

# yoloV1原理分析及代码实现

yolo是计算机视觉方面进行目标检测的算法

计算机视觉能解决的问题有

1分类

2检测

3分割

分类通过图像判断出有哪些类别但是不知道位置

引入位置把单独类别进行分类加定位

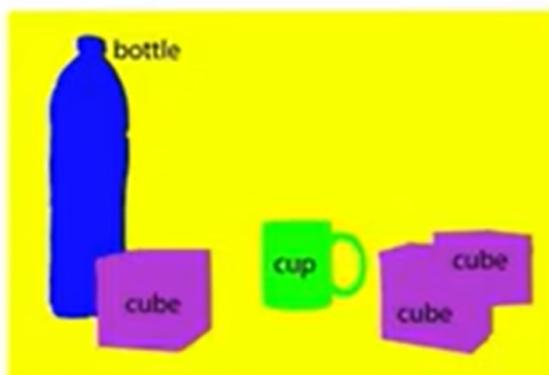
多个类别的多个物体进行分类以及框出来即目标检测问题

精确到更细粒度到抠图部分的即为分割

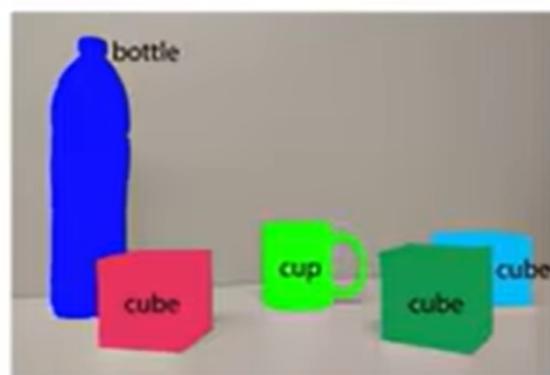
分割分为了语义分割和实例分割

语义分割对每一个像素分类只分类

实例分割将同一个类别的不同实例再进行分割



(c) Semantic segmentation



(d) Instance segmentation

例子

除此之外计算机视觉还解决了很多问题



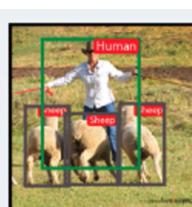
## Image Classification

Classify an image based on the dominant object inside it.  
datasets: MNIST, CIFAR, ImageNet



## Object Localization

Predict the image region that contains the dominant object. Then image classification can be used to recognize object in the region  
datasets: ImageNet



## Object Recognition

Localize and classify all objects appearing in the image. This task typically includes: proposing regions then classify the object inside them.  
datasets: PASCAL, COCO



## Semantic Segmentation

Label each pixel of an image by the object class that it belongs to, such as human, sheep, and grass in the example.  
datasets: PASCAL, COCO



## Instance Segmentation

Label each pixel of an image by the object class and object instance that it belongs to.  
datasets: PASCAL, COCO

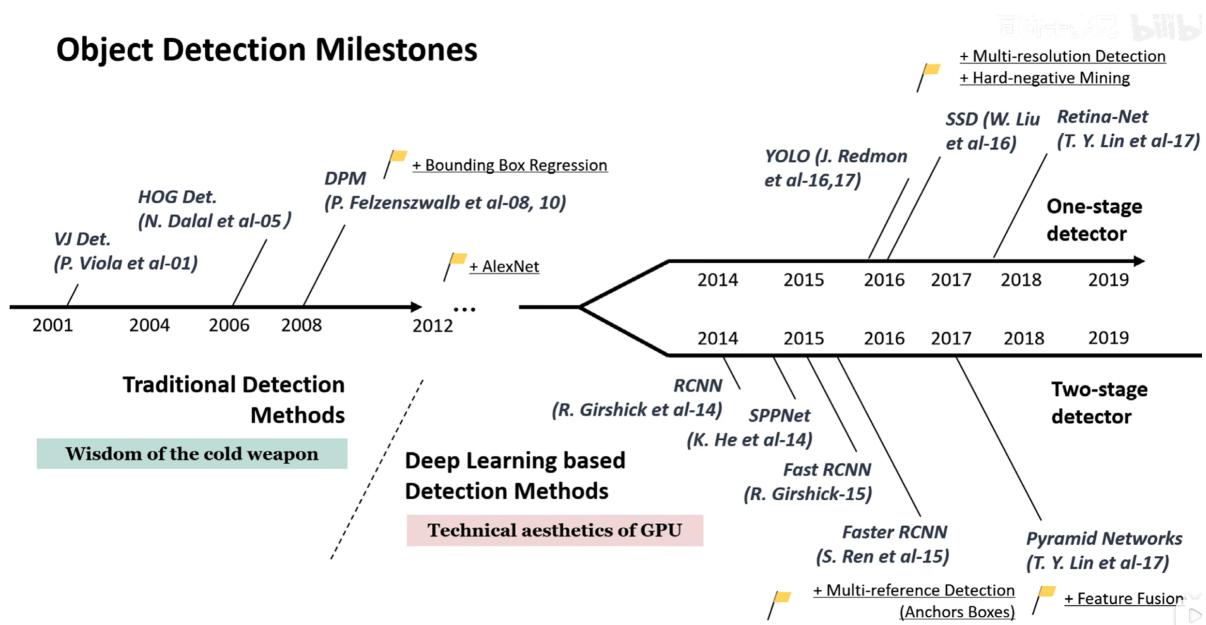


## Keypoint Detection

Detect locations of a set of predefined keypoints of an object, such as keypoints in a human body, or a human face.  
datasets: COCO

目标检测的两种模式

## Object Detection Milestones



单阶段模型以及两阶段模型

两阶段先提取候选框再逐一甄别

单阶段直接把全图输入

是端到端的模型，通过一次前向推断得到bbox定位以及分类结果

**单阶段模型:** 单阶段模型指的是在一次前向传播中，直接从输入数据得到最终的预测结果。这类模型通常较为简单、直接。这个MNIST模型就是一个典型的单阶段模型，因为它从输入图像直接输出分类结果，没有分成多个阶段或步骤。

**双阶段模型:** 双阶段模型通常包括两个主要步骤。第一个阶段生成一组初步的候选区域或特征（如目标检测中的Region Proposal Network），第二个阶段对这些候选区域或特征进行进一步的处理或分类。这类模型通常较为复杂，适用于一些需要精细处理的任务，如目标检测、实例分割等。

## 预测阶段（前向推断）

当模型已经训练好

输入是448\* 448 \*3的图像

输出是7\* 7 \*30维的张量

每一个30维的向量代表了一个grid cell的信息，一共有7\*7个向量

这30维之中有10个维度表示一个grid cell生成的两个bounding box每一个都有 (x, y, h, w, c)

另外20维代表了类别

首先将图片划分成s\*s个grid cell

每一个grid cell 能够预测b个bounding box (预测框)

bounding box的中心点落在该grid cell里说明该bounding box是由该grid cell生成的

预测框包含了四个定位坐标：中心点坐标以及框的宽高

以及另一个变量是不是object的置信度（使用线的粗细进行表示）

(x, y, h, w, c)

每一个grid cell预测一组条件类别的概率

假设已经包含物体时是某一个类别的概率

每一个bounding box的置信度乘以类别的条件概率，就能获得每一个bounding box各类别的概率（即概率论中的条件概率）

而后后处理会进行

1.低置信度过滤

2.非极大值抑制

## 预测阶段（后处理）

已经训练好的模型

低置信度过滤

把重复的过滤框去掉

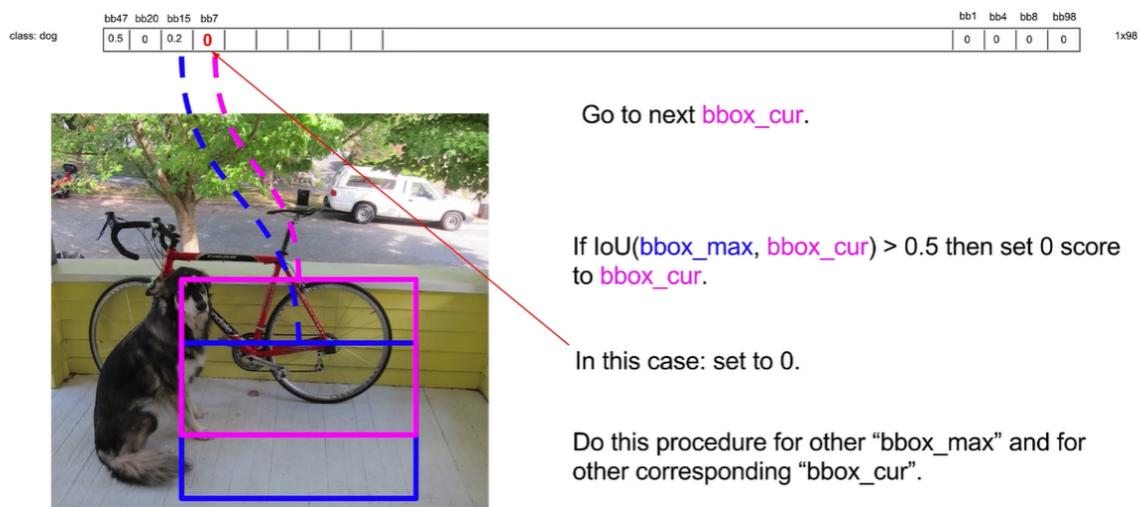
即非极大值抑制NMS

将张量变成最终的检测结果

将获得的98个竖条每一维进行查看，设置一个阈值将小于一部分的舍去再进行排序

再对排序后的部分进行非极大值抑制

如果两个框的交并比大于0.5则认为是在重复识别，将较小的设置为0



例如

最终结果是稀疏矩阵

不为零的类别索引找出来进行可视化

## 训练阶段（反向传播）

监督学习通过梯度下降和反向传播使得损失函数最小化

人工通过labelme或者labelimage进行画框

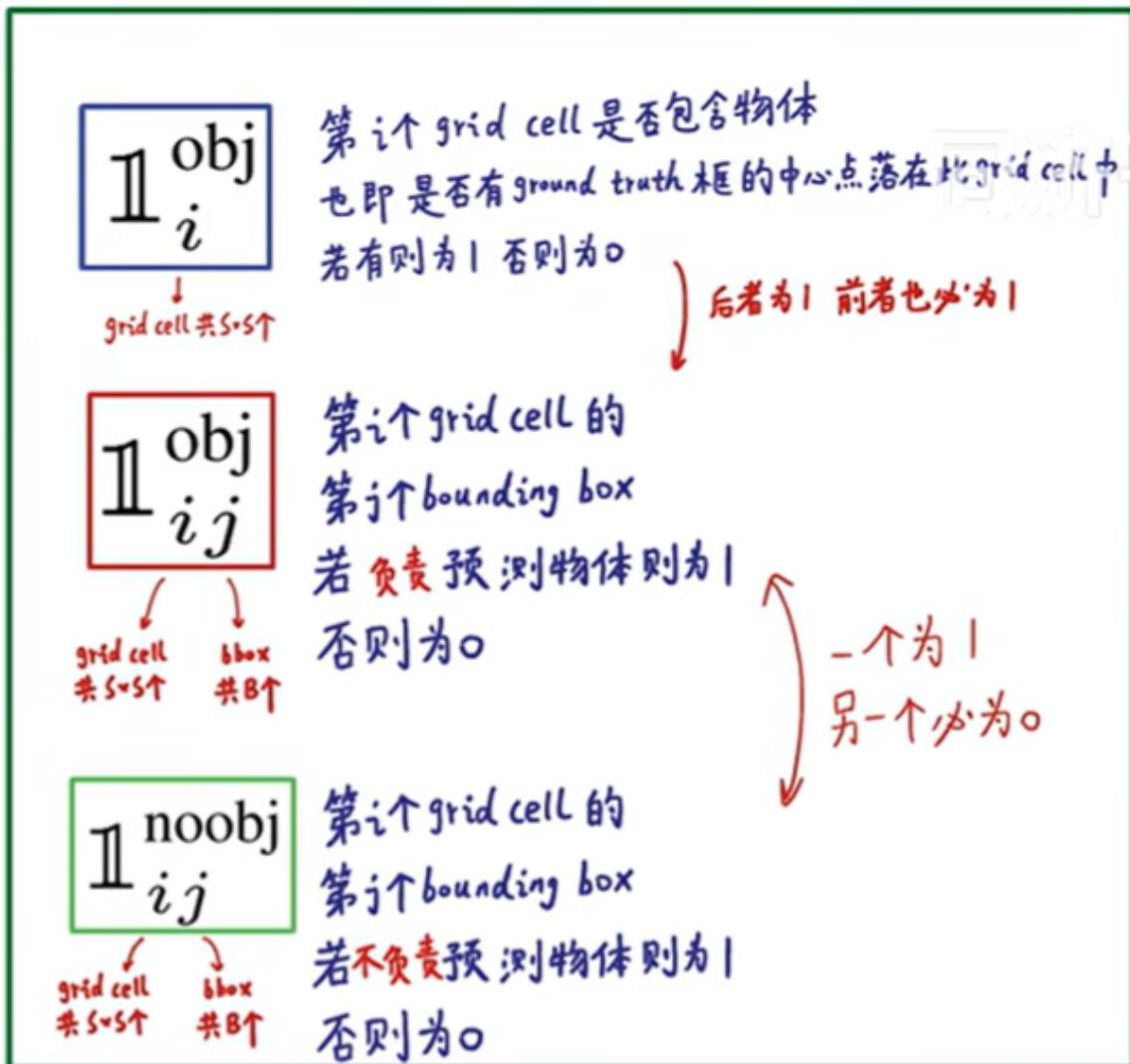
画出的groundchoice作为标准答案

由bounding box来进行拟合这个标准答案

## loss function

回归问题损失函数，预测出一个连续的值，把这个值和标注值进行比较，越接近越好

yolo将目标检测问题当作回归问题解决



loss function:

The diagram illustrates the loss function (3) with handwritten annotations explaining each term:

- 负责检测物体的bbox 中心点定位误差**:  $\lambda_{\text{coord}} \sum_{i=0}^S \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2]$
- 负责检测物体的bbox 宽高定位误差**:  $+ \lambda_{\text{coord}} \sum_{i=0}^S \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$
- 负责检测物体的bbox Confidence 误差**:  $+ \sum_{i=0}^S \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2$
- 不负责检测物体的bbox Confidence 误差**:  $+ 0.5 \lambda_{\text{noobj}} \sum_{i=0}^S \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2$
- 类别预测误差**:  $+ \sum_{i=0}^S \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \quad (3)$

Annotations explain the terms:
 

- $\mathbb{1}_{ij}^{\text{obj}}$ : 第*i*个grid cell 的所有 bounding box 的负责检测物体的bbox
- $\mathbb{1}_{ij}^{\text{noobj}}$ : 第*i*个grid cell 的所有 bounding box 的不负责检测物体的bbox
- $C = \text{Pr}(\text{object}) \times \text{IoU}^{\text{pred}}$ : 计算这个 bbox 与 ground truth 的 IOU
- $C_i$ : 第*i*个grid cell 的所有 bounding box 的负责检测物体的bbox 的预测值
- $\hat{C}_i$ : 第*i*个grid cell 的所有 bounding box 的负责检测物体的bbox 的标签值
- $\mathbb{1}_i^{\text{obj}}$ : 第*i*个grid cell 的所有负责检测物体的bbox
- $\mathbb{1}_i^{\text{noobj}}$ : 第*i*个grid cell 的所有不负责检测物体的bbox

## 位置误差

### 1. 负责检测物体的bounding box中心点定位误差

### 2. 负责检测物体的bounding box宽高定位误差

小框对误差更加敏感

大框损失函数更小

小框定位出错就更加明显，大框区别没有那么大

## 置信度误差

### 1. 负责检测物体bounding box的置信度误差

### 2. 不负责检测物体bounding box的置信度误差

不负责的最好都为0，负责的最好都为1

## 分类误差

负责检测物体的grid cell的分类误差

loss function:

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

坐标预测

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

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

置信度预测  
(有中心点方格损失)

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

+  
无中心点方格损失)

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

类别预测

CSDN @天真

## ✓ YOLO-V1



优点：快速，简单！

问题1：每个Cell只预测一个类别，如果重叠无法解决

问题2：小物体检测效果一般，长宽比可选的但单一

## 代码辅助理解

网络结构部分

```
[net]
# Testing
batch=1
subdivisions=1

# Training
# batch=64
# subdivisions=8
height=448
width=448
channels=3
momentum=0.9
decay=0.0005
```

```
saturation=.75
exposure=.75
hue = .1

learning_rate=0.0005
policy=steps
steps=200,400,600,800,20000,30000
scales=2.5,2,2,2,.1,.1
max_batches = 40000

[detection]
classes=20
coords=4
rescore=1
side=7
num=2
softmax=0
sqrt=1
jitter=.2

object_scale=1
noobject_scale=.5
class_scale=1
coord_scale=5
```

training部分

输入448\*448 \* 3的图像

动量设置为0.9，用于加速梯度下降

decay设置为0.0005，用于正则化，防止过拟合

```
height=448
width=448
channels=3
momentum=0.9
decay=0.0005
```

数据增强的参数，表示了饱和度、曝光度和色调的变化范围

```
saturation=.75
exposure=.75
hue = .1
```

学习率和调整学习率的策略

初始0.0005

按照迭代次数到达steps对应的时候

进行学习率调整

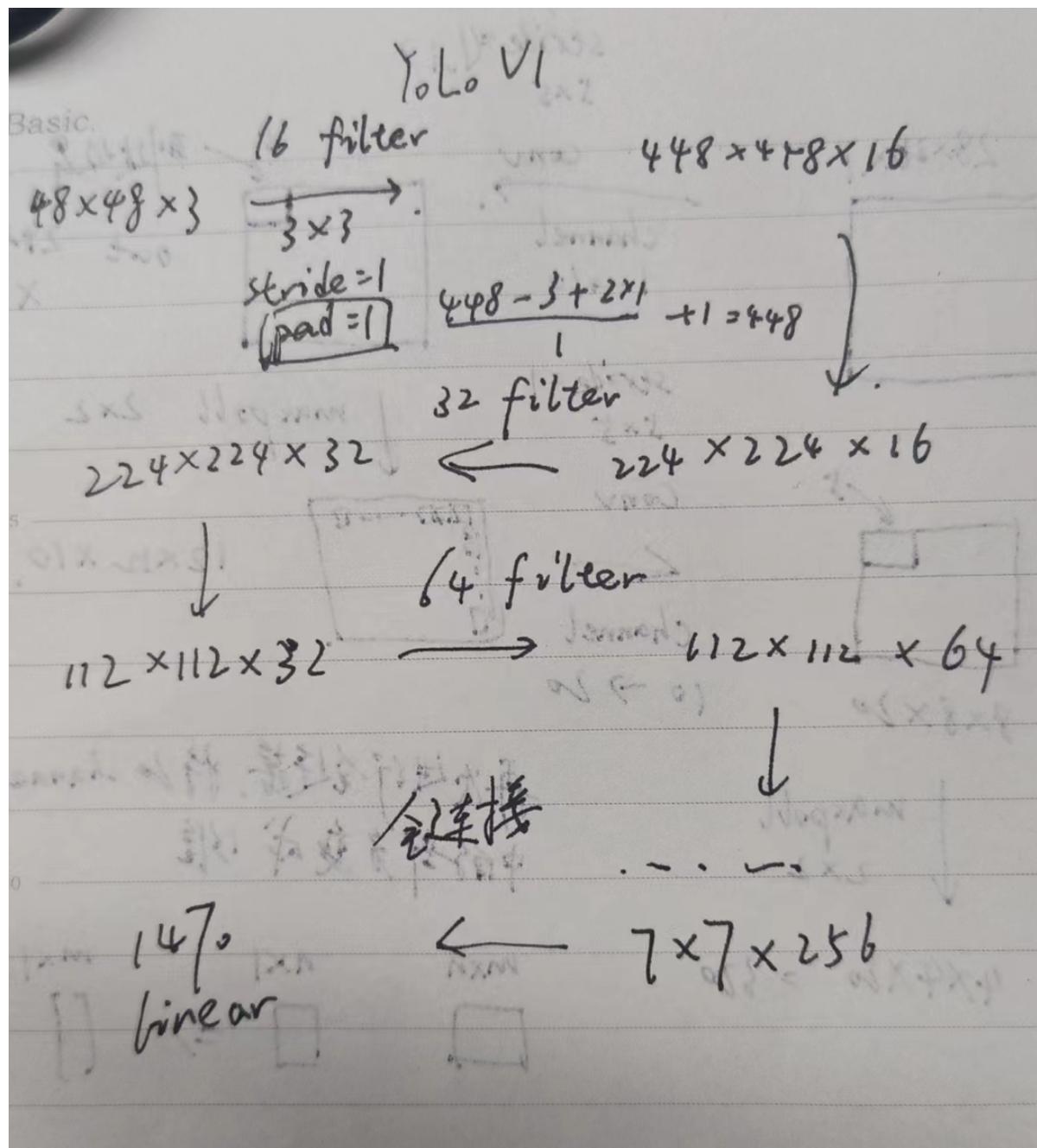
最大迭代次数40000

```

learning_rate=0.0005
policy=steps
steps=200,400,600,800,20000,30000
scales=2.5,2,2,2,.1,.1
max_batches = 40000

```

卷积和池化和全连接



检测层

[detection] 部分定义了检测层：

- `classes=20`: 检测的类别数量。
- `coords=4`: 每个边界框的坐标数 ( $x, y, w, h$ ) 四个变量
- `rescore=1`: 是否对边界框进行重评分。1表示在训练过程中会对边界框的置信度进行重新评分，进一步优化边界框的位置和尺寸
- `side=7`: 网格划分尺寸 ( $7 \times 7$ )。将图片进行网格分成 $7 \times 7$
- `num=2`: 每个网格预测的边界框数量。每个grid cell预测两个bounding box

- `softmax=0`: 是否使用softmax分类。不使用softmax分类，而是使用多标签分类。对于目标检测任务，通常不需要softmax，因为一个边界框可能同时包含多个类别
- `sqrt=1`: 是否对边界框宽高取平方根。有助于稳定训练过程，因为对大尺度的边界框预测会更加准确
- `jitter=.2`: 数据增强的抖动参数。对输入图像进行随机抖动，增强模型的泛化能力

损失函数权重

`object scale`代表存在目标的时候，权重1

`noobject`代表没有的时候权重0.5，没有目标的框没有那么重要但也需要考虑

`class scale`是类别分类损失的权重

`coord scale`是位置损失的权重

## 代码完整自写

---

思路

首先定义一个通用的数据集dataset类型

再对本人的数据集进行加载到dataset中（使用data loader）

然后将数据放入网络进行训练

训练之中会用到对应参数比如batchsize和epoch等

该部分的参数可以单独提出写一个py或者在data加载时同步赋值

一个函数

```
files = sorted(glob.glob('%s/*.*' % folder_path))
```

`glob`可以直接获取路径

过程中的一些要点

1.在数据集构建的时候，由于标注的数据是json格式，需要转成txt通过换行符提取，我是直接使用json对应键值对处理，有时候不是那么方便，当然标注类型还有很多种，可以把这个固定为某一种格式，并且单独进行每一种格式和这个格式的转换，这样可扩展性更强

2.图片的标注和size是相对的，为了训练效果大部分模型都进行从原始图片填充或是删减来放缩，放缩图片的同时一定要将label也进行处理

3.对于YOLO通常输入模型的参数是标注的物体位置，种类，还有图片（转换之后的数），在参数传递和不同模块连接时需要考虑一致

4.在传入模型的时候target不是简单的label和location，会通过batchsize将一整个batch变成一块一起输入，需要用到view以及unsqueeze

5.最后结果用图表绘出更方便观察与理解

6.在loss function部分加上了平方根的计算就突然变成了NAN，但是按理来说这部分loss应该比较小的，后续试过学习率和别的优化器，也没有改变，又尝试了修改平方根切片的对应小块，还是没办法实现比较困惑，再分布调试发现问题只出现在平方根这个部分没有影响到其他内容

网上查阅了许多资料，做了各种尝试，比如调整学习率、调整batch大小、调整网络复杂度、梯度裁剪、过滤脏数据、检查是否存在除0、 $\log(0)$ ，加入BatchNormalization层等，无奈还是会出现loss变为nan的问题

我又怀疑是硬件问题调到云服务器跑历经千辛万苦还是不行

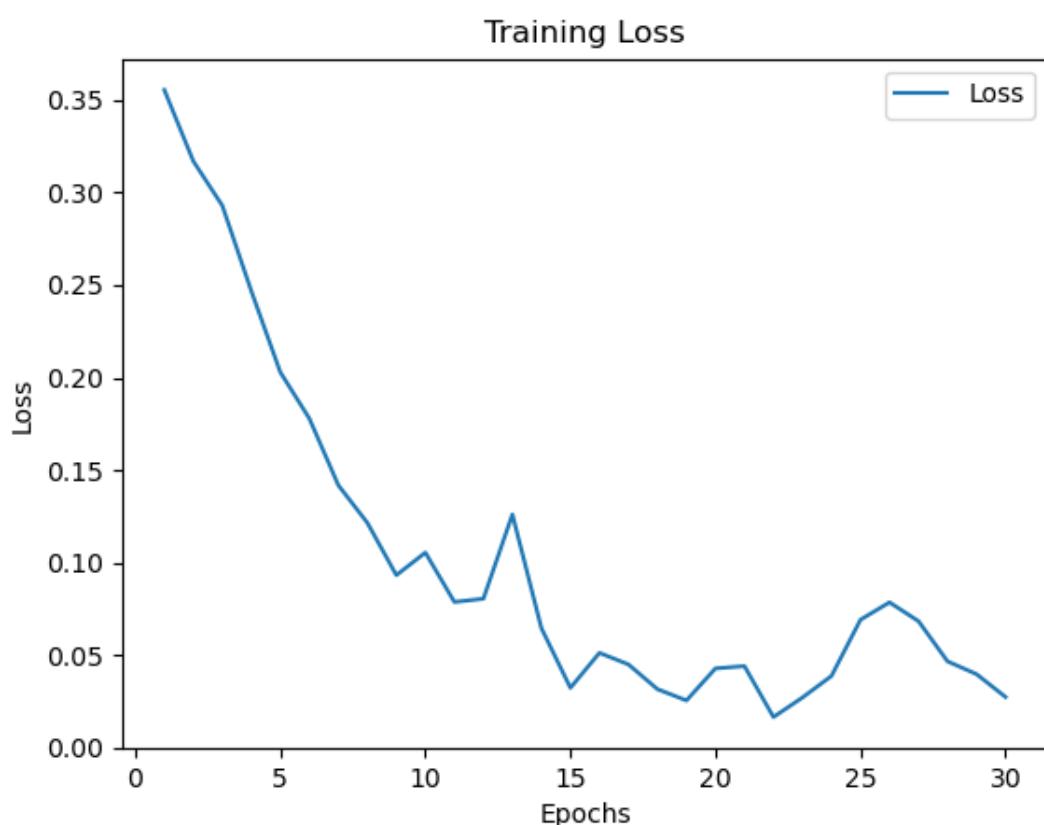
最终搜索到一个原因如下：

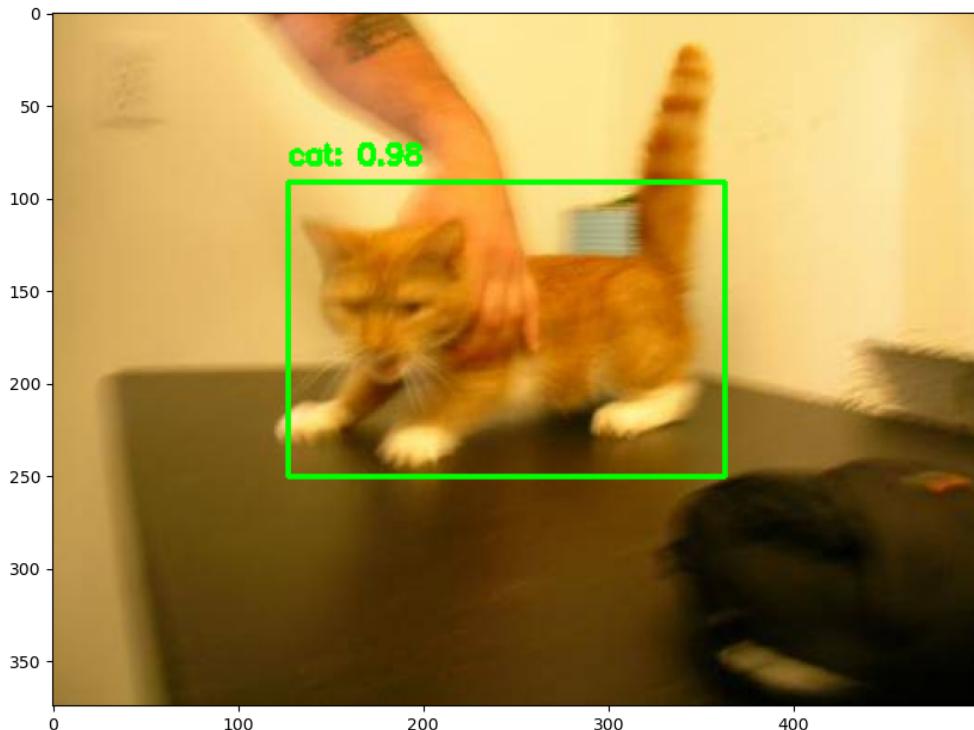
$\text{sqrt}()$ 即 $x^{1/2}$ ，在 $x=0$ 处不可导，前向传播过程中，loss的计算不会出问题，但在反向传播进行梯度计算的时候可能会遇到在0处求导的情况，这也是loss突然变为nan的原因，在 $\text{sqrt}()$ 添加一个极小数之后得到解决

在将数量级调到1的时候发现成功跑通（为什么说1也算小数量级因为我的loss没有进行归一化，初始到达了 $1e6$ 数量级所以此处1也就是跟 $1e-6$ 差不多）

然后发现有非常明显的改善，然后再对loss进行归一化

最终简易框架搞定后





当然这只是极度简易版，此处稍微有过拟合嫌疑

后续还可以优化

1. 图像组成batch以后进行裁剪拼接等
2. 网络结构完善不再是tiny版
3. 测试IOU并且优化bounding box划区范围
4. 多层非极大值抑制
5. 结合attention以及1\*1卷积
6. dropout层

具体代码部分由于工程太多此处只列举test，完整部分将上传博客

```
from Myyolo import YoloV1Model
import cv2
import torch
from datasets import CustomDataset, preprocess
```

```

import matplotlib.pyplot as plt
from PIL import Image
import numpy as np
# model = Yolov1Model()
# state_dict = torch.load(r"C:\Users\jajnw\Desktop\yolov5-
master\catndog\labels\try.pt")
# model.load_state_dict(state_dict)
#     # outputs = net(img_)
# dataset = CustomDataset(images_dir="C:/Users/jajnw/Desktop/yolov5-
master/catndog/images/train",
#                         annotations_dir="C:/Users/jajnw/Desktop/yolov5-
master/catndog/labels/json",
#                         transform=preprocess)
# print(dataset[0])
# images,targets = dataset[0]
# model.eval()
# inputs_resize_unsqueeze = torch.unsqueeze(images, 0)
# # 进行模型的评估操作，例如前向传播和计算预测结果
# output = model(inputs_resize_unsqueeze)
# batch_size = output.size(0)
# preds = output.view(batch_size, 7, 7, 30)
# print(preds.shape)
# print(output[...,20:30])

def preprocess_image(image, target_size=(448, 448)):
    """Resize and normalize the image for YOLO model."""
    image = cv2.resize(image, target_size)
    image = image / 255.0 # Normalize to [0, 1]
    image = np.transpose(image, (2, 0, 1)) # HWC to CHW
    image = np.expand_dims(image, axis=0) # Add batch dimension
    image = torch.tensor(image, dtype=torch.float32) # Convert to tensor
    return image
def extract_boxes_and_classes(preds, img_width, img_height,
confidence_threshold=0.5):
    """
    Extract bounding boxes, class ids and confidences from YOLO predictions.

    preds: [batch_size, 7*7*30] tensor from the model output
    img_width: width of the original image
    img_height: height of the original image
    confidence_threshold: threshold to filter boxes with low confidence
    """
    batch_size = preds.size(0)
    preds = preds.view(batch_size, 7, 7, 30) # Reshape to [batch_size, 7, 7, 30]

    all_boxes = []

    for batch_idx in range(batch_size):
        boxes = []

        for i in range(7):
            for j in range(7):
                # Extract class probabilities and bounding boxes for this grid
cell
                class_probs = preds[batch_idx, i, j, :20]
                pred_box1 = preds[batch_idx, i, j, 20:25]

```

```

pred_box2 = preds[batch_idx, i, j, 25:30]

# Choose the box with higher confidence
if pred_box1[0] > pred_box2[0]:
    box = pred_box1
else:
    box = pred_box2

# Calculate the final confidence
final_confidence = box[0]

if final_confidence < confidence_threshold:
    continue

# Get the class with the highest probability
class_id = torch.argmax(class_probs)
class_confidence = class_probs[class_id]

# Convert (x, y, w, h) to actual image coordinates
x_center = (box[1].item() + j) / 7.0 * img_width
y_center = (box[2].item() + i) / 7.0 * img_height
width = box[3].item() * img_width
height = box[4].item() * img_height
x_min = int(x_center - width / 2)
y_min = int(y_center - height / 2)
x_max = int(x_center + width / 2)
y_max = int(y_center + height / 2)

boxes.append([x_min, y_min, x_max, y_max, class_id.item(),
final_confidence.item()])

all_boxes.append(boxes)

return all_boxes

def draw_boxes(image, boxes, class_names):
    """Draw bounding boxes on the image."""
    for box in boxes:
        for item in box:
            # print(item)
            x_min, y_min, x_max, y_max, class_id, confidence = item
            color = (0, 255, 0) # Green color for bounding boxes
            label = f'{class_names[class_id]}: {confidence:.2f}'
            cv2.rectangle(image, (x_min, y_min), (x_max, y_max), color, 2)
            cv2.putText(image, label, (x_min, y_min - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

    return image

# 使用示例:
# 假设 model 是已经训练好的 YOLOv1 模型, image 是你要测试的图像

# preds 是模型输出的 [batch_size, 7*7*30] tensor
model = Yolov1Model()
state_dict = torch.load(r"C:\Users\jajnw\Desktop\yolov5-
master\catndog\labels\try.pt")
model.load_state_dict(state_dict)

```

```

# outputs = net(img_)

dataset = CustomDataset(images_dir="C:/Users/jajnw/Desktop/yolov5-
master/catndog/images/train",
                       annotations_dir="C:/Users/jajnw/Desktop/yolov5-
master/catndog/labels/json",
                       transform=preprocess)
print(dataset[0])
images, targets = dataset[0]
model.eval()
inputs_resize_unsqueeze = torch.unsqueeze(images, 0)
# 进行模型的评估操作，例如前向传播和计算预测结果

output = model(inputs_resize_unsqueeze)
batch_size = output.size(0)
preds = output.view(batch_size, 7, 7, 30)
print(preds.shape)
class_names=
['cat', 'cat', 'cat', 'cat', 'cat', 'cat', 'cat', 'cat', 'cat', 'cat', 'cat']
boxes = extract_boxes_and_classes(preds, 448, 448)
for i in range(8):
    images, targets = dataset[i]
    model.eval()
    inputs_resize_unsqueeze = torch.unsqueeze(images, 0)
    # 进行模型的评估操作，例如前向传播和计算预测结果

    output = model(inputs_resize_unsqueeze)
    batch_size = output.size(0)
    preds = output.view(batch_size, 7, 7, 30)
    print(preds.shape)
    class_names = ['cat', 'cat', 'cat', 'cat', 'cat', 'cat', 'cat', 'cat', 'cat',
'cat', 'cat', 'cat']
    boxes = extract_boxes_and_classes(preds, 448, 448)
    image_path="C:/Users/jajnw/Desktop/yolov5-master/catndog/images/train/cat."+str(i)+".jpg"
    image = cv2.imread(image_path)
    original_size = image.shape[:2] # Original size (height, width)
    new_size = (448, 448) # Resize to 448x448 for YOLOv1

    image_resized = cv2.resize(image, new_size)
    # image = image.resize((448, 448))
    # image = preprocess_image(image)

    output_image = draw_boxes(image_resized, boxes, class_names)
    plt.figure(figsize=(10, 10))
    plt.imshow(cv2.cvtColor(output_image, cv2.COLOR_BGR2RGB))
    # plt.imshow(output_image)
    plt.axis('on')
    plt.show()
    # 打印或者处理 boxes， boxes 是一个包含多个框的列表，每个框格式为 [x_min, y_min,
x_max, y_max, class_id, confidence]
    for batch_idx, batch_boxes in enumerate(boxes):
        print(f"Image {batch_idx}:")
        for box in batch_boxes:
            print(f"Box: {box}")

```

## 模型训练通用规则:

- 如果 train 效果挺好, 但 test 或 predict 效果较差, 说明过拟合(overfit)了. 原因有: (1) 模型太复杂了, 这时候应该减少 epoch 或者使用更小 scale 的模型. (2) 数据集太小, 这时候需要增加训练数据. (3) 增大 val 数据集的比例
- 如果 train 效果不佳, 可以使用更大规模的模型, 或者增加训练数据, 或者增大 epoch.
- 大模型在小数据集上更容易过拟合.
- 数据集的分类物体, 如果特征较少, 推荐使用小模型. 如何确定特征的多少, 看这个维度: (1) 物体的形状是否变化多样, (2) 颜色是否多样, (3) 纹理是否复杂, (4) 大小是否差异较大.
- 小模型通常能更好地过滤噪声信息, 如果图像质量不高, 使用小模型效果可能会更好.
- 大目标检测, 模型深度越高, 感受野越大, 效果会更好, 推荐使用 yolov8x 或 yolov8x6 模型.
- 小目标检测, 如果模型太深, 优势反而不利, 推荐使用 yolov8s 或 yolov8m 或 yolov8l.
- 大小目标混合检测, 推荐使用 yolov8l6 或 yolov8x6
- 计算过程中不用关心 loss 是否太大, 重要的是 loss 是否可以收敛. 如果始终不收敛, 尝试调整优化器类型或降低学习率.
- from scratch 训练模型需要足够的算力和数据集, 基本上每种 object 需要训练 2000 多次, 另外需要有强大的调参能力, 否则毫无意义.
- 调整神经网络结构. 选择合适的模型规格( n/s/m/l/x 以及 p2/p6), 增加网络深度和宽度可以增强模型的表达能力.
- 数据集应该大于 500, 否则效果很难上来.
- 数据集应有较高质量, 标签不能错误, box 边框要准确, 正负样本要平衡, 增加数据多样性.
- 调整预测的阈值, 降低 conf 阈值可以提升 recall, 提高 conf 阈值可以提升 precision.
- 调整超参, 学习率/batch/优化器/epoch 等参数