我们在前面讲解了如何使用打码平台来识别验证码，简单高效。但是也有一些缺点，比如效率可能没那么高，准确率也不一定能做到完全可控，并且需要付出一定的费用。

本课时我们就来介绍使用深度学习来识别验证码的方法，训练好对应的模型就能更好地对验证码进行识别，并且准确率可控，节省一定的成本。

本课时我们以深度学习识别滑块验证码为例来讲解深度学习对于此类验证码识别的实现。

滑块验证码是怎样的呢？如图所示，验证码是一张矩形图，图片左侧会出现一个滑块，右侧会出现一个缺口，下侧会出现一个滑轨。左侧的滑块会随着滑轨的拖动而移动，如果能将左侧滑块匹配滑动到右侧缺口处，就算完成了验证。

由于这种验证码交互形式比较友好，且安全性、美观度上也会更高，像这种类似的验证码也变得越来越流行。另外不仅仅是“极验”，其他很多验证码服务商也推出了类似的验证码服务，如“网易易盾”等，上图所示的就是“网易易盾”的滑动验证码。

没错，这种滑动验证码的出现确实让很多网站变得更安全。但是做爬虫的可就苦恼了，如果想采用自动化的方法来绕过这种滑动验证码，关键点在于以下两点：

- 找出目标缺口的位置。
- 模拟人的滑动轨迹将滑块滑动到缺口处。

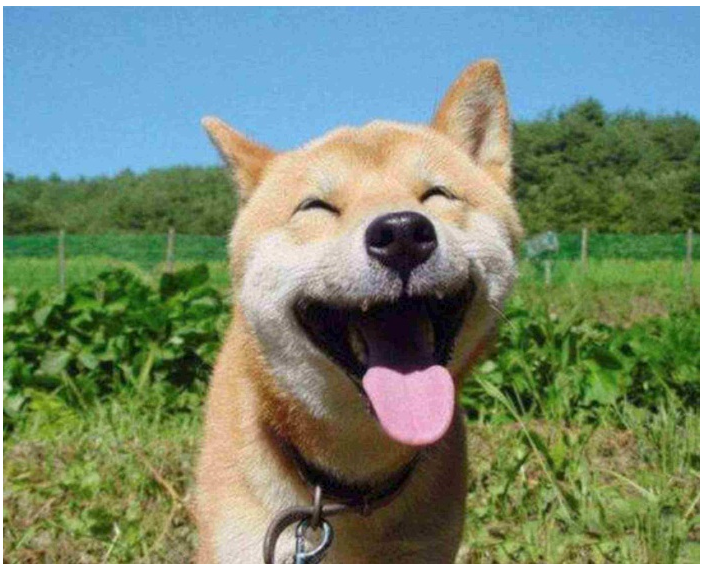
那么问题来了，第一步怎么做呢？

接下来我们就来看看如何利用深度学习来实现吧。

目标检测

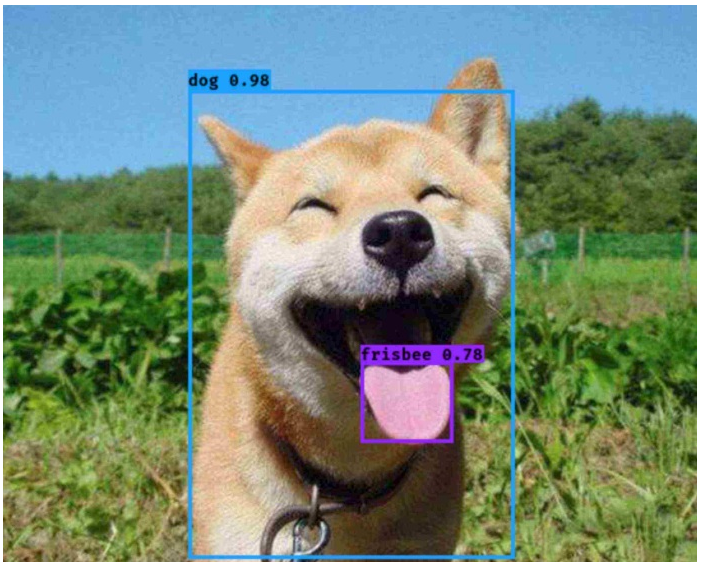
我们的目标就是输入一张图，输出缺口的的位置，所以只需要将这个问题归结成一个深度学习的“目标检测”问题就好了。

首先在开始之前简单说下目标检测。什么叫目标检测？顾名思义，就是把我们想找的东西找出来。比如给一张“狗”的图片，如图所示：



我们想知道这只狗在哪，它的舌头在哪，找到了就把它们框选出来，这就是目标检测。

经过目标检测算法处理之后，我们期望得到的图片是这样的：



可以看到这只狗和它的舌头就被框选出来了，这样就完成了一个不错的目标检测。

当前做目标检测的算法主要有两个方向，有一阶段式和两阶段式，英文分别叫作 One stage 和 Two stage，简述如下。

- Two Stage：算法首先生成一系列目标所在位置的候选框，然后再对这些框选出来的结果进行样本分类，即先找出来在哪，然后再分出来是什么，俗话说叫“看两眼”，这种算法有 R-CNN、Fast R-CNN、Faster R-CNN 等，这些算法架构相对复杂，但准确率上有优势。
- One Stage：不需要产生候选框，直接将目标定位和分类的问题转化为回归问题，俗话说叫“看一眼”，这种算法有 YOLO、SSD，这些算法虽然准确率上不及 Two stage，但架构相对简单，检测速度更快。

所以这次我们选用 One Stage 的有代表性的目标检测算法 YOLO 来实现滑动验证码缺口的识别。

YOLO，英文全称叫作 You Only Look Once，取了它们的首字母就构成了算法名，目前 YOLO 算法最新的版本是 V3 版本，这里算法的具体流程我们就不过多介绍了，如果你感兴趣可以搜一下相关资料了解下，另外也可以了解下 YOLO V1～V3 版本的不同和改进之处，这里列几个参考链接。

- YOLO V3 论文：https://pjreddie.com/media/files/papers/YOLOv3.pdf
- YOLO V3 介绍：https://zhuanlan.zhihu.com/p/34997279
- YOLO V1-V3 对比介绍：https://www.cnblogs.com/makefile/p/yolov3.html

数据准备

回归我们本课时的主题，我们要做的是缺口的位置识别，那么第一步应该做什么呢？

我们的目标是要训练深度学习模型，那我们总得需要让模型知道要学点什么东西吧，这次我们做缺口识别，那么我们需要让模型学的就是找到这个缺口在哪里。由于一张验证码图片只有一个缺口，要分类就是一类，所以我们只需要找到缺口位置就行了。

好，那模型要学如何找出缺口的位置，就需要我们提供样本数据让模型来学习才行。样本数据怎样的呢？样本数据就得有带缺口的验证码图片以及我们自己标注的缺口位置。只有把这两部分都告诉模型，模型才能去学习。等模型学好了，当我们再给个新的验证码时，就能检测出缺口在哪里了，这就是一个成功的模型。

OK，那我们就开始准备数据和缺口标注结果吧。

数据这里用的是网易盾的验证码，验证码图片可以自行收集，写个脚本批量保存下来就行。标注的工具可以使用 LabelImg，GitHub 链接为：https://github.com/tzutalin/labelImg，利用它我们可以方便地进行检测目标位置的标注和类别的标注，如这里验证码和标注示例如下：

标注完了会生成一系列 xml 文件，你需要解析 xml 文件把位置的坐标和类别等处理一下，转成训练模型需要的数据。

在这里我已经整理好了我的数据集，完整 GitHub 链接为：https://github.com/Python3WebSpider/DeepLearningSlideCaptcha，我标注了 200 多张图片，然后处理了 xml 文件，变成训练 YOLO 模型需要的数据格式，验证码图片和标注结果见 data/captcha 文件夹。

如果要训练自己的数据，数据格式准备见：https://github.com/eriklindernoren/PyTorch-YOLOv3#train-on-custom-dataset

初始化

上一步我已经把标注好的数据处理好了，可以直接拿来训练了。

由于 YOLO 模型相对比较复杂，所以这个项目我就直接基于开源的 PyTorch-YOLOV3 项目来进行修改了，模型使用的深度学习框架为 PyTorch，具体的 YOLO V3 模型的实现这里不再阐述了。

另外推荐使用 GPU 训练，不然拿 CPU 直接训练速度会很慢。我的 GPU 是 P100，几乎十几秒就训练完一轮。

下面就直接把代码克隆下来吧。

由于本项目我把训练好的模型也放上去了，使用了 Git LFS，所以克隆时间较长，克隆命令如下：

```
git clone https://github.com/Python3WebSpider/DeepLearningSlideCaptcha.git
```

如果想加速克隆，可以暂时先跳过大文件模型下载，可以执行命令：

```
GIT_LFS_SKIP_SMUDGE=1 git clone https://github.com/Python3WebSpider/DeepLearningSlideCaptcha.git
```

环境安装

代码克隆下载之后，我们还需要下载一些预训练模型。

YOLOV3 的训练要加载预训练模型才能有不错的训练效果，预训练模型下载命令如下：

```
bash prepare.sh
```

执行这个脚本，就能下载 YOLO V3 模型的一些权重文件，包括 yolov3 和 weights，还有 darknet 的 weights，在训练之前我们需要用这些权重文件初始化 YOLO V3 模型。

注意：Windows 下建议使用 Git Bash 来运行上述命令。

另外还需要安装一些必须的库，如 PyTorch、TensorBoard 等，建议使用 Python 虚拟环境，运行命令如下：

```
pip3 install -r requirements.txt
```

这些库都安装好了之后，就可以开始训练了。

训练

本项目已经提供了标注好的数据集，在 data/captcha，可以直接使用。

当前数据训练脚本：

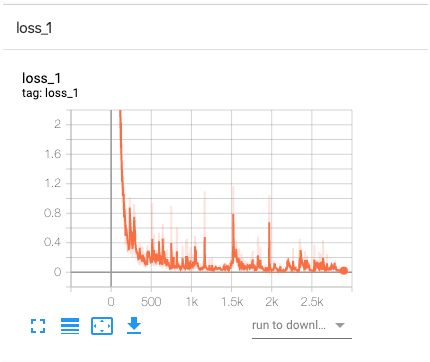
```
bash train.sh
```

实测 P100 训练时长约 15 秒一个 epoch，大约几分钟即可训练出较好效果。

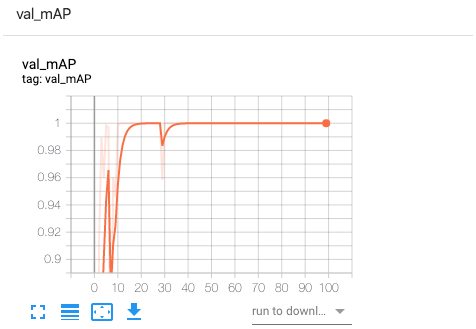
训练差不多了，我们便可以使用 TensorBoard 来看看 loss 和 mAP 的变化，运行 TensorBoard：

```
tensorboard --logdir='logs' --port=6006 --host 0.0.0.0
```

loss_1 变化如下：



val_mAP 变化如下：



可以看到 loss 从最初的非常高下降到了很低，准确率也逐渐接近 100%。

另外训练过程中还能看到如下的输出结果：

```
---- [Epoch 99/100, Batch 27/29] ----
+------------+--------------+--------------+--------------+
| Metrics    | YOLO Layer 0 | YOLO Layer 1 | YOLO Layer 2 |
+------------+--------------+--------------+--------------+
| grid_size  | 14           | 28           | 56           |
| loss       | 0.028268     | 0.046053     | 0.043745     |
| x          | 0.002108     | 0.005267     | 0.008111     |
| y          | 0.004561     | 0.002016     | 0.009047     |
| w          | 0.001284     | 0.004618     | 0.000207     |
| h          | 0.000594     | 0.000528     | 0.000946     |
| conf       | 0.019700     | 0.033624     | 0.025432     |
| cls        | 0.000022     | 0.000001     | 0.000002     |
| cls_acc    | 100.00%      | 100.00%      | 100.00%      |
| recall50   | 1.000000     | 1.000000     | 1.000000     |
| recall75   | 1.000000     | 1.000000     | 1.000000     |
| precision  | 1.000000     | 0.800000     | 0.666667     |
| conf_obj   | 0.994271     | 0.999249     | 0.997762     |
| conf_noobj | 0.000126     | 0.000158     | 0.000140     |
+------------+--------------+--------------+--------------+
Total loss 0.11806630343198776
```

这里显示了训练过程中各个指标的变化情况，如 loss、recall、precision、confidence 等，分别代表训练过程的损失（越小越好）、召回率（能识别出的结果占应该识别出结果的比例，越高越好）、精确率（识别出的结果中正确的比率，越高越好）、置信度（模型有把握识别对的概率，越高越好），可以作为参考。

测试

训练完毕之后会在 checkpoints 文件夹生成 pth 文件，可直接使用模型来预测生成标注结果。

如果你没有训练自己的模型的话，这里我已经把训练好的模型放上去了，可以直接使用我训练好的模型来测试。如之前跳过了 Git LFS 文件下载，则可以使用如下命令下载 Git LFS 文件：

```
git lfs pull
```

此时 checkpoints 文件夹会生成训练好的 pth 文件。

测试脚本：

```
sh detect.sh
```

该脚本会读取 captcha 下的 test 文件夹所有图片，并将处理后的结果输出到 result 文件夹。

运行结果样例：

```
Performing object detection:
        + Batch 0, Inference Time: 0:00:00.044223
        + Batch 1, Inference Time: 0:00:00.028566
        + Batch 2, Inference Time: 0:00:00.029764
        + Batch 3, Inference Time: 0:00:00.032430
        + Batch 4, Inference Time: 0:00:00.033373
        + Batch 5, Inference Time: 0:00:00.027861
        + Batch 6, Inference Time: 0:00:00.031444
        + Batch 7, Inference Time: 0:00:00.032110
        + Batch 8, Inference Time: 0:00:00.029131

Saving images:
(0) Image: 'data/captcha/test/captcha_4497.png'
        + Label: target, Conf: 0.99999
(1) Image: 'data/captcha/test/captcha_4498.png'
        + Label: target, Conf: 0.99999
(2) Image: 'data/captcha/test/captcha_4499.png'
        + Label: target, Conf: 0.99997
(3) Image: 'data/captcha/test/captcha_4500.png'
        + Label: target, Conf: 0.99999
(4) Image: 'data/captcha/test/captcha_4501.png'
        + Label: target, Conf: 0.99997
(5) Image: 'data/captcha/test/captcha_4502.png'
        + Label: target, Conf: 0.99999
(6) Image: 'data/captcha/test/captcha_4503.png'
        + Label: target, Conf: 0.99997
(7) Image: 'data/captcha/test/captcha_4504.png'
        + Label: target, Conf: 0.99998
(8) Image: 'data/captcha/test/captcha_4505.png'
        + Label: target, Conf: 0.99998
```

拿几个样例结果看下：

这里我们可以看到，利用训练好的模型我们就成功识别出缺口的位置了，另外程序还会打印输出这个边框的中心点和宽高信息。

有了这个边界信息，我们再利用某些手段拖动滑块即可通过验证了，比如可以模拟加速减速过程，或者可以录制人的轨迹再执行都是可以的，由于本课时更多是介绍深度学习识别相关内容，所以关于拖动轨迹不再展开讲解。

总结

本课时我们介绍了使用深度学习识别滑动验证码缺口的方法，包括标注、训练、测试等环节都进行了阐述。有了它，我们就能轻松方便地对缺口进行识别了。

代码：https://github.com/Python3WebSpider/DeepLearningSlideCaptcha