电 子 科 技 大 学

实 验 报 告

学生姓名：刘文晨　　学 号：2018080901006　　指导教师：沈复民，徐行

一、实验项目名称：

基于深度学习的鱼类检测（Fish Detection with Deep Learning）

二、实验原理：

1. YOLOv3

YOLOv3 是 YOLO (You Only Look Once)系列目标检测算法中的第三版，相比之前的算法，在保持速度优势的前提下，提升了预测精度，尤其是加强了对小物体的识别能力。YOLOv3 的网络结构如图 1 所示。

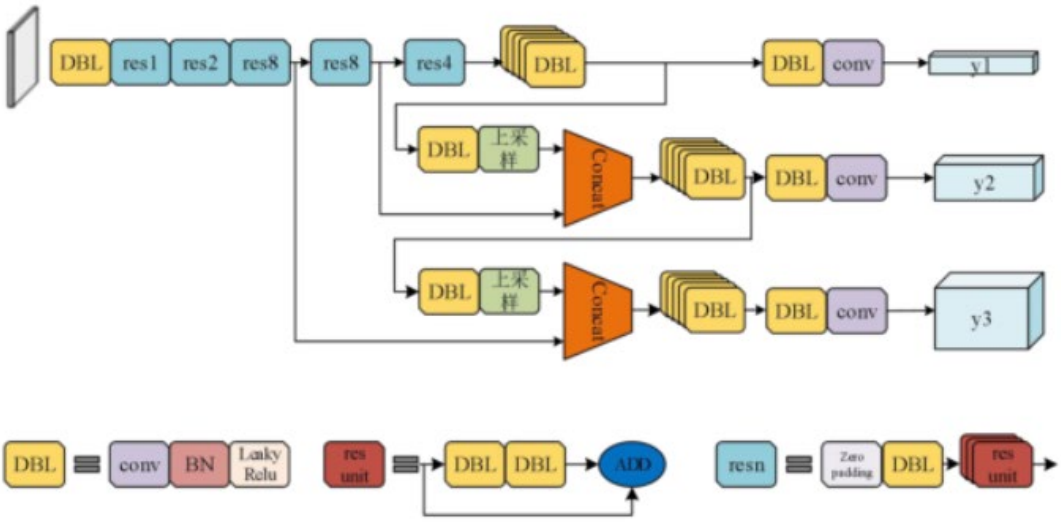


图 1 YOLOv3 网络结构

1) Darknet-53 网络结构

在基本的图像特征提取方面，YOLOv3 采用了称之为 Darknet-53 的网络结构，如图 2 所示。它含有 53 个卷积层，借鉴了残差网络

（Residual Network）的做法，在一些层之间设置了快捷链路（Shortcut Connections）。

	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					

图 2 Darknet-53 网络结构

Darknet-53 网络采用 256*256*3 作为输入，最左侧那一列的 1、2、8 等数字表示多少个重复的残差组件。每个残差组件有两个卷积层和一个快捷链路，示意图如图 3 所示。

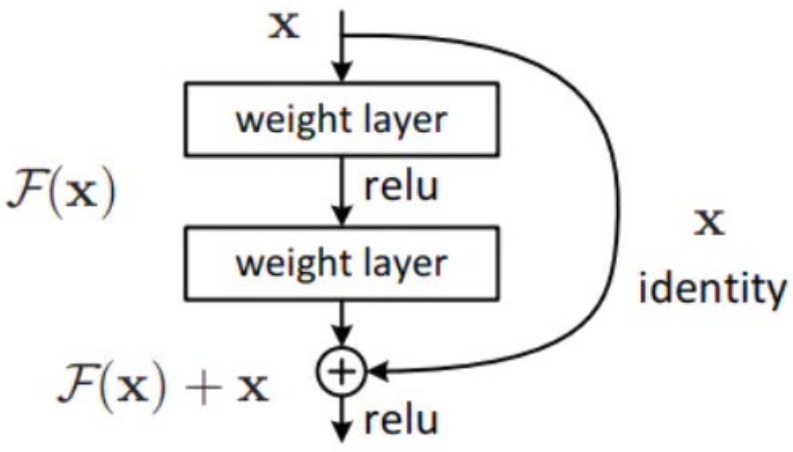


图 3 残差结构

2) 利用多尺度特征进行对象检测

YOLO2 曾采用 passthrough 结构来检测细粒度特征，在 YOLO3 更进一步采用了 3 个不同尺度的特征图来进行对象检测。

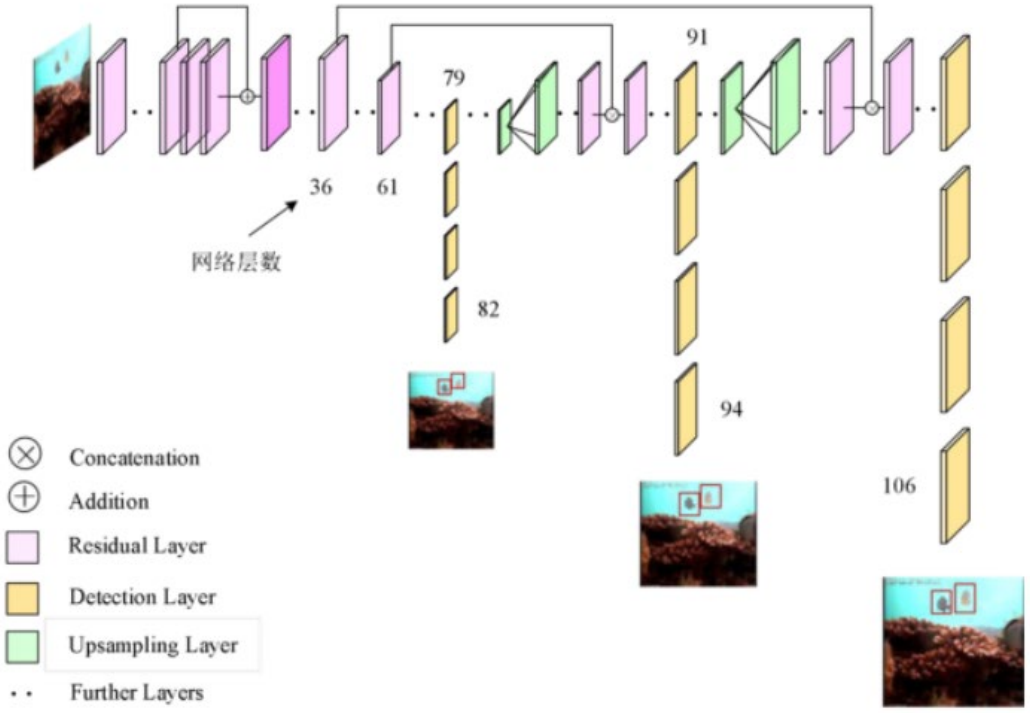


图 4 多尺度检测

结合图 4 看，卷积网络在 79 层后，经过下方几个黄色的卷积层得到一种尺度的检测结果。相比输入图像，这里用于检测的特征图有 32 倍的下采样。比如输入是 416*416 的话，这里的特征图就是 13*13 了。由于下采样倍数高，这里特征图的感受野比较大，因此适合检测图像中尺寸比较大的对象。为了实现细粒度的检测，第 79 层的特征图又开始作上采样（从 79 层往右开始上采样卷积），然后与第 61 层特征图融合，这样得到第 91 层较细粒度的特征图，同样经过几个卷积层后得到相对输入图像 16 倍下采样的特征图。它具有中等尺度的感受野，适合检测中等尺度的对象。最后，第 91 层特征图再次上采样，并与第 36 层特征图融合，最后得到相对输入图像 8

倍下采样的特征图。它的感受野最小，适合检测小尺寸的对象。

3) 9 种尺度的先验框

随着输出的特征图的数量和尺度的变化，先验框的尺寸也需要相应的调整。YOLOv2 已经开始采用 K-means 聚类得到先验框的尺寸，YOLOv3 延续了这种方法，为每种下采样尺度设定 3 种先验框，总共聚类出 9 种尺寸的先验框。在 COCO 数据集这 9 个先验框是：(10x13)，(16x30)，(33x23)，(30x61)，(62x45)，(59x119)，(116x90)，(156x198)，(373x326)。

分配上，如图 5 所示。在最小的 13*13 特征图上（有最大的感受野）应用较大的先验框(116x90)，(156x198)，(373x326)，适合检测较大的对象。中等的 26*26 特征图上（中等感受野）应用中等的先验框(30x61)，(62x45)，(59x119)，适合检测中等大小的对象。较大的 52*52 特征图上（较小的感受野）应用较小的先验框(10x13)，(16x30)，(33x23)，适合检测较小的对象。

特征图	13*13			26*26			52*52		
感受野	大			中			小		
先验框	(116x90)	(156x198)	(373x326)	(30x61)	(62x45)	(59x119)	(10x13)	(16x30)	(33x23)

图 5 特征图与先验框

4) 对象分类 softmax 改成 logistic

预测对象类别时不使用 softmax，改成使用 logistic 的输出进行预测。这样能够支持多标签对象。

不考虑神经网络结构细节的话，总的来说，对于一个输入图像，YOLOv3 将其映射到 3 个尺度的输出张量，代表图像各个位置存在各种对象的概率。

相比于 YOLOv2，YOLOv3 的尝试预测边框数量增加了 10 多倍，而且是在不同分辨率上进行，所以对 mAP 以及小物体的检测效果有一定的提升。

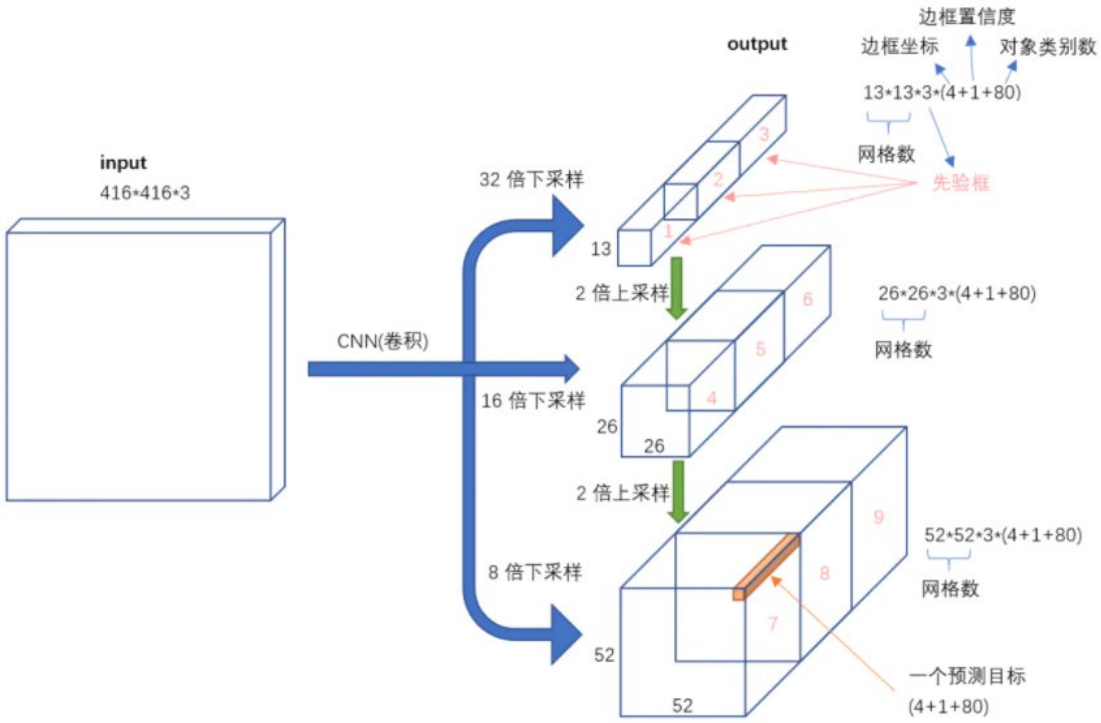


图 6 输入映射到输出

5) 模型对比

YOLOv3 借鉴了残差网络结构，形成更深的网络层次，以及多尺度检测，提升了 mAP 及小物体的检测效果。如果采用 COCO mAP50 做评估指标（不是太介意预测框的准确性的话），YOLOv3 的表现相当惊人，如图 7 所示。在精确度相当的情况下，YOLOv3 的速度是其它模型的 3 到 4 倍。

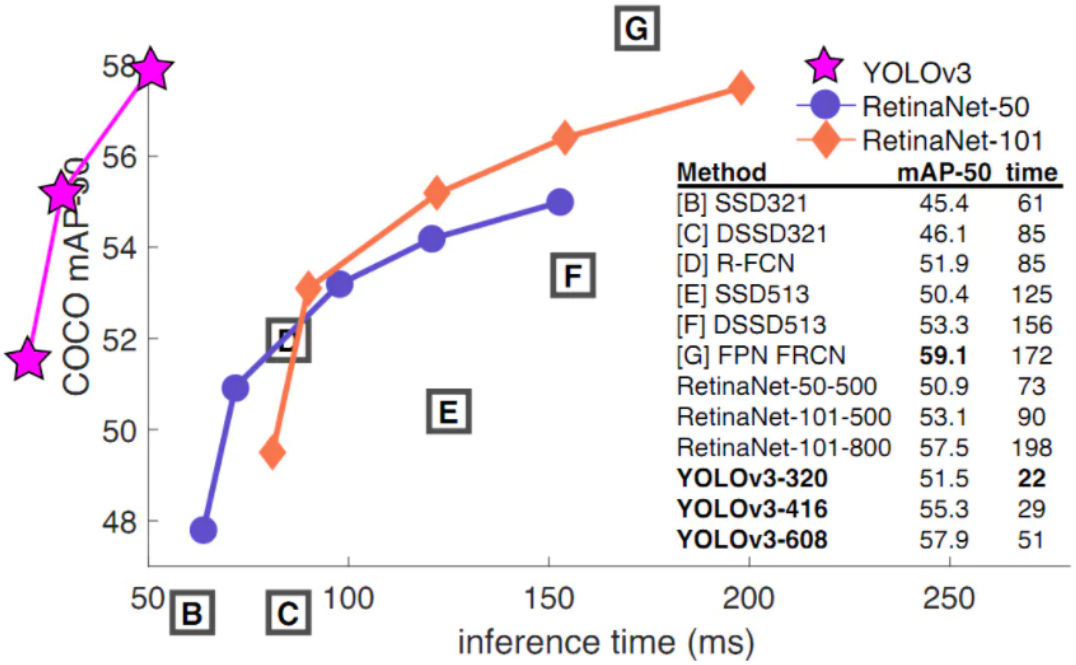
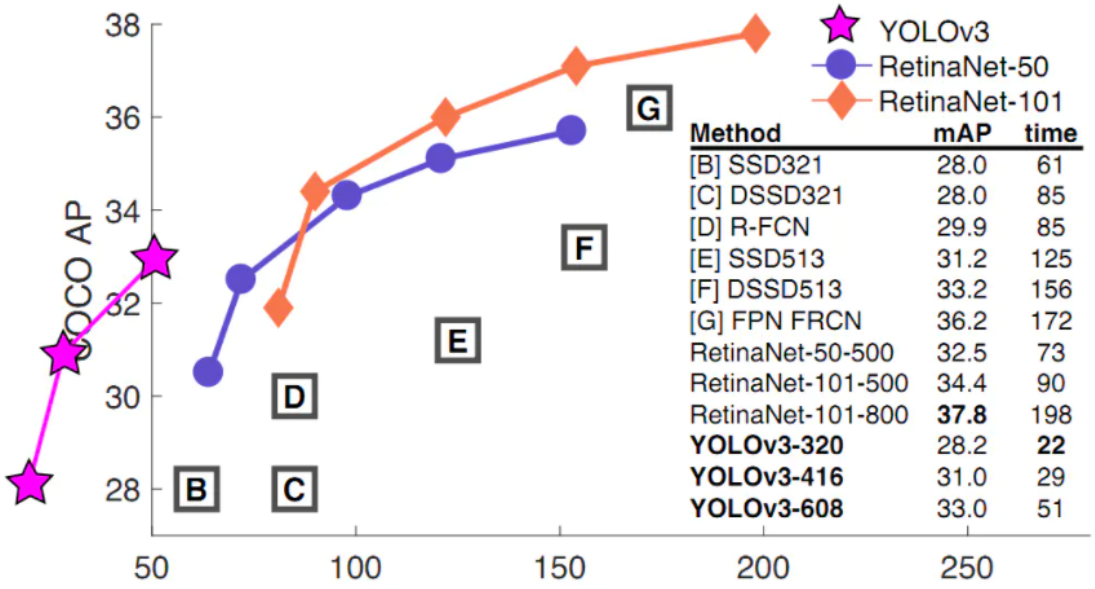


图 7 YOLOv3 与其它模型的性能对比（COCO mAP50 做评估指标）

不过如果要求更精准的预测边框，采用 COCO AP 做评估标准的话，YOLO3 在精确率上的表现就弱了一些。如图 8 所示。



Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

图 8 YOLOv3 与其它模型的性能对比（COCO AP 做评估指标）

2. YOLOv3-tiny

YOLOv3-tiny 就是在 YOLOv3 的基础上去掉了一些特征层，只保留了 2 个独立预测分支，具体的结构图如图 9 所示。对于速度要求比较高的项目，YOLOv3-tiny 才是首要选择。

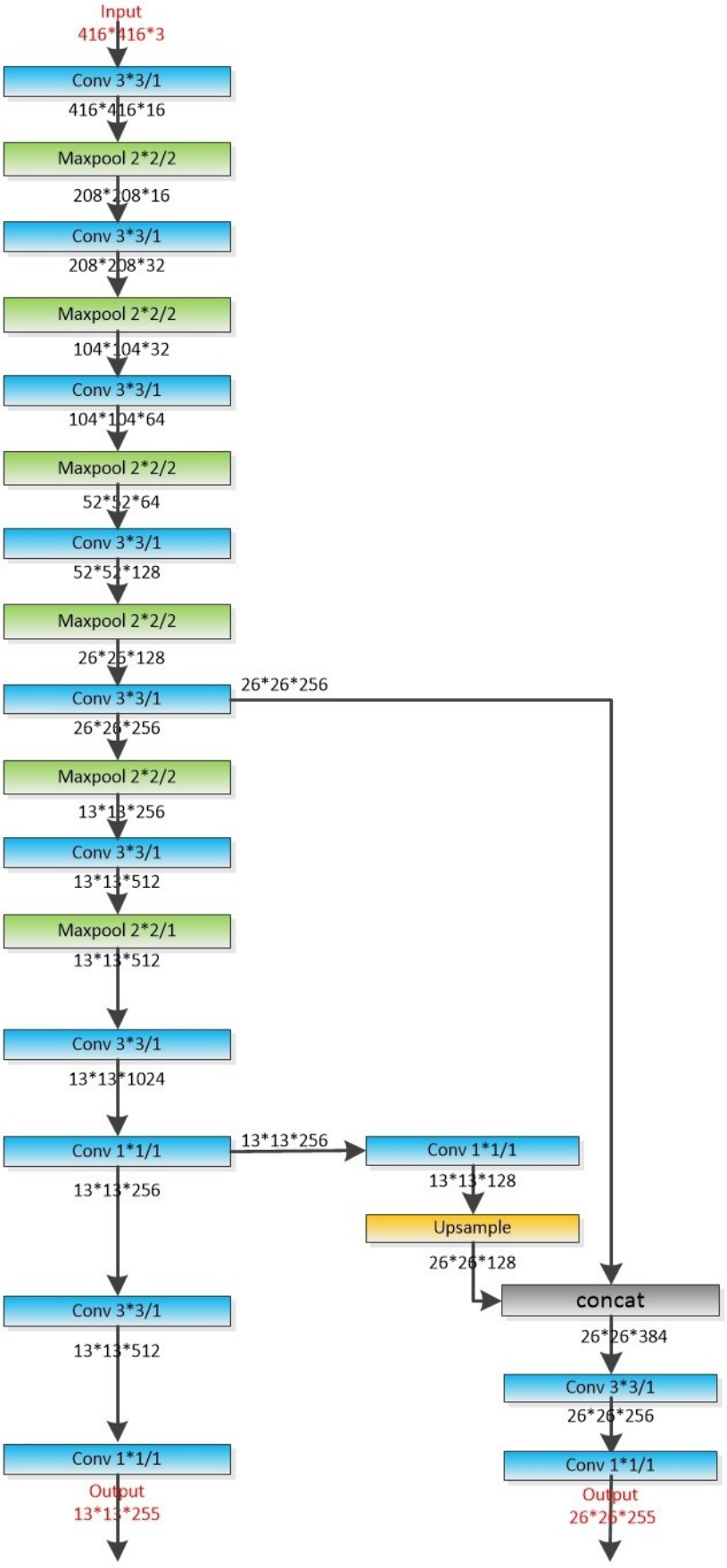


图 9 YOLOv3-tiny 结构图

三、实验目的：

基于 YOLOv3-tiny 模型实现鱼类检测算法，熟悉深度学习在目标检测上的应用方式。

三、实验内容：

1. 了解并学习基于深度学习的鱼类检测的知识和操作.
2. 实现数据预处理、训练模型等功能。

四、实验步骤：

1. 数据预处理

完善“proj5.ipynb”文件的 Data Preprocess 部分的代码，生成有效的 Train 数据集和 Val 数据集，使用 data/custom/images 和 data/custom/labels 中的 data 生成 data/custom/中的 path file train.txt 和 val.txt。经过完善的代码如下所示。

```
################################################################################
#                                      Your Code                               #
################################################################################
# You should generate valid Train dataset and Val dataset.
# Use data in data/custom/images and data/custom/labels to generate the path file train.txt and
# val.txt in data/custom/
# a qualified val dataset is smaller than the train dataset and
# most time there are no overlapped data between two sets.
val_percent = 0.1
train_percent = 0.9
filepath = 'data/custom/labels'
txtsavepath = 'data/custom'
total_txt = os.listdir(filepath)

num = len(total_txt)
list = range(num)
```

```
tv = int(num * val_percent)  # 12
tr = int(num * train_percent)  # 110
val = random.sample(list, tv)  # 12
train = random.sample(list, tr)  # 110

ftrain = open('data/custom/train.txt', 'w')
fval = open('data/custom/valid.txt', 'w')

for i in list:
    name = total_txt[i][:-4] + '.jpg' + '\n'
    if i in train:
        ftrain.write("data\custom\images\\" + name)
    else:
        fval.write("data\custom\images\\" + name)
ftrain.close()
fval.close()
################################################################################
#                                             End                              #
################################################################################
```

2. 训练模型

完善“proj5.ipynb”文件的 Train your model 部分的代码，经过完善的代码如下所示。

```
for epoch in range(opt["epochs"]):
    print("\n---- Training Model ----")
    model.train()
    ################################################################################
    #                                             Your Code                        #
    ################################################################################
    # Your code need to execute forward and backward steps.您的代码需要执行前进和后退步骤。
    # Use 'enumerate' to get a batch[_, images, targets]
    # some helpful function
    # - outputs = model.__call__(imgs)(use it by model(imgs))
    # - loss, _ = cumpte_loss(outputs, targets, model)
    # - loss.backward() (backward step)
```

```
    # - optimizer.step() (execute params updating)
    # - optimizer.zero_grad() (reset gradients)
    # if you want to see how loss changes in each mini-batch step:
    # -eg print(f'Epoch:{epoch+1}, Step{step+1}/{len(dataloader)}, loss:{loss.item()}')
    size =len(dataloader.dataset)
    for batch,(img_path,img,bb_targets) in enumerate(dataloader):
        # print(img.shape)
        outputs=model(img) # 计算预测输出
        loss,_=compute_loss(outputs,bb_targets,model) # 计算 loss function
        optimizer.zero_grad()# 将优化梯度归零
        loss.backward()
        optimizer.step()
        # clear()
        #print(f'Epoch:{epoch+1}, Step{batch+1}/{size)}, loss:{loss.item()}')
        print(f'Epoch:{epoch+1},Step{batch+1}/{size},loss:{loss.item()}')

    ################################################################################
    #                                                End                           #
    ################################################################################
```

五、实验数据及结果分析：

运行“proj5.ipynb”文件，运行结果如图 10 所示。

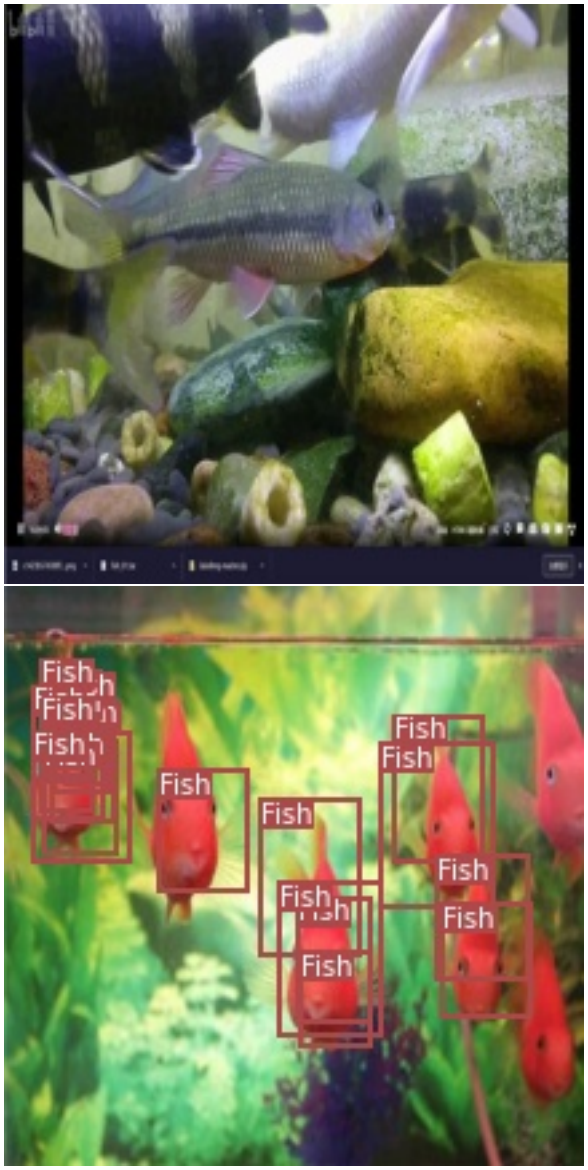
```
---- Training Model ----
Epoch:1,Step1/109,loss:5.6883649826049805
Epoch:1,Step2/109,loss:5.128265380859375
Epoch:1,Step3/109,loss:4.41349983215332
Epoch:1,Step4/109,loss:3.960383415222168
Epoch:1,Step5/109,loss:3.5438008308410645
Epoch:1,Step6/109,loss:2.993985891342163
Epoch:1,Step7/109,loss:2.6540727615356445
Epoch:1,Step8/109,loss:2.3840296268463135
Epoch:1,Step9/109,loss:1.8891925811767578
Epoch:1,Step10/109,loss:1.7955983877182007
Epoch:1,Step11/109,loss:2.089334487915039
Epoch:1,Step12/109,loss:1.4843039512634277
Epoch:1,Step13/109,loss:1.3394604921340942
Epoch:1,Step14/109,loss:1.2395603656768799
Epoch:1,Step15/109,loss:1.0651031732559204
Epoch:1,Step16/109,loss:1.0466809272766113
Epoch:1,Step17/109,loss:1.031522274017334
Epoch:1,Step18/109,loss:0.7342434525489807
Epoch:1,Step19/109,loss:1.064943552017212
Epoch:1,Step20/109,loss:0.6988198161125183
Epoch:1,Step21/109,loss:0.814171314239502
Epoch:1,Step22/109,loss:0.649197518825531
Epoch:1,Step23/109,loss:0.7970384359359741
Epoch:1,Step24/109,loss:0.6010976433753967
Epoch:1,Step25/109,loss:0.5510596632957458
Epoch:1,Step26/109,loss:0.5846338272094727
Epoch:1,Step27/109,loss:0.6532424688339233
```

```
Detecting objects: 100%|██████████| 13/13 [00:04<00:00,  2.67it/s]
Computing AP: 100%|██████████| 1/1 [00:00<?, ?it/s]

---- Evaluating Model ----
['Fish'] 0
+-------+------------+---------+
| Index | Class name | AP      |
+-------+------------+---------+
| 0     | Fish       | 0.03148 |
+-------+------------+---------+
---- mAP 0.031479631479631474
```

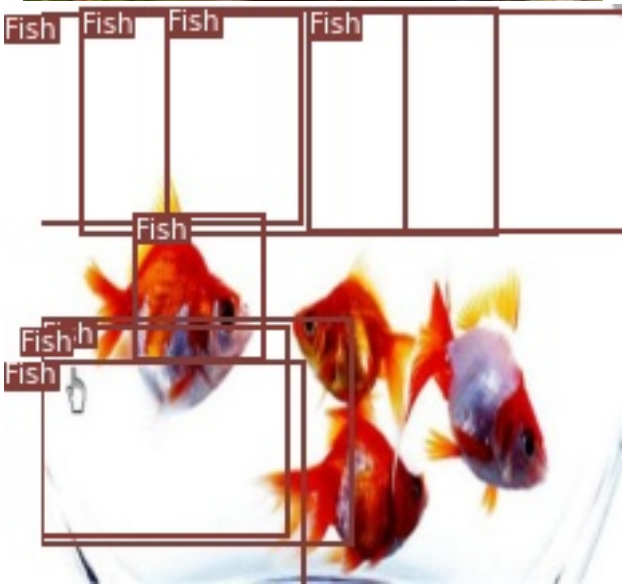
```
Performing object detection:

Saving images:
(0) Image: 'data/samples\test (1).jpg'
(1) Image: 'data/samples\test (10).jpg'
        + Label: Fish, Conf: 0.38541
        + Label: Fish, Conf: 0.36426
        + Label: Fish, Conf: 0.32340
        + Label: Fish, Conf: 0.26004
        + Label: Fish, Conf: 0.25825
        + Label: Fish, Conf: 0.23418
(2) Image: 'data/samples\test (11).jpg'
        + Label: Fish, Conf: 0.30449
(3) Image: 'data/samples\test (2).jpg'
        + Label: Fish, Conf: 0.55987
        + Label: Fish, Conf: 0.54847
        + Label: Fish, Conf: 0.49720
        + Label: Fish, Conf: 0.33117
        + Label: Fish, Conf: 0.32074
        + Label: Fish, Conf: 0.30763
        + Label: Fish, Conf: 0.29559
        + Label: Fish, Conf: 0.27445
        + Label: Fish, Conf: 0.26168
        + Label: Fish, Conf: 0.25817
        + Label: Fish, Conf: 0.24872
        + Label: Fish, Conf: 0.24143
        + Label: Fish, Conf: 0.23582
        + Label: Fish, Conf: 0.22316
        + Label: Fish, Conf: 0.21560
        + Label: Fish, Conf: 0.21167
        + Label: Fish, Conf: 0.20755
        + Label: Fish, Conf: 0.20519
        + Label: Fish, Conf: 0.20149
        + Label: Fish, Conf: 0.20126
(4) Image: 'data/samples\test (3).jpg'
        + Label: Fish, Conf: 0.49681
        + Label: Fish, Conf: 0.41957
        + Label: Fish, Conf: 0.26982
        + Label: Fish, Conf: 0.20163
(5) Image: 'data/samples\test (4).jpg'
(6) Image: 'data/samples\test (5).jpg'
        + Label: Fish, Conf: 0.29362
        + Label: Fish, Conf: 0.25193
        + Label: Fish, Conf: 0.23736
        + Label: Fish, Conf: 0.23427
        + Label: Fish, Conf: 0.21618
        + Label: Fish, Conf: 0.21151
        + Label: Fish, Conf: 0.20704
        + Label: Fish, Conf: 0.20459
(7) Image: 'data/samples\test (6).jpg'
        + Label: Fish, Conf: 0.63114
        + Label: Fish, Conf: 0.37615
        + Label: Fish, Conf: 0.27502
        + Label: Fish, Conf: 0.26442
        + Label: Fish, Conf: 0.22914
        + Label: Fish, Conf: 0.21456
(8) Image: 'data/samples\test (7).jpg'
        + Label: Fish, Conf: 0.56022
        + Label: Fish, Conf: 0.44568
        + Label: Fish, Conf: 0.27522
        + Label: Fish, Conf: 0.21318
(9) Image: 'data/samples\test (8).jpg'
        + Label: Fish, Conf: 0.36027
        + Label: Fish, Conf: 0.27259
        + Label: Fish, Conf: 0.26646
        + Label: Fish, Conf: 0.23637
(10) Image: 'data/samples\test (9).jpg'
        + Label: Fish, Conf: 0.57055
        + Label: Fish, Conf: 0.39253
        + Label: Fish, Conf: 0.26552
        + Label: Fish, Conf: 0.25294
        + Label: Fish, Conf: 0.21540
```

图 10 “proj5.ipynb”文件运行结果

“data”文件夹下的“sample”文件夹中鱼类图片的测试结果如图 11 所示。




Fish
Fish
Fish


Fish
Fish
Fish
Fish
Fish
Fish
Fish


Fish
Fish
Fish
Fish
Fish
Fish

图 11 “sample”文件夹中鱼类图片的测试结果

六、实验结论：

本实验基本实现了基于深度学习的鱼类检测，实现了数据预处理和训练模型等功能。程序的平均运行时间为 40 分钟，平均准确率为 3%。鱼类图片的检测效果不容乐观，有时，鱼没有被检测到；有时，同一条鱼上重复出现了检测框；有时，检测框出现在错误的位置。

七、总结及心得体会：

本实验使用的 PyTorch 的计算平台为 CPU，若采用 CUDA 作为计算平

台，开启 GPU 进行计算，可能对降低程序运行时间和提高准确率都有不小的帮助。

八、对本实验过程及方法的改进建议：

提供实验指导书。

报告评分：

指导教师签字：