華中科技大學

论文

基于 Fast RCNN 对图片物体的分类和定位

小 组 成 员： 刘泊、季慧、雷紫薇

班 级： 种子班 1601 班

指 导 教 师： 王兴刚

完 成 日 期： 2019 年 5 月 19 日

摘 要

从二十世纪中叶起，非结构化数据就以一个很快地速度在增长。然而，所产生的数据中有80％是非结构化的多媒体内容，未能把重点放在组织大数据的举措上，这个多媒体内容的很大一部分是图像。智能无线设备的迅速普及以及通过互联网共享图像的兴起，对这类内容的大规模增长作出了重大贡献。图像反映了人类知识、互动和对话的很大一部分。如今，图像数据中的大量知识为创造新的使用案例、应用和产品创造了极大的机会。几十年来，对图像的处理，理解、识别和定位一直是人工智能（AI）和机器学习（ML）中的一个巨大的技术挑战，但在过去的十年里，已经有了一些突破。图片比文字更具有影响力，因为它们往往更具吸引力。图像更可能被共享和转发。人们利用图像/视频来捕捉他们的特殊时刻。但是，图像已经发展成为一种交流手段。基于14年R-CNN的大获全胜，微软研究所的Ross Girshick提出了改进算法 Fast RCNN 。Fast R-CNN 在VGG16 network下训练速度比R-CNN快了9倍，测试时间更是缩短了213倍，不仅如此，mAP也达到了最高水平。本文主要沿用了Fast RCNN算法进行图片的识别和定位工作，期望能正确分类图像并对图像中的对象进行准确定位。

关键词：图片识别、物体定位、Fast RCNN

Abstract

From the middle of the twentieth century, unstructured data has grown at a rapid rate. However, 80% of the data generated is unstructured multimedia content, failing to focus on the initiative to organize big data. A large part of this multimedia content is images. The rapid spread of smart wireless devices and the rise of images shared over the Internet have contributed significantly to the massive growth of such content. The image reflects a large part of human knowledge, interaction and dialogue. Today, the vast amount of knowledge in image data creates tremendous opportunities for creating new use cases, applications, and products. For decades, the processing, understanding, identification and positioning of images has been a huge technical challenge in artificial intelligence (AI) and machine learning (ML), but in the past decade, there have been some breakthroughs. Pictures are more influential than words because they tend to be more attractive. Images are more likely to be shared and forwarded. People use images/video to capture their special moments. However, images have evolved into a means of communication. Based on the 14-year R-CNN win, Ross Girshick of the Microsoft Research Institute proposed the improved algorithm Fast R-CNN. Fast R-CNN is 9 times faster than R-CNN in VGG16 network, and the test time is 213 times shorter. Not only that, mAP has reached the highest level. In this paper, the Fast RCNN algorithm is used to identify and locate the image. It is expected that the image can be correctly classified and the objects in the image can be accurately located.

Key words： picture recognition, object location, Fast RCNN

目　录

1 实验概述

本次实验我们选取的是老师在 ppt 上展示的图像分类与定位的题目，使用 CNN 进行图片物体的分类和定位，图片集中的目标种类共五种，分别是猫、鸟、乌龟、狗和蜥蜴，其中每个种类的图片各有 180 张。需处理的图片大小为 128*128。评价指标为一个正确的定位意味着预测的类别标签是正确的同时预测 IoU>0.5，最后需报告平均正确率并展示一些定位结果。

2 相关原理及基本流程

2.1 实验原理

传统 RCNN 的基础思想是：

- 在图像中确定约 1000-2000 个候选框
- 对于每个候选框内图像块，使用深度网络提取特征
- 对候选框中提取出的特征，使用分类器判别是否属于一个特定类
- 对于属于某一特征的候选框，用回归器进一步调整其位置

Fast RCNN 针对 RCNN 在训练时是 multi-stage pipeline 和训练的过程中很耗费时间空间的问题进行改进。它主要是将深度网络和后面的 SVM 分类两个阶段整合到一起，使用一个新的网络直接做分类和回归。主要做以下改进:

1、 最后一个卷积层后加了一个 ROI pooling layer。ROI pooling layer 首先可以将 image 中的 ROI 定位到 feature map，然后是用一个单层的 SPP layer 将这个 feature map patch 池化为固定大小的 feature 之后再传入全连接层。

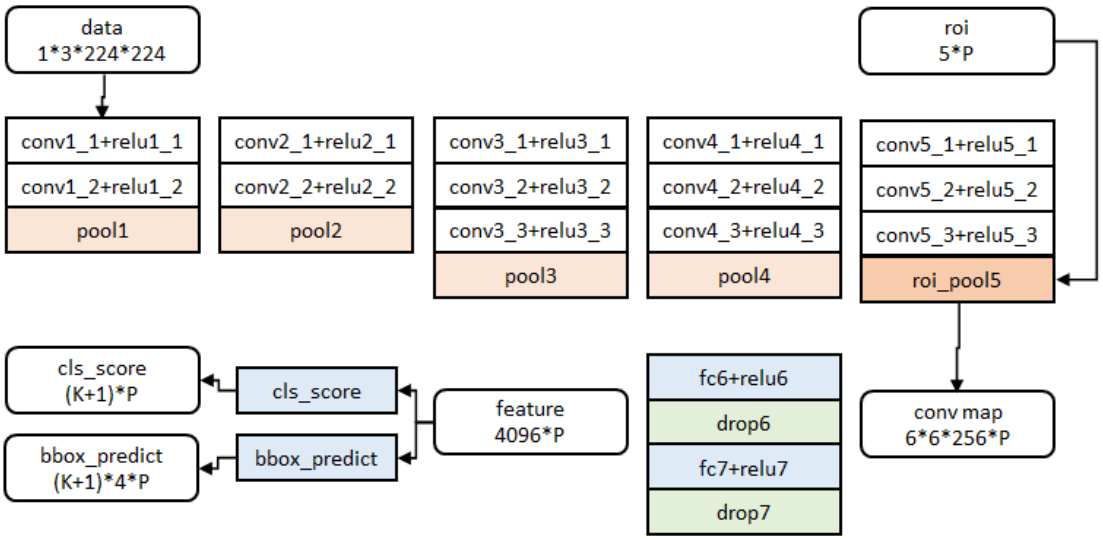
2、 损失函数使用了多任务损失函数(multi-task loss)，将边框回归直接加入到 CNN 网络中训练。

继 2014 年的 RCNN 之后，Ross Girshick 在 15 年推出 Fast RCNN，构思精巧，流程更为紧凑，大幅提升了目标检测的速度。

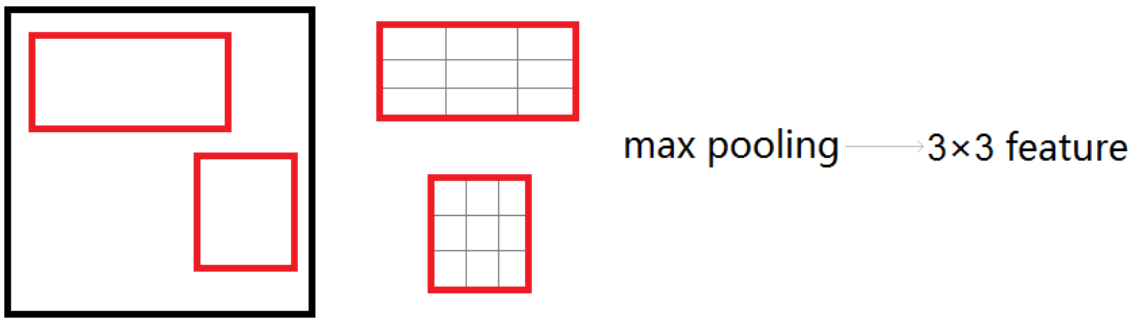
2.1.1 Fast RCNN 原理

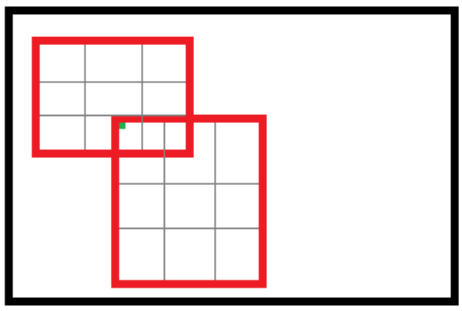
2. 1. 1. 1 特征提取网络

先构造基本结构，图像归一化直接送入网络。前五阶段是基础的 conv+relu+pooling 形式，在第五阶段结尾，输入 P 个候选区域。



Roi_ pool 层的测试，Roi_ pool 层将每个候选区域均匀分成 M×N 块，对每块进行 max pooling。将特征图上大小不一的候选区域转变为大小统一的数据，送入下一层。





2.1.1.2 网络参数训练

先进行参数初始化，网络除去末尾部分如下图，在 ImageNet 上训练 1000 类分类器。结果参数作为相应层的初始化参数。其余参数随机初始化。接下来对数据分层，在调优训练时，每一个 mini-batch 中首先加入 N 张完整图片，而后加入从 N 张图片中选取的 R 个候选框。这 R 个候选框可以复用 N 张图片前 5 个阶段的网络特征。最后构成训练数据。

2.1.1.3 分类与位置调整

第五阶段的特征输入到两个并行的全连层中，cls_score 层用于分类，输出 K+1 维数组 p，表示属于 K 类和背景的概率。 bbox_prdict 层用于调整候选区域位置，输出 4*K 维数组 t，表示分别属于 K 类时，应该平移缩放的参数。代价函数由两部分构成，loss_cls 层评估分类代价，loss_bbox 评估检测框定位代价。总代价为两者加权和。最后用全连接层提速。

2.1.2 实验指导

在 Fast RCNN 原理指导下，我们针对数据集给出了相应的指导

- 输入：3 x 128 x 128 RGB 图像
- 卷积层：使用卷积层提取图像的特征
- ROI 层：ROI 层将搜索框框住的部分应用最大池，转化成统一的数据发送到下一层
- 全链接层：通过全链接层分类
- cls_score：得到图像的分类信息
- bbox_predict：得到图像的位置信息

2.1.3 实验基本流程

使用 Fast RCNN 作为框架模型。Fast RCNN 模型首先使用 VGG16 作为基础模型，然后对卷积层的最后一层修改，并加入 ROI 层，ROI 层将 location 数据与图片数据相结合，得到感兴趣的区域。然后通过线性分类层，在最后输出加上两个并行的线性分类层，分别得到 5 个 label 的概率信息和 4 x 5 的 location 信息。使用 CrossEntropyLoss 计算得分的 Loss，使用 SmoothL1Loss

计算位置信息的 Loss，将两个 Loss 按比例相加得到最后的 Loss

代码实现如下

```
class FAST_RCNN(nn.Module):
    def __init__(self):
        super().__init__()

        # use vgg 16 bn <bn use batch norm layer>
        raw_net = torchvision.models.vgg16_bn(pretrained=True).cpu()
        # conv layer sequence
        self.seq = nn.Sequential(*list(raw_net.features.children())[:-1])
        self.roi_pool = SlowROIPool(output_size=(7, 7))

        # classifier layer sequence
        self.feature = nn.Sequential(*list(raw_net.classifier.children())[:-1])
        # print(self.feature)

        _x = Variable(torch.Tensor(1, 3, 128, 128))
        _r = np.array([[0., 0., 1., 1.]])
        _x = self.feature(self.roi_pool(self.seq(_x), _r).view(1, -1))
        feature_dim = _x.size(1)
        self.cls_score = nn.Linear(feature_dim, N_CLASS + 1)
        self.bbox = nn.Linear(feature_dim, 4 * (N_CLASS + 1))

        self.cel = nn.CrossEntropyLoss()
        self.sl1 = nn.SmoothL1Loss()
```

```
    def forward(self, imgs, rois):
        res = self.seq(imgs)
        res = self.roi_pool(res, rois)
        res = res.detach()
        res = res.view(res.size(0), -1)
        feat = self.feature(res)

        cls_score = self.cls_score(feat)
        bbox = self.bbox(feat).view(-1, N_CLASS + 1, 4)
        return cls_score, bbox
```

3 实验过程

3.1 数据处理

给定数据是一些图片和对应的 location.txt 文件，读取图片并存储到变量中，同时读取 location 文件，将读取到的字符分割并将后四个位置存储到对应的变量中，然后将 list 转换成 numpy 数组，

此时一共读取了 750 张图片作为训练集，150 张图片作为测试集，shape 为（750,128,128,3）和（150,128,128,3）；对应的 750 组 location 训练数据，150 组测试数据，shape 为（750,4）和（150，4），由于图片的 shape 与需要的 shape 有出入，所以使用 transpose 做维度变换，将图片测训练集和测试集的 shape 转换为（750,3,128,128）和（150,3,128,128）。

最后将测试集，训练集对应的图片数据，label 数据和 location 数据用同样的方式打乱，得到最终的训练数据和测试数据/

加载图片：

```
type_flag = 0
for dir_path in img_dir:
    img_list = os.listdir(dir_path)
    index = 0

    for img in img_list:
        if index < 150:
            train_img_set_x.append(mpimg.imread(os.path.join(dir_path, img)))
            train_img_set_y.append(type_flag)
        elif index >= 150 and not index >= 180:
            test_img_set_x.append(mpimg.imread(os.path.join(dir_path, img)))
            test_img_set_y.append(type_flag)
        index += 1
    type_flag += 1
```

加载位置数据：

```
for obj in file_obj:
    for index in range(180):
        line = [int(x) for x in obj.readline().strip().split()]
        if index < 150:
            train_box_set.append(line[1:])
        else:
            test_box_set.append(line[1:])
```

对数据重新排序：

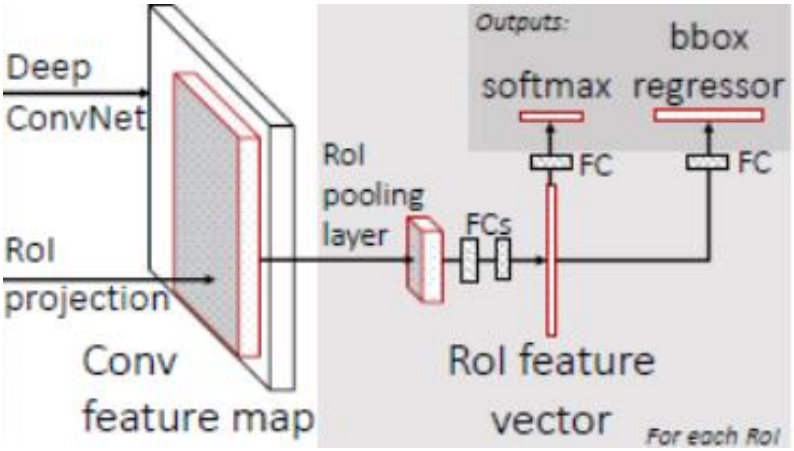
```
# random dataset
randomIndices = random.sample(range(x_img_tr.shape[0]), x_img_tr.shape[0])
x_img_tr = x_img_tr[randomIndices]
y_img_tr = y_img_tr[randomIndices]
box_tr = box_tr[randomIndices]

randomIndices = random.sample(range(x_img_te.shape[0]), x_img_te.shape[0])
x_img_te = x_img_te[randomIndices]
y_img_te = y_img_te[randomIndices]
box_te = box_te[randomIndices]
```

3.2 网络结构

输入的是 3 x 128 x 128 RGB 固定大小的图像，经过 5 个卷积层+2 个降采样层（分别跟在第一和第二个卷积层后面），进入 ROIPooling 层（其输入是 conv5 层的输出和 region proposal，region proposal 个数大约为 2000 个），再经过两个 output 都为 4096 维的全连接层，分别经过 output 各为 5 和 20 维的全连接层（并列的，前者是分类输出，后者是回归输出），最后接上两个损失层（分类是 softmax，回归是 smoothL1）。

Fast RCNN 模型的流程图如下：



3.2.1 ROIPooling

由于 region proposal 的尺度各不相同，而期望提取出来的特征向量维度相同，因此需要某种特殊的技术来做保证。ROIPooling 的提出便是为了解决这一问题的。其思路如下：

1、 将 region proposal 划分为 H×WH×W 大小的网格

2、 对每一个网格做 MaxPooling（即每一个网格对应一个输出值）

3、 将所有输出值组合起来便形成固定大小为 H×WH×W 的 feature map

4、 本次实验简化流程，采用 region proposal 然后做 MaxPooling 直接输出，得到固定大小的图像。代码如下图所示。

```
def forward(self, images, rois):
    roi_num = rois.shape[0]
    h = images.size(2)
    w = images.size(3)
    x1 = rois[:, 0]
    y1 = rois[:, 1]
    x2 = rois[:, 2]
    y2 = rois[:, 3]

    x1 = np.floor(x1 * w).astype(int)
    x2 = np.ceil(x2 * w).astype(int)
    y1 = np.floor(y1 * h).astype(int)
    y2 = np.ceil(y2 * h).astype(int)

    res = []
    for i in range(roi_num):
        img = images[i].unsqueeze(0)
        img = img[:, :, y1[i]:y2[i], x1[i]:x2[i]]
        img = self.maxpool(img)
        res.append(img)
    res = torch.cat(res, dim=0)
    return res
```

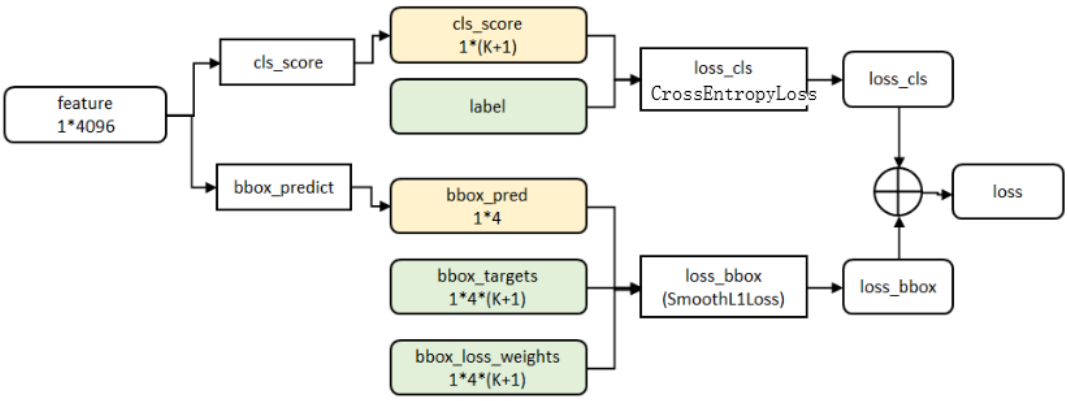
3.2.2 训练样本

训练过程中每个 mini-batch 包含 50 张图像和 50 个对应的 region proposal（即 ROI，1 个 ROI/张）。将图片数据和 label 数据和位置数据转换为 Tensor 之后，输入到网络中。

3.2.3 特征提取结构

1、 cls_score 层用于分类，输出 K+1 维数组 p，表示属于 K 类和背景的概率。

2、 bbox_prdict 层用于调整候选区域位置，输出 4*K 维数组 t，表示分别属于 K 类时，应该平移缩放的参数。



3.2.4 损失函数

使用 CrossEntropyLoss 计算分类的 Loss，使用 SmoothL1Loss 计算候选区位置的 Loss。

交叉熵损失 CrossEntropyLoss 测量分类模型的性能，该分类模型的输出是 0 到 1 之间的概率值。交叉熵损失随着预测概率偏离实际标签而增加。

Smooth L1 损失用于对某些物体检测系统进行盒式回归（SSD，快速/快速 RCNN），这种损失对异常值的敏感性低于其他回归损失。

```
def calc_loss(self, probs, bbox, labels, gt_bbox):
    loss_sc = self.cel(probs, labels)
    lbl = labels.view(-1, 1, 1).expand(labels.size(0), 1, 4)
    mask = (labels != 0).float().view(-1, 1).expand(labels.size(0), 4)

    loss_loc = self.sl1(bbox.gather(1, lbl).squeeze(1) * mask, gt_bbox * mask)
    lmb = 1.0
    loss = loss_sc + lmb * loss_loc
    return loss, loss_sc, loss_loc
```

3.2.5 IOU 计算

IOU 计算如下，首先计算实际的框的面积和标出的框的面积，然后计算出重叠的面积，然后通过 重叠面积/（总面积 – 重叠面积）计算出 IOU 大小

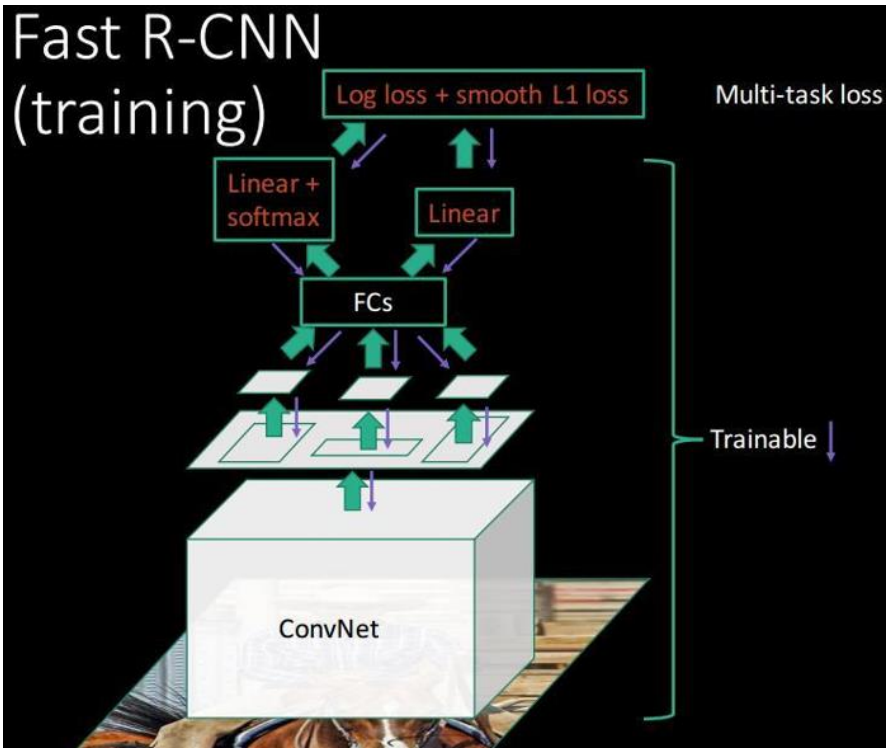
```
def calc_ious(ex_rois, gt_rois):
    ex_area = (1. + ex_rois[:, 2] - ex_rois[:, 0]) * (1. + ex_rois[:, 3] - ex_rois[:, 1])
    gt_area = (1. + gt_rois[:, 2] - gt_rois[:, 0]) * (1. + gt_rois[:, 3] - gt_rois[:, 1])
    area_sum = ex_area + gt_area

    lb = np.maximum(ex_rois[:, 0], gt_rois[:, 0])
    rb = np.minimum(ex_rois[:, 2], gt_rois[:, 2])
    tb = np.maximum(ex_rois[:, 1], gt_rois[:, 1])
    ub = np.minimum(ex_rois[:, 3], gt_rois[:, 3])

    width = np.maximum(1. + rb - lb, 0.)
    height = np.maximum(1. + ub - tb, 0.)
    area_i = width * height
    area_u = area_sum - area_i

    ious = area_i / area_u
    return ious.sum() / ex_rois.shape[0]
```

3.2.6 训练整体框架



3.3 测试

测试与训练类似，将 150 x 3 x 128 x 128 大小的图片输入到训练好的网络中，得到 5 种类的分数，和 4 x 5 的位置信息。

将得到的分数，选择一个最大的概率的坐标就是该图片的种类，然后与 label 数据进行对比，

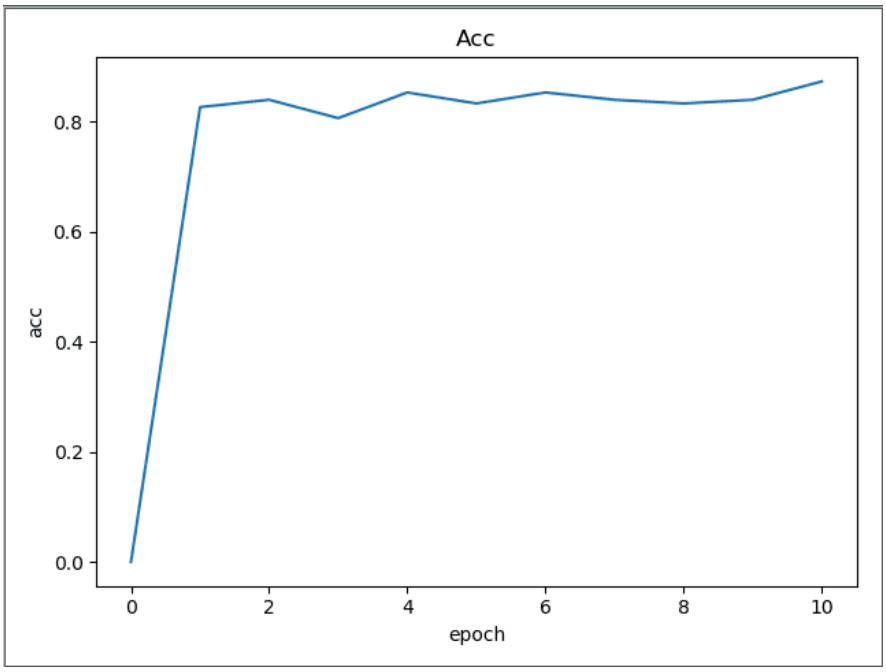
得到正确率信息。

将得到的位置信息，由上面得到的种类信息，选择一个位置信息，然后与正确的位置信息对比计算得到 IOU

3.4 测试结果分析

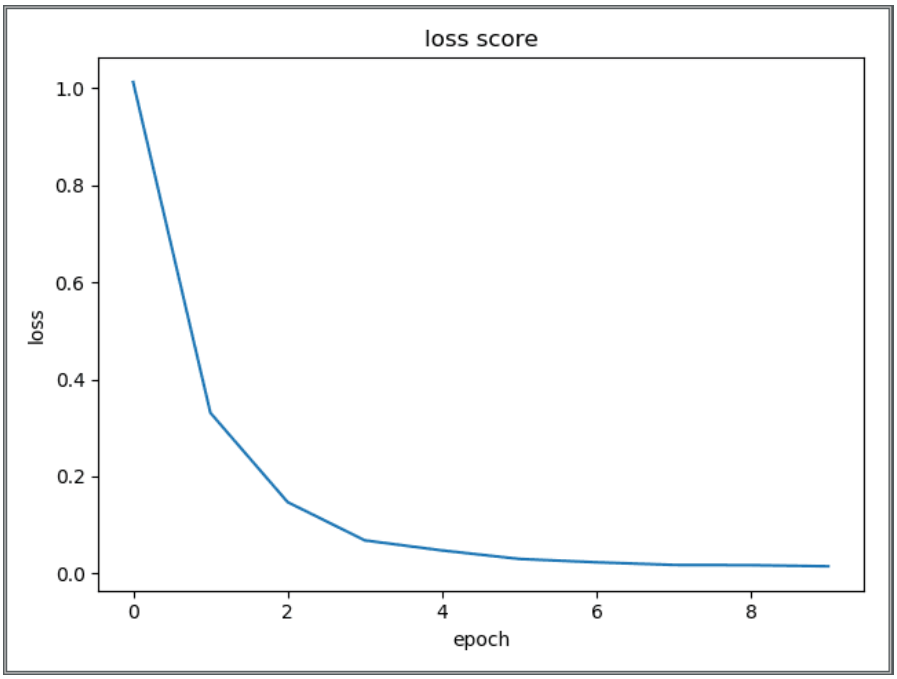
3.4.1 正确率

由于我们使用的基础网络 VGG 是经过预训练的，所以一开始正确率就比较高，都在 0.8 以上，如下图所示。

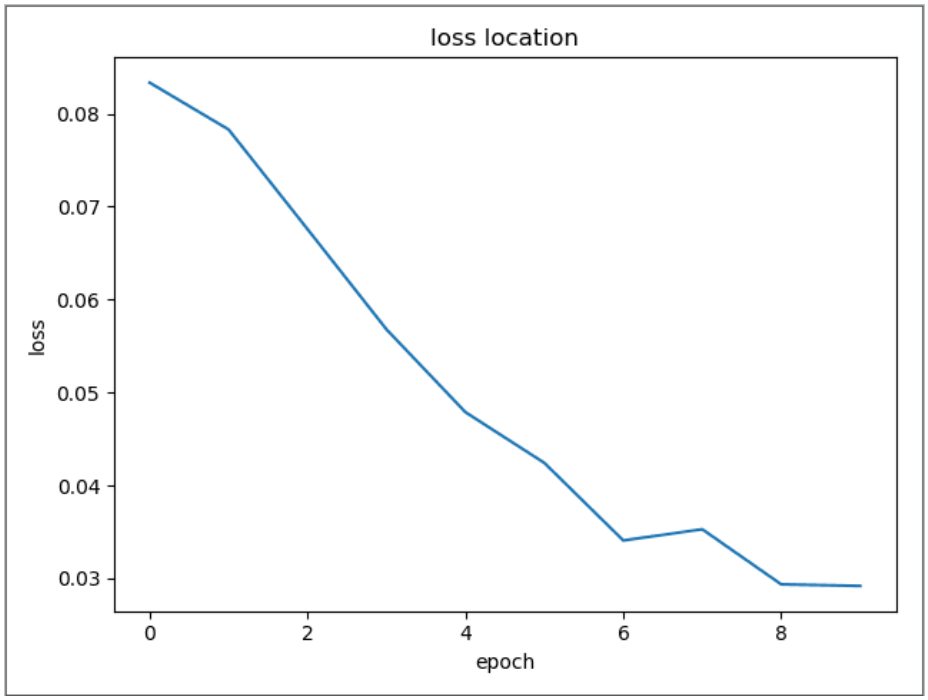


3.4.2 LOSS

种类的分类 loss 随着 epoch 增大变化情况如下图所示。

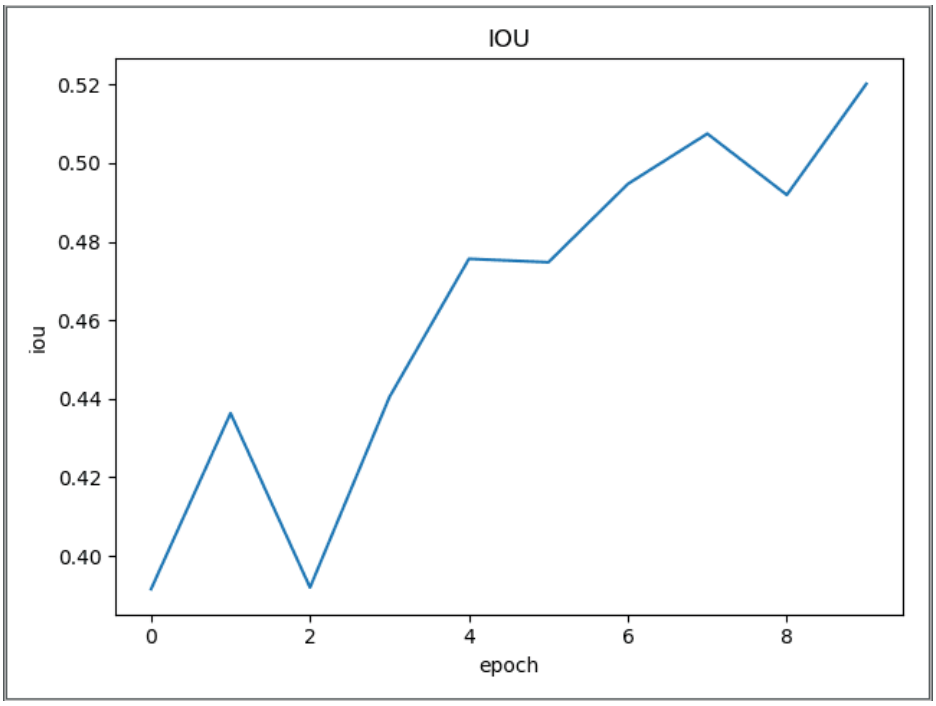


位置 Loss 随着训练的 epoch 增大变化情况如下图所示。

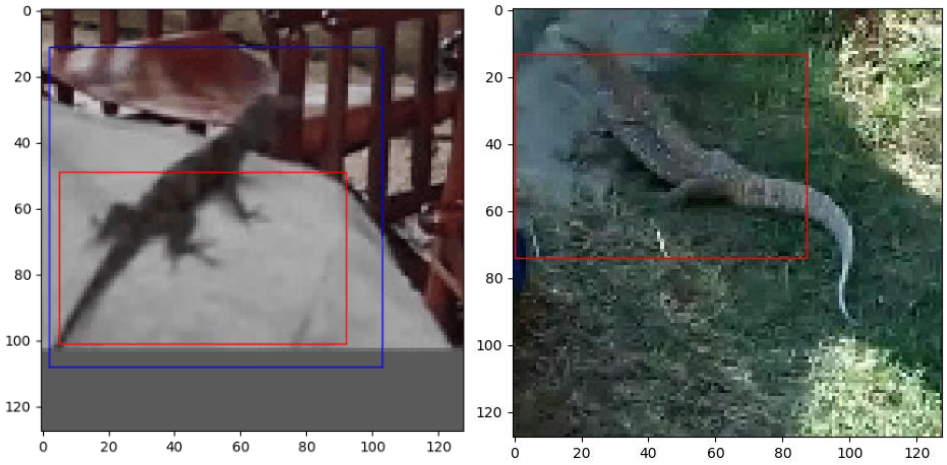


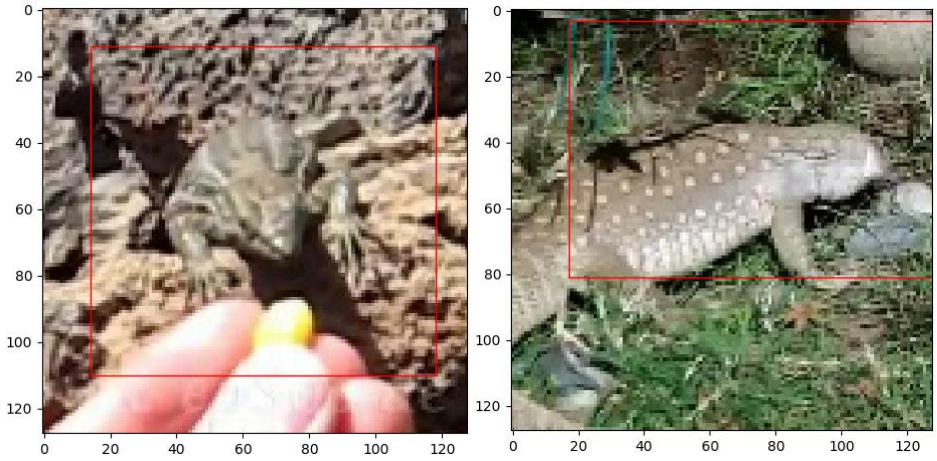
3.4.3 IOU

随着 epoch 的增大，IOU 变化情况如下图所示，最终为 0.52 > 0.5



抽取部分图片，查看标注的框的信息，如下图所示：




0
20
40
60
80
100
120
0
20
40
60
80
100
120
0
20
40
60
80
100
120
0
20
40
60
80
100
120

4 理解和体会

在本次实验中，首先遇到了网络选择的问题，由于 FAST-RCNN 出现比较早资料可能比较多，而且训练速度和训练结果也比较可观，Fast-RCNN 借鉴了 SPP 的思路，在不用图像金字塔池化的方式下，提出简化版的 ROI 池化层。通过加入候选框映射功能，使网络能够反向传播。YOLO 不再采用 two-stage 的这种思路，将图像分类和位置检测完全集成在一个网络里，将边框位置设计称可以回归的参数，直接由网络回归得出。速度极大的得到提升，但是检测精度跟 faster-RCNN 相比，降低了不少。所以就选择了 FAST-RCNN 作为我们的基础网络，后来发现关于 FAST-RCNN 网络的资料很少，研究这个网络的人也比较少，可能是因为 FASTER-RCNN 的出现比较快然后就把 FAST-RCNN 给淹没了吧。然后我们根据论文和前人做的实验，经过修改以适应我们的训练模型，得到了 FAST-RCNN 的基本网络结构，将数据集加载并转换成一定的可输入的格式，最后将数据输入到网络并对训练的网络进行测试。

这个实验最关键的部分在于 ROI 部分，其余的网络模型例如 FASTER-RCNN 等与 FAST-RCNN 的区别就在于 ROI 部分，FAST-RCNN 的 ROI 层中一张图片中是有很多的区域可以选择的，但是我们的数据集每张图片只有一个标注区域，所以在这部分对原模型进行了修改，每个图片只提取出一个 ROI 然后经过最大池化后输出到下一层中。

经过实验我们对图像识别这个领域有了一定的初步理解，从服务器环境的构建到处理数据，从理论学习和动手实验，真切感受到了算法的一点一点实现过程，收获良多。

致谢

本次实验过程中，我们得到了很多在帮助。有很多机器学习的老师在网上公布了他们的学习资料和笔记，而很多框架也已经开发完善，正是因为高度活跃的开源社区，我们的学习才会这样便利，感谢他们。同时，也很感谢小组成员在项目压力十分大的情况下仍然勇于担当，共同帮助，一起交流学习，最后完成实验，感谢！