Processing Timer

一些废话

本程序是浙江大学2022-2023（秋冬）普物实验II-真空实验装置改进与研究课程中的部分产出。

很遗憾，由于疫情原因，本次实验原定目标的实时读取真空装置仪表的部分未能完成。在网课环境下我们选则尝试实时拍摄并读取平板的时钟，并将读取到的数据进行实时写入`Excel`表格。

~~由于临近期末考试，以及负责代码的同学接连阳性，本次实验的结项的成果比较差强人意。~~

结项时，我们读取拍摄后视频（ProcessingTimer/videos/2.MP4）获得了100%的准确率，实时读取大约有70~80%左右的准确率。这显然距离真正具有实用性还有很远的距离。而疫情前实现的`InstantProcessing`部分则识别率更低一些，但可以实时识别真空装置四个表格。

同时，为了项目的泛用性，我们还编写了一个脚本`GenerateMyDataset.py`，对于一个新的表，数字形状不同的情况下，它可以根据截取少量的图片，生成大量的数据集，重新训练神经网络，来提高神经网络的准确性。

~~这些困难就留给学弟学妹们了~~

本项目正常运行依赖的必须的库： `opencv-python`，`matplotlib`，`torch`，`numpy`，`torchvision`，`xlwt`，`Pillow`。本实验在`PyTorch`环境下进行，建议使用`Anaconda`虚拟环境。

特别鸣谢一下前一届的学长，尤其是彭博学长，我们的项目很大程度上基于他们优秀的工作，这里是他们的项目链接。

项目部分简介

`template images`:存放用于`MatchTemplate`操作的图片
`trainset`:用于存放训练神经网络的数据集
`PaddingIMG`:用于存放那些需要生成数据集的原图
`pth`:用于存放训练完成的神经网络
`resultData`:用于存放生成完毕的数据集
`videos`:非实时模式时，用于存放需要识别的视频

`GenerateMyDataset.py`:用于根据新的表，生成大量的数据集，来提高神经网络的准确性
`ProcessingTimer.py`:实时处理并读取时间表
`openCVdemo.py`:用来测试和调试摄像头
`TimerExpNetwork.py`:为`ProcessingTimer.py`必要的依赖
`TrainNetwork.ipynb`:用来训练新的神经网络模型

> 注： `InstantProcessing.py`是用于实时读取真空实验装置的，于后期的读取时间表无关。

主要程序部分介绍

ProcessingTimer（或InstantProcessing）部分

使用方法及注意事项

本部分可以实现实时拍摄电子表并读取其中的数字，或者读取已拍摄的一个视频中的数字。

- 如果你希望实时拍摄电子表并读取其中的数字，那就更改522行的参数`VideoNumber`为你的摄像头号码，其中，你的摄像头号码可以通过`openCVdemo.py`获得
- 如果你希望读取已拍摄的一个视频中的数字，那就更改522行的参数`VideoNumber`为你的视频路径，并注释掉代码266行，并解开265行的注释

本程序的运行可以直接在命令行中输入

```
python ProcessingTimer.py
```

请保证在运行前你**已经**安装了必要的依赖。

正常运行时，程序会跳出一个**720p**的实时画面，同时命令行中会打印参数。

- 如果你看到的是`--Processing!--`，那么程序会正常运行
- 如果你看到的是`Can't Find`，那么请根据实时画面适当调整拍摄角度，直到出现`--Processing!--`

请注意，画面必须是720P的，的，因为这涉及到对数字像素裁切时的裁切宽度，错误的分辨率可能导致识别错误或者程序直接报错强制退出。
当然，我们的程序已经保证画面参数为720P，但是摄像头的更换或者错误的参数调整可能会影

响这一点。

在你需要结束程序时，你可以通过按压键盘上的'q'键来quit退出程序。

算法基本原理

本次实验中，所有实验用图像均由摄像头从屏幕或其他显示器上实时读取，**也可以通过改变调用路径直接从本地视频中进行读取**。

首先，在程序中，我们定义了一个`find_marker`函数，用以确定要计算距离的物体。我们在图片中找到目标物体我们通过调用`OpenCV`中的`cvtColor`、`GaussianBlur`、`Canny`函数进行轻微高斯模糊处理以去除高频噪声，然后进行边缘检测。接下来调用`findContours`函数找到图片中的众多轮廓，**然后根据需要找到最大的目标轮廓**。由于真空表和平板电脑都是画面中的最大轮廓。在找到轮廓后，我们通过将实时视频中的轮廓宽度除以已经调整的参数（通过已拍摄的视频得到），得到一个比例尺，已解决实时处理时距离电子表的远近问题。

然后，我们通过调用`opencv-python`库中的`MatchTemplate`函数，匹配画面中最契合图片`E1`的位置（本次实验中是iPad OS底部的小白条），并通过相对位置的画面裁切，裁切出几个数字的位置。
值得一提的是，如果处于某些偏僻的相对位置，可能导致裁切像素超出画面，这种情况下我们会打印`Can't Find`，并希望使用者能够调整拍摄，尽量使画面居中。
处于泛用性考虑，这张用于匹配图片`E1`也可以根据需要实时读取的表格的不同而替换。

最后，我们每隔1s抽取一帧并根据比例尺裁切，通过`matplotlib`库将采集到的图像转化为灰度图，每次获取图像时重新校准比例尺，利用训练得到效果最好的神经网络对数字进行识别并保存。在接收到`q`的信号后，程序停止运行，使用`OS`库操作生成对应的Excel表格。

GenerateMyDataset和TrainNetwork部分

使用方法及注意事项

通过少量实时画面中截取或者说其他形式（例如照片等），得到0~9以及小数点（如果有的话）的照片，并且用一定方式命名后存放于`PaddingIMG`文件夹中。然后修改`GenerateMyDataset.py`的参数，选择生成一定数量的图片以形成数据集。

数据集完备后，运行`TrainNetwork`的`Jupiter`文件，更改一下新的神经网络名称，得到一个新的神经网络。

算法基本原理

在开始正式读取数据以前，出于泛用性考量，**更换需要读取的表格时**，由于数字的形状往往变化巨大。如果仍然调用旧版的神经网络，会导致识别率其低无比。所以，我们往往需要重新训练新的卷积神经网络。

我们通过少量实时画面中截取或者说其他形式（例如照片等）得到的表格的数字图片，通过`padding`操作，添加一个随机数，让新的图片在画面的左右上下一定的随机位置，通过少量的几张图片可以padding出数以千计的图片，这就生成了一个简单的数据集。

我们直接运行`TrainNetwork`的`Jupiter`文件，更改一下新的神经网络名称，得到一个新的神经网络，并通过拍摄好的视频，在`ProcessingTimer.py`的读取已拍摄视频的模式下，查看生成的表格的数字是否准确。我们认为非实时做到100%正确时，新的神经网络训练完成。