山东大学

崇新学堂

2023－2024 学年第一学期

实 验 报 告

课程名称：信息基础 II

专 业 班 级：崇新学堂 21 级

学 生 姓 名：刘浩

个 人 学 号：202120120312

实验五：Yolo

一、 实验要求

要求自己编程实现 YOLO（v3-v7 任一种）检测算法，本实验采取 YOLOv5 结合 CCPD 数据集实现车牌检测。

二、 实验原理

由于之前并未接触过 YOLO 这类模型，所以对我来说是比较陌生的，我系统学习了[1]课程了解了 YOLO 的原理后才开始进行的实验。

本次实验选择的是 YOLOv5s,其主要特点是模型小,速度快,其中 YOLOv5s 的网络结构如图 1 所示:

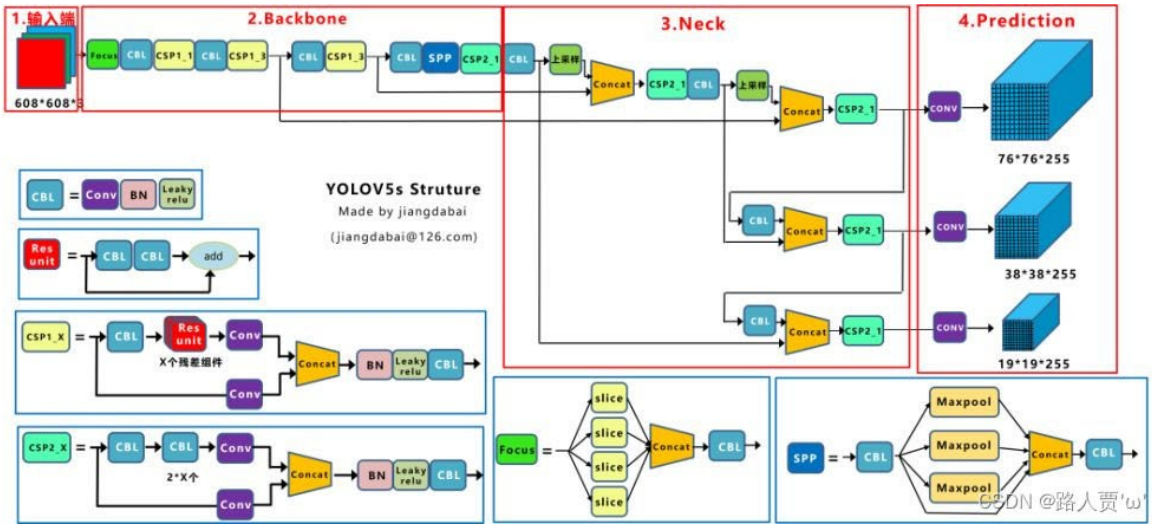


图 1 YOLOv5 网络结构

物体检测中的评价指标分析**(Precision 是精度，Recall 是召回率)**：

$$Precision = \frac{TP}{TP + FP}$$
$$\text{Re}call = \frac{TP}{TP + FN} \tag{1}$$

除此以外，评价指标还有**置信度**，最终输出的时候采取极大值抑制策略，对预选框的置信度进行排序，最终只选取较大的预选框。

下面说一下我对 **YOLO** 的核心思想的理解：

从 **YOLOv1** 开始说起，它首先对划分后的每个格子，要产生两种框，记为 B1B2，最终模型输出得到的是$7\times7\times30$的格子

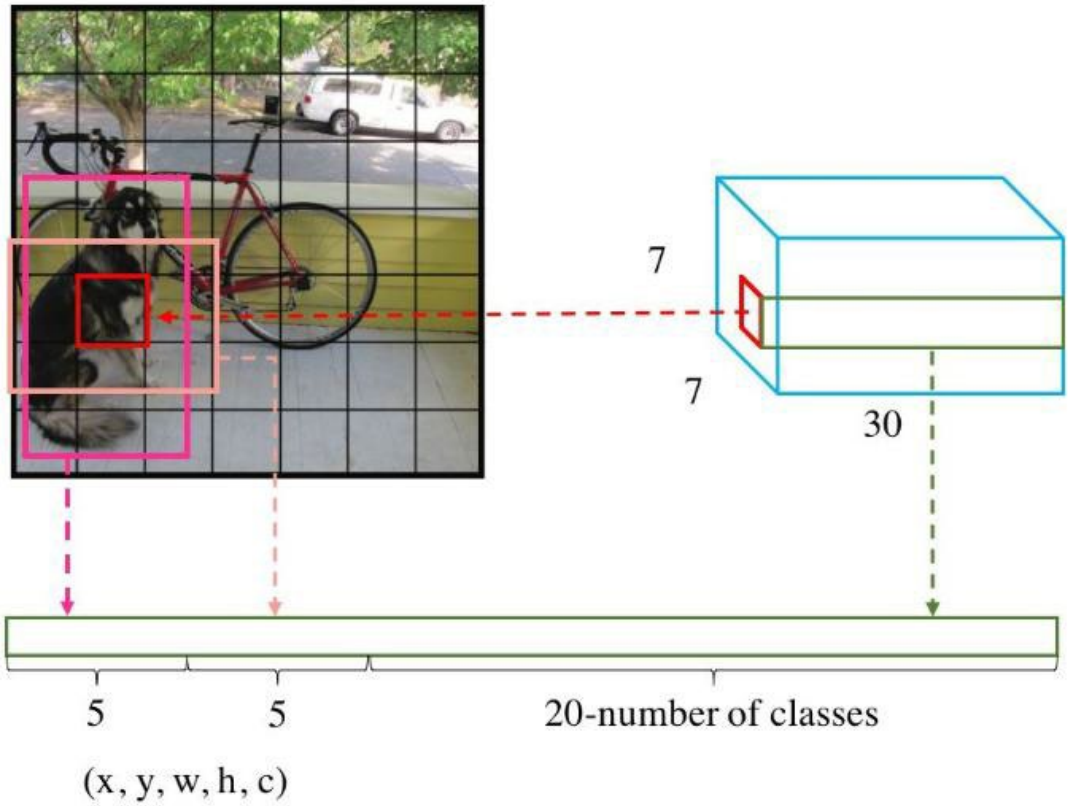


图 2 YOLOv1 检测过程示意图

其实可以看作7×7个 30 个数，也就是说每个1×30 中的前 5 个是生成的第一种框 B1 的位置、长宽以及置信度，5-10 个是生成的第二种框 B2 的位置、长宽以及置信度，剩下的 20 个表示类别，即：

$$(X,Y,H,W,C) * B \tag{2}$$

损失函数：

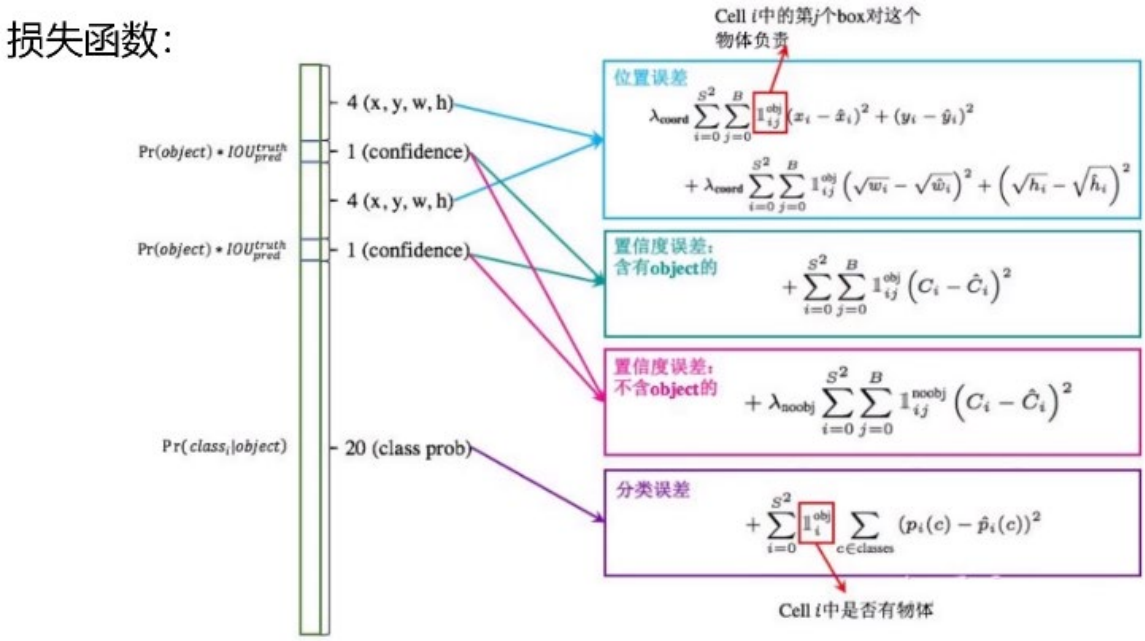


图 3 损失函数

后续其实都是基于 **YOLOv1** 进行，如 **YOLOv2** 采取公式(3)聚类的方式

$$d(box, centroids) = 1 - IOU(box, centroids) \quad (3)$$

下面介绍 YOLOv5 的基本结构：

输入的图片会先经过 Backbone 再经过 Head，最终输出检测框。

输入端采用和 Yolov4 一样的 Mosaic 数据增强方式

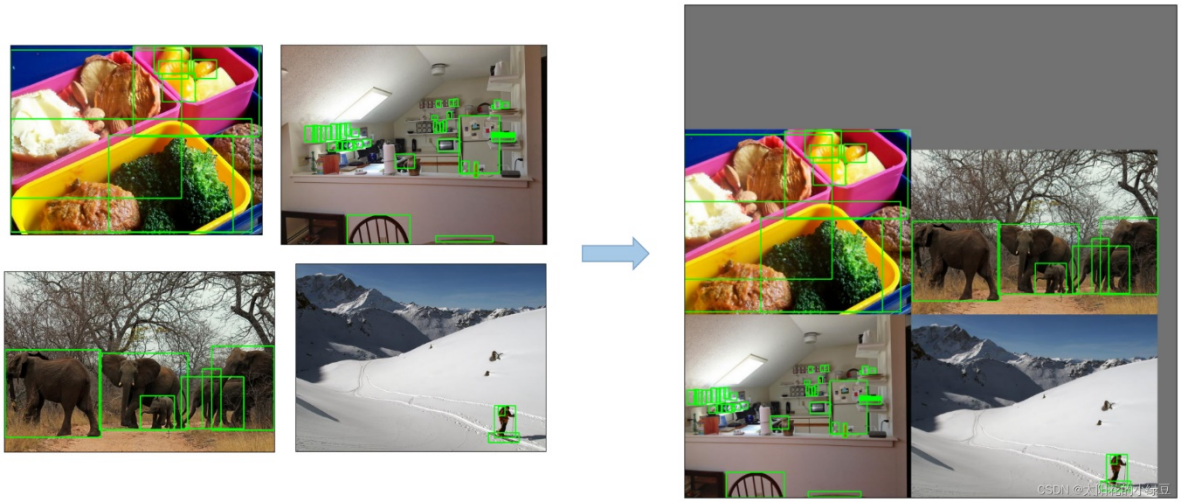


图 4 Mosaic 数据增强方式

自适应锚框计算，在每次训练开始之前，YOLOv5 都会根据不同的数据集来自适应计算不同训练集中的最佳锚框值。

Backbone(骨干网络)主要由 Focus 结构以及 CSP 结构组成，Focus 结构是一种用于特征提取的卷积神经网络层，用于将输入特征图中的信息进行压缩和组合，从而提取出更高层次的特征表示。具体来说，Focus 结构可以将输入特征图划分成四个子图，并将这四个子图进行通道拼接，从而得到一个更小的特征图。

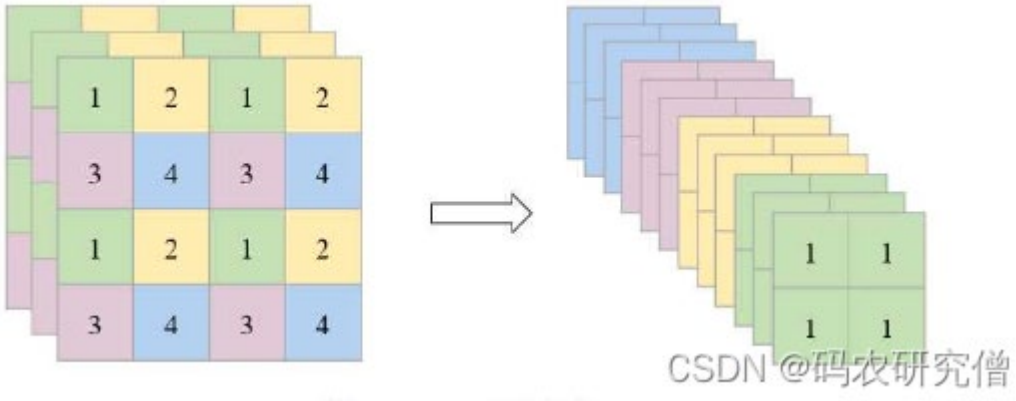


图 5 Focus 结构示意图

CSP 结构，其核心思想是将输入特征图分成两部分：

一部分经过一个小的卷积网络（称为子网络）进行压缩，然后进行一系列卷积操作，最后再使用一个卷积层进行扩张。这样可以提取出相对较少的高层次特征。

另一部分则直接进行下一层的处理。

然后将经过子网络处理的特征图与直接处理的特征图进行拼接，然后再进行一系列卷积操作。这样可以将低层次的细节特征和高层次的抽象特征结合起来，提高特征提取的效率。

同时 **YOLOv5** 中采用了两种不同的 **Neck** 层(在 Backbone 和输出层之间)：SPP 和 PAN，与 YOLOv4 采用的普通卷积操作不同的是，YOLOv5 借鉴 CSPnet 设计的 CSP2 结构，从而加强网络特征融合能力。

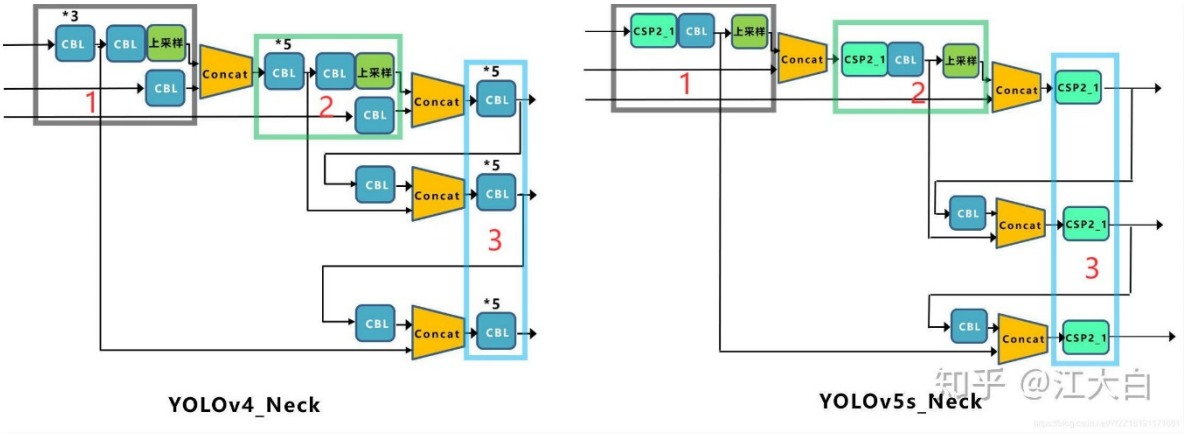
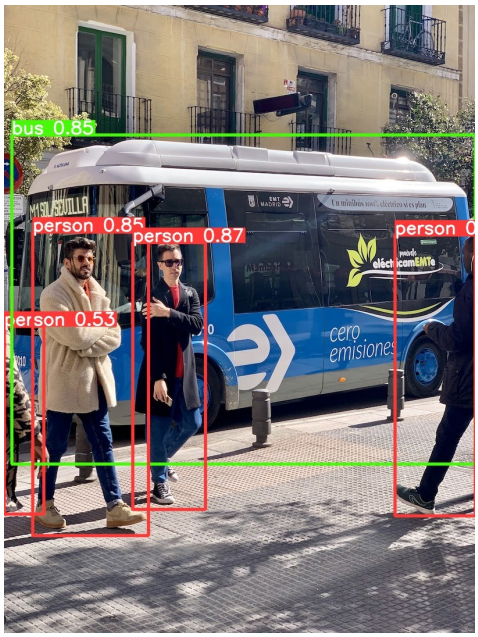


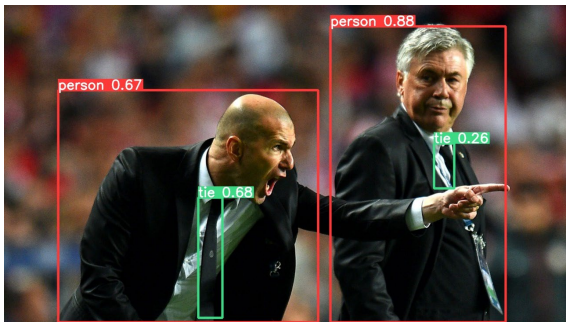
图 6 Neck 层对比

三、 实验步骤

本实验采用的代码以 github 上 [YOLOv5 的官方代码](#)为基础，在读懂的基础上进行修改，同时构建了自己的数据集进行训练和实际测试。

完成环境配置后运行 YOLOv5 的 demo，即 **detect.py**，直接用训练好的模型,得到的结果如下：





由于代码过于复杂，我在这里仅仅介绍一下代码中的关键部分：

detect 部分：

下面的 parse_opt 函数主要是在运行的时候指定参数，我主要讲解一下比较关键的几个参数：

参数	含义
--weights	训练的权重路径
--source	测试数据
--data	配置文件的路径
-- imgsz	预测时输入网络的图大小
--conf-thres	置信度阈值
--iou-thres	IOU 阈值
--max-det	最大检测框的个数

```
def parse_opt():
    parser = argparse.ArgumentParser()
    parser.add_argument('--
weights', nargs='+', type=str, default=ROOT / 'yolov5s.pt', help='model path or triton UR
L')
    parser.add_argument('--
source', type=str, default=ROOT / 'data/images', help='file/dir/URL/glob/screen/0(webcam)
')
    parser.add_argument('--
data', type=str, default=ROOT / 'data/coco128.yaml', help='(optional) dataset.yaml path')

    parser.add_argument('--imgsz', '--img', '--img-
size', nargs='+', type=int, default=[640], help='inference size h,w')
    parser.add_argument('--conf-
thres', type=float, default=0.25, help='confidence threshold')
    parser.add_argument('--iou-
thres', type=float, default=0.45, help='NMS IoU threshold')
    parser.add_argument('--max-
det', type=int, default=1000, help='maximum detections per image')
    parser.add_argument('--
device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--view-img', action='store_true', help='show results')
    parser.add_argument('--save-
txt', action='store_true', help='save results to *.txt')
    parser.add_argument('--save-
csv', action='store_true', help='save results in CSV format')
    parser.add_argument('--save-conf', action='store_true', help='save confidences in --
save-txt labels')
    parser.add_argument('--save-
crop', action='store_true', help='save cropped prediction boxes')
    parser.add_argument('--
nosave', action='store_true', help='do not save images/videos')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --
classes 0, or --classes 0 2 3')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-
agnostic NMS')
```

```
    parser.add_argument('--augment', action='store_true', help='augmented inference')
    parser.add_argument('--visualize', action='store_true', help='visualize features')
    parser.add_argument('--update', action='store_true', help='update all models')
    parser.add_argument('--
project', default=ROOT / 'runs/detect', help='save results to project/name')
    parser.add_argument('--name', default='exp', help='save results to project/name')
    parser.add_argument('--exist-
ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--line-
thickness', default=3, type=int, help='bounding box thickness (pixels)')
    parser.add_argument('--hide-
labels', default=False, action='store_true', help='hide labels')
    parser.add_argument('--hide-
conf', default=False, action='store_true', help='hide confidences')
    parser.add_argument('--half', action='store_true', help='use FP16 half-
precision inference')
    parser.add_argument('--
dnn', action='store_true', help='use OpenCV DNN for ONNX inference')
    parser.add_argument('--vid-stride', type=int, default=1, help='video frame-
rate stride')
    opt = parser.parse_args()
    opt.imgsz *= 2 if len(opt.imgsz) == 1 else 1  # expand
    print_args(vars(opt))
    return opt
```

首先会根据 source 的类型对输入进行判断，通过一些布尔值区分是图片还是视频还是其他的东西：

```
source = str(source)
save_img = not nosave and not source.endswith('.txt')  # save inference images
is_file = Path(source).suffix[1:] in (IMG_FORMATS + VID_FORMATS)
is_url = source.lower().startswith(('rtsp://', 'rtmp://', 'http://', 'https://'))
webcam = source.isnumeric() or source.endswith('.streams') or (is_url and not is_file)
screenshot = source.lower().startswith('screen')
if is_url and is_file:
    source = check_file(source)  # download
```

然后将模型加载进来：

```
# Load model
device = select_device(device)
model = DetectMultiBackend(weights, device=device, dnn=dnn, data=data, fp16=half)
stride, names, pt = model.stride, model.names, model.pt
imgsz = check_img_size(imgsz, s=stride)  # check image size
```

参数解析：

参数	含义
stride	推理时所用到的步长，默认为 32， 大步长适合于大目标，小步长适合于小目标
names	保存推理结果名的列表，比如默认模型的值是['person', 'bicycle', 'car', ...]
pt	加载的是否是 pytorch 模型

数据加载部分：

```
# Dataloader
 bs = 1  # batch_size
 if webcam:
     view_img = check_imshow(warn=True)
     dataset = LoadStreams(source, img_size=imgsz, stride=stride, auto=pt, vid_stride=vid
_stride)
     bs = len(dataset)
 elif screenshot:
     dataset = LoadScreenshots(source, img_size=imgsz, stride=stride, auto=pt)
 else:
```

```
    dataset = LoadImages(source, img_size=imgsz, stride=stride, auto=pt, vid_stride=vid_
stride)
 vid_path, vid_writer = [None] * bs, [None] * bs
```

然后进行推理：

先对模型进行热身(传入一张空图片)后开始正式的预测，将输入图片转化为 torch 可以使用的张量的后，对像素进行归一化操作，然后对图片进行预测并进行极大值抑制，包括：可信度限制(默认 0.25)，IOU 限制(默认 0.45)，框的数量限制(默认 1000)

```
model.warmup(imgsz=(1 if pt or model.triton else bs, 3, *imgsz))  # warmup
 seen, windows, dt = 0, [], (Profile(), Profile(), Profile())
 for path, im, im0s, vid_cap, s in dataset:
    with dt[0]:
        im = torch.from_numpy(im).to(model.device)
        im = im.half() if model.fp16 else im.float()  # uint8 to fp16/32
        im /= 255  # 0 - 255 to 0.0 - 1.0
        if len(im.shape) == 3:
            im = im[None]  # expand for batch dim

    # Inference
    with dt[1]:
        visualize = increment_path(save_dir / Path(path).stem, mkdir=True) if visualize
else False
        pred = model(im, augment=augment, visualize=visualize)

    # NMS
    with dt[2]:
        pred = non_max_suppression(pred, conf_thres, iou_thres, classes, agnostic_nms, m
ax_det=max_det)
```

由于 **YOLOv5** 对处理的图片大小有要求，所以代码对输入图片进行了 resize 操作，因此标注的时候，需要将坐标映射回原图，代码中通过 **Annotator** 类完成。

YOLO 的网络模型在哪呢？

在 **model** 文件夹下由于 **yolo.py** 文件以及 **common.py**，这里面完成了 YOLO 网络主要框架的搭建，如果对模型做修改的话也应该在该文件下进行，下面介绍一下关键代码：

common 下的卷积模块：

```
class Conv(nn.Module):
    # Standard convolution
    def __init__(self, c1, c2, k=1, s=1, p=None, g=1, act=True):  # ch_in, ch_out, kernel
, stride, padding, groups
        """
        @Pargm c1: 输入通道数
        @Pargm c2: 输出通道数
        @Pargm k : 卷积核大小(kernel_size)
        @Pargm s : 卷积步长 (stride)
        @Pargm p : 特征图填充宽度 (padding)
        @Pargm g : 控制分组，必须整除输入的通道数(保证输入的通道能被正确分组)
        """
        super().__init__()
        # https://oneflow.readthedocs.io/en/master/generated/oneflow.nn.Conv2d.html?highl
ight=Conv
        self.conv = nn.Conv2d(c1, c2, k, s, autopad(k, p), groups=g, bias=False)
        self.bn = nn.BatchNorm2d(c2)
        self.act = nn.SiLU() if act is True else (act if isinstance(act, nn.Module) else
nn.Identity())

    def forward(self, x):
        return self.act(self.bn(self.conv(x)))

    def forward_fuse(self, x):
        return self.act(self.conv(x))
```

backbone 中的 focus 结构：

```
class Focus(nn.Module):
    # Focus wh information into c-space
    def __init__(self, c1, c2, k=1, s=1, p=None, g=1, act=True):  # ch_in, ch_out, kernel
, stride, padding, groups
        super().__init__()
        self.conv = Conv(c1 * 4, c2, k, s, p, g, act=act)
        # self.contract = Contract(gain=2)

    def forward(self, x):  # x(b,c,w,h) -> y(b,4c,w/2,h/2)
        return self.conv(torch.cat((x[..., ::2, ::2], x[..., 1::2, ::2], x[..., ::2, 1::2
], x[..., 1::2, 1::2]), 1))
        # return self.conv(self.contract(x))
```

CSP 结构：

```
class BottleneckCSP(nn.Module):
    # CSP Bottleneck https://github.com/WongKinYiu/CrossStagePartialNetworks
    def __init__(self, c1, c2, n=1, shortcut=True, g=1, e=0.5):  # ch_in, ch_out, number,
 shortcut, groups, expansion
        super().__init__()
        c_ = int(c2 * e)  # hidden channels
        self.cv1 = Conv(c1, c_, 1, 1)
        self.cv2 = nn.Conv2d(c1, c_, 1, 1, bias=False)
        self.cv3 = nn.Conv2d(c_, c_, 1, 1, bias=False)
        self.cv4 = Conv(2 * c_, c2, 1, 1)
        self.bn = nn.BatchNorm2d(2 * c_)  # applied to cat(cv2, cv3)
        self.act = nn.SiLU()
        self.m = nn.Sequential(*(Bottleneck(c_, c_, shortcut, g, e=1.0) for _ in range(n)
))

    def forward(self, x):
        y1 = self.cv3(self.m(self.cv1(x)))
        y2 = self.cv2(x)
        return self.cv4(self.act(self.bn(torch.cat((y1, y2), 1))))

class BottleneckCSP(nn.Module):
    # CSP Bottleneck https://github.com/WongKinYiu/CrossStagePartialNetworks
    def __init__(self, c1, c2, n=1, shortcut=True, g=1, e=0.5):  # ch_in, ch_out, number,
 shortcut, groups, expansion
        super().__init__()
        c_ = int(c2 * e)  # hidden channels
        self.cv1 = Conv(c1, c_, 1, 1)
        self.cv2 = nn.Conv2d(c1, c_, 1, 1, bias=False)
        self.cv3 = nn.Conv2d(c_, c_, 1, 1, bias=False)
        self.cv4 = Conv(2 * c_, c2, 1, 1)
        self.bn = nn.BatchNorm2d(2 * c_)  # applied to cat(cv2, cv3)
        self.act = nn.SiLU()
        self.m = nn.Sequential(*(Bottleneck(c_, c_, shortcut, g, e=1.0) for _ in range(n)
))

    def forward(self, x):
        y1 = self.cv3(self.m(self.cv1(x)))
        y2 = self.cv2(x)
        return self.cv4(self.act(self.bn(torch.cat((y1, y2), 1))))

class C3(nn.Module):
    # CSP Bottleneck with 3 convolutions
    def __init__(self, c1, c2, n=1, shortcut=True, g=1, e=0.5):  # ch_in, ch_out, number,
 shortcut, groups, expansion
        super().__init__()
        c_ = int(c2 * e)  # hidden channels
        self.cv1 = Conv(c1, c_, 1, 1)
        self.cv2 = Conv(c1, c_, 1, 1)
        self.cv3 = Conv(2 * c_, c2, 1)  # optional act=FReLU(c2)
        self.m = nn.Sequential(*(Bottleneck(c_, c_, shortcut, g, e=1.0) for _ in range(n)
))
```

```
    def forward(self, x):
        return self.cv3(torch.cat((self.m(self.cv1(x)), self.cv2(x)), 1))
```

Neck 层结构：

```
class Bottleneck(nn.Module):
    # Standard bottleneck
    def __init__(self, c1, c2, shortcut=True, g=1, e=0.5):  # ch_in, ch_out, shortcut, gr
oups, expansion
        super().__init__()
        c_ = int(c2 * e)  # hidden channels
        self.cv1 = Conv(c1, c_, 1, 1)
        self.cv2 = Conv(c_, c2, 3, 1, g=g)
        self.add = shortcut and c1 == c2

    def forward(self, x):
        return x + self.cv2(self.cv1(x)) if self.add else self.cv2(self.cv1(x))
```


```
class Bottleneck(nn.Module):
    # Standard bottleneck
    def __init__(self, c1, c2, shortcut=True, g=1, e=0.5):  # ch_in, ch_out, shortcut, gr
oups, expansion
        super().__init__()
        c_ = int(c2 * e)  # hidden channels
        self.cv1 = Conv(c1, c_, 1, 1)
        self.cv2 = Conv(c_, c2, 3, 1, g=g)
        self.add = shortcut and c1 == c2

    def forward(self, x):
        return x + self.cv2(self.cv1(x)) if self.add else self.cv2(self.cv1(x))
```


至此，YOLOv5 的网络结构关键部分介绍完毕。

Train 部分：

该部分我主要介绍一下如何用 YOLOv5 训练**其他数据集**

例如采用 CCPD(中国城市车牌开源数据集)进行车牌识别训练。

数据集下载链接(大小:12G)：https://pan.baidu.com/s/1i5AOjAbtkwb17Zy-NQGqkw

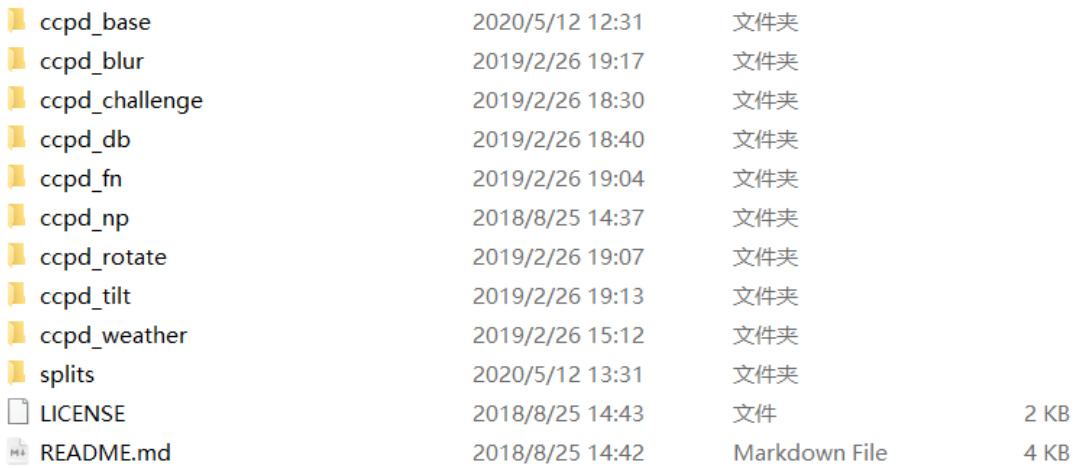


图 7 数据集结构

鉴于数据集比较大，我采用了老师的服务器进行训练，将数据集上传到服务器中，其中 YOLO 对于数据的结构要求如下：

```
data_set
└─ score
├─ images
│  ├─ test # 放测试集图片
│  ├─ train # 放训练集图片
│  └─ val # 放验证集图片
```

```
└─ labels
├─ test # 放测试集标签
├─ train # 放训练集标签
├─ val # 放验证集标签
```

且 CCPD 需要进行格式转化才可被 YOLO 识别：

首先按照 6:2:2 的比例划分训练集验证集和测试集：

```
import shutil
import os
import random
from shutil import copy2
trainfiles = os.listdir("/root/yolov5-CCPD/data/CCPD2019/ccpd_fn/")
num_train = len(trainfiles)
print( "num_train: " + str(num_train) )
index_list = list(range(num_train))
print(index_list)
random.shuffle(index_list)
num = 0

trainDir = "/root/yolov5-CCPD/data/ccpd_train/images/train/"
validDir = "/root/yolov5-CCPD/data/ccpd_train/images/val/"
detectDir = "/root/yolov5-CCPD/data/test_dataset/images/"

for i in index_list:
    fileName = os.path.join("/root/yolov5-CCPD/data/CCPD2019/ccpd_fn/", trainfiles[i])
    if num < num_train*0.6:
        print(str(fileName))
        copy2(fileName, trainDir)
    elif num < num_train*0.8:
        print(str(fileName))
        copy2(fileName, detectDir)
    else:
        copy2(fileName, validDir)
    num += 1
```

再将其转化为 YOLO 可以识别的格式：

```
import shutil
import cv2
import os

def txt_file(img_path):
    x = img_path.split("/", 9)
    if x[6] == "train":
        y = '/'.join(x[0:5]) + '/labels/' + 'train/'
    else:
        y = '/'.join(x[0:5]) + '/labels/' + 'val/'
    return y



def txt_translate(path, txt_path):
    for filename in os.listdir(path):
        print(filename)

        list1 = filename.split("-", 3)  # 第一次分割，以减号'-'做分割
        subname = list1[2]
        list2 = filename.split(".", 1)
        subname1 = list2[1]
        if subname1 == 'txt':
            continue
        lt, rb = subname.split("_", 1)  # 第二次分割，以下划线'_'做分割
        lx, ly = lt.split("&", 1)
        rx, ry = rb.split("&", 1)
        width = int(rx) - int(lx)
        height = int(ry) - int(ly)  # bounding box 的宽和高
        cx = float(lx) + width / 2
```

```
        cy = float(ly) + height / 2  # bounding box 中心点

        img = cv2.imread(path + filename)
        if img is None:  # 自动删除失效图片（下载过程有的图片会存在无法读取的情况）
            os.remove(os.path.join(path, filename))
            continue
        width = width / img.shape[1]
        height = height / img.shape[0]
        cx = cx / img.shape[1]
        cy = cy / img.shape[0]

        txtname = filename.split(".", 1)
        txtfile = txt_path + txtname[0] + ".txt"
        with open(txtfile, "w") as f:
            f.write(str(0) + " " + str(cx) + " " + str(cy) + " " + str(width) + " " + str
(height))


if __name__ == '__main__':
    trainDir = "/root/yolov5-CCPD/data/ccpd_train/images/train/"
    validDir = "/root/yolov5-CCPD/data/ccpd_train/images/val/"
    txt_path1 = txt_file(trainDir)
    print(txt_path1)
    txt_path2 = txt_file(validDir)
    print(txt_path2)
    txt_translate(trainDir, txt_path1)
    txt_translate(validDir, txt_path2)
```

至此数据集构建完成。

另外需要创建 yaml 文件，只需要按照提供的格式修改一下即可：

```
train: data/ccpd_train/images/train  # train images (relative to 'path') 128 images
val: data/ccpd_train/images/val  # val images (relative to 'path') 128 images
test:  # test images (optional)

# Classes
names:
  0: license
```

最后修改 train 文件中的部分参数（权重参数采用 yolov5s）

```
parser.add_argument('--
weights', type=str, default=ROOT / 'yolov5s.pt', help='initial weights path')
parser.add_argument('--
cfg', type=str, default='models/yolov5s.yaml', help='model.yaml path')
parser.add_argument('--
data', type=str, default='data/ccpd.yaml', help='dataset.yaml path')
parser.add_argument('--hyp', type=str, default=ROOT / 'data/hyps/hyp.scratch-
low.yaml', help='hyperparameters path')
parser.add_argument('--epochs', type=int, default=15, help='total training epochs')
parser.add_argument('--batch-
size', type=int, default=8, help='total batch size for all GPUs, -1 for autobatch')
parser.add_argument('--imgsz', '--img', '--img-
size', type=int, default=640, help='train, val image size (pixels)')
parser.add_argument('--rect', action='store_true', help='rectangular training')
parser.add_argument('--
resume', nargs='?', const=True, default=False, help='resume most recent training')
parser.add_argument('--nosave', action='store_true', help='only save final checkpoint')
parser.add_argument('--noval', action='store_true', help='only validate final epoch')
parser.add_argument('--noautoanchor', action='store_true', help='disable AutoAnchor')
parser.add_argument('--noplots', action='store_true', help='save no plot files')
parser.add_argument('--
evolve', type=int, nargs='?', const=300, help='evolve hyperparameters for x generations')

parser.add_argument('--bucket', type=str, default='', help='gsutil bucket')
parser.add_argument('--cache', type=str, nargs='?', const='ram', help='image --
cache ram/disk')
```

```
parser.add_argument('--image-
weights', action='store_true', help='use weighted image selection for training')
parser.add_argument('--
device', default='0', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
parser.add_argument('--multi-scale', action='store_true', help='vary img-
size +/- 50%%')
parser.add_argument('--single-cls', action='store_true', help='train multi-
class data as single-class')
parser.add_argument('--
optimizer', type=str, choices=['SGD', 'Adam', 'AdamW'], default='SGD', help='optimizer')

parser.add_argument('--sync-
bn', action='store_true', help='use SyncBatchNorm, only available in DDP mode')
parser.add_argument('--
workers', type=int, default=2, help='max dataloader workers (per RANK in DDP mode)')
parser.add_argument('--
project', default=ROOT / 'runs/train', help='save to project/name')
parser.add_argument('--name', default='exp', help='save to project/name')
parser.add_argument('--exist-
ok', action='store_true', help='existing project/name ok, do not increment')
parser.add_argument('--quad', action='store_true', help='quad dataloader')
parser.add_argument('--cos-lr', action='store_true', help='cosine LR scheduler')
parser.add_argument('--label-
smoothing', type=float, default=0.0, help='Label smoothing epsilon')
parser.add_argument('--
patience', type=int, default=100, help='EarlyStopping patience (epochs without improvemen
t)')
parser.add_argument('--
freeze', nargs='+', type=int, default=[0], help='Freeze layers: backbone=10, first3=0 1 2
')
parser.add_argument('--save-period', type=int, default=-
1, help='Save checkpoint every x epochs (disabled if < 1)')
parser.add_argument('--seed', type=int, default=0, help='Global training seed')
parser.add_argument('--local_rank', type=int, default=-1, help='Automatic DDP Multi-
GPU argument, do not modify')
```

至此，可以开始训练了！

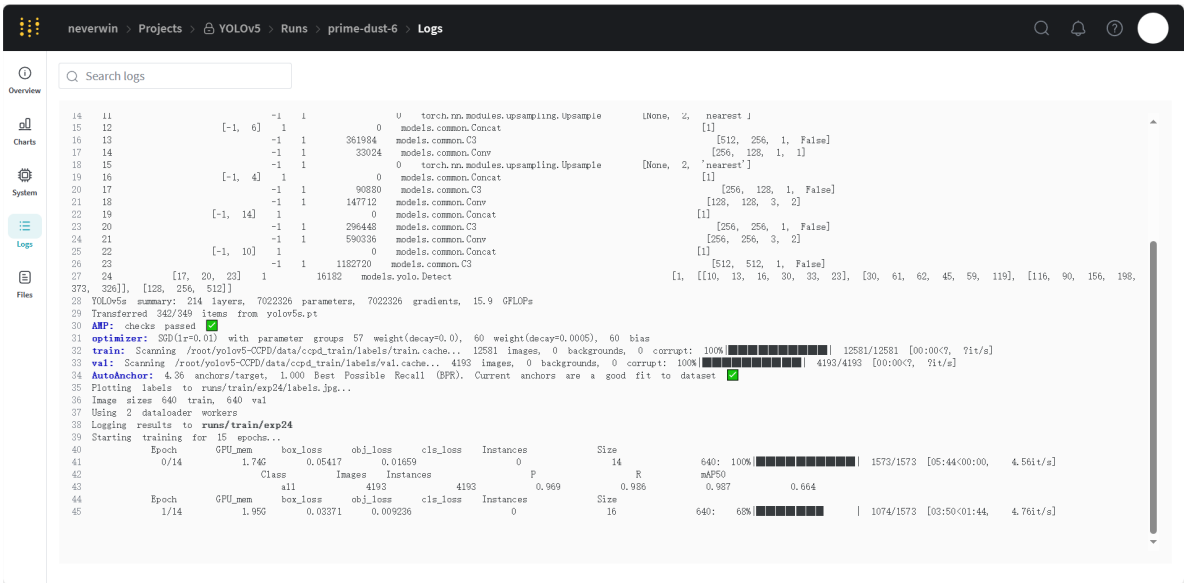


图 8 训练过程示意图

训练过程采用 **wandb** 进行可视化

pip install wandb 进行安装，然后创建好自己的账户，再次训练即可查看训练的可视化结果

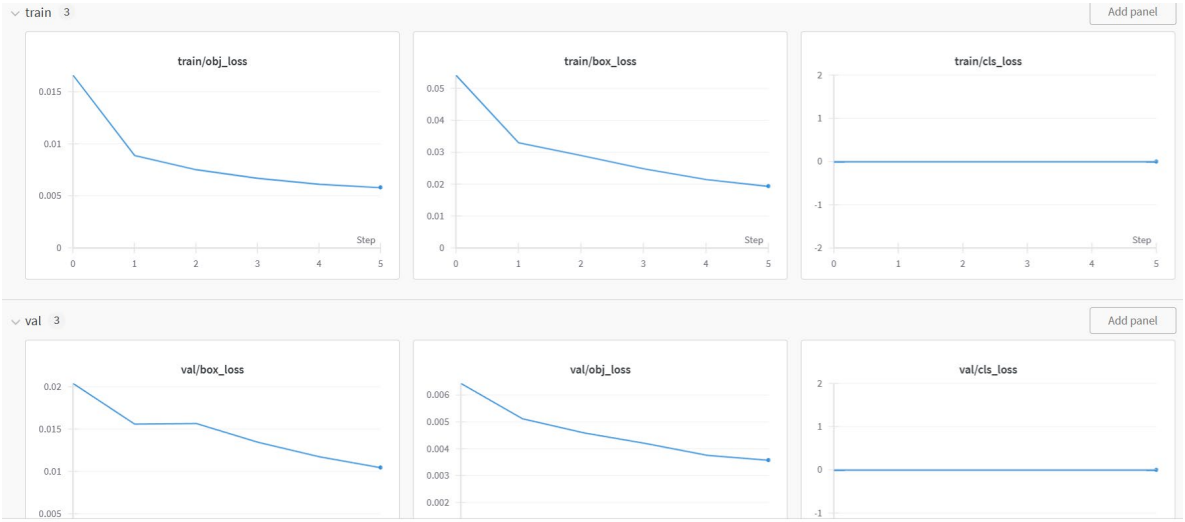


图 9 训练过程可视化示意图

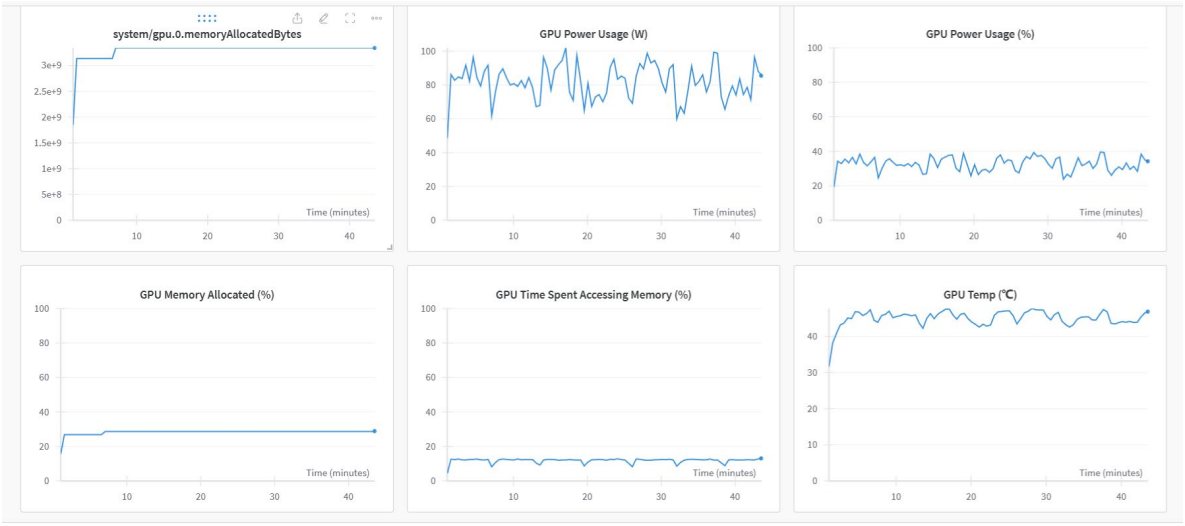


图 10 可视化示意图续

注：训练 epoch＝15 最终的结果放在**实验结果**部分，此处图 11，图 12 仅作示意图

训练完成后权重文件保存在/root/yolov5-CCPD/runs/train/exp24/weights 中，其中 best.pt 是**最佳**模型参数,last.pt 是**最新**的模型参数

best.pt	2023/11/28 19:26	PT 文件	14,027 KB
last.pt	2023/11/28 19:26	PT 文件	14,027 KB

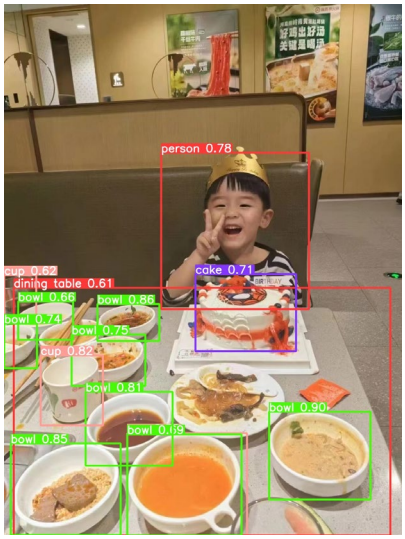
训练完成后该**如何测试**？修改 detect 文件的部分参数：

```
parser.add_argument('--
weights', nargs='+', type=str, default=ROOT /'runs/train/exp24/weights/best.pt', help='mo
del path or triton URL')
parser.add_argument('--
source', type=str, default=ROOT / 'data/images', help='file/dir/URL/glob/screen/0(webcam)
')
parser.add_argument('--
data', type=str, default='data/ccpd.yaml', help='(optional) dataset.yaml path')
parser.add_argument('--imgsz', '--img', '--img-
size', nargs='+', type=int, default=[640], help='inference size h,w')
parser.add_argument('--conf-
thres', type=float, default=0.25, help='confidence threshold')
```

```
parser.add_argument('--iou-thres', type=float, default=0.45, help='NMS IoU threshold')
parser.add_argument('--max-
det', type=int, default=1000, help='maximum detections per image')
parser.add_argument('--
device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
parser.add_argument('--view-img', action='store_true', help='show results')
parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
parser.add_argument('--save-
csv', action='store_true', help='save results in CSV format')
parser.add_argument('--save-conf', action='store_true', help='save confidences in --save-
txt labels')
parser.add_argument('--save-
crop', action='store_true', help='save cropped prediction boxes')
parser.add_argument('--nosave', action='store_true', help='do not save images/videos')
parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --
classes 0, or --classes 0 2 3')
parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
parser.add_argument('--augment', action='store_true', help='augmented inference')
parser.add_argument('--visualize', action='store_true', help='visualize features')
parser.add_argument('--update', action='store_true', help='update all models')
parser.add_argument('--
project', default=ROOT / 'runs/detect', help='save results to project/name')
parser.add_argument('--name', default='exp', help='save results to project/name')
parser.add_argument('--exist-
ok', action='store_true', help='existing project/name ok, do not increment')
parser.add_argument('--line-
thickness', default=3, type=int, help='bounding box thickness (pixels)')
parser.add_argument('--hide-
labels', default=False, action='store_true', help='hide labels')
parser.add_argument('--hide-
conf', default=False, action='store_true', help='hide confidences')
parser.add_argument('--half', action='store_true', help='use FP16 half-
precision inference')
parser.add_argument('--
dnn', action='store_true', help='use OpenCV DNN for ONNX inference')
parser.add_argument('--vid-stride', type=int, default=1, help='video frame-
rate stride')
```

四、 实验结果

首先对于本地图片使用 **YOLOv5** 检测结果展示(另外也测试了 video 版等，此处不再展示)：



训练过程记录：

➢ 完整的训练部分放在 train.pdf 中一并提交

训练结果如下：

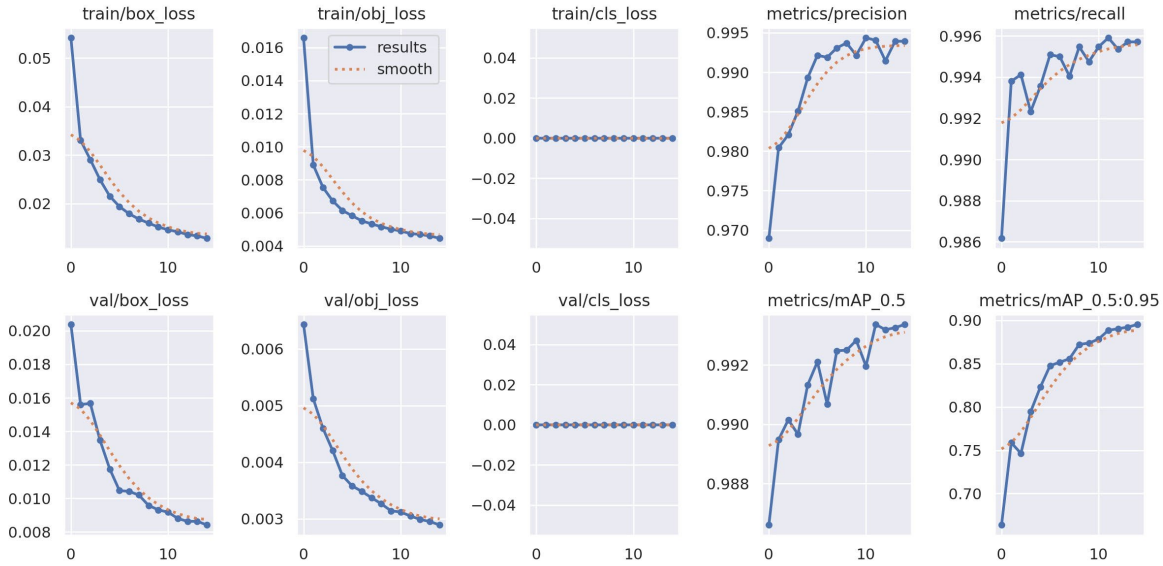


图 11 训练结果

batch_size=8，下面为训练部分和验证部分的 batch 记录：

图 12 train batch0 记录

图 13 train batch1 记录

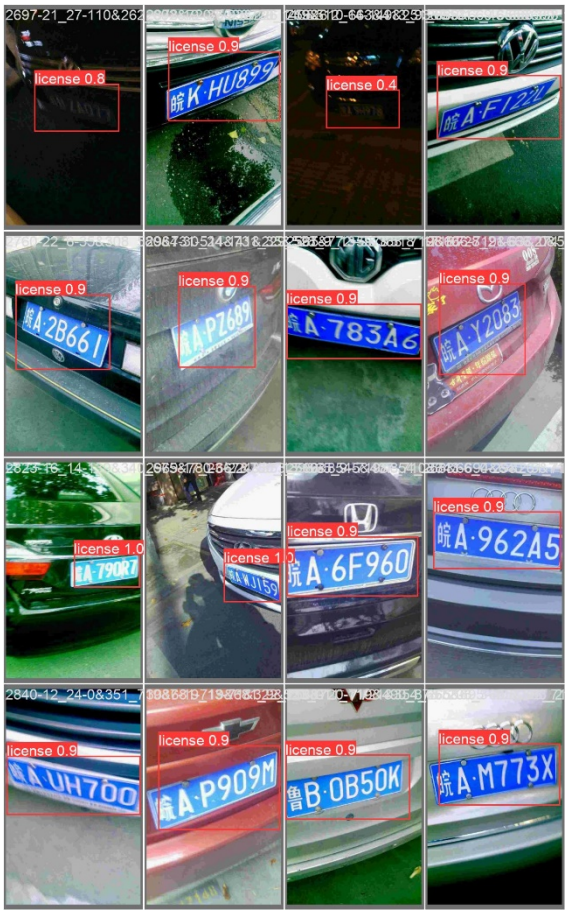
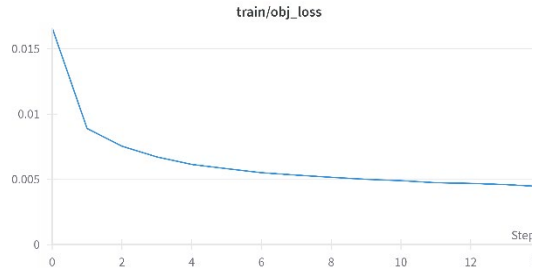
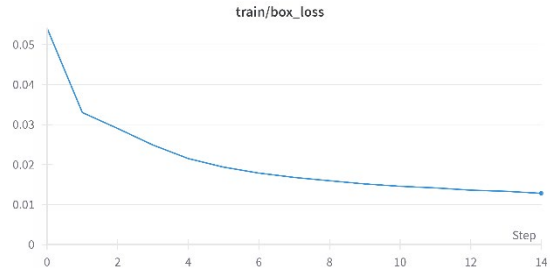


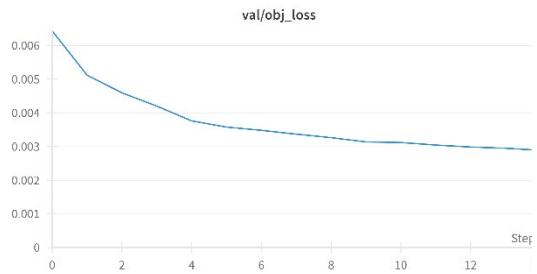
图 14 val batch pred

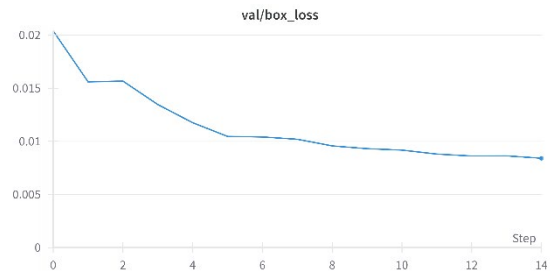
图 15 val batch labels

下面为通过 wandb 可视化的结果：









采取我训练的最好模型进行车牌识别检测结果：

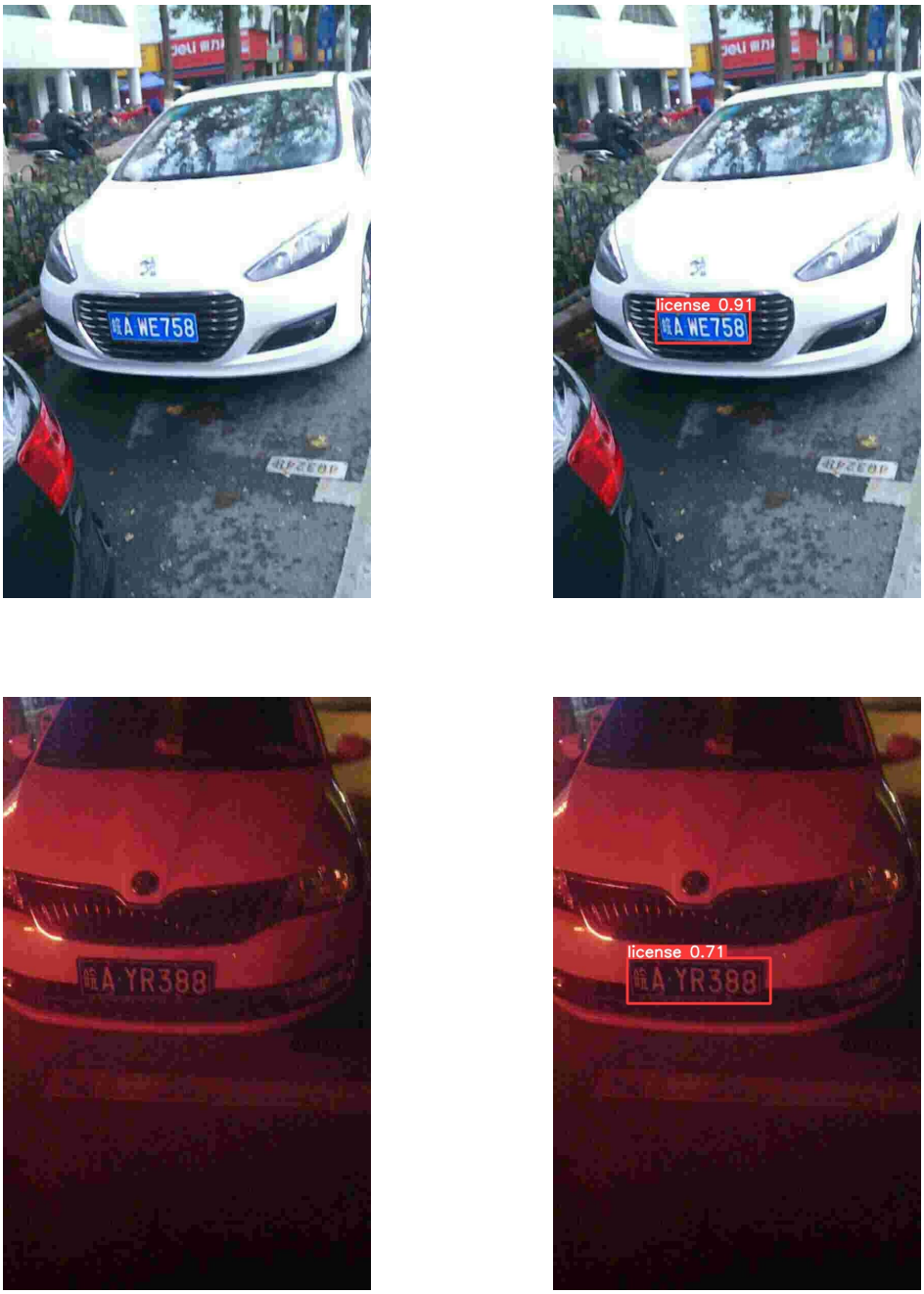


图 16 车牌识别检测结果

五、 实验探究

5.1 如何构建自己的数据集

安装：直接 **pip install labelimg** 即可
工具：**labelimg**

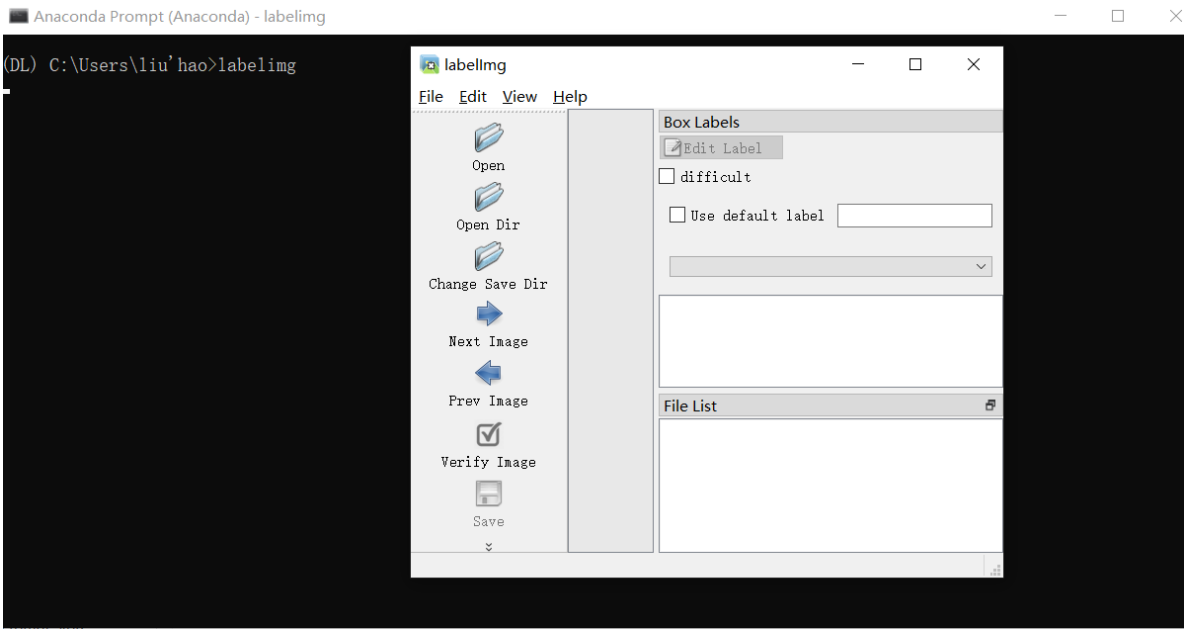


图 17 labelimg 启动示意图

使用介绍：

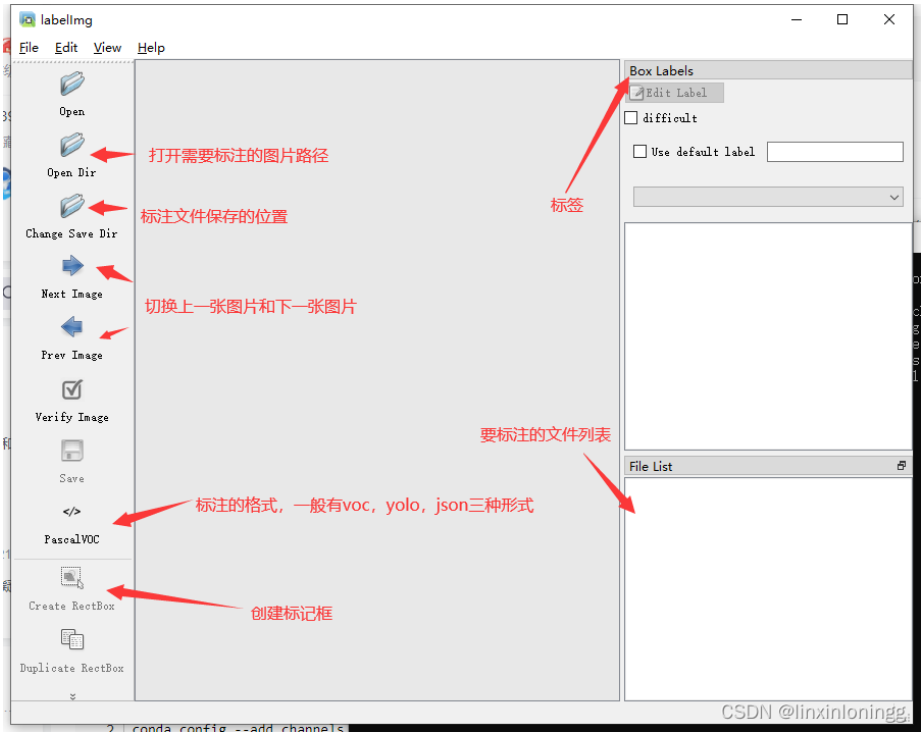
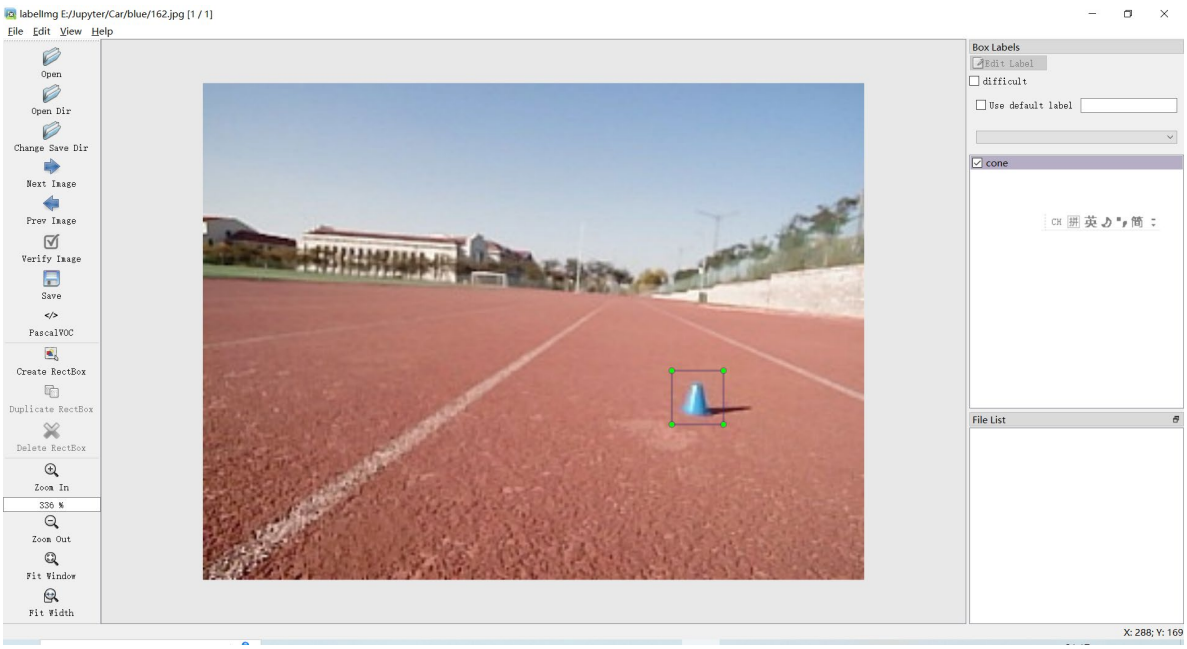


图 18 labelimg 软件说明

标注示例(手动把目标框起来并加入标签即可)：



手动打标签的如何训练？

和实验原理中的**类似**，将数据集导入后划分训练集验证集和测试集，并进行格式的转化，创建新的 yaml 文件修改 train.py 中的文件即可训练，测试过程同理。

六、 实验感想

YOLO 是一个目标检测模型，在本次实验的过程中，我学会了如何构建自己的数据集训练，体会到了目标检测的神奇之处，我使用开源的 CCPD 数据集进行了测试，由于训练时间过长，我在实验的过程中仅仅训练了 15 轮，不过取得的效果也不错，Yolo 的代码比较复杂，读代码花了不少时间，但是最终训练的时候，收获满满，我用 YOLO 标注了自己的数据集(识别操场上的蓝色锥桶)，效果也很好！

在提交代码时，为避免文件过大，并未提交 CCPD 数据集，如测试代码，需下载 CCPD 数据集并运行 ToYOLO.ipynb(data 文件夹下)代码完成数据集构造。

七、 参考资料

[1]. https://www.bilibili.com/video/BV1XG411j7L3?t=307.8&p=15
[2]. http://t.csdnimg.cn/a5jLZ
[3]. http://t.csdnimg.cn/gpSFY