

Python 期末车牌识别大作业报告

枚辉煌

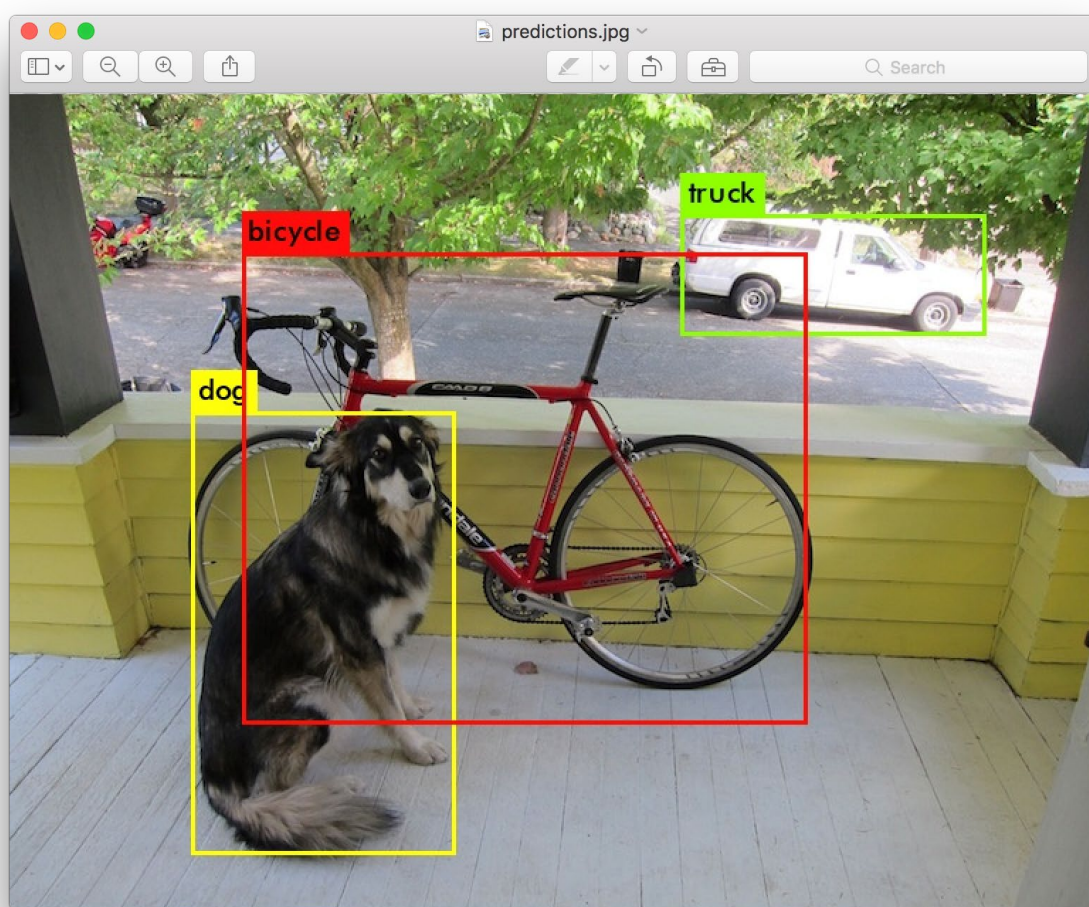
1800094810

子任务 1：车牌定位

我训练一个 yolov3-tiny 网络模型来识别检测定位车牌。

YOLO 网络官方网页：<https://pjreddie.com/darknet/yolo/>

YOLO 是一个实时物体检测，可以在一张图片中定位出物体的位置，然后分类物体，可视化结果是这样：



官方网页也说明 yolo 原理：

“Prior detection systems repurpose classifiers or localizers to perform detection. They apply the model to an image at multiple locations and scales. High scoring regions of the image are considered detections.

We use a totally different approach. We apply a single neural network to the full image. This network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.

Our model has several advantages over classifier-based systems. It looks at the whole image at test time so its predictions are informed by global context in the image. It also makes predictions with a single network evaluation unlike systems like R-CNN which require thousands for a single image. This makes it extremely fast, more than 1000x faster than R-CNN and 100x faster than Fast R-CNN“

我用的网络模型是 yolov3-tiny，是 yolov3 的一个简化版本，主要使用 1×1 和 3×3 卷积核获取特征。网络结构如下：

Layer	Type	Filters	Size/Stride	Input	Output
0	Convolutional	16	$3 \times 3/1$	$416 \times 416 \times 3$	$416 \times 416 \times 16$
1	Maxpool		$2 \times 2/2$	$416 \times 416 \times 16$	$208 \times 208 \times 16$
2	Convolutional	32	$3 \times 3/1$	$208 \times 208 \times 16$	$208 \times 208 \times 32$
3	Maxpool		$2 \times 2/2$	$208 \times 208 \times 32$	$104 \times 104 \times 32$
4	Convolutional	64	$3 \times 3/1$	$104 \times 104 \times 32$	$104 \times 104 \times 64$
5	Maxpool		$2 \times 2/2$	$104 \times 104 \times 64$	$52 \times 52 \times 64$
6	Convolutional	128	$3 \times 3/1$	$52 \times 52 \times 64$	$52 \times 52 \times 128$
7	Maxpool		$2 \times 2/2$	$52 \times 52 \times 128$	$26 \times 26 \times 128$
8	Convolutional	256	$3 \times 3/1$	$26 \times 26 \times 128$	$26 \times 26 \times 256$
9	Maxpool		$2 \times 2/2$	$26 \times 26 \times 256$	$13 \times 13 \times 256$
10	Convolutional	512	$3 \times 3/1$	$13 \times 13 \times 256$	$13 \times 13 \times 512$
11	Maxpool		$2 \times 2/1$	$13 \times 13 \times 512$	$13 \times 13 \times 512$
12	Convolutional	1024	$3 \times 3/1$	$13 \times 13 \times 512$	$13 \times 13 \times 1024$
13	Convolutional	256	$1 \times 1/1$	$13 \times 13 \times 1024$	$13 \times 13 \times 256$
14	Convolutional	512	$3 \times 3/1$	$13 \times 13 \times 256$	$13 \times 13 \times 512$
15	Convolutional	255	$1 \times 1/1$	$13 \times 13 \times 512$	$13 \times 13 \times 255$
16	YOLO				
17	Route 13				
18	Convolutional	128	$1 \times 1/1$	$13 \times 13 \times 256$	$13 \times 13 \times 128$
19	Up-sampling		$2 \times 2/1$	$13 \times 13 \times 128$	$26 \times 26 \times 128$
20	Route 19 8				
21	Convolutional	256	$3 \times 3/1$	$13 \times 13 \times 384$	$13 \times 13 \times 256$
22	Convolutional	255	$1 \times 1/1$	$13 \times 13 \times 256$	$13 \times 13 \times 256$
23	YOLO				

网络实现的框架我利用 github 上的一个项目：<https://github.com/ultralytics/yolov3>

网络结构定义在 ./cfg/yolov3-1cls.cfg

首先要将 plate_data 转化为 darknet 格式的数据集。我写了一个代码

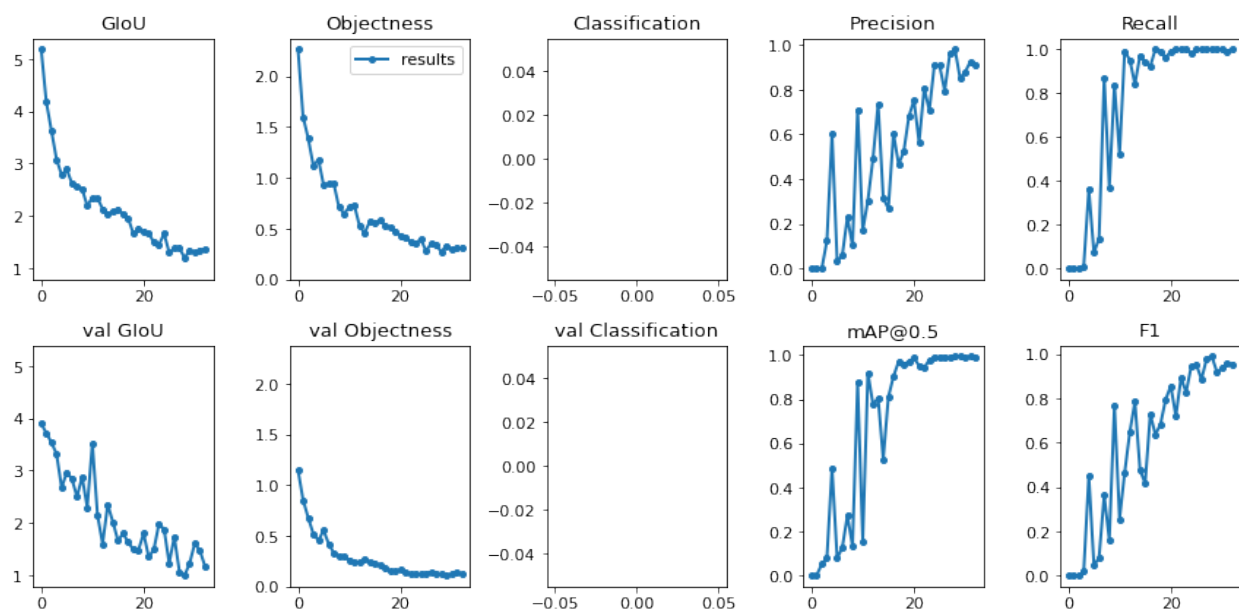
在 ./data/convert_plate_data.py 实现这个任务，主要是将图片和 ground_truth 分别放在两个目录 images 和 labels，另外要把这些路径写在 ./data/plate_test.txt 和 ./data/plate_train.txt

里。./data/plate.names 定义类名，这里只有一个类是 license_plate。./data/plate.data 定义类的个数为 1，训练集和测试集的文件和类名。

训练网络阶段，因为我笔记本的计算力不强所以我把代码放到 Google Colab 上训练，训练 40 个 epoch 要大概一个多小时。训练命令如下：

```
python3 train.py --data data/plate.data --cfg cfg/yolov3-tiny-1cls.cfg --weights '' --single-cls
```

训练过程如下：



可以看到 30 多个 epoch 以后，precision 和 mean average precision (IoU > 0.5) 都达到。最好的 model 会在 ./weights 存下来它的 weights 文件叫 yolov3-tiny-best.pt。有了这个 weights 我可以在本地上运行定位车牌，plate_detection.py 主要任务是创建一个 Darknet 网络模型，读取 weights 文件然后进行检测定位，bounding box 的四个坐标保存在 ./data/plate_data/test/output/*.txt 里，其中 * 是图片的文件名。Plate_detection 命令如下：

```
python plate_detection.py --cfg ./cfg/yolov3-tiny-1cls.cfg  
--weights ./weights/best.pt --names ./data/plate.names  
--source ./data/plate_data/test/images
```

eval_mission1.py 输出的结果：

所有样本的平均 iou:0.9246258994479812

检测正确的样本数目:100

检测定位车牌的结果:



子任务 2：字符分割和识别

1. 字符分割 在 character_segmentation.py 中实现，基本思路是：先用 Otsu threshold 将车牌二值化，像这样：



因为车牌的边界有比较多的 noise，所以我写一个 remove_border 函数将边界的一些行和列删除，经过调参最后对四个边界的参数为 top, bot, left, right = 6, 4, 3, 3。

然后用 opencv 中的 findContours 函数将这些字符轮廓找出来，再用 cv.boundingRect 得到每个字符的 bounding box。对于每个 bounding box 我们知道 x, y, w, h 分别是 top left corner, width, height，然后计算出 $roi_ratio = (w * h) / img_area$, $box_ratio = w / h$ ，分别表示这个 box 在车牌中面积占比和宽度与高度的比例。我调参以后设定 $roi_ratio \geq 0.015$ and $box_ratio \geq 0.1$ and $h > 10$ 才有可能是一个字符。

这个做法有一个问题：在二值化的车牌中会有比较多的 noise，将一些字符连起来，所以 `boundRect` 就将几个字符放在一个框格里面。比如上面第二个车牌 Y7 被放在一块，4445 被放在一块。不难看出字符是等高的，除了 1 以外字符也是等宽的，所以我根据 `box_ratio` 把这些大框格等宽切成几个小框。经过调参最后的参数是 `[0.83, 1.4, 1.9, 2.6, 3.2, inf]`，比如 `box_ratio <= 0.83` 就是一个字符，`0.83 < box_ratio <= 1.4` 切成两个字符。一下是一些比较好等宽分割的结果：



但是也有几个车牌因为位置不正或者 noise 导致 `box_ratio` 算错：



我在调 `box_ratio` 参数的时候，评价指标是只要能切成 6 个小框格就算对，对于 test 中的 100 个图片有 96 个算对。结果保存在 `output` 目录里，每个车牌有二值化图片、已画框格的图片和 `chars` 文件保存每个框格的信息。比如第一张图片在 `output` 里会对应 5 个文件：

1.jpg: 检测定位车牌的图片

1.txt: 车牌的位置 (`xmin, ymin, xmax, ymax`)

1_binary.jpg: 车牌二值化图片

1_chars.txt: 六个框格的位置，每行是一个框格 (`xmin, ymin, xmax, ymax`)

1_grey.txt: 车牌在 gray scale 图片。

2. 字符识别 在 `character_recognition.py` 实现

我用 tensorflow 训练一个 CNN 网络，结果如下：

Input: 20 x 20 x 1: 车牌的二值化图片

Conv2D: 32 filters, kernel_size = (3, 3), relu

MaxPooling2D: kernel_size = (2, 2)

Conv2D: 64 filters, kernel_size = (3, 3), relu

MaxPooling2D: kernel_size = (2, 2)

Conv2D: 64 filters, kernel_size = (3, 3), relu

Flatten layers



FC layers: output size 64, relu

FC layers: output size 34/10/24, softmax

我训练 3 个 model，对应着识别字母，数字，字母与数字，分别保存在 `./model/all_model.h5`, `number_model.h5`, `letter_model.h5`。训练三个 model 的理由是车牌中的两个字符大概都是数字，所以用 `number_model` 特判这两个位置正确率会提高一点。

参数 `batch_size = 32`, `epochs = 20`, 测试准确率大概 0.92 ~ 0.95。

我一开始用的已给的 `chars_data`，车牌识别准确率是 0.14。我发现几个经常判错的字符，比如 D 判成 0，6 判成 G，8 判成 B。这些字符对比较相似的，但是判错主要原因是 `chars_data` 中字符和

车牌中字符的字体不一样，比如 chars_data 中的 6  和车牌中的 6 。另外因为 chars_data 中的字符比较整齐且没有 noise，车牌中的切完字符可能缺少一部分并且有比较多的 noise，例如：

K  0  D  R  R  U  6  

所以我就自己搞一个训练集，用 character_segmentation.py 将 train 中的车牌字符分割然后按照 xml 中的 platetext 生成 chars_data_from_plate 的数据。上面几个字符都是从这个训练集。这样可以解决两个问题：字体不同和 noise 问题，因为训练和测试的图片是有一致性。生成训练集我写在 ./data/make_chars_data.py 里。判断的文字写在 output 目录 *_text.txt。

另外我写一个代码 ./data/convert_test_output.py，任务是将 output 目录中的车牌定位和车牌字符识别文件写入 xml_pred 里，方便用 eval_mission2.py。eval_mission2.py 的输出如下：

```
Plate 15 wrong, expected T51735, predicted AAAAAA
Plate 32 wrong, expected HZ6217, predicted UZ62T7
Plate 51 wrong, expected 6998DB, predicted 96980B
Plate 54 wrong, expected JZ0942, predicted AAAAAA
Plate 55 wrong, expected 0329HB, predicted AAAAAA
Plate 61 wrong, expected 0X6511, predicted XH2511
Plate 68 wrong, expected 3D0061, predicted AAAAAA
Plate 8 wrong, expected YY8818, predicted YY881N
Plate 92 wrong, expected OH1357, predicted OH1357
Plate 95 wrong, expected 3A7705, predicted 3A7715
Plate 96 wrong, expected FL4063, predicted FE4063
```

车牌预测准确率为 0.89

对 5 个字符以上的准确率为 0.93

字符识别准确率为 0.94

其中几个 AAAAAA 是分割失败的车牌，其他几个车牌最多错 3 个字符。车牌识别准确率提高了很多，到了 0.89，这说明我自己搞的 chars_data_from_plate 效果不错，而且字符识别准确率为 0.94 符合训练时的 test_acc，这说明训练和测试集的一致性很重要。上面结果有几个 predicted AAAAAA 是由于字符分割失败，其他车牌最多只错了 3 个字符，Plate 92 错的原因是因为训练集没有 O (字母) 所以被判成 0 (数字)，所以准确率应该是 90/100。我对这个结果比较满意了。