深 圳 大 学 实 验 报 告

课程名称：人工智能课程实训

实验项目名称：实验 3 模型优化实践

学院：计算机与软件学院

专业：软件工程（腾班）

指导教师：王旭

报告人：洪子敬 学号：2022155033 班级：腾班

实验时间：2024 年 10 月 17 日至 11 月 06 日

实验报告提交时间：2024 年 11 月 05 日

教务处制

实验目的与要求：

目标：

1. 了解模型训练前数据预处理和增强方法；
2. 掌握对象检测模型 YOLO 的训练与优化基本方法；
3. 掌握有方向的 YOLO-OBB 模型的基本原理与优化方法。

基本要求：

1. 将众包完成的口算数据集通过脚本/代码处理的方式，完成数据集的准备，训练集/测试集的比例划分；
2. 在训练前完成预处理操作，不仅限于翻转，加噪等视觉数据增强方法；
3. 基于 YOLO 预训练模型在口算数据集进行模型微调训练，在实验内容中给出预处理前后不同模型在测试集上的性能对比。
4. 使用 Weights & Biases 工具完成模型训练的可视化。

高级要求：

5. 通过代码处理的方式对数据集中的图片进行旋转，扭曲等几何失真（opencv 库中图像处理仿射变换等算子）仿真，以模仿真实场景的拍摄失真。
6. 基于 YOLO-OBB 模型在增强后的数据集上进行模型微调训练，在实验内容中给出预处理前后不同模型在测试集上的性能对比。

方法、步骤：

1. 准备数据集，按照 7：3 比例划分训练集和测试集，并转换.xml 格式文件为对应图片的标签数据；
2. 选取已经划分好的训练集中 20%的数据进行数据增强（本实验主要运用翻转、加噪和颜色变换等视觉数据增强方法）；
3. 在 Colab 平台上对 YOLOv5 预训练模型进行部署，并用上述准备好的数据集进行性能测试，比较数据增强前后的性能差异；
4. 使用 W&B 工具对上述训练过程进行可视化，并分析训练结果。

实验过程及内容：

1. 数据集准备

实验二我们已经完成了对口算数据集的标注并转换成 yolo 格式，而本实验提供了这些数据集的.xml 格式文件，记录了图片的算式框位置。首先我们需要先对原始图片进行划分，按照 7：3 的比例划分训练集和测试集（随机数据分布），设置取原始图片的前 500 张图片（即训练集 350 张，测试集 150 张）；接着通过编写函数 convert_voc_to_yolo 获得这些图片种算术框 yolo 格式的标签数据，函数编写代码如下所示：

```python
def convert_voc_to_yolo(xml_file, class_names):
    tree = ET.parse(xml_file)
    root = tree.getroot()

    # 获取图像文件名
    filename = root.find('filename').text
    width = int(root.find('size/width').text)
    height = int(root.find('size/height').text)

    yolo_data = []

    # 遍历每个对象
    for obj in root.findall('object'):
        class_name = obj.find('name').text
        if class_name not in class_names:
            continue  # 如果类名不在定义的类名列表中，跳过

        class_id = class_names.index(class_name)  # 获取类的索引
        bbox = obj.find('bndbox')
        x_min = int(bbox.find('xmin').text)
        y_min = int(bbox.find('ymin').text)
        x_max = int(bbox.find('xmax').text)
        y_max = int(bbox.find('ymax').text)

        # 计算中心点和宽高
        x_center = (x_min + x_max) / 2 / width
        y_center = (y_min + y_max) / 2 / height
        w = (x_max - x_min) / width
        h = (y_max - y_min) / height

        # 添加到YOLO 格式数据列表
        yolo_data.append(f"{class_id} {x_center} {y_center} {w} {h}")

    return filename, yolo_data
```

图 1. convert_voc_to_yolo 函数代码图

得到的数据集存放格式如下所示：

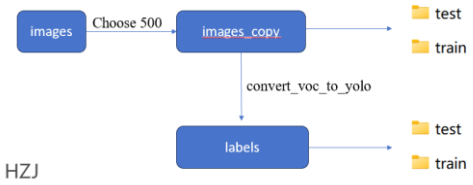

图 2. 数据集格式要求图

2. 数据增强

数据增强(Data Augmentation)是一种常用的数据处理技术，主要用于解决机器学习和深度学习中的数据不足或者数据不平衡问题。数据增强的核心思想是通过对现有数据进行一定程度的变换，生成新的数据样本，从而扩大数据集的规模和样本的多样性，以提高模型的泛化能力。

根据增强方法的不同，数据增强可以分为以下几类：

a) 随机增强：通过随机旋转、翻转、剪裁等方式对现有数据进行处理；

b) 基于模型的增强：通过训练生成模型（如 GAN）来生成高质量数据样本；

c) 基于规则的增强：通过定义一系列规则对现有数据进行处理，生成新数据。

由于做实验的时间较为紧张，本实验采用了较为基础的数据增强方法，即基于 opencv 的方法，包括翻转、加噪声和颜色变换（亮度、对比度变换），并按照 3：3：4 的比例进行图像的随机增强。此外，本实验采用使用训练集中的 20%进行数据增强，图像也是随机挑选的。三个变换函数代码如下：

```python
def augment_flip(image_path, output_path):
    image = Image.open(image_path)
    aug_image = image.transpose(Image.FLIP_LEFT_RIGHT)
    aug_image.save(output_path)


1 usage
def augment_noise(image_path, output_path):
    image = Image.open(image_path)
    np_image = np.array(image)
    noise = np.random.normal( loc: 0,  scale: 25, np_image.shape).astype(np.uint8)
    noisy_image = np.clip(np_image + noise,  a_min: 0,  a_max: 255)
    Image.fromarray(noisy_image).save(output_path)
    # print(f"保存噪声图像到: {output_path}")  # 添加打印语句
```

```python
def augment_color(image_path, output_path):
    image = Image.open(image_path)
    # 随机调整亮度、对比度和饱和度
    enhancer = ImageEnhance.Brightness(image)
    image = enhancer.enhance(random.uniform( a: 0.5,  b: 1.5))
    enhancer = ImageEnhance.Contrast(image)
    image = enhancer.enhance(random.uniform( a: 0.5,  b: 1.5))
    enhancer = ImageEnhance.Color(image)
    image = enhancer.enhance(random.uniform( a: 0.5,  b: 1.5))
    image.save(output_path)
```

图 3. 三个变换函数代码图

具体实现是将选择到的数据增强图片，按照比例随机地进行上述变换，并将变换结果替换原图片，保持训练集大小的一致；此外，由于这些图片均是算式图片，所以标签均是一些算术框的位置，三种变换中只有翻转会改变框的位置，于是将标签中算术框的位置进行对称反转即可，变换代码如下：

```
filename, yolo_format = convert_voc_to_yolo(xml_file_path, class_names)
output_file_path = os.path.join(output_train_folder, os.path.splitext(xml_file)[0] + '.txt')
with open(output_file_path, 'w') as f:
    for line in yolo_format:
        # 更新标注中的 x_center
        class_id, x_center, y_center, w, h = line.split()
        new_x_center = 1 - float(x_center)  # 翻转后中心点x坐标变化
        f.write(f"{class_id} {new_x_center} {y_center} {w} {h}\n")
```

图 4. 翻转时标签变换代码图

以下是部分图片的变换结果：

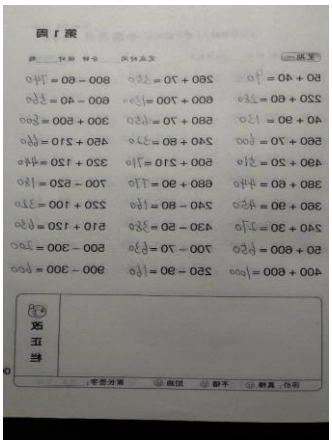

图 5.flip

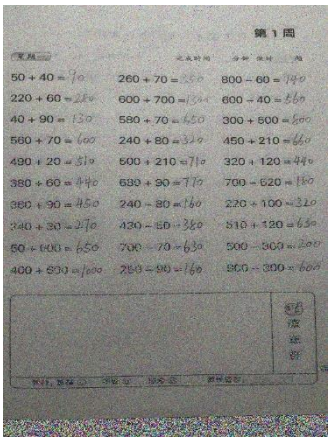

图 6.noise

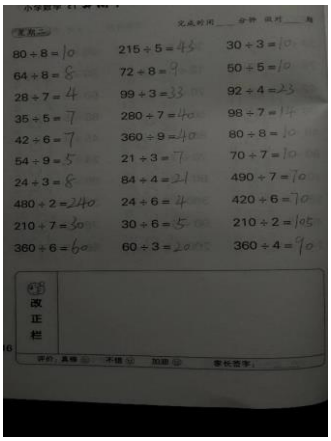

图 7.color

3. 模型训练

在实验一中我们已经复现了 Yolov5s 模型，本实验同样是在 Github 上进行 Git clone 操作，运行在 Colab 笔记本中。详细步骤本报告不再说明，只描述再复现过程中遇到的一些问题和解决方法。

（1）inference 文件夹的建立：在 git clone Yolov5s 模型后，由于其没有自带 inference 文件夹，无法使用 detect 函数进行测试，所以我们需要在 yolov5 目录下，新建一个名为“inference”的目录，并在其下再新建一个名为“images”的文件夹，里面可以存放用来测试的图片（格式一般选用“jpg”或者“jpeg”）。

（2）数据集的存放：我们需要在 content 目录下，新建一个和 yolov5 同级的文件夹，取名为“datasets”，并将我们想要测试的数据集上传到此目录下（解压缩也要解压到此目录下）。

（3）配置文件的编写：我们需要在 data 目录下编写一个新的.yaml 配置文件，对应于我们要训练的数据集。我们实验主要是对算式框的检测，因而只有单一类别是“equation”，标记为 0；剩下的 path、train、val 和 test（可有可无）修改成对应的数据集的路径即可，剩下的均可不要。

执行下面的命令进入模型训练，训练 5 轮，单次 batch 训练 16 张图片，1 轮大概 22 个 batch，每张图片像素大小为限制为 640，指定预训练权重 yolov5s.pt 和对应训练集的配置文件进行训练：

```
!python train.py --img 640 --batch 16 --epochs 5 --data ./data/Equation.yaml --cfg ./models/yolov5s.yaml --weights ./yolov5s.pt
```

实验结果展示：用增强前后的数据分别用于此模型，每次数据训练在 CPU 上耗时大致在一个小时左右，在 GPU 上耗时大致在六分钟左右。

对于增强前的数据，我们可以得到 5 个 epoches 后的结果如下：

```
5 epochs completed in 0.106 hours.
Optimizer stripped from yolov5/experiment1/weights/last.pt, 14.4MB
Optimizer stripped from yolov5/experiment1/weights/best.pt, 14.4MB

Validating yolov5/experiment1/weights/best.pt...
Fusing layers...
YOLOv5s summary: 157 layers, 7012822 parameters, 0 gradients, 15.8 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 5/5 [00:24<00:00,  4.93s/it]
                   all        150       4983      0.621      0.835      0.711      0.274
Results saved to yolov5/experiment1                                                          HZJ
wandb: 4.918 MB of 21.652 MB uploaded
wandb: 8.710 MB of 21.652 MB uploaded
```

图 8. 数据增强前运行结果图

由上述结果可知，在 150 张测试图像中，识别到 4983 个目标算式框，模型总的准确率为 62.1%，模型召回率为 83.5%，在 IoU 阈值为 0.5 时的平均精度为 0.711，在 IoU 从 0.5 到 0.95 的平均精度为 0.274，说明此模型在此测试集上能有效识别到目标，但匹配的准确率不高，且在阈值条件较高的情况下表现不佳，模型在检测精度上还有待提升。

对于增强后的数据，我们可以得到 5 个 epoches 后的结果如下：

```
5 epochs completed in 1.080 hours.
Optimizer stripped from yolov5/experiment22/weights/last.pt, 14.4MB
Optimizer stripped from yolov5/experiment22/weights/best.pt, 14.4MB

Validating yolov5/experiment22/weights/best.pt...
Fusing layers...
YOLOv5s summary: 157 layers, 7012822 parameters, 0 gradients, 15.8 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 5/5 [01:29<00:00, 17.85s/it]
                   all        150       5207      0.713       0.79      0.777      0.261
Results saved to yolov5/experiment22
wandb: 21.502 MB of 21.502 MB uploaded                                                       HZJ
wandb: 21.502 MB of 21.502 MB uploaded
```

图 9. 数据增强后运行结果图

由上述结果可知，在 150 张测试图像中，识别到 5207 个目标算式框，模型总的准确率为 71.3%，模型召回率为 79%，在 IoU 阈值为 0.5 时的平均精度为 0.777，在 IoU 从 0.5 到 0.95 的平均精度为 0.261。

数据增强前后对比可知，模型的精度有所提升，召回率虽然下降，但仍然保持在较高的水平，阈值条件较高时模型表现仍然较差。整体而言，数据增强后的模型相比之前在**精度和整体识别性能有所改善，泛化能力有所提升，整体性能往好的方向发展**，进一步说明了数据增强的作用所在。

4. W&B 工具的使用

Yolov5s 模型中的 train 板块已经使用了 W&B 工具，只需要我们绑定对应的账号后（即运行时输入账号的密钥），登录网页 https://wandb.ai/进入自己的账号“Project”处，选择进入对应的项目和运行名称即可看到运行过程的可视化结果。

具体实现需要我们在前面进行训练时命令加上“--project”指定项目名称，“--name”指定该项目下的运行名称，比如本次实验设置项目名称为 yolov5s，运行名称为 experiment1、experiment2 等等，如下所示：

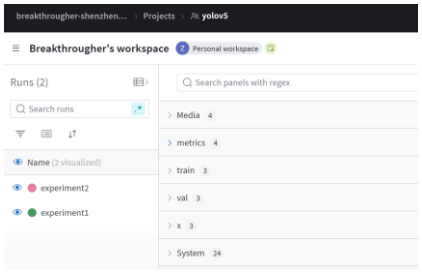

图 10. W&B 网页界面图

使用了 W&B 工具之后，在执行命令、运行结束后会在输出运行的过程信息，包括召回率、精度等的变化，以及过程最好的 epoches 等信息（具体可见提交的.ipynb 文件）；数据增强前后两次实验的输出结果信息如下：

```
wandb: Run history:
wandb:      metrics/mAP_0.5 ▁▂▃▅▆▇▇▇
wandb: metrics/mAP_0.5:0.95 ▁▂▃▄▅▆▇▇
wandb:    metrics/precision ▁▂▃▅▆▇▇▇
wandb:       metrics/recall ▁▂▃▆▇▇▇▇
wandb:       train/box_loss ▇▆▅▄▃▂▂▁
wandb:       train/cls_loss ▁▁▁▁▁▁▁▁
wandb:       train/obj_loss ▇▅▃▂▂▂▂▁
wandb:         val/box_loss ▇▆▅▅▄▃▂▁
wandb:         val/cls_loss ▁▁▁▁▁▁▁▁
wandb:         val/obj_loss ▇▅▃▂▂▂▂▁
wandb:                x/lr0 ▇▆▅▄▃▂▂▁
wandb:                x/lr1 ▁▅▆▇▇▇▇▇
wandb:                x/lr2 ▁▅▆▇▇▇▇▇
wandb:
```

图 11. 数据增强前

```
wandb: Run history:
wandb:      metrics/mAP_0.5 ▁▂▃▅▆▇▇▇
wandb: metrics/mAP_0.5:0.95 ▁▂▃▄▅▆▇▇
wandb:    metrics/precision ▁▂▃▅▆▇▇▇
wandb:       metrics/recall ▁▂▃▆▇▇▇▇
wandb:       train/box_loss ▇▆▅▄▃▂▂▁
wandb:       train/cls_loss ▁▁▁▁▁▁▁▁
wandb:       train/obj_loss ▇▅▃▂▂▂▂▁
wandb:         val/box_loss ▇▆▅▅▄▃▂▁
wandb:         val/cls_loss ▁▁▁▁▁▁▁▁
wandb:         val/obj_loss ▇▅▃▂▂▂▂▁
wandb:                x/lr0 ▇▆▅▄▃▂▂▁
wandb:                x/lr1 ▁▅▆▇▇▇▇▇
wandb:                x/lr2 ▁▅▆▇▇▇▇▇
wandb:
```

图 12.数据增强后

除了直接看输出结果以外，我们可以使用 W&B 工具平台到对应的项目运行里面看，该平台上面已经将运行过程数据进行了可视化，结果如下所示：

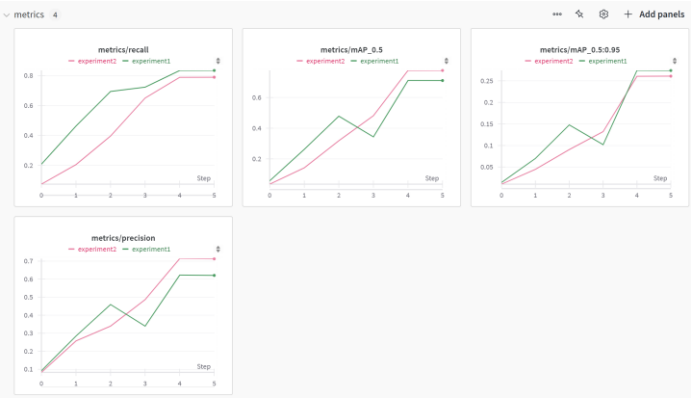

图 13. 召回率、精度、mAP0.5 和 mAP0.5:0.95 的变化图

图 14. 训练过程中 cls_loss、box_loss 和 obj_loss 的变化图

图 15. 验证过程中 cls_loss、box_loss 和 obj_loss 的变化图

由图 13 不难发现数据增强后的召回率、精度等各项指标都变化的比较平滑，说明其泛化性能得到了提升，精度和 mAP0.5 的提升侧面佐证了这一点。图 14 显示了训练过程中数据增强后的 loss 会变少，但图 15 显示了验证过程中数据增强后的 loss 又变大了，进一步说明了数据增强带来的一些副作用，增强后的数据不好拟合验证数据集，存在可能使得结果的精度下降的风险。

实验结论：

本次实验对 Yolov5s 模型进行部署和优化，通过划分所给数据集、进行数据增强，用数据增强前后的数据集成功进行了模型训练，也利用 W&B 工具成功可视化了结果，并对实验结果进行了分析，说明了数据增强的作用。本次实验圆满结束。

心得体会：

本次实验相比之前的简单模型部署新增了使用自己的数据集进行试验，通过本实验，学会了如何写配置文件，熟练了在.ipynb 笔记本中运行命令、如何去做简单的数据增强和使用 W&B 工具去观察自己的模型训练结果。通过本次实验，个人对模型部署和优化实践有了更加深刻的理解，不再停留于知识层面，提升了个人的动手实践能力，收获满满。

指导教师批阅意见：

成绩评定：

指导教师签字：王旭

2024 年 11 月 07 日

备注：