基于YOLOv3的目标检测实验报告

目录

小组成员及分工

姓名	学号	贡献
马家昱	1950509	数据集搜索与整合、图片处理
陈冠忠	1950638	模型修改、调试、训练
陶思月	1951858	数据集拍摄、标记
黄继宣	1951857	数据集拍摄、标记
周婉莹	1950579	数据集拍摄、标记
罗格峰	1952222	数据集拍摄、标记

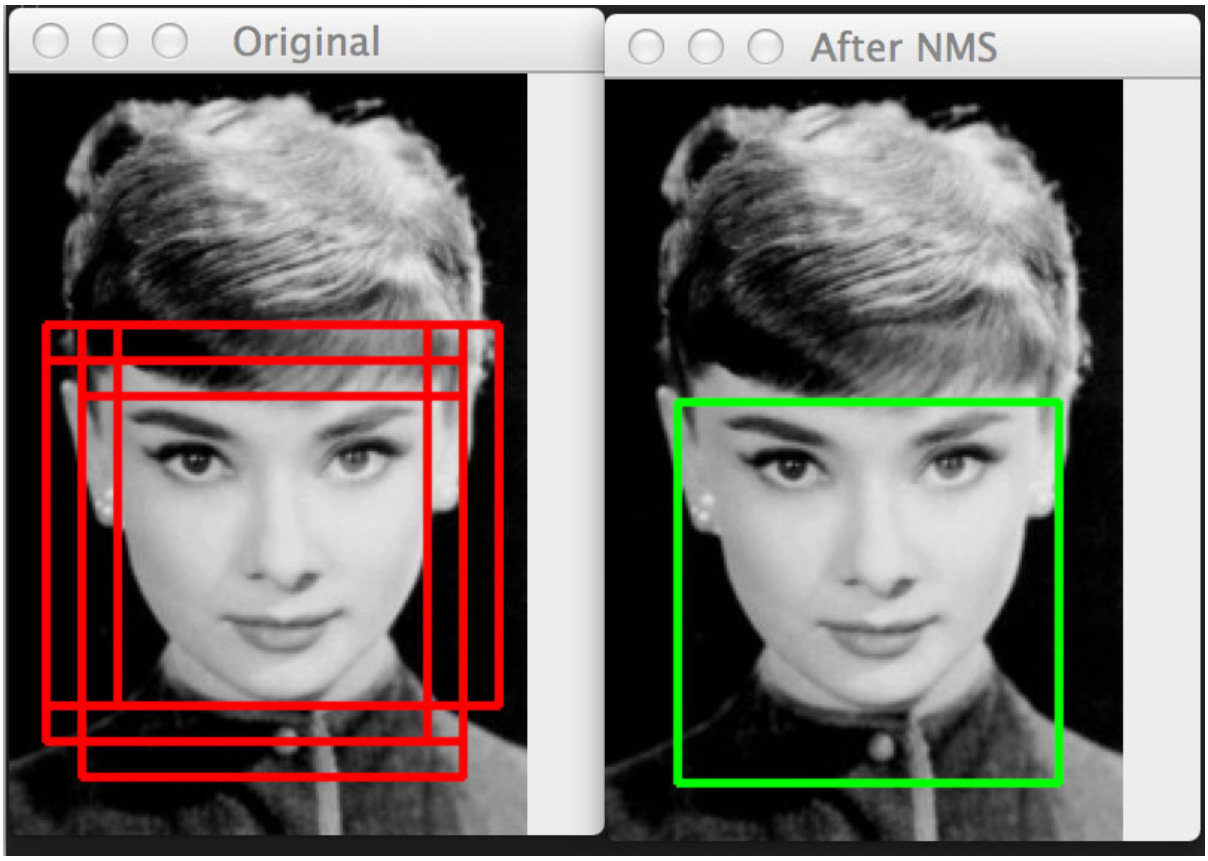
YOLOv3目标检测网络

YOLO算法简介

- 相关算法

1. 滑动窗口

采用滑动窗口的目标检测算法将检测问题转化为了图像分类问题。其基本原理就是采用不同大小和比例（宽高比）的窗口在整张图片上以一定的步长进行滑动，然后对这些窗口对应的区域做图像分类，这样就可以实现对整张图片的检测了。



2. 非极大值抑制

首先从所有的检测框中找到置信度最大的那个框，然后挨个计算其与剩余框的交并比(IOU)，如果其值大于一定阈值（重合度过高），那么就将该框剔除；然后对剩余的检测框重复上述过程，直到处理完所有的检测框。

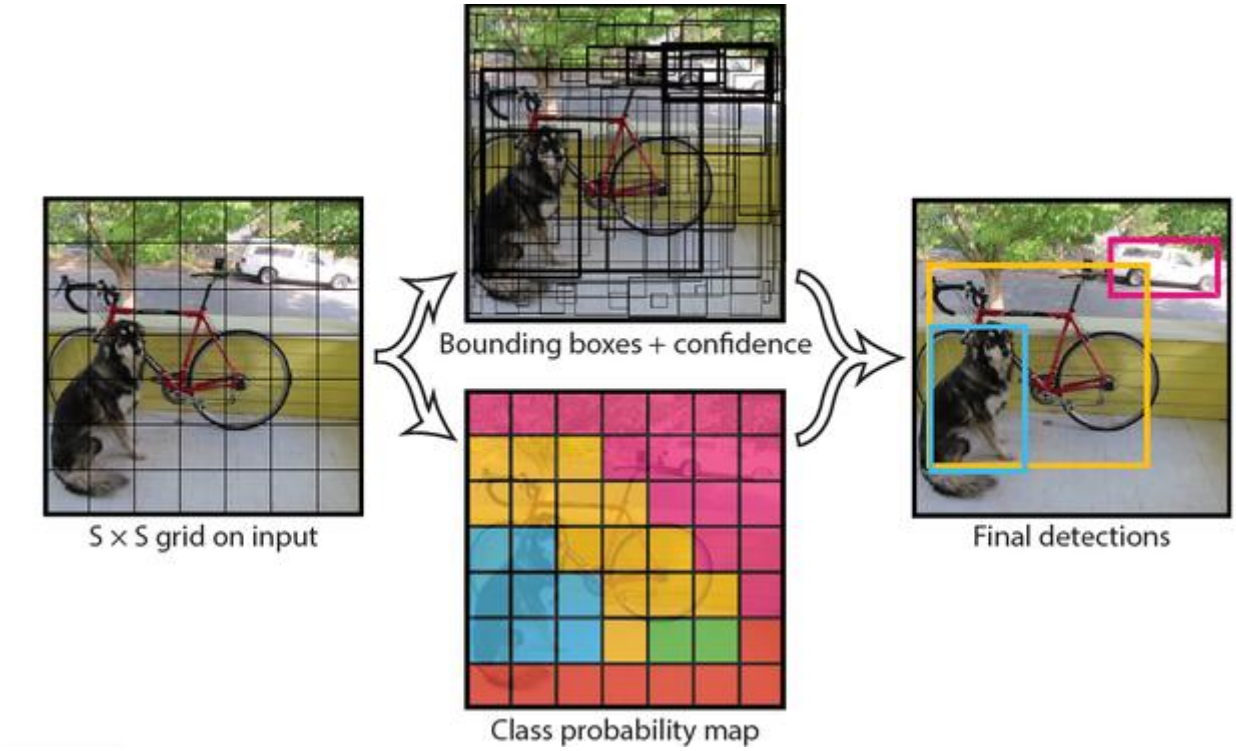
- YOLO算法

YOLO将对象检测重新定义为一个回归问题。它将单个卷积神经网络(CNN)应用于整个图像，将图像分成网格，并预测每个网格的类概率和边界框。对于每个网格，网络都会预测一个边界框和与每个类别（汽车，行人，交通信号灯等）相对应的概率。每个边界框可以使用四个描述符进行描述：

1. 边界框的中心
2. 高度
3. 宽度
4. 值映射到对象所属的类

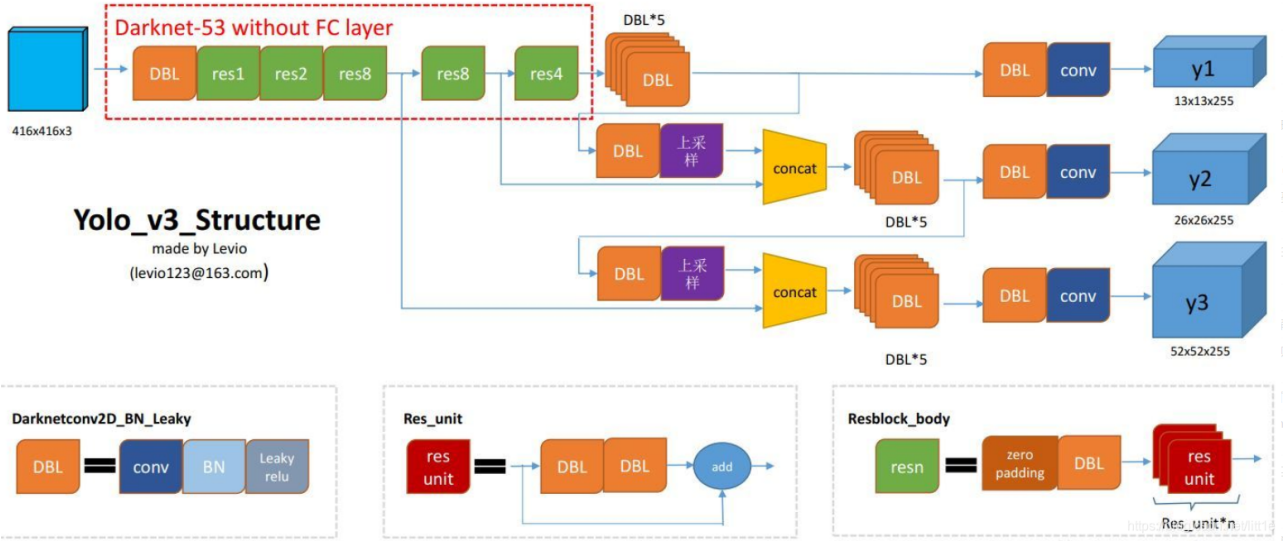
此外，该算法还可以预测边界框中存在对象的概率。如果一个对象的中心落在一个网格单元中，则该网格单元负责检测该对象。每个网格中将有多个边界框。在训练时，我们希望每个对象只有一个边界框。因此，我们根据哪个Box与ground truth box的重叠度最高，从而分配一个Box来负责预测对象。

最后，对每个类的对象应用非最大值抑制的方法来过滤出“置信度”小于阈值的边界框。这为我们提供了图像预测。



网络结构

- YOLOv3采用了称之为Darknet-53的网络结构（含有53个卷积层），它借鉴了残差网络的做法，在一些层之间设置了快捷链路。下图展示了其基本结构。



其中Darknet-53的具体结构如下，其采用448*448*3作为输入，左侧数字表示多重复的残差组件个数，每个残差组件有两个卷积层和一个快捷链路。

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

PaddlePaddle代码实现

主要参数

```
train_params = {
    "data_dir": "data/data6045",  # 数据目录
    "train_list": "train.txt",  # 训练集文件
    "eval_list": "eval.txt",
    "class_dim": -1,
    "label_dict": {},  # 标签字典
    "num_dict": {},
    "image_count": -1,
    "continue_train": True,  # 是否加载前一次的训练参数，接着训练
    "pretrained": False,  # 是否预训练
```

```
    "pretrained_model_dir": "./pretrained-model",
    "save_model_dir": "./yolo-model",  # 模型保存目录
    "model_prefix": "yolo-v3",  # 模型前缀
    "freeze_dir": "freeze_model",
    "use_tiny": False,  # 是否使用 裁剪 tiny 模型
    "max_box_num": 8,  # 一幅图上最多有多少个目标
    "num_epochs": 15,  # 训练轮次
    "train_batch_size": 12,  # 对于完整yolov3，每一批的训练样本不能太多，内存会炸掉；
如果使用tiny，可以适当大一些
    "use_gpu": True,  # 是否使用GPU
    "yolo_cfg": {  # YOLO模型参数
        "input_size": [3, 448, 448],  # 原版的边长大小为608，为了提高训练速度和预测速
度，此处压缩为448
        "anchors": [7, 10, 12, 22, 24, 17, 22, 45, 46, 33, 43, 88, 85, 66, 115,
146, 275, 240],  # 锚点??
        "anchor_mask": [[6, 7, 8], [3, 4, 5], [0, 1, 2]]
    },
    "yolo_tiny_cfg": {  # YOLO tiny 模型参数
        "input_size": [3, 256, 256],
        "anchors": [6, 8, 13, 15, 22, 34, 48, 50, 81, 100, 205, 191],
        "anchor_mask": [[3, 4, 5], [0, 1, 2]]
    },
    "ignore_thresh": 0.7,
    "mean_rgb": [127.5, 127.5, 127.5],
    "mode": "train",
    "multi_data_reader_count": 4,
    "apply_distort": True,  # 是否做图像扭曲增强
    "nms_top_k": 300,
    "nms_pos_k": 300,
    "valid_thresh": 0.01,
    "nms_thresh": 0.40,  # 非最大值抑制阈值
    "image_distort_strategy": {  # 图像扭曲策略
        "expand_prob": 0.5,  # 扩展比率
        "expand_max_ratio": 4,
        "hue_prob": 0.5,  # 色调
        "hue_delta": 18,
        "contrast_prob": 0.5,  # 对比度
        "contrast_delta": 0.5,
        "saturation_prob": 0.5,  # 饱和度
        "saturation_delta": 0.5,
        "brightness_prob": 0.5,  # 亮度
        "brightness_delta": 0.125
    },
    "sgd_strategy": {  # 梯度下降配置
        "learning_rate": 0.002,
        "lr_epochs": [30, 50, 65],  # 学习率衰减分段（3个数字分为4段）
        "lr_decay": [1, 0.5, 0.25, 0.1]  # 每段采用的学习率，对应lr_epochs参数4段
    },
    "early_stop": {
        "sample_frequency": 50,
        "successive_limit": 3,
        "min_loss": 2.5,
        "min_curr_map": 0.84
```

```
    }
}
```

模型建立

```
class YOLOv3(object):
    def __init__(self, class_num, anchors, anchor_mask):
        self.outputs = []  # 网络最终模型
        self.downsample_ratio = 1  # 下采样率
        self.anchor_mask = anchor_mask  # 计算卷积核？？？
        self.anchors = anchors  # 锚点
        self.class_num = class_num  # 类别数量

        self.yolo_anchors = []
        self.yolo_classes = []

        for mask_pair in self.anchor_mask:
            mask_anchors = []
            for mask in mask_pair:
                mask_anchors.append(self.anchors[2 * mask])
                mask_anchors.append(self.anchors[2 * mask + 1])
            self.yolo_anchors.append(mask_anchors)
            self.yolo_classes.append(class_num)

    def name(self):
        return 'YOLOv3'

    # 获取anchors
    def get_anchors(self):
        return self.anchors

    # 获取anchor_mask
    def get_anchor_mask(self):
        return self.anchor_mask

    def get_class_num(self):
        return self.class_num

    def get_downsample_ratio(self):
        return self.downsample_ratio

    def get_yolo_anchors(self):
        return self.yolo_anchors

    def get_yolo_classes(self):
        return self.yolo_classes

    # 卷积正则化函数：卷积、批量正则化处理、leakrelu
    def conv_bn(self,
                input,  # 输入
                num_filters,  # 卷积核数量
```

```
                filter_size,  # 卷积核大小
                stride,  # 步幅
                padding,  # 填充
                use_cudnn=True):
        # 2d卷积操作
        conv = fluid.layers.conv2d(input=input,
                                   num_filters=num_filters,
                                   filter_size=filter_size,
                                   stride=stride,
                                   padding=padding,
                                   act=None,
                                   use_cudnn=use_cudnn,  # 是否使用cudnn，cudnn利用
cuda进行了加速处理

param_attr=ParamAttr(initializer=fluid.initializer.Normal(0., 0.02)),
                                   bias_attr=False)

        # batch_norm中的参数不需要参与正则化，所以主动使用正则系数为0的正则项屏蔽掉
        # 在batch_norm中使用leaky的话，只能使用默认的alpha=0.02；如果需要设值，必须提
出去单独来
        # 正则化的目的，是为了防止过拟合，较小的L2值能防止过拟合
        param_attr = ParamAttr(initializer=fluid.initializer.Normal(0., 0.02),
                               regularizer=L2Decay(0.))
        bias_attr = ParamAttr(initializer=fluid.initializer.Constant(0.0),
                              regularizer=L2Decay(0.))
        out = fluid.layers.batch_norm(input=conv, act=None,
                                      param_attr=param_attr,
                                      bias_attr=bias_attr)
        # leaky_relu: Leaky ReLU是给所有负值赋予一个非零斜率
        out = fluid.layers.leaky_relu(out, 0.1)
        return out
```

训练与迭代

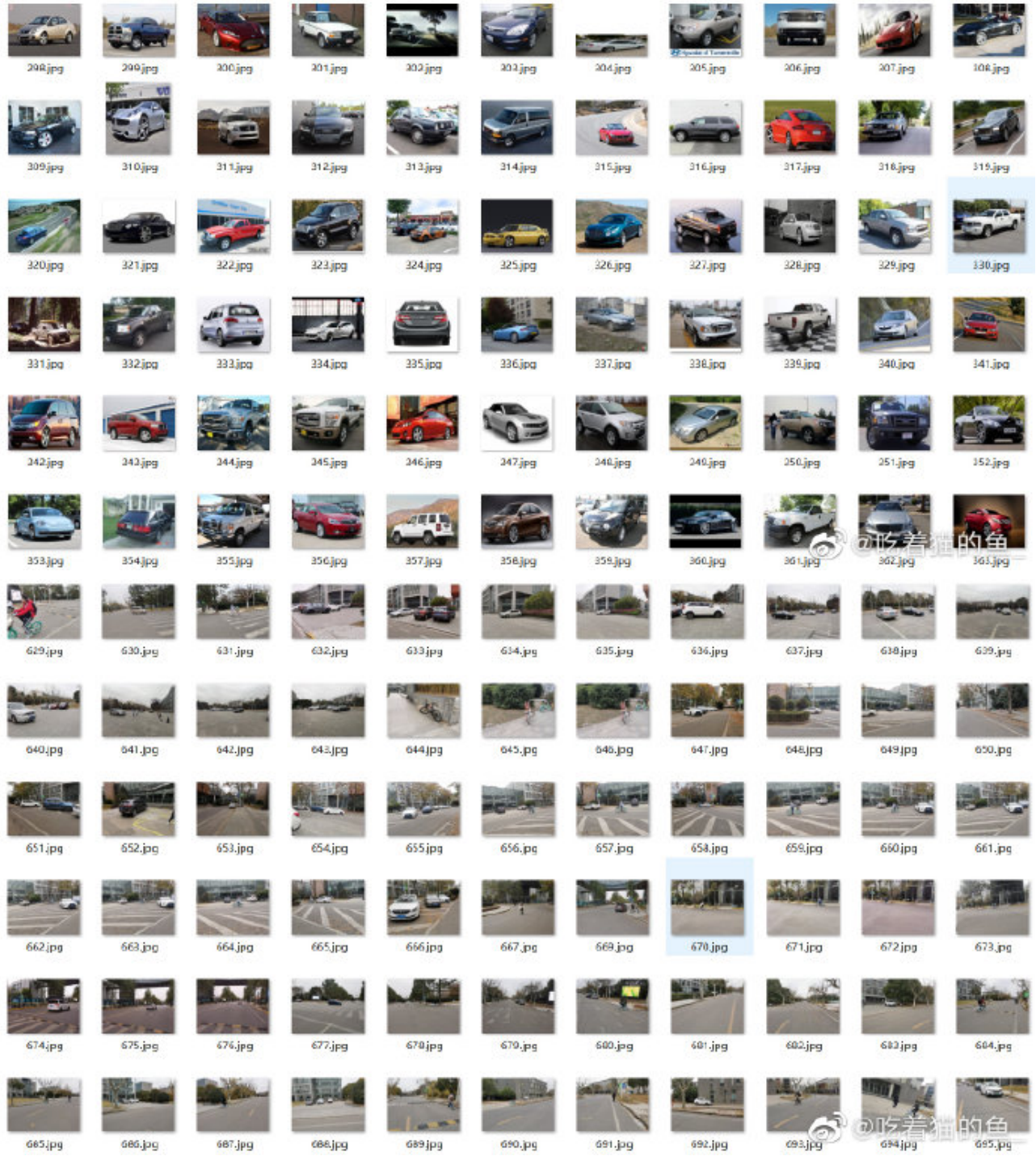
```
# 执行训练
def train():
    init_log_config()
    init_train_parameters()

    logger.info("start train YOLOv3, train params:%s", str(train_params))
    logger.info("create place, use gpu:" + str(train_params['use_gpu']))

    place = fluid.CUDAPlace(0) if train_params['use_gpu'] else fluid.CPUPlace()

    logger.info("build network and program")
    train_program = fluid.Program()
    start_program = fluid.Program()
    feeder, reader, loss = build_program_with_feeder(train_program, start_program,
place)

    logger.info("build executor and init params")
```

```
    exe = fluid.Executor(place)
    exe.run(start_program)
    train_fetch_list = [loss.name]
    load_pretrained_params(exe, train_program)  # 加载模型及参数

    stop_strategy = train_params['early_stop']
    successive_limit = stop_strategy['successive_limit']
    sample_freq = stop_strategy['sample_frequency']
    min_curr_map = stop_strategy['min_curr_map']
    min_loss = stop_strategy['min_loss']
    stop_train = False
    successive_count = 0
    total_batch_count = 0
    valid_thresh = train_params['valid_thresh']
    nms_thresh = train_params['nms_thresh']
    current_best_loss = 10000000000.0

    # 开始迭代训练
    for pass_id in range(train_params["num_epochs"]):
        logger.info("current pass: {}, start read image".format(pass_id))
        batch_id = 0
        total_loss = 0.0

        for batch_id, data in enumerate(reader()):
            t1 = time.time()

            loss = exe.run(train_program,
                           feed=feeder.feed(data),
                           fetch_list=train_fetch_list)  # 执行训练

            period = time.time() - t1
            loss = np.mean(np.array(loss))
            total_loss += loss
            batch_id += 1
            total_batch_count += 1

            if batch_id % 10 == 0:  # 调整日志输出的频率
                logger.info(
                    "pass {}, trainbatch {}, loss {} time {}".format(pass_id,
batch_id, loss, "%2.2f sec" % period))

        pass_mean_loss = total_loss / batch_id
        logger.info("pass {0} train result, current pass mean loss:
{1}".format(pass_id, pass_mean_loss))

        # 采用每训练完一轮停止办法，可以调整为更精细的保存策略
        if pass_mean_loss < current_best_loss:
            logger.info("temp save {} epcho train result, current best pass loss
{}".format(pass_id, pass_mean_loss))
            fluid.io.save_persistables(dirname=train_params['save_model_dir'],
main_program=train_program,
                                       executor=exe)
            current_best_loss = pass_mean_loss
```

```
    logger.info("training till last epcho, end training")
    fluid.io.save_persistables(dirname=train_params['save_model_dir'],
main_program=train_program, executor=exe)
```

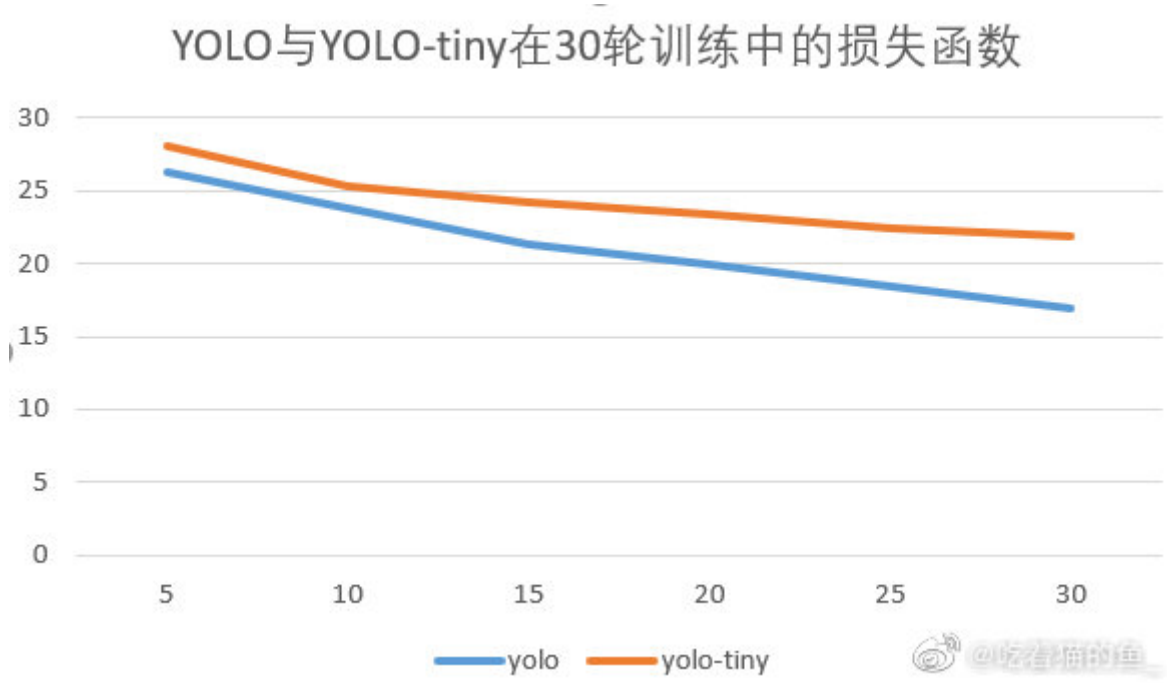
数据集基本信息

- 本组使用的数据集共有900张图片，其中500张来自校园拍摄实景，其余为下载的特定分类图片。
- 所有图片宽高比均为4：3,分辨率为800*600。数据集图片主要分四类，包括单独的行人、自行车与汽车与前三类混杂在一起的图片。



训练过程中的参数调整与模型优化

YOLO和YOLO-tiny对比



模型	训练30轮所用时长
YOLO	2h9m
YOLO-tiny	1h41m

参数调整

- max_box_num": 8
- nms_thresh": 0.40
- valid_thresh": 0.015
- 优化显存
  - os.environ["FLAGS_fraction_of_gpu_memory_to_use"] = '0.92'
  - os.environ["FLAGS_eager_delete_tensor_gb"] = '0'
  - os.environ["FLAGS_memory_fraction_of_eager_deletion"] = '1'
  - os.environ["FLAGS_fast_eager_deletion_mode"]='True'

模型优化

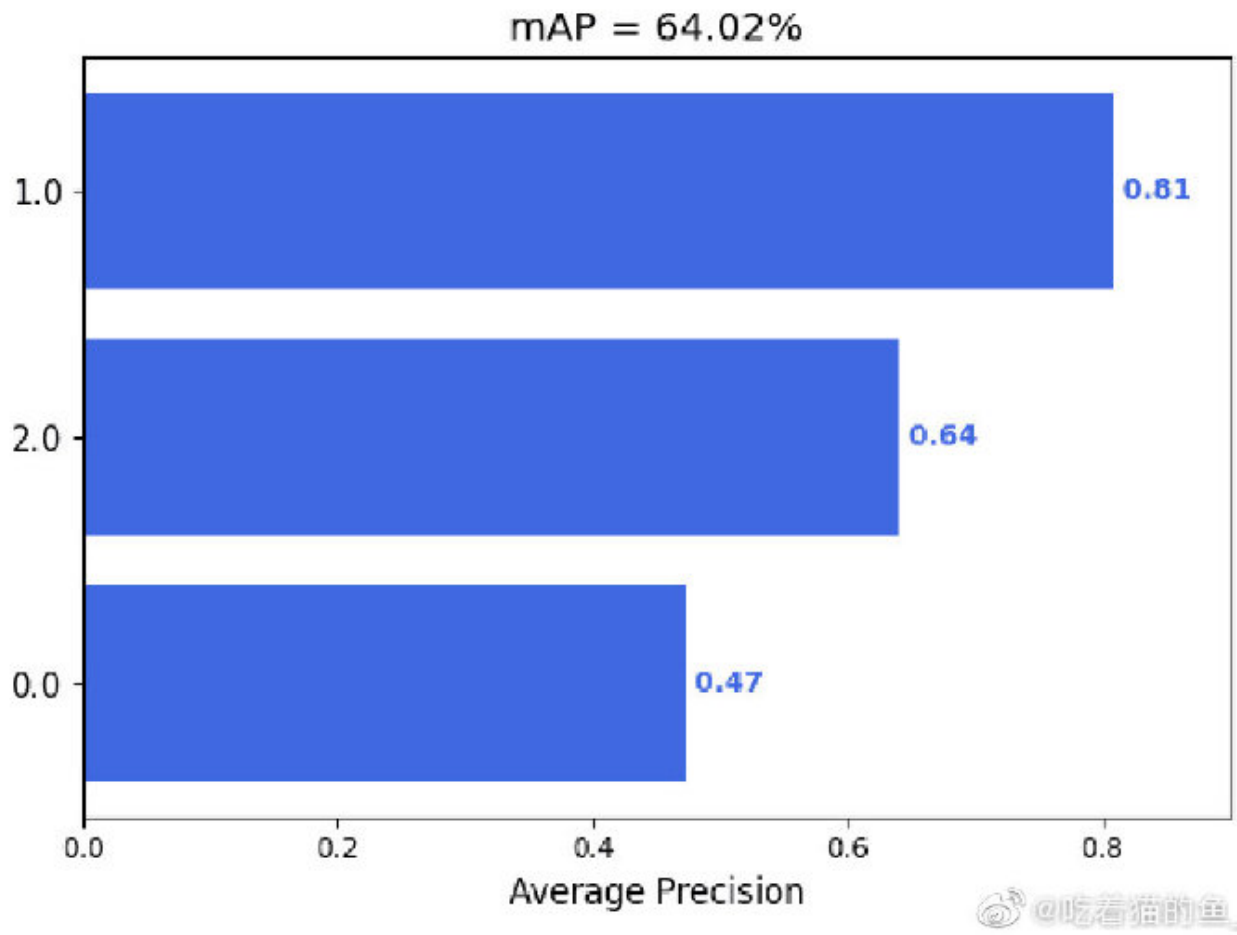
- 优化器更改：原优化器为SGD

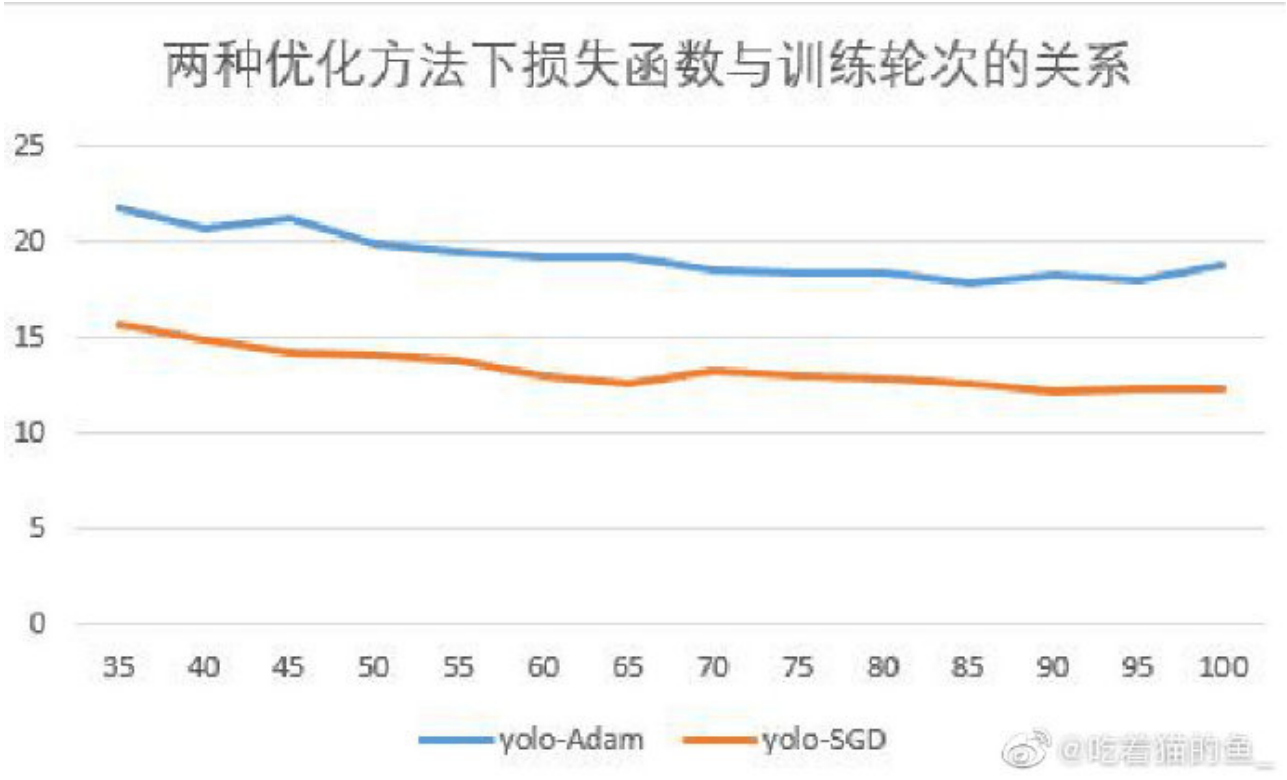
```
optimizer=fluid.optimizer.SGDOptimizer(
        learning_rate=fluid.layers.piecewise_decay(boundaries, values),
regularization=fluid.regularizer.L2Decay(0.00005))
```

- 变更为Adam算法

```
optimizer=fluid.optimizer.AdamOptimizer(learning_rate=0.01,beta1=0.9,beta2=0.999,r
egularization=fluid.regularizer.L2Decay(0.00005))
```

- Adam优化对比分析：

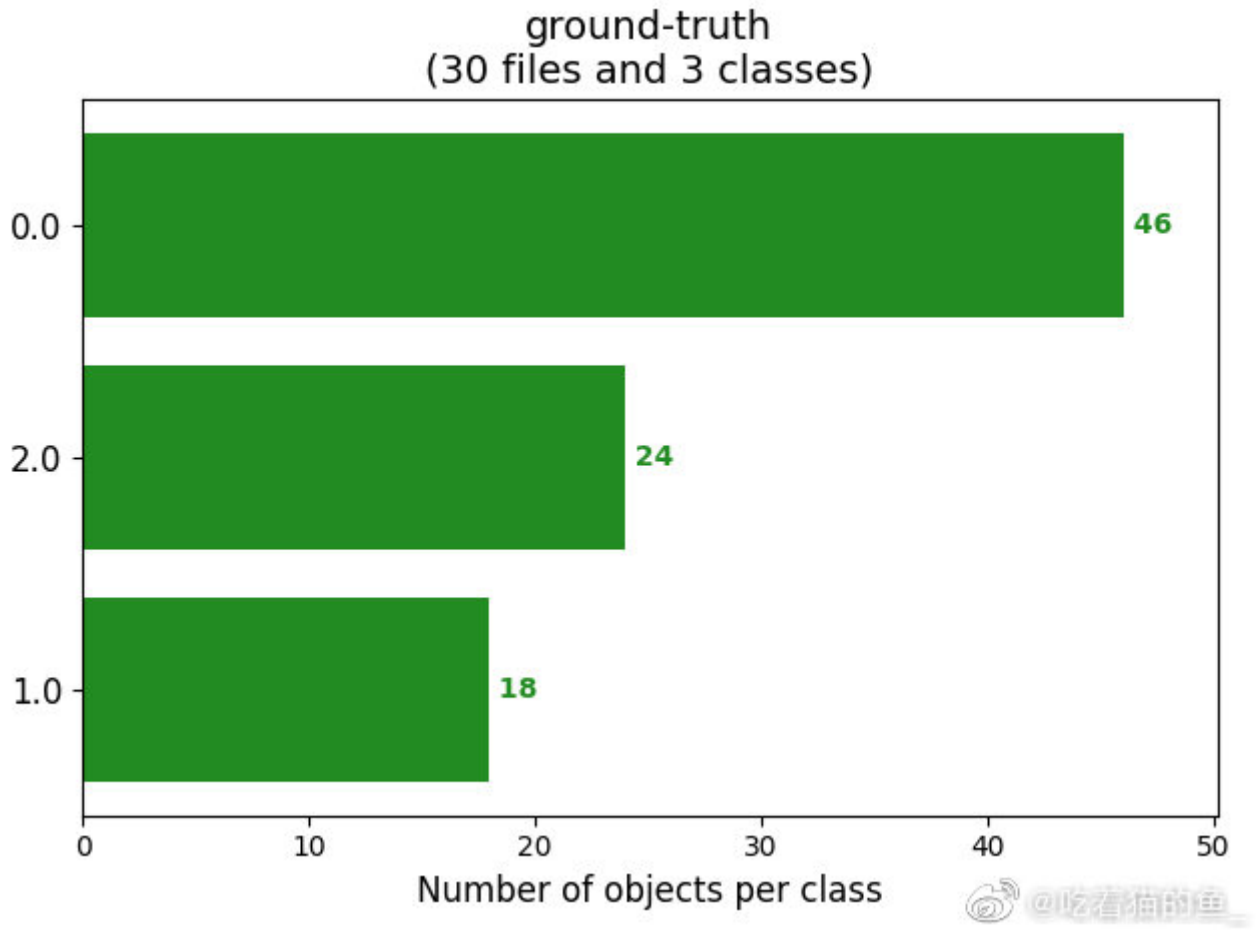


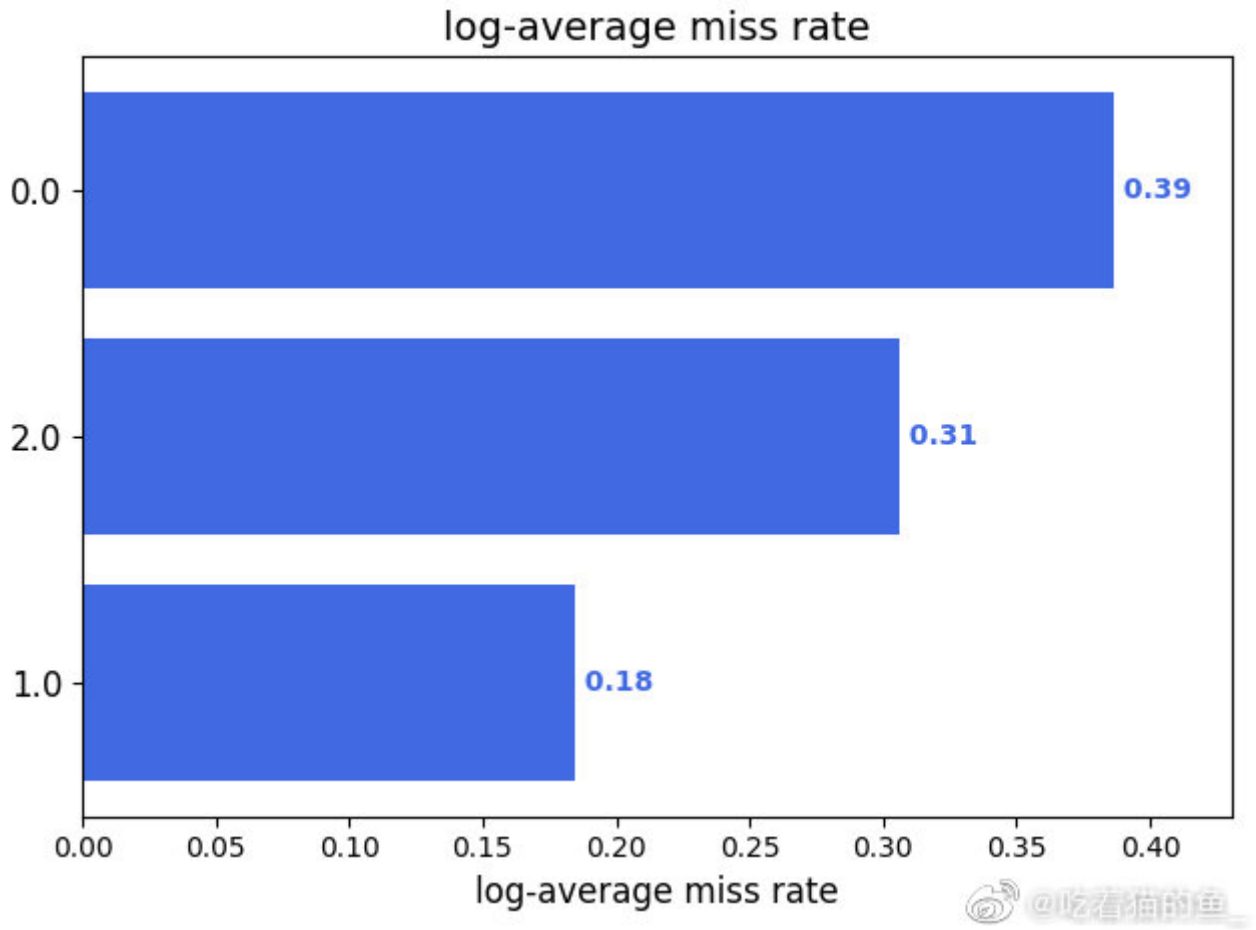


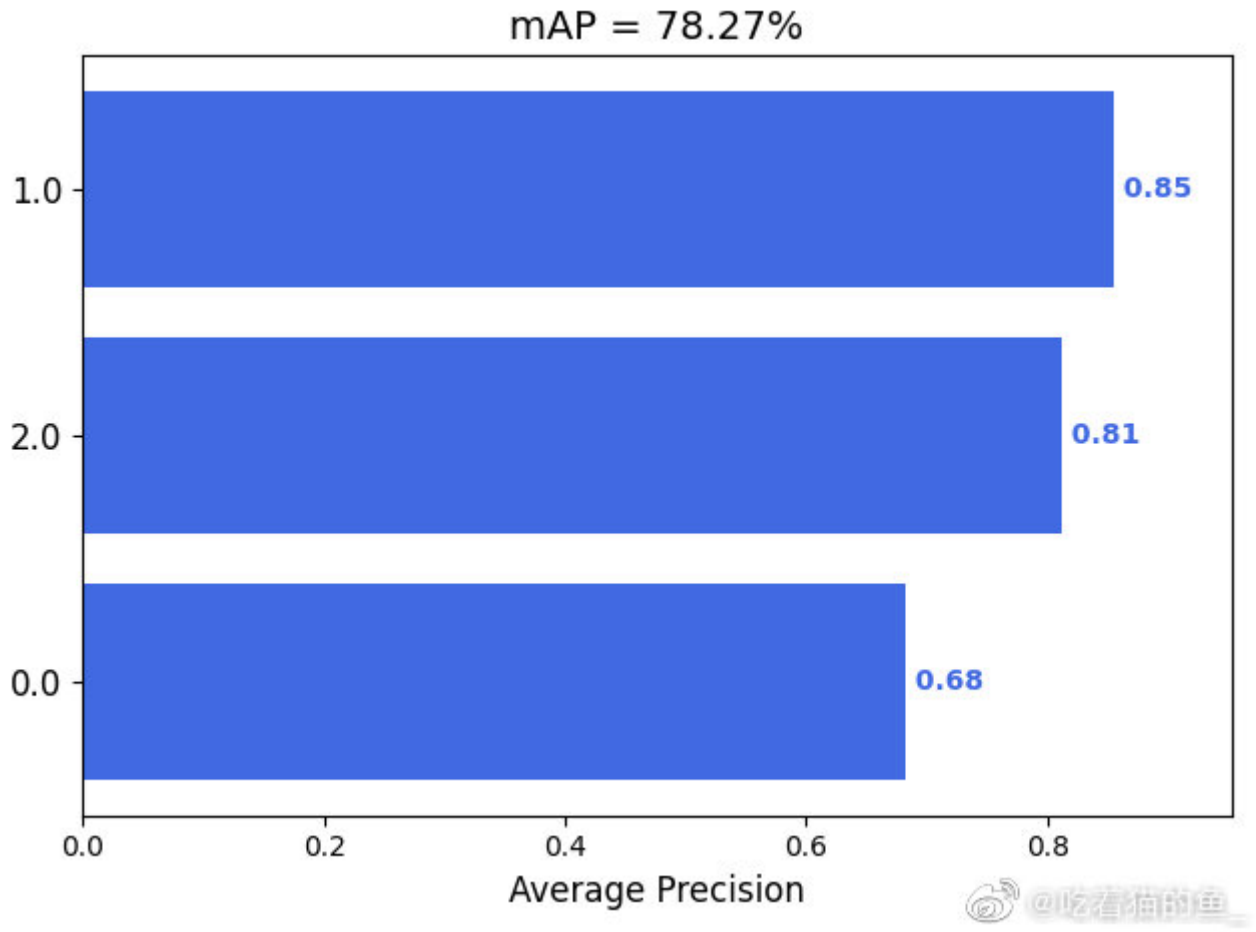
网络性能分析

- 挑战集测试分析


detection-results
(30 files and 3 detected classes)
2.0
0.0
1.0
58 23
45 34
35 16
False Positive
True Positive
0
20
40
60
80
Number of objects per class


ground-truth
(30 files and 3 classes)
0.0
2.0
1.0
46
24
18
0
10
20
30
40
50
Number of objects per class


log-average miss rate
0.0
2.0
1.0
0.39
0.31
0.18
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
log-average miss rate


mAP = 78.27%
1.0
2.0
0.0
0.85
0.81
0.68
0.0
0.2
0.4
0.6
0.8
Average Precision

- 实际结果

- 
@吃着猫的鱼_
- 
@吃着猫的鱼_