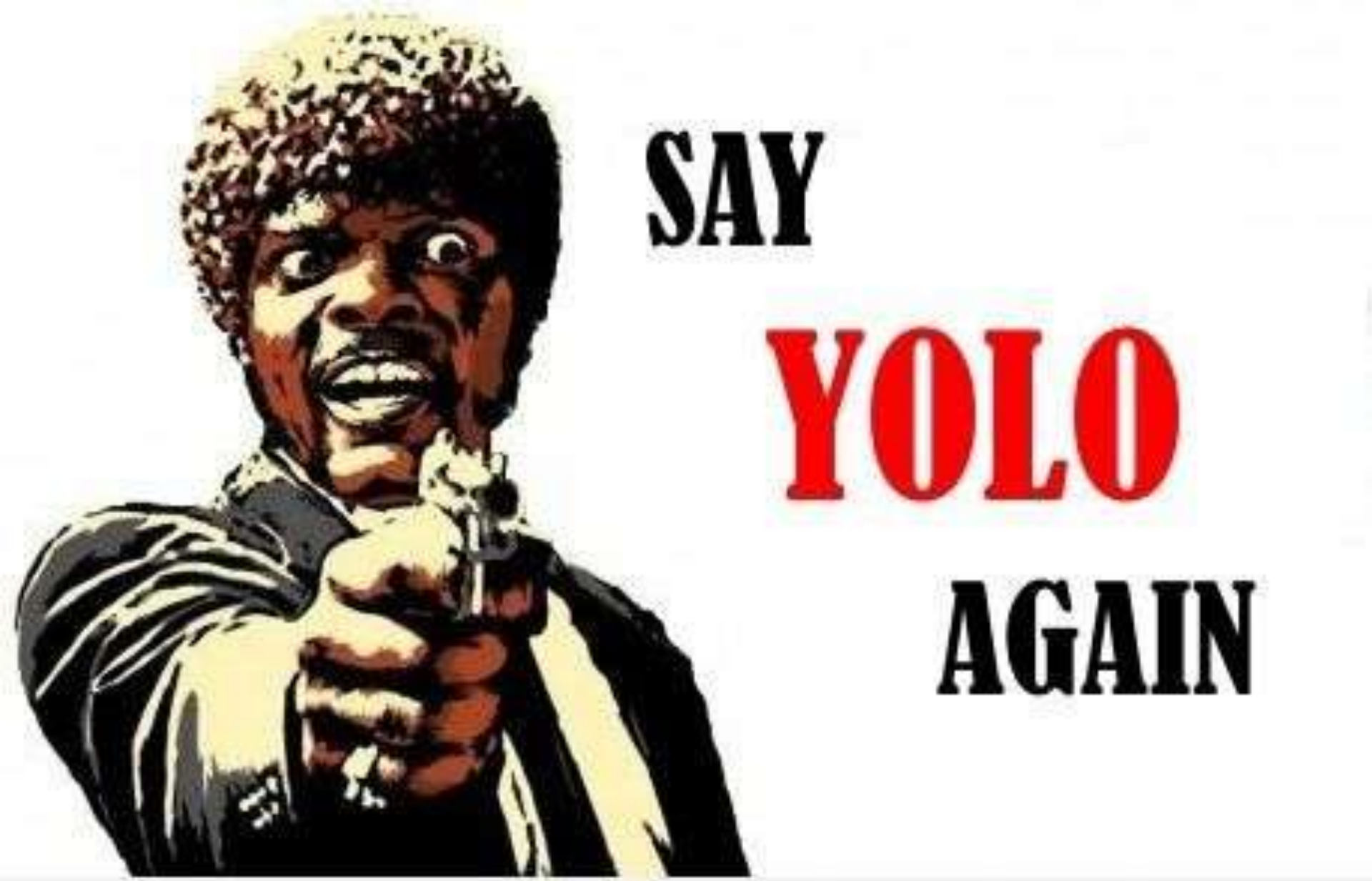
```
   ...: scores, boxes, classes = yolo_eval(yolo_outputs, image_shape)
   ...:
   ...: out_scores, out_boxes, out_classes = predict(sess, image_file)
S:\Anaconda\envs\tensorflow\lib\site-packages\keras\engine\saving.py:269:
UserWarning: No training configuration found in save file: the model was *not*
compiled. Compile it manually.
  warnings.warn('No training configuration found in save file: '
Found 3 boxes for dog.jpg
dog 0.78 (137, 214) (323, 540)
truck 0.80 (462, 82) (694, 168)
bicycle 0.84 (81, 112) (554, 469)
__main__:160: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
```




```
In [3]:
```

IPython console | History log

Permissions: RW | End-of-lines: CRLF | Encoding: ASCII | Line: 166 | Column: 65 | Memory: 62 %



YOLO-V3介绍：http://www.sohu.com/a/226481716_473283

YOLO-V3

首先在性能上，YOLOv3还是一如既往的快速与准确，虽然YOLOv1性能不及SSD、YOLOv2在性能略超SSD算法，但是YOLOv3的出现可以说是一举击败SSD成为目前开源通用目标检测算法的领头羊。

1. 使用Darknet-53网络
2. 从3个尺寸进行检测
3. 善于检测更小的物品
4. Anchor Box的选择
5. 图像有更多的边界框
6. 损失函数发生变化
7. 不再使用Softmax进行分类

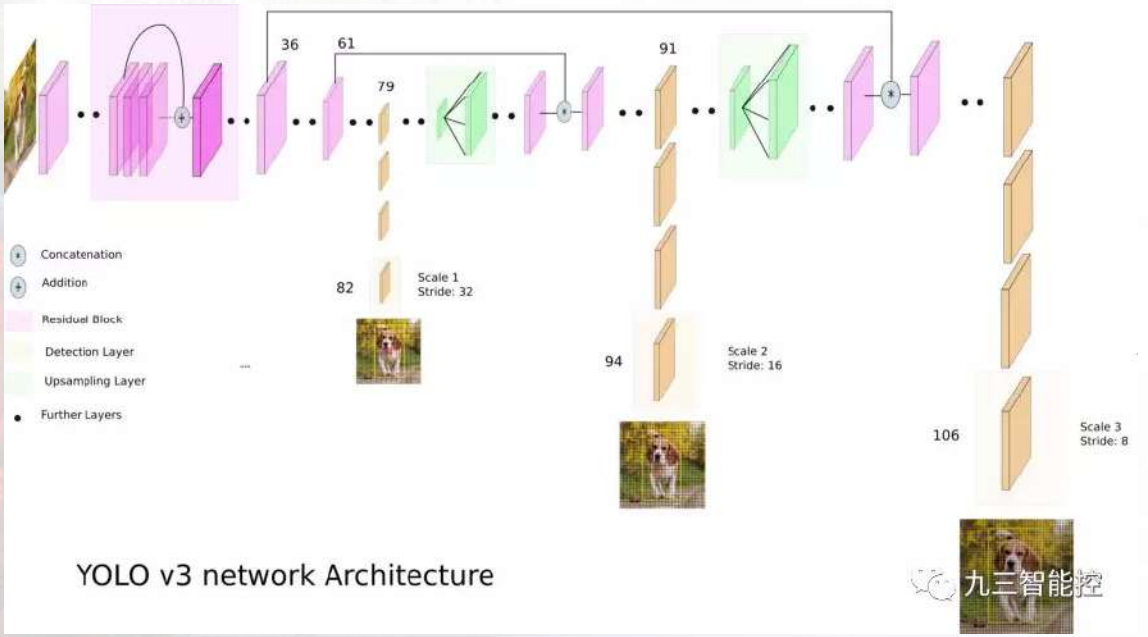
论文地址：https://pjreddie.com/media/files/papers/YOLOv3.pdf
代码地址：https://pjreddie.com/darknet/yolo/

使用Darknet-53网络

首先，YOLO v3融合了残差网络和上采样等先进的算法思想。其次，对Darknet进行了改进，将网络深度扩展到53层，并在Imagenet上进行了预训练，物体识别则通过另外的53层网络来实现，YOLO v3完整的卷积底层结构共有106层，这也是YOLO v3较YOLO v2训练慢的原因。

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Table 1. **Darknet-53.**



从3个尺寸进行检测

YOLO-V3最显著的特点是在三个不同的尺度上对图像展开检测。YOLO-V1是一个全卷积神经网络，其通过在特征图(feature map)上应用1×1的卷积核来产生最终输出结果。在YOLO-V3中，物体检测则是通过在网络不同位置、不同尺度的特征图应用1×1卷积核来实现的。

YOLO-V3在3个不同尺寸的特征图上进行预测，准确的说，即是对输入图像分别进行32、16和8倍的降采样后再进行预测。

第一次检测发生在第82层。对于前面的81层网络，图像被逐渐的降采样，也就是说对于第81层，其降采样步长为32。假设输入的图像尺寸为416×416，那么输出的特征图尺寸为13×13。然后应用1×1卷积核进行第一次检测，输出尺寸为13×13×255的检测特征层。

第二次检测，输出的特征层尺寸为26×26×255。然后对第79层的特征图先进行一系列卷积处理，再进行2倍的上采样，将尺寸扩展到26×26。然后再将其与第61层网络在第3个维度上进行拼接。对拼接后的特征层应用一系列的1×1卷积核进行处理，将第61层的特征进一步融合。

第三次检测的处理方法相同，即将第36层和第91层的特征进行融合，并在第106层进行第三次特征检测，输出的特征层尺寸为52×52×255。

善于检测更小的物品

- YOLO-V3将上采样层跟前面的层进行拼接融合，该处理方法保留了图像中更细致的特征，这一点对检测微小尺寸物品很有帮助。

- 对应的，YOLO-V3的13×13特征层适用于检测大尺寸物品，26×26特征层适用于检测中等尺寸物品，52×52特征层则用来检测小尺寸物品。

Anchor Box的选择

YOLO-V3总共使用了9个锚箱(anchor box)，每个尺度各3个锚箱。

如果你在自己的数据集上应用YOLO，你需要使用K平均聚类方法自行创建9个锚箱。

然后将各锚箱按降序排列，最大的3个应用到第一类特征层，接下来的3个对于第二类特征层，最后3个处理最后的特征层。

- YOLO-V1 2个Archor Box
- YOLO-V2 5个Archor Box
- YOLO-V3 9个Archor Box

损失函数发生变化

YOLO-V2损失函数的后三项是平方误差，而YOLO-V3则更改为交叉熵误差项(Cross Entropy Loss)，也就是说YOLO-V3的物品置信度和分离预测使用的是逻辑回归(Logistic Regression)算法。

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$= -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)}\log h_\theta(x^{(i)}) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

http://blog.csdn.net/syyyy712

06

目前计算机视觉领域的挑战

Challenges & Problems

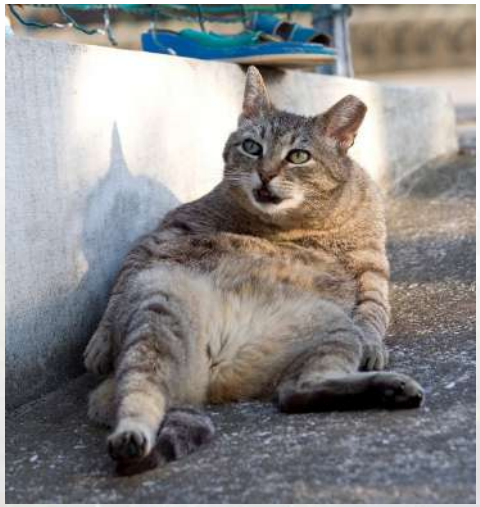
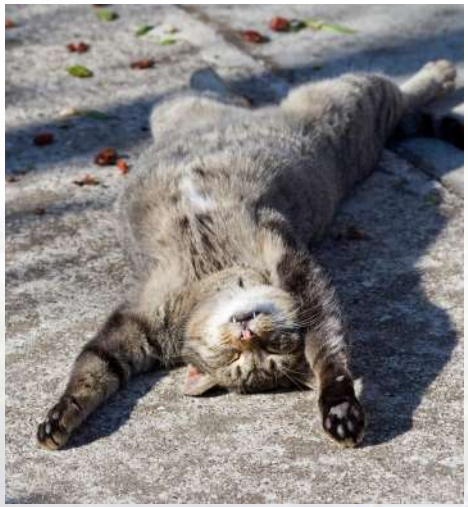
1. Illumination 亮度&照度
2. Deformation 变形
3. Occlusion 遮挡
4. Background Clutter 背景混乱
5. Intraclass Variation 多物体极高相似度
6. Small Object Detection 小物体识别

Challenges:亮度&照度





Challenges:变形

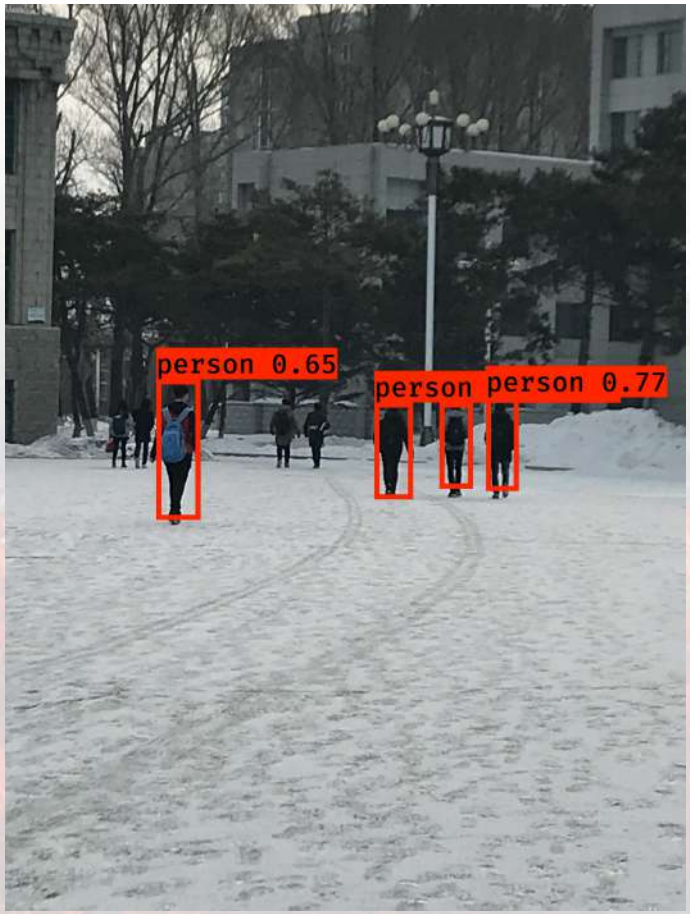


Challenges:遮挡

Challenges:背景混乱

Challenges:多物体极高相似度

Challenges: 小物体目标识别



07

个人未来研究兴趣与方向

目前个人感兴趣的话题 & 学习方向：

1. 远距离小物体目标识别 （Small Object Detection）细粒度
2. YOLO预测输出带角度的Bounding Box（再研究）
3. Image Segmentation 图像分割（先实现Musk-RCNN）
4. Visual Tracking 动态图像追踪（先下载好数据集）
5. One-Shot Learning 一眼学习（用到运动想象脑电数据分类）
6. Robotics Imitation Learning 机器人模仿学习（V-REP）

08

Deep Mind大牛对未来的展望

Frontiers： (Nando de Freitas, Oxford University & Deep Mind)

1. Reinforcement learning 强化学习
2. Meta learning 元学习
3. Imitation 模仿学习
4. Robotics 机器人
5. Concepts and abstraction 概念与抽象
6. Awareness and consciousness 感知与意识
7. Causal reasoning 因果推理

感谢大家聆听，送上C罗经典倒挂金钩！