模式识别大作业报告

——李昭阳 2021013445

一、 xml 文件解析

XML 是一种用于存储和传输数据的文本文件格式。它的设计目标是提供一种通用的标记语言，用于描述数据的结构和内容。根据其文本格式的特点，解析其标签，编写代码如下。

```
def convert_annotation(year, image_id, list_file):
    file = open(os.path.join(VOCdevkit_path, 'VOC' + year + '/Annotations/' + image_id + '.xml'))
    tree = ET.parse(file)
    root = tree.getroot()

    for obj in root.iter('object'):
        cls = obj.find('name').text
        if cls not in classes:
            continue
        cls_id = classes.index(cls)
        xmlbox = obj.find('bndbox')
        box = (int(float(xmlbox.find('xmin').text)), int(float(xmlbox.find('ymin').text)),
            int(float(xmlbox.find('xmax').text)), int(float(xmlbox.find('ymax').text)))
        list_file.write(" " + ",".join([str(coord) for coord in box]) + ',' + str(cls_id))

        nums[classes.index(cls)] = nums[classes.index(cls)] + 1
```

在以上的过程中，我读取了 xml 文件的数据，通过 xml.etree.ElementTree 遍历 xml 文件中的 object 标签，跳过不属于样本标签集，将 bndbox 的对角坐标、对应类别 cls 名称拼凑起来。由于数据集几乎不变，因此此处单独编写脚本，将读取到的数据写入一个 txt 文件储存，并在训练初始化时直接读取 dataset 而非生成，有利于降低每次初始化的时间。

训练、预测数据使用了 Pytorch 的 Dataloder，get item 的方法如下。对训练数据进行随机的数据增强，返回值为[变换后的图，变换后的框，框内标签]。

```
def __getitem__(self, index):
    index = index % self.length

    # 训练时进行数据的随机增强(random=self.train)
    # 验证时不进行数据的随机增强
    image, the_box = self.get_data(self.annotation_lines[index], self.input_shape[0:2], random=self.train)
    image = np.transpose(preprocess_input(np.array(image, dtype=np.float32)), (2, 0, 1))
    bndbox_data = np.zeros((len(the_box), 5))
    if len(the_box) > 0:
        bndbox_data[:len(the_box)] = the_box

    box = bndbox_data[:, :4]
    label = bndbox_data[:, -1]
    return image, box, label
```

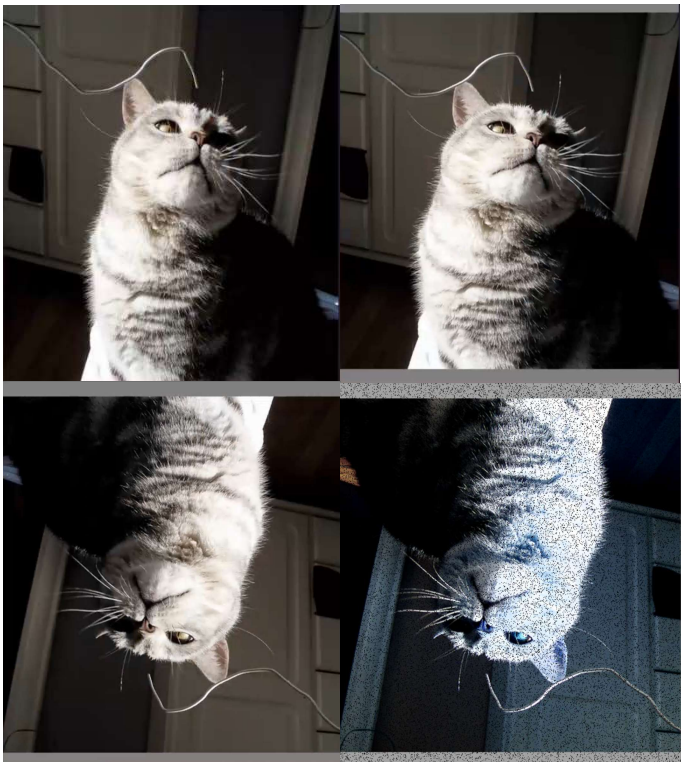
在数据增强方面，通过对图像的随机拉伸压缩、随机左右翻转、颜色随机变换、添加随机噪声等，制造了丰富的新训练数据集。

```
# 计算新的宽高比并且进行随机的扭曲
new_ar = image_w / image_h * self.rand(1 - jitter, 1 + jitter) / self.rand(1 - jitter, 1 + jitter)
scale = self.rand(.25, 2)
if new_ar < 1:
    new_h = int(scale * h)
    new_w = int(new_h * new_ar)
else:
    new_w = int(scale * w)
    new_h = int(new_w / new_ar)
image = image.resize((new_w, new_h), Image.BICUBIC)
```

```python
# 随机翻转
flip_lr = self.rand() < .5
flip_tb = self.rand() < .5
if flip_lr:
    image = image.transpose(Image.FLIP_LEFT_RIGHT)
if flip_tb:
    image = image.transpose(Image.FLIP_TOP_BOTTOM)

data = np.array(image, np.uint8)

# 将图像映射到HSV色域，随机变色，色相（hue）、饱和度（sat）和亮度（val）
r = np.random.uniform(-1, 1, 3) * [hue, sat, val] + 1
hue, sat, val = cv2.split(cv2.cvtColor(data, cv2.COLOR_RGB2HSV))

x = np.arange(0, 256, dtype=r.dtype)
rand_hue = ((x * r[0]) % 180).astype(data.dtype)
rand_sat = np.clip(x * r[1], 0, 255).astype(data.dtype)
rand_val = np.clip(x * r[2], 0, 255).astype(data.dtype)

data = cv2.merge((cv2.LUT(hue, rand_hue), cv2.LUT(sat, rand_sat), cv2.LUT(val, rand_val)))

# 生成随机噪声
noise = self.rand() < .5
if noise:
    # 生成一个与图像大小相同的随机二值遮罩
    mask = np.random.rand(data.shape[0], data.shape[1]) > 0.1
    # 将遮罩应用于图像
    data[np.where(mask==False)] = 0

data = cv2.cvtColor(data, cv2.COLOR_HSV2RGB)
```

根据以上代码，得到以下为数据增强效果，左上为原始图片，右上为拉伸压缩变换，左下为左右、上下翻转，右下为随机颜色变换以及随机噪声。

事实上还可以进行沿对角线翻转以及中心旋转，但是前者为了简单起见仅采取左右、上下翻转，后者考虑到中心旋转可能会损失一些图像边缘信息，故确定如上数据增强策略。

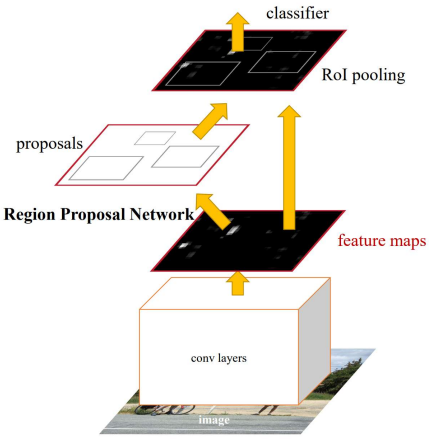


二、 搭建目标检测神经网络

在目标检测方面，经过调研，我选取了 Faster RCNN 作为目标检测的神经网络模型。Faster RCNN 的核心思想是通过引入 RPN 实现端到端的目标检测。

首先，通过 CNN 提取图像特征。接着，RPN 在这些特征上生成候选目标框的提议，为每个提议分配得分。候选框经过 ROI Pooling 操作后，送入分类器和边界框回归器，分别用于判断是否包含目标和调整框的位置。最后，通过非极大值抑制，筛选出最终的目标检测结果。

其算法如下图，



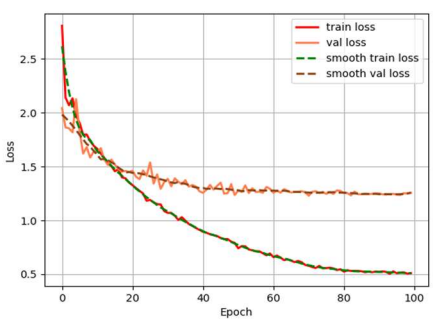
对于 CNN 网络，我选择了 VGG16 作为主干，通过多个“映射组”（两个 n 通道 3x3 卷积层 + 一个步长为 2 的 2x2 最大值池化层），不断提取图像特征。每次池化之后，图像尺寸都减小一半。CNN 将 600x600 的图像映射为 37x37 的 feature map。

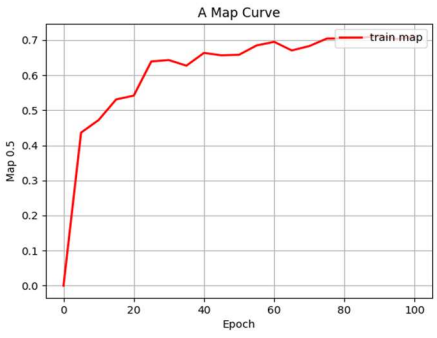
对于 RPN 网络，首先在 feature map 上将每个整数点取为特征点，每个特征点对应原图像的一个区域。对每个特征点，以其为中心以不同的长宽比生成 9 个矩形先验框（anchor）；接下来将 RPN 分为两条，一条通过 softmax 层判断先验框内是否存在物体，另一条用一个 1x1 卷积层（输出大小 9x4=36，即每个点的 9 个框都有 4 个回归参数——左上角 x、y、宽度 w、高度 h），进行回归预测，得到先验框的建议偏移量。最后通过 proposal layer 综合计算精确的建议框（RoI）。

对于 RoI Pooling 网络，目的是统一建议框的大小，以便于接下来的 Classify。此处采用了 Pytorch 内置的 RoI Pool 函数，得到了固定大小的 RoI（7x7）。

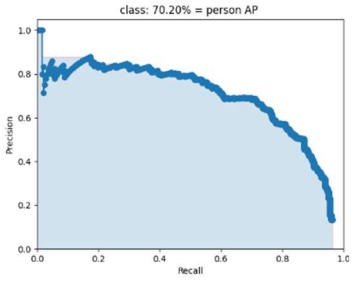
对于 Classify 网络，使用多个全连接层，将 RoI 映射出精确的偏移量、RoI 包含物体的分类。

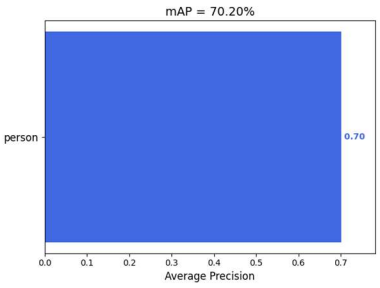
训练网络，训练效果如下。可以看到在 100 个 epoch 内，训练集和验证集的 loss 均快速下降，同时测试集 mAP 快速上升，逐渐趋向于 71%。



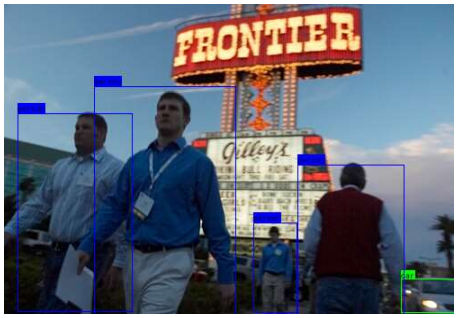


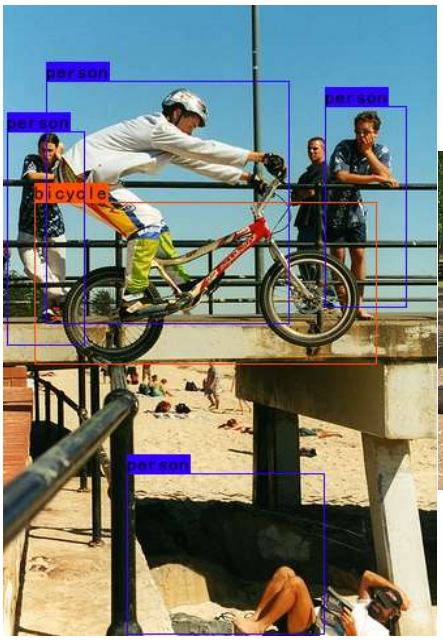
由于测试集属性问题，数据极度偏向于 person 类，其最终 ap、map 图像如下

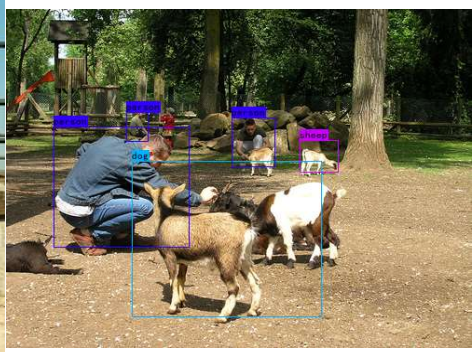




将测试集输入网络，得到预测效果如下。







通过上两图可知，该网络有较好的性能，即使一些重叠、边角、远小物体也能取得较好的检测效果；但是由下两图可知，该网络还存在提升空间，如左图靠下建议框偏移错误、右图将 sheep 识别为 dog。我认为还可以通过调节一些超参数、微调网络结构来提升预测效果。

预测核心代码如下。在挑选最终预测框时采用了非极大抑制(nms)：首先，对所有的边界框按照其检测得分进行降序排序。选择得分最高的边界框，将其保留，并从候选框列表中移除。接下来遍历剩余的边界框，去除与已选择的框有较大重叠（IoU 大于设定阈值）的框。这样可以确保在同一个目标位置不会有多个边界框。重复进行上述步骤，直到处理完所有的边界框。这样做可以降低目标检测算法输出中冗余边界框，同时由于有 confidence 的限制，可选的边界框进一步减少。最后可得一系列置信度较高的预测框。

```
for c in range(1, self.num_classes):
    # 取出属于该类的所有框的置信度判断是否大于门限
    c_confs = prob[:, c]
    c_confs_m = c_confs > confidence

    if len(c_confs[c_confs_m]) > 0:
        # 取出得分高于confidence的框
        boxes_to_process = cls_bbox[c_confs_m, c]
        confs_to_process = c_confs[c_confs_m]

        keep = nms(
            boxes_to_process,
            confs_to_process,
            nms_iou
        )

        # 取出在非极大抑制中效果较好的内容
        good_boxes = boxes_to_process[keep]
        confs = confs_to_process[keep][:, None]
        labels = (c - 1) * torch.ones((len(keep), 1)).cuda() if confs.is_cuda else (c - 1) * torch.ones(
            (len(keep), 1))

        # 拼装label、置信度、框的位置
        c_pred = torch.cat((good_boxes, confs, labels), dim=1).cpu().numpy()
        # 添加进result里
        results[-1].extend(c_pred)
```

三、 构想实例分割神经网络

本部分未实现代码，但是有一个简单的分割思路。在查阅资料过程中，我发现 FCN 可以实现语义分割，于是我想是否有可能先训练好一个 FCN，然后在预测时将图像通过 RoI 划分为多个小图像，分别通过该 FCN 求得掩膜，最后拼接在同一张图片上。

由于 FCN 训练的 loss 是求取预测掩膜和实际掩膜的交叉熵，不适用于基于边框的弱监督。那么借鉴 BoxInst[1]文章的 loss 设定，将 loss 改为如下两式之和。

$$\begin{aligned} L_{proj} &= L(\mathrm{Proj}_x(\tilde{\boldsymbol{m}}), \mathrm{Proj}_x(\boldsymbol{b})) + L(\mathrm{Proj}_y(\tilde{\boldsymbol{m}}), \mathrm{Proj}_y(\boldsymbol{b})) \\ &= L(\max_y(\tilde{\boldsymbol{m}}), \max_y(\boldsymbol{b})) + L(\max_x(\tilde{\boldsymbol{m}}), \max_x(\boldsymbol{b})) \\ &= L(\tilde{\boldsymbol{l}}_x, \boldsymbol{l}_x) + L(\tilde{\boldsymbol{l}}_y, \boldsymbol{l}_y), \end{aligned}$$

$$\begin{aligned} L_{pairwise} = -\frac{1}{N} \sum_{e \in E_{in}} & y_e \log P(y_e = 1) \\ & + (1 - y_e) \log P(y_e = 0), \end{aligned}$$

L_{proj} 是反映了掩膜和边框的在 x、y 轴上投影的重合度，$L_{pairwise}$ 刻画了边界的颜色变化，结合以上应该可以取得一个较好的结果。

参考文献

[1] Z. Tian, C. Shen, X. Wang and H. Chen, "BoxInst: High-Performance Instance Segmentation with Box Annotations," 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Nashville, TN, USA, 2021, pp. 5439-5448, doi: 10.1109/CVPR46437.2021.00540.