**人脸性别识别：二分类问题**

**班级 无08**

**学号 2020010841**

**姓名 李煜彤**

**邮箱 liyutong20@mails.tsinghua.edu.cn**

**实验报告**

目录

[一、实验目的及要求 3](#_Toc81863572)

[二、实验过程 4](#_Toc81863573)

[1. 准备工作 4](#_Toc81863574)

[2. 尝试逻辑回归算法 4](#_Toc81863575)

[3. 尝试KNN算法 5](#_Toc81863576)

[4. 尝试CNN算法 6](#_Toc81863577)

[5. 调试KNN算法 13](#_Toc81863578)

[6. 调试CNN 15](#_Toc81863579)

[三、最终模型 20](#_Toc81863580)

[四、结果与分析 22](#_Toc81863581)

[五、源代码 27](#_Toc81863582)

[六、总结 40](#_Toc81863583)

# 一、实验目的及要求

大作业提供人脸图像，数据集包括了白种人、黄种人、黑种人等多种种族数据。数据集存在人脸姿态、光照、年龄等多种干扰，具有一定的挑战性。要求实现：基于人脸图像自动识别该人性别，数据集划分要求为：随机选择40％、10％、50％的数据，分别作为训练、验证和测试集。

① 要求撰写一份实验报告，报告中说明使用了何种模型及方法，以及对应的测试结果；

② 在训练模型时所参考的学术信息需要给出适当的文献引用；

③ 不允许直接使用开源项目提供的已训练好的模型或已写好的现有代码。

④ 附加任务（供参考选做）：设计界面，可在界面上实现输入数据集中的图像后，显示该人的性别。

# 二、实验过程

### 1. 准备工作

首先弄清楚作业需要完成什么，阅读了作业文件，配置了环境；之后复习KNN和逻辑回归算法，初步弄懂助教的示例代码。

### 2. 尝试逻辑回归算法

依据提示，重新写了DataLoader，用以加载LWF数据集。修改的部分主要包括：

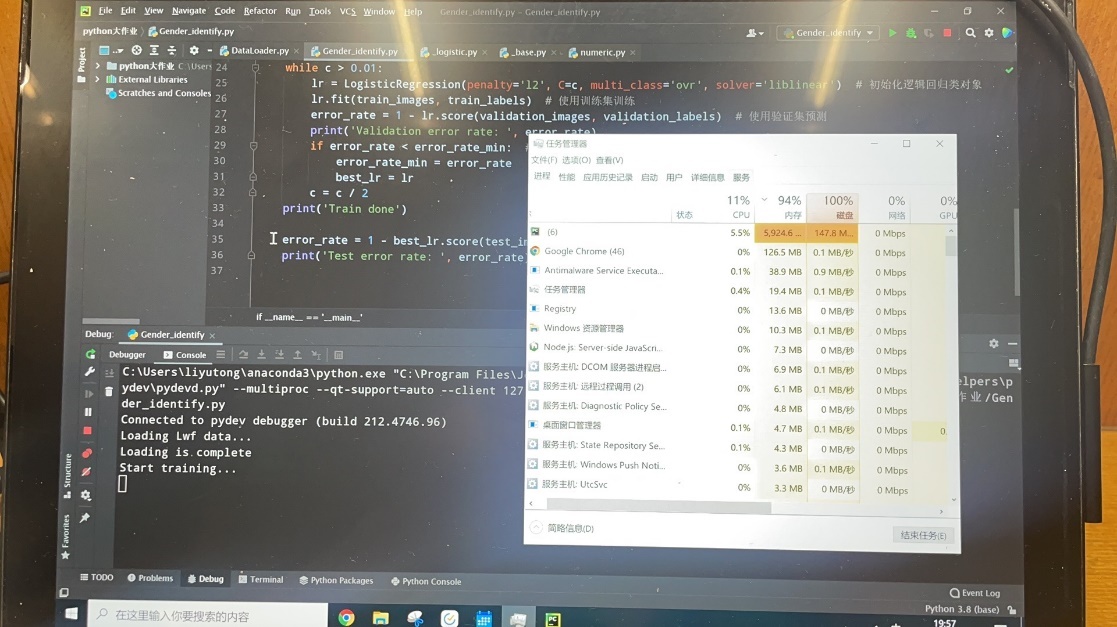
* 没有使用one-hot编码；因为是二分类问题，所以只用1和0作为label。
* 修改数据集的数量及比例
* 修改数据的加载方式：读取男女名单文件，分别加载后合并。

首先尝试逻辑回归算法。

在复习KNN算法时，其中提到KNN算法的缺点可能有计算量大的问题：因为KNN算法会计算新的样本和所有测试样本的距离，进行排序选出最小的K个。私以为LWF的样本量偏大。并且样本男女差距悬殊，恐有盲猜男性的可能。所以优先考虑了逻辑回归算法。

起初我延续助教的代码进行了尝试，但是无论如何调试，都会报错。报错信息如下：

Process finished with exit code -1073740791 (0xC0000409)

这个错误非常的奇怪，因为它只给了我一个错误码，没有给出任何的错误提示，调试过程便突然直接结束。我开始逐行地运行，但是也总是突然退出。上网查了许多资料，大多数说是内存问题。Debug的时候打开任务管理器，有如下界面，可以看到磁盘和内存几乎已经满了：

尝试了许多办法也没有效果。之后尝试了修改一个参数，就是正则化的选择。在上网查资料时，有一个资料说一般使用l2正则化即可，复杂一些的也可以选择l1正则化。原代码中使用的l1，因此我尝试了l1正则化。

终于不报这个错误了。但是运行的时候，在第一个正则化参数训练的时候出现了时间过长的问题（一个多小时都没有出任何结果，我中间甚至睡了一觉）。效率这样低，就算有结果也是差强人意的，遂按下停止按钮。

### 3. 尝试KNN算法

于是转而尝试KNN算法。

依旧先尝试延续助教的代码进行尝试，在其中加入了一些输出内容。以下是运行结果：

Loading Lwf data...

Loading is complete

Start training...

\* \* \* \* \* \* \* \* \* \*

k = 1

Validation error rate: 0.27210884353741494

Total time: 64.59212017059326 seconds

\* \* \* \* \* \* \* \* \* \*

k = 3

Validation error rate: 0.23053665910808763

Total time: 81.94681453704834 seconds

\* \* \* \* \* \* \* \* \* \*

k = 5

Validation error rate: 0.2146636432350718

Total time: 82.62141275405884 seconds

\* \* \* \* \* \* \* \* \* \*

k = 7

Validation error rate: 0.2108843537414966

Total time: 84.6814296245575 seconds

\* \* \* \* \* \* \* \* \* \*

k = 9

Validation error rate: 0.20181405895691606

Total time: 80.4258782863617 seconds

\* \* \* \* \* \* \* \* \* \*

k = 11

Validation error rate: 0.19879062736205588

Total time: 80.05250859260559 seconds

\* \* \* \* \* \* \* \* \* \*

k = 13

Validation error rate: 0.2033257747543462

Total time: 86.75002098083496 seconds

\* \* \* \* \* \* \* \* \* \*

k = 15

Validation error rate: 0.20030234315948603

Total time: 74.16994547843933 seconds

Start testing...

\* \* \* \* \* \* \* \* \* \*

k = KNeighborsClassifier(n\_neighbors=11)

Test error rate: 0.19550434454098975

Total time: 349.1709837913513 seconds

运行了几次，正确率均在80%左右，运行时间也比较慢；但是暂时我也不知道应该如何调试，因此决定尝试CNN算法。

Total time: 80.05250859260559 seconds

\* \* \* \* \* \* \* \* \* \*

k = 13

Validation error rate: 0.2033257747543462

Total time: 86.75002098083496 seconds

\* \* \* \* \* \* \* \* \* \*

k = 15

Validation error rate: 0.20030234315948603

Total time: 74.16994547843933 seconds

Start testing...

\* \* \* \* \* \* \* \* \* \*

Best: KNeighborsClassifier(n\_neighbors=11)

Test error rate: 0.19550434454098975

Total time: 349.1709837913513 seconds

### 4. 尝试CNN算法

CNN算法相对来说门槛高了很多，有了机器学习的感觉。我打算先把助教给的参考代码跑起来，看看效果如何。没想到这一步就耗费了我三天。

首先，由于MNIST是Pytorch自带的数据集，而LWF不是，因此首先要写一个Dataset的派生类，用以载入数据。依据参考文献，有三个函数必须重载，它们的功能分别有：

* \_\_init\_\_

构造函数。除了初始化参数，还包括读取包含名字的名单。

* \_\_len\_\_

返回数据集的大小。

* \_\_getitem\_\_

用于返回某一条数据。在这个函数中，不仅承担了读取数据的功能，还承担了图形变换的功能，变换取决于类的属性transforms。

然后顺利地形成第一个大bug：

TypeError: pic should be PIL Image or ndarray. Got <class 'dict'>

* 原因应该是：在\_\_getitem\_\_函数中，我返回的sample是一个字典，这不符合它的要求。于是有如下更改：

原代码：

尝试了许多办法，这样修改是有效的：

if self.transform:

image = self.transform(image)

return image, label

sample = {'image': image, 'label': label}

if self.transform:

sample = self.transform(sample)

return sample

这个bug解决了。之后开始和第二个大bug的斗争：

ValueError: too many values to unpack (expected 2)

我想这个是说，我要接两个参数，但是我给它传了多于两个。问题出在这行代码：

for inputs, labels in tqdm(train\_loader):

记忆有些模糊了，下面说得可能不是百分百正确：

调试的时候发现传的第一个参数是int，这并不是我想要的。于是修改为：

for i, inputs, labels in tqdm(train\_loader):

然后它居然说我传的参数太少了！

后来又查了很多网页，这样修改是有效的：

for i, (inputs, labels) in tqdm(enumerate(test\_loader)):

这个解决了，然后有一些不太顽固的问题：

* 矩阵尺寸不匹配问题，改变尺寸即可解决
* 输入了三个通道，在读取图片时把其转化为灰度图即可
* ……

然后出现了一个问题：

在定义MyDataset的时候，它是依赖于一个名单文件的。那个文件上有的名字，所对应的图片就是它的数据，不太容易再进行内部的划分。于是我先按比例，分别随机划分了男女名单，然后实例化了四个类，分别是男女的训练集和测试集。

由于我比较信赖python的高级性，我想把两个MyDataset直接进行相加，得到训练集和测试集，后续直接使用这两个数据集就可以了。事实证明我只是痴心妄想。

于是我重新写了一份加载数据的dataset，这次叫LfwDataset。同样是继承Dataset类并重载三个函数。与初代版本区别如下：

* 首先生成一份包含所有名字的名单文件
* 构造函数\_\_init\_\_增加了两个参数：train和test，类型为bool，用以表征它们是训练集还是测试集。如果均为false，则说明是验证集。有了这两个参数，就可以直接对数据集进行删减。
* 构造函数中承担读取数据和图像变换的功能。
* \_\_getitem\_\_直接从其图片列表中返回数据，而不是读取文件。

然后来了这个非常非常非常折磨人的错误，耗了我一天的时间：

RuntimeError: expected scalar type Double but found Float

首先是弄懂哪里出的问题，这个就花了很多功夫。问题出在：读取到的数据经过转化为灰度图之后，就变成了float64（也就是double）。但之后的默认是float32，这两个无法在一起做运算。

然后是寻求解决办法。尝试了太多！都没有用！最后这个解决了问题：在图片变换的时候加上：

image = np.array(image, dtype=np.float32)

天知道我找了这个方法多久！泪目了。

之后尝试运行，得到这样的结果：

Epoch 0/9

----------

0%| | 0/21 [00:00<?, ?it/s]C:\Users\liyutong\anaconda3\lib\site-packages\torch\nn\functional.py:718: UserWarning: Named tensors and all their associated APIs are an experimental feature and subject to change. Please do not use them for anything important until they are released as stable. (Triggered internally at ..\c10/core/TensorImpl.h:1156.)

return torch.max\_pool2d(input, kernel\_size, stride, padding, dilation, ceil\_mode)

100%|██████████| 21/21 [01:45<00:00, 5.02s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.1069 Acc: 0.9667

100%|██████████| 104/104 [03:06<00:00, 1.80s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 8800.7780 Acc: 0.5512

Epoch 1/9

----------

100%|██████████| 21/21 [01:43<00:00, 4.94s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:10<00:00, 1.84s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11590.8875 Acc: 0.5512

Epoch 2/9

----------

100%|██████████| 21/21 [02:07<00:00, 6.09s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:33<00:00, 2.06s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11928.6479 Acc: 0.5512

Epoch 3/9

----------

100%|██████████| 21/21 [02:10<00:00, 6.21s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:33<00:00, 2.06s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11966.0086 Acc: 0.5512

Epoch 4/9

----------

100%|██████████| 21/21 [02:09<00:00, 6.16s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:49<00:00, 2.21s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1013 Acc: 0.5512

Epoch 5/9

----------

100%|██████████| 21/21 [02:04<00:00, 5.91s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:35<00:00, 2.08s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1374 Acc: 0.5512

Epoch 6/9

----------

100%|██████████| 21/21 [02:06<00:00, 6.04s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:47<00:00, 2.19s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1377 Acc: 0.5512

Epoch 7/9

----------

100%|██████████| 21/21 [02:10<00:00, 6.22s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:26<00:00, 1.98s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1376 Acc: 0.5512

Epoch 8/9

----------

100%|██████████| 21/21 [01:43<00:00, 4.92s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [02:58<00:00, 1.72s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1377 Acc: 0.5512

Epoch 9/9

----------

100%|██████████| 21/21 [01:49<00:00, 5.21s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:30<00:00, 2.03s/it]

test Loss: 11970.1376 Acc: 0.5512

Training complete in 54m 27s

Best Acc: 0.551156

Epoch 1/9

----------

100%|██████████| 21/21 [01:43<00:00, 4.94s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:10<00:00, 1.84s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11590.8875 Acc: 0.5512

Epoch 2/9

----------

100%|██████████| 21/21 [02:07<00:00, 6.09s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:33<00:00, 2.06s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11928.6479 Acc: 0.5512

Epoch 3/9

----------

100%|██████████| 21/21 [02:10<00:00, 6.21s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:33<00:00, 2.06s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11966.0086 Acc: 0.5512

Epoch 4/9

----------

100%|██████████| 21/21 [02:09<00:00, 6.16s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:49<00:00, 2.21s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1013 Acc: 0.5512

Epoch 5/9

----------

100%|██████████| 21/21 [02:04<00:00, 5.91s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:35<00:00, 2.08s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1374 Acc: 0.5512

Epoch 6/9

----------

100%|██████████| 21/21 [02:06<00:00, 6.04s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:47<00:00, 2.19s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1377 Acc: 0.5512

Epoch 7/9

----------

100%|██████████| 21/21 [02:10<00:00, 6.22s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:26<00:00, 1.98s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1376 Acc: 0.5512

Epoch 8/9

----------

100%|██████████| 21/21 [01:43<00:00, 4.92s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [02:58<00:00, 1.72s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1377 Acc: 0.5512

Epoch 9/9

----------

100%|██████████| 21/21 [01:49<00:00, 5.21s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:30<00:00, 2.03s/it]

test Loss: 11970.1376 Acc: 0.5512

Training complete in 54m 27s

Best Acc: 0.551156

可以发现，train loss为0且准确率为100%，但是测试集的loss大得离谱，并且准确率也太窝囊了些。运行时间也非常久。

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:47<00:00, 2.19s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1377 Acc: 0.5512

Epoch 7/9

----------

100%|██████████| 21/21 [02:10<00:00, 6.22s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:26<00:00, 1.98s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1376 Acc: 0.5512

Epoch 8/9

----------

100%|██████████| 21/21 [01:43<00:00, 4.92s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [02:58<00:00, 1.72s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1377 Acc: 0.5512

Epoch 9/9

----------

100%|██████████| 21/21 [01:49<00:00, 5.21s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:30<00:00, 2.03s/it]

test Loss: 11970.1376 Acc: 0.5512

Training complete in 54m 27s

Best Acc: 0.551156

此时开始准备着手调这些不知所云的参数。上网搜索资料，可以调节学习率试试。于是将学习率从0.1调整到0.00001。结果如下，似乎看到了一点希望的曙光：

Epoch 0/9

----------

0%| | 0/21 [00:00<?, ?it/s]C:\Users\liyutong\anaconda3\lib\site-packages\torch\nn\functional.py:718: UserWarning: Named tensors and all their associated APIs are an experimental feature and subject to change. Please do not use them for anything important until they are released as stable. (Triggered internally at ..\c10/core/TensorImpl.h:1156.)

return torch.max\_pool2d(input, kernel\_size, stride, padding, dilation, ceil\_mode)

100%|██████████| 21/21 [01:56<00:00, 5.55s/it]

train Loss: 2.1394 Acc: 0.5775

100%|██████████| 104/104 [03:10<00:00, 1.84s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 2.0061 Acc: 0.5512

Epoch 1/9

----------

100%|██████████| 21/21 [01:58<00:00, 5.62s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 1.3775 Acc: 1.0000

100%|██████████| 104/104 [03:15<00:00, 1.88s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 1.6796 Acc: 0.5512

Epoch 2/9

----------

100%|██████████| 21/21 [01:48<00:00, 5.16s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.6820 Acc: 1.0000

100%|██████████| 104/104 [03:03<00:00, 1.77s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 1.6598 Acc: 0.5512

Epoch 3/9

----------

100%|██████████| 21/21 [01:54<00:00, 5.45s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.3342 Acc: 1.0000

100%|██████████| 104/104 [03:04<00:00, 1.78s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 1.8188 Acc: 0.5512

Epoch 4/9

----------

100%|██████████| 21/21 [01:42<00:00, 4.87s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.1919 Acc: 1.0000

100%|██████████| 104/104 [03:08<00:00, 1.81s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 1.9856 Acc: 0.5512

Epoch 5/9

----------

100%|██████████| 21/21 [02:01<00:00, 5.78s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.1488 Acc: 1.0000

100%|██████████| 104/104 [03:35<00:00, 2.07s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 1.9996 Acc: 0.5512

Epoch 6/9

----------

100%|██████████| 21/21 [02:35<00:00, 7.41s/it]

train Loss: 0.1478 Acc: 1.0000

100%|██████████| 104/104 [03:44<00:00, 2.16s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 2.0127 Acc: 0.5512

Epoch 7/9

----------

100%|██████████| 21/21 [02:19<00:00, 6.66s/it]

train Loss: 0.1426 Acc: 1.0000

100%|██████████| 104/104 [03:31<00:00, 2.03s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 2.0255 Acc: 0.5512

Epoch 8/9

----------

100%|██████████| 21/21 [02:03<00:00, 5.89s/it]

train Loss: 0.1401 Acc: 1.0000

100%|██████████| 104/104 [03:38<00:00, 2.10s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 2.0383 Acc: 0.5512

Epoch 9/9

----------

100%|██████████| 21/21 [01:56<00:00, 5.53s/it]

train Loss: 0.1338 Acc: 1.0000

100%|██████████| 104/104 [03:20<00:00, 1.93s/it]

test Loss: 2.0508 Acc: 0.5512

Training complete in 53m 51s

Best Acc: 0.551156

Process finished with exit code 0at ..\c10/core/TensorImpl.h:1156.)

return torch.max\_pool2d(input, kernel\_size, stride, padding, dilation, ceil\_mode)

100%|██████████| 21/21 [01:56<00:00, 5.55s/it]

train Loss: 2.1394 Acc: 0.5775

100%|██████████| 104/104 [03:10<00:00, 1.84s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 2.0061 Acc: 0.5512at ..\c10/core/TensorImpl.h:1156.)

return torch.max\_pool2d(input, kernel\_size, stride, padding, dilation, ceil\_mode)

100%|██████████| 21/21 [01:45<00:00, 5.02s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.1069 Acc: 0.9667

100%|██████████| 104/104 [03:06<00:00, 1.80s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 8800.7780 Acc: 0.5512

Epoch 1/9

----------

100%|██████████| 21/21 [01:43<00:00, 4.94s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:10<00:00, 1.84s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11590.8875 Acc: 0.5512

Epoch 2/9

----------

100%|██████████| 21/21 [02:07<00:00, 6.09s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:33<00:00, 2.06s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11928.6479 Acc: 0.5512

Epoch 3/9

----------

100%|██████████| 21/21 [02:10<00:00, 6.21s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:33<00:00, 2.06s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11966.0086 Acc: 0.5512

Epoch 4/9

----------

100%|██████████| 21/21 [02:09<00:00, 6.16s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:49<00:00, 2.21s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1013 Acc: 0.5512

Epoch 5/9

----------

100%|██████████| 21/21 [02:04<00:00, 5.91s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:35<00:00, 2.08s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1374 Acc: 0.5512

Epoch 6/9

----------

100%|██████████| 21/21 [02:06<00:00, 6.04s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:47<00:00, 2.19s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1377 Acc: 0.5512

Epoch 7/9

----------

100%|██████████| 21/21 [02:10<00:00, 6.22s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:26<00:00, 1.98s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1376 Acc: 0.5512

Epoch 8/9

----------

100%|██████████| 21/21 [01:43<00:00, 4.92s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [02:58<00:00, 1.72s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1377 Acc: 0.5512

Epoch 9/9

----------

100%|██████████| 21/21 [01:49<00:00, 5.21s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:30<00:00, 2.03s/it]

test Loss: 11970.1376 Acc: 0.5512

Training complete in 54m 27s

Best Acc: 0.551156

at ..\c10/core/TensorImpl.h:1156.)

return torch.max\_pool2d(input, kernel\_size, stride, padding, dilation, ceil\_mode)

100%|██████████| 21/21 [01:56<00:00, 5.55s/it]

train Loss: 2.1394 Acc: 0.5775

100%|██████████| 104/104 [03:10<00:00, 1.84s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 2.0061 Acc: 0.5512

Epoch 1/9

----------

100%|██████████| 21/21 [01:58<00:00, 5.62s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 1.3775 Acc: 1.0000

100%|██████████| 104/104 [03:15<00:00, 1.88s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 1.6796 Acc: 0.5512

Epoch 2/9

----------

100%|██████████| 21/21 [01:48<00:00, 5.16s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.6820 Acc: 1.0000

100%|██████████| 104/104 [03:03<00:00, 1.77s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 1.6598 Acc: 0.5512

Epoch 3/9

----------

100%|██████████| 21/21 [01:54<00:00, 5.45s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.3342 Acc: 1.0000

100%|██████████| 104/104 [03:04<00:00, 1.78s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 1.8188 Acc: 0.5512

Epoch 4/9

----------

100%|██████████| 21/21 [01:42<00:00, 4.87s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.1919 Acc: 1.0000

100%|██████████| 104/104 [03:08<00:00, 1.81s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 1.9856 Acc: 0.5512

Epoch 5/9

----------

100%|██████████| 21/21 [02:01<00:00, 5.78s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.1488 Acc: 1.0000

100%|██████████| 104/104 [03:35<00:00, 2.07s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 1.9996 Acc: 0.5512

Epoch 6/9

----------

100%|██████████| 21/21 [02:35<00:00, 7.41s/it]

train Loss: 0.1478 Acc: 1.0000

100%|██████████| 104/104 [03:44<00:00, 2.16s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 2.0127 Acc: 0.5512

Epoch 7/9

----------

100%|██████████| 21/21 [02:19<00:00, 6.66s/it]

train Loss: 0.1426 Acc: 1.0000

100%|██████████| 104/104 [03:31<00:00, 2.03s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 2.0255 Acc: 0.5512

Epoch 8/9

----------

100%|██████████| 21/21 [02:03<00:00, 5.89s/it]

train Loss: 0.1401 Acc: 1.0000

100%|██████████| 104/104 [03:38<00:00, 2.10s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 2.0383 Acc: 0.5512

Epoch 9/9

----------

100%|██████████| 21/21 [01:56<00:00, 5.53s/it]

train Loss: 0.1338 Acc: 1.0000

100%|██████████| 104/104 [03:20<00:00, 1.93s/it]

test Loss: 2.0508 Acc: 0.5512

Training complete in 53m 51s

Best Acc: 0.551156

Process finished with exit code 0at ..\c10/core/TensorImpl.h:1156.)

return torch.max\_pool2d(input, kernel\_size, stride, padding, dilation, ceil\_mode)

100%|██████████| 21/21 [01:56<00:00, 5.55s/it]

train Loss: 2.1394 Acc: 0.5775

100%|██████████| 104/104 [03:10<00:00, 1.84s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 2.0061 Acc: 0.5512at ..\c10/core/TensorImpl.h:1156.)

return torch.max\_pool2d(input, kernel\_size, stride, padding, dilation, ceil\_mode)

100%|██████████| 21/21 [01:45<00:00, 5.02s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.1069 Acc: 0.9667

100%|██████████| 104/104 [03:06<00:00, 1.80s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 8800.7780 Acc: 0.5512

Epoch 1/9

----------

100%|██████████| 21/21 [01:43<00:00, 4.94s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:10<00:00, 1.84s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11590.8875 Acc: 0.5512

Epoch 2/9

----------

100%|██████████| 21/21 [02:07<00:00, 6.09s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:33<00:00, 2.06s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11928.6479 Acc: 0.5512

Epoch 3/9

----------

100%|██████████| 21/21 [02:10<00:00, 6.21s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:33<00:00, 2.06s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11966.0086 Acc: 0.5512

Epoch 4/9

----------

100%|██████████| 21/21 [02:09<00:00, 6.16s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:49<00:00, 2.21s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1013 Acc: 0.5512

Epoch 5/9

----------

100%|██████████| 21/21 [02:04<00:00, 5.91s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:35<00:00, 2.08s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1374 Acc: 0.5512

Epoch 6/9

----------

100%|██████████| 21/21 [02:06<00:00, 6.04s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:47<00:00, 2.19s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1377 Acc: 0.5512

Epoch 7/9

----------

100%|██████████| 21/21 [02:10<00:00, 6.22s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:26<00:00, 1.98s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1376 Acc: 0.5512

Epoch 8/9

----------

100%|██████████| 21/21 [01:43<00:00, 4.92s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [02:58<00:00, 1.72s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1377 Acc: 0.5512

Epoch 9/9

----------

100%|██████████| 21/21 [01:49<00:00, 5.21s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:30<00:00, 2.03s/it]

test Loss: 11970.1376 Acc: 0.5512

Training complete in 54m 27s

Best Acc: 0.551156

之后还尝试了调节其它参数，比如：

transforms.Normalize(

mean=[0.1307, ],

std=[0.3081, ])

----------

100%|██████████| 21/21 [02:35<00:00, 7.41s/it]

train Loss: 0.1478 Acc: 1.0000

100%|██████████| 104/104 [03:44<00:00, 2.16s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 2.0127 Acc: 0.5512

Epoch 7/9

----------

100%|██████████| 21/21 [02:19<00:00, 6.66s/it]

train Loss: 0.1426 Acc: 1.0000

100%|██████████| 104/104 [03:31<00:00, 2.03s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 2.0255 Acc: 0.5512

Epoch 8/9

----------

100%|██████████| 21/21 [02:03<00:00, 5.89s/it]

train Loss: 0.1401 Acc: 1.0000

100%|██████████| 104/104 [03:38<00:00, 2.10s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 2.0383 Acc: 0.5512

Epoch 9/9

----------

100%|██████████| 21/21 [01:56<00:00, 5.53s/it]

train Loss: 0.1338 Acc: 1.0000

100%|██████████| 104/104 [03:20<00:00, 1.93s/it]

test Loss: 2.0508 Acc: 0.5512

Training complete in 53m 51s

Best Acc: 0.551156

100%|██████████| 21/21 [01:49<00:00, 5.21s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:30<00:00, 2.03s/it]

test Loss: 11970.1376 Acc: 0.5512

Training complete in 54m 27s

Best Acc: 0.551156

transforms.Normalize(

mean=[0.1307, ],

std=[0.3081, ])

这两个数字是Mnist官方给出的数值。将其改为Imagenet的均值和标准差：

但是并没有什么明显的效果。还尝试其它的各种参数，都是如无头苍蝇一般撞，收效微乎其微，没有副作用便谢天谢地。

transforms.Normalize(

mean=[0.485, ],

std=[0.229, ])

之后便想，是否要继续调试CNN呢？我想我跑出这样烂的结果已经耗费了三四天，如果要写一个很圆满的程序，这样调试肯定是不可行的，最好是能系统地学习一下pytorch，或者机器学习？但是这样恐怕会耗费更多的时间，最后的结果如何对我来说就像冒险。因此我决定回头继续调试KNN。

### 5. 调试KNN算法

怎么调试KNN呢？算法上来看，它就只有一个参数k。因此我想到了在尝试CNN的时候有一步是图像标准化或者归一化。因此我在载入图像的时候进行了标准化：

def normalize(i):

"""

图片标准化

"""

mean = np.mean(i)

var = np.mean(np.square(i - mean))

i = (i - mean) / np.sqrt(var)

max\_i = abs(i).max()

i = i / max\_i

return i

运行结果竟然出奇得好，我惊呆了：

Loading Lwf data...

Loading is complete

Start training...

\* \* \* \* \* \* \* \* \* \*

k = 1

Validation error rate: 0.0022675736961451642

Total time: 74.44413447380066 seconds

\* \* \* \* \* \* \* \* \* \*

k = 3

Validation error rate: 0.0015117157974300355

Total time: 108.46373724937439 seconds

\* \* \* \* \* \* \* \* \* \*

k = 5

Validation error rate: 0.0015117157974300355

Total time: 102.76729249954224 seconds

\* \* \* \* \* \* \* \* \* \*

k = 7

Validation error rate: 0.0015117157974300355

Total time: 69.94577932357788 seconds

\* \* \* \* \* \* \* \* \* \*

k = 9

Validation error rate: 0.0007558578987150177

Total time: 64.555002450943 seconds

\* \* \* \* \* \* \* \* \* \*

k = 11

Validation error rate: 0.0007558578987150177

Total time: 91.70549368858337 seconds

\* \* \* \* \* \* \* \* \* \*

k = 13

Validation error rate: 0.0007558578987150177

Total time: 63.905200719833374 seconds

\* \* \* \* \* \* \* \* \* \*

k = 15

Validation error rate: 0.0

Total time: 60.181222915649414 seconds

Start testing...

\* \* \* \* \* \* \* \* \* \*

Best: KNeighborsClassifier(n\_neighbors=15)

Test error rate: 0.000755572346052169

Total time: 498.4267656803131 seconds

Process finished with exit code 0at ..\c10/core/TensorImpl.h:1156.)

return torch.max\_pool2d(input, kernel\_size, stride, padding, dilation, ceil\_mode)

100%|██████████| 21/21 [01:56<00:00, 5.55s/it]

train Loss: 2.1394 Acc: 0.5775

100%|██████████| 104/104 [03:10<00:00, 1.84s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 2.0061 Acc: 0.5512

Epoch 1/9

----------

100%|██████████| 21/21 [01:58<00:00, 5.62s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 1.3775 Acc: 1.0000

100%|██████████| 104/104 [03:15<00:00, 1.88s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 1.6796 Acc: 0.5512

Epoch 2/9

----------

100%|██████████| 21/21 [01:48<00:00, 5.16s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.6820 Acc: 1.0000

100%|██████████| 104/104 [03:03<00:00, 1.77s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 1.6598 Acc: 0.5512

Epoch 3/9

----------

100%|██████████| 21/21 [01:54<00:00, 5.45s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.3342 Acc: 1.0000

100%|██████████| 104/104 [03:04<00:00, 1.78s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 1.8188 Acc: 0.5512

Epoch 4/9

----------

100%|██████████| 21/21 [01:42<00:00, 4.87s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.1919 Acc: 1.0000

100%|██████████| 104/104 [03:08<00:00, 1.81s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 1.9856 Acc: 0.5512

Epoch 5/9

----------

100%|██████████| 21/21 [02:01<00:00, 5.78s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.1488 Acc: 1.0000

100%|██████████| 104/104 [03:35<00:00, 2.07s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 1.9996 Acc: 0.5512

Epoch 6/9

----------

100%|██████████| 21/21 [02:35<00:00, 7.41s/it]

train Loss: 0.1478 Acc: 1.0000

100%|██████████| 104/104 [03:44<00:00, 2.16s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 2.0127 Acc: 0.5512

Epoch 7/9

----------

100%|██████████| 21/21 [02:19<00:00, 6.66s/it]

train Loss: 0.1426 Acc: 1.0000

100%|██████████| 104/104 [03:31<00:00, 2.03s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 2.0255 Acc: 0.5512

Epoch 8/9

----------

100%|██████████| 21/21 [02:03<00:00, 5.89s/it]

train Loss: 0.1401 Acc: 1.0000

100%|██████████| 104/104 [03:38<00:00, 2.10s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 2.0383 Acc: 0.5512

Epoch 9/9

----------

100%|██████████| 21/21 [01:56<00:00, 5.53s/it]

train Loss: 0.1338 Acc: 1.0000

100%|██████████| 104/104 [03:20<00:00, 1.93s/it]

test Loss: 2.0508 Acc: 0.5512

Training complete in 53m 51s

Best Acc: 0.551156

Process finished with exit code 0at ..\c10/core/TensorImpl.h:1156.)

return torch.max\_pool2d(input, kernel\_size, stride, padding, dilation, ceil\_mode)

100%|██████████| 21/21 [01:56<00:00, 5.55s/it]

train Loss: 2.1394 Acc: 0.5775

100%|██████████| 104/104 [03:10<00:00, 1.84s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 2.0061 Acc: 0.5512at ..\c10/core/TensorImpl.h:1156.)

return torch.max\_pool2d(input, kernel\_size, stride, padding, dilation, ceil\_mode)

100%|██████████| 21/21 [01:45<00:00, 5.02s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.1069 Acc: 0.9667

100%|██████████| 104/104 [03:06<00:00, 1.80s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 8800.7780 Acc: 0.5512

Epoch 1/9

----------

100%|██████████| 21/21 [01:43<00:00, 4.94s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:10<00:00, 1.84s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11590.8875 Acc: 0.5512

Epoch 2/9

----------

100%|██████████| 21/21 [02:07<00:00, 6.09s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:33<00:00, 2.06s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11928.6479 Acc: 0.5512

Epoch 3/9

----------

100%|██████████| 21/21 [02:10<00:00, 6.21s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:33<00:00, 2.06s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11966.0086 Acc: 0.5512

Epoch 4/9

----------

100%|██████████| 21/21 [02:09<00:00, 6.16s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:49<00:00, 2.21s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1013 Acc: 0.5512

Epoch 5/9

----------

100%|██████████| 21/21 [02:04<00:00, 5.91s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:35<00:00, 2.08s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1374 Acc: 0.5512

Epoch 6/9

----------

100%|██████████| 21/21 [02:06<00:00, 6.04s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:47<00:00, 2.19s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1377 Acc: 0.5512

Epoch 7/9

----------

100%|██████████| 21/21 [02:10<00:00, 6.22s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:26<00:00, 1.98s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1376 Acc: 0.5512

Epoch 8/9

----------

100%|██████████| 21/21 [01:43<00:00, 4.92s/it]

0%| | 0/104 [00:00<?, ?it/s]train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [02:58<00:00, 1.72s/it]

0%| | 0/21 [00:00<?, ?it/s]test Loss: 11970.1377 Acc: 0.5512

Epoch 9/9

----------

100%|██████████| 21/21 [01:49<00:00, 5.21s/it]

train Loss: 0.0000 Acc: 1.0000

100%|██████████| 104/104 [03:30<00:00, 2.03s/it]

test Loss: 11970.1376 Acc: 0.5512

Training complete in 54m 27s

Best Acc: 0.551156

检查代码，发现我只对女性图片进行了这样的操作，男性忘加了。但是如果对男女都进行标准化的操作，效果反而不好。准确率反而回到了80%左右。

Total time: 108.46373724937439 seconds

\* \* \* \* \* \* \* \* \* \*

k = 5

Validation error rate: 0.0015117157974300355

Total time: 102.76729249954224 seconds

\* \* \* \* \* \* \* \* \* \*

k = 7

Validation error rate: 0.0015117157974300355

Total time: 69.94577932357788 seconds

\* \* \* \* \* \* \* \* \* \*

k = 9

Validation error rate: 0.0007558578987150177

Total time: 64.555002450943 seconds

\* \* \* \* \* \* \* \* \* \*

k = 11

Validation error rate: 0.0007558578987150177

Total time: 91.70549368858337 seconds

\* \* \* \* \* \* \* \* \* \*

k = 13

Validation error rate: 0.0007558578987150177

Total time: 63.905200719833374 seconds

\* \* \* \* \* \* \* \* \* \*

k = 15

Validation error rate: 0.0

Total time: 60.181222915649414 seconds

Start testing...

\* \* \* \* \* \* \* \* \* \*

Best: KNeighborsClassifier(n\_neighbors=15)

Test error rate: 0.000755572346052169

Total time: 498.4267656803131 seconds

为什么只对女性进行操作可以提高准确率呢？我一开始想会不会是因为女性图片比较复杂呢？应该不是，对于机器来说都是同样复杂的呀。

后来我和一位同学聊天的时候他启发了我。他跟我说KNN不行，不是他打击我。我想了想，确实啊，KNN怎么可能这么优秀呢？而后我想明白了，只对女性做标准化，相当于人为把两份样本区分开了！看来还是得好好学习一下才行！

### 6. 调试CNN

首先我在网上找到了PyTorch的教程，总算知道了各个参数的含义，以及网络搭建的过程以及反馈机制。

重新改写了某些参数，比如：

* 使用三通道
* 最后线性化要变成2个…（我以为这点很重要，但没有）
* 增加了padding

但是结果依然是令人绝望的0.5512…

Epoch 0/9

----------

0%| | 0/83 [00:00<?, ?it/s]C:\Users\liyutong\anaconda3\lib\site-packages\torch\nn\functional.py:718: UserWarning: Named tensors and all their associated APIs are an experimental feature and subject to change. Please do not use them for anything important until they are released as stable. (Triggered internally at ..\c10/core/TensorImpl.h:1156.)

return torch.max\_pool2d(input, kernel\_size, stride, padding, dilation, ceil\_mode)

100%|██████████| 83/83 [09:45<00:00, 7.05s/it]

train Loss: 1.3991 Acc: 0.8755

100%|██████████| 104/104 [06:59<00:00, 4.03s/it]

test Loss: 2.3391 Acc: 0.5512

Epoch 1/9

----------

100%|██████████| 83/83 [09:42<00:00, 7.02s/it]

train Loss: 0.3446 Acc: 0.9987

100%|██████████| 104/104 [04:36<00:00, 2.66s/it]

test Loss: 3.1320 Acc: 0.5512

Epoch 2/9

----------

100%|██████████| 83/83 [09:17<00:00, 6.71s/it]

train Loss: 0.1638 Acc: 0.9996

100%|██████████| 104/104 [05:03<00:00, 2.91s/it]

test Loss: 3.7110 Acc: 0.5512

私以为是因为过拟合造成的，但是之前增加dropout没有对结果造成任何影响。这次增加一层卷积层试试呢？截取前两个EPOCH：

Loading data...

EPOCH:1/10

训练次数：10, Loss:0.7115321755409241

训练次数：20, Loss:0.6890419125556946

训练次数：30, Loss:0.6619747281074524

训练次数：40, Loss:0.6307131052017212

训练次数：50, Loss:0.5940333604812622

训练次数：60, Loss:0.5543129444122314

训练次数：70, Loss:0.5363575220108032

训练次数：80, Loss:0.5016556978225708

整体测试集上的Loss:72.6392268538475

整体测试集上的正确率：0.5516095161437988

Saved!

EPOCH:2/10

训练次数：90, Loss:0.4595687985420227

训练次数：100, Loss:0.4364284873008728

训练次数：110, Loss:0.4031129777431488

训练次数：120, Loss:0.38502058386802673

训练次数：130, Loss:0.35639312863349915

训练次数：140, Loss:0.34936395287513733

训练次数：150, Loss:0.3039506673812866

训练次数：160, Loss:0.3279275596141815

整体测试集上的Loss:81.79509449005127

整体测试集上的正确率：0.5511561036109924

Saved!

EPOCH:3/10

训练次数：170, Loss:0.3070454001426697

训练次数：180, Loss:0.28160035610198975

训练次数：190, Loss:0.2696661651134491

训练次数：200, Loss:0.2387031614780426

训练次数：210, Loss:0.23376138508319855

训练次数：220, Loss:0.2437240481376648

训练次数：230, Loss:0.2062584012746811

训练次数：240, Loss:0.21605455875396729

整体测试集上的Loss:94.5999944806099

整体测试集上的正确率：0.5511561036109924

Saved!

EPOCH:4/10

训练次数：250, Loss:0.23431824147701263

训练次数：260, Loss:0.1606861650943756

训练次数：270, Loss:0.17231033742427826

训练次数：280, Loss:0.16667398810386658

训练次数：290, Loss:0.16859933733940125

训练次数：300, Loss:0.1538567841053009

训练次数：310, Loss:0.14209997653961182

训练次数：320, Loss:0.14273601770401

训练次数：330, Loss:0.14827945828437805

整体测试集上的Loss:109.1605179309845

整体测试集上的正确率：0.5511561036109924

Saved!

EPOCH:5/10

训练次数：340, Loss:0.15169040858745575

训练次数：350, Loss:0.12080876529216766

训练次数：360, Loss:0.12370860576629639

训练次数：370, Loss:0.12408115714788437

训练次数：380, Loss:0.1285608410835266

训练次数：390, Loss:0.12507322430610657

训练次数：400, Loss:0.12475623935461044

训练次数：410, Loss:0.11311765015125275

整体测试集上的Loss:123.06934529542923

整体测试集上的正确率：0.5511561036109924

Saved!

终于，不再是那个0.5512了。变成0.5516了！起码知道它能变了…

之后我又加了些东西，但是出现了Loss一直是0.69左右的情况，与此同时准确率在50%，这不就是随便猜么。并且0.69=-ln(0.5)，很有文章，说明什么都没有学到。把学习率调高，又回到了0.5516。做了许多尝试，都不行。以下只摘录有明显变化的步骤：

* Dropout

由于准确率一直不变，且train loss非常小，猜测是过拟合。在卷积层后加dropout，就算到达80%也没有太大改观。考虑在第一层卷积后加上，颇有些效果。以下是运行结果：

此后又调整了学习率，效果已经非常好了，呜呜呜好感动！！！

EPOCH:1/10

整体测试集上的正确率：0.4278428554534912

EPOCH:2/10

整体测试集上的正确率：0.6331696510314941

EPOCH:3/10

整体测试集上的正确率：0.5434454083442688

EPOCH:4/10

整体测试集上的正确率：0.5277673006057739

EPOCH:5/10

整体测试集上的正确率：0.44673216342926025

EPOCH:6/10

整体测试集上的正确率：0.5536456108093262

EPOCH:7/10

整体测试集上的正确率：0.5032112002372742

EPOCH:8/10

整体测试集上的正确率：0.6514922380447388

EPOCH:9/10

整体测试集上的正确率：0.7174159288406372

EPOCH:10/10

整体测试集上的正确率：0.6786928772926331

Loading data...

EPOCH:1/10

Train Loss:0.6537600755691528

Train Acc:0.16346153616905212

Test Loss:51.883450388908386

Test Acc:0.763883650302887

EPOCH:2/10

Train Loss:0.7031839489936829

Train Acc:0.11538461595773697

Test Loss:50.86209851503372

Test Acc:0.8571968078613281

EPOCH:3/10

Train Loss:0.731357753276825

Train Acc:0.08653846383094788

Test Loss:49.14549112319946

Test Acc:0.9578768610954285

EPOCH:4/10

Train Loss:0.7010369896888733

Train Acc:0.11538461595773697

Test Loss:49.97032576799393

Test Acc:0.9852663278579712

EPOCH:5/10

Train Loss:0.6645208597183228

Train Acc:0.1538461595773697

Test Loss:49.03929138183594

Test Acc:0.9926331639289856

EPOCH:6/10

Train Loss:0.67194664478302

Train Acc:0.16346153616905212

Test Loss:50.09390050172806

Test Acc:0.9761994481086731

EPOCH:7/10

Train Loss:0.6581960320472717

Train Acc:0.17307692766189575

Test Loss:50.115226328372955

Test Acc:0.9909331202507019

EPOCH:8/10

Train Loss:0.6765804290771484

Train Acc:0.14423076808452606

Test Loss:47.85217642784119

Test Acc:0.996977686882019

EPOCH:9/10

Train Loss:0.6629800200462341

Train Acc:0.14423076808452606

Test Loss:48.80183082818985

Test Acc:0.9952776432037354

EPOCH:10/10

Train Loss:0.7413228750228882

Train Acc:0.06730769574642181

Test Loss:50.42632430791855

Test Acc:0.9459765553474426

随后我设计了GUI界面，先添加了一个功能：随机在数据集中抽取一张照片，用训练好的模型来输出结果，我发现这个正确率差得离谱，根本达不到90%；并且原程序与此同时出现了正确率不变的情况，甚至逼近100%，我觉得有些不对。经过调试与检查，发现了一个致命的错误：我只打乱了图片，但是没有打乱其相应的label！！！那我之前都在训练啥？？？赶紧更正，并且又经历了一番调试。经过GUI的验证，终于确定没什么太大的问题了。**最后的模型及结果见下两个部分。**

Test Loss:47.85217642784119

Test Acc:0.996977686882019

EPOCH:9/10

Train Loss:0.6629800200462341

Train Acc:0.14423076808452606

Test Loss:48.80183082818985

Test Acc:0.9952776432037354

EPOCH:10/10

Train Loss:0.7413228750228882

Train Acc:0.06730769574642181

Test Loss:50.42632430791855

Test Acc:0.9459765553474426

GUI采取python自带的tkinter。可以实现两个功能：

1. 从数据集中随机抽取一张图片，显示人名及判断的性别结果
2. 使用者输入一张数据集中的图片名称，若存在，显示人名及判断的性别结果

其中需要定义一个小的SimpleDataset，继承Dataset，只含有一张图片。

# 三、最终模型

使用CNN算法。

加载数据时加载全部数据，并将图片进行变换：中心裁剪为150\*150，这样有助于学习（排除干扰），并且有利于加快训练速度；标准化；转化为Tensor数据类型。将数据集划分为训练集和测试集，比例为4：6，不设置验证集。每个数据都有图片信息和标签信息，0代表女性，1代表男性。

整个网络包括以下层：

* 卷积层。

输入通道：3

输出通道：64

卷积核大小：5\*5

卷积步长：1

填充：至相同大小

* 激活函数：线性整流函数
* 池化：最大池化

池化核：2\*2

池化步长：2

* Dropout层，比例0.5
* 卷积层。

输入通道：64

输出通道：128

卷积核大小：5\*3

卷积步长：1

填充：至相同大小

* 激活函数：线性整流函数
* 池化：最大池化

池化核：2\*2

池化步长：2

* Dropout层，比例0.25
* Flatten层
* 线性层：175232至64
* Dropout层，比例0.25
* 线性层：175232至2

超参数设置：

BATCH\_SIZE = 64

学习率为0.1，每5个epoch，学习率变为之前的0.1

训练轮数EPOCH = 20

损失函数：交叉熵损失函数

优化器：随机梯度下降法

环境：

Python3.8.8

pytorch1.9.0

torchvision0.10.0

在cpu上训练

# 四、结果与分析

最后一次运行结果：

Loading data...

Epoch 1/20

train Loss: 0.6145 Acc: 0.7682

test Loss: 0.4485 Acc: 0.7906

----------

Epoch 2/20

train Loss: 0.4313 Acc: 0.8107

test Loss: 0.3655 Acc: 0.8299

----------

Epoch 3/20

train Loss: 0.3476 Acc: 0.8475

test Loss: 0.3571 Acc: 0.8703

----------

Epoch 4/20

train Loss: 0.2811 Acc: 0.8889

test Loss: 0.2556 Acc: 0.9035

----------

Epoch 5/20

train Loss: 0.2141 Acc: 0.9150

test Loss: 0.2276 Acc: 0.9084

----------

Epoch 6/20

train Loss: 0.1929 Acc: 0.9286

test Loss: 0.2022 Acc: 0.9229

----------

Epoch 7/20

train Loss: 0.1601 Acc: 0.9397

test Loss: 0.1960 Acc: 0.9292

----------

Epoch 8/20

train Loss: 0.1415 Acc: 0.9431

test Loss: 0.1885 Acc: 0.9317

----------

Epoch 9/20

train Loss: 0.1174 Acc: 0.9547

test Loss: 0.2017 Acc: 0.9263

----------

Epoch 10/20

train Loss: 0.1108 Acc: 0.9581

test Loss: 0.1858 Acc: 0.9336

----------

Epoch 11/20

train Loss: 0.0817 Acc: 0.9707

test Loss: 0.2021 Acc: 0.9338

----------

Epoch 12/20

train Loss: 0.0869 Acc: 0.9677

test Loss: 0.4169 Acc: 0.8464

----------

Epoch 13/20

train Loss: 0.0850 Acc: 0.9715

test Loss: 0.2018 Acc: 0.9407

----------

Epoch 14/20

train Loss: 0.0938 Acc: 0.9671

test Loss: 0.2117 Acc: 0.9290

----------

Epoch 15/20

train Loss: 0.0636 Acc: 0.9747

test Loss: 0.2023 Acc: 0.9406

----------

Epoch 16/20

train Loss: 0.0441 Acc: 0.9834

test Loss: 0.2174 Acc: 0.9432

----------

Epoch 17/20

train Loss: 0.0505 Acc: 0.9828

test Loss: 0.2116 Acc: 0.9440

----------

Epoch 18/20

train Loss: 0.0419 Acc: 0.9849

test Loss: 0.2204 Acc: 0.9377

----------

Epoch 19/20

train Loss: 0.0294 Acc: 0.9913

test Loss: 0.2442 Acc: 0.9431

----------

Epoch 20/20

train Loss: 0.0249 Acc: 0.9904

test Loss: 0.2527 Acc: 0.9417

Best Acc: 0.943962

Epoch 9/20

train Loss: 0.1174 Acc: 0.9547

test Loss: 0.2017 Acc: 0.9263

----------

Epoch 10/20

train Loss: 0.1108 Acc: 0.9581

test Loss: 0.1858 Acc: 0.9336

----------

Epoch 11/20

train Loss: 0.0817 Acc: 0.9707

test Loss: 0.2021 Acc: 0.9338

----------

Epoch 12/20

train Loss: 0.0869 Acc: 0.9677

test Loss: 0.4169 Acc: 0.8464

----------

Epoch 13/20

train Loss: 0.0850 Acc: 0.9715

test Loss: 0.2018 Acc: 0.9407

----------

Epoch 14/20

train Loss: 0.0938 Acc: 0.9671

test Loss: 0.2117 Acc: 0.9290

----------

Epoch 15/20

train Loss: 0.0636 Acc: 0.9747

test Loss: 0.2023 Acc: 0.9406

----------

Epoch 16/20

train Loss: 0.0441 Acc: 0.9834

test Loss: 0.2174 Acc: 0.9432

----------

Epoch 17/20

train Loss: 0.0505 Acc: 0.9828

test Loss: 0.2116 Acc: 0.9440

----------

Epoch 18/20

train Loss: 0.0419 Acc: 0.9849

test Loss: 0.2204 Acc: 0.9377

----------

Epoch 19/20

train Loss: 0.0294 Acc: 0.9913

test Loss: 0.2442 Acc: 0.9431

----------

Epoch 20/20

train Loss: 0.0249 Acc: 0.9904

test Loss: 0.2527 Acc: 0.9417

Best Acc: 0.943962

可以看出train loss不断下降、train acc不断上升；test loss没有明显的变化规律，但是test acc上升后稳定在0.94左右。

test Loss: 0.2204 Acc: 0.9377

----------

Epoch 19/20

train Loss: 0.0294 Acc: 0.9913

test Loss: 0.2442 Acc: 0.9431

----------

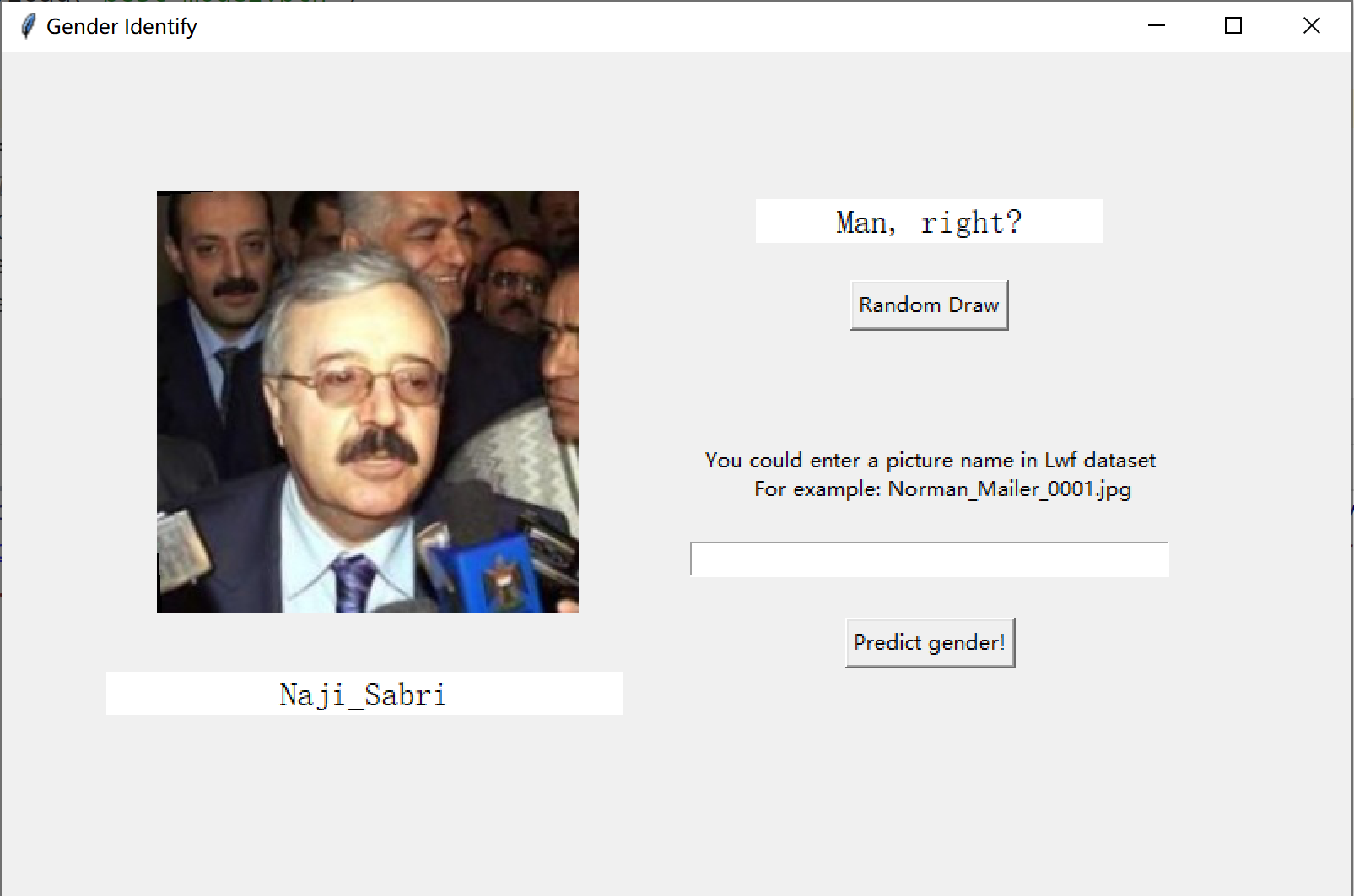
Epoch 20/20

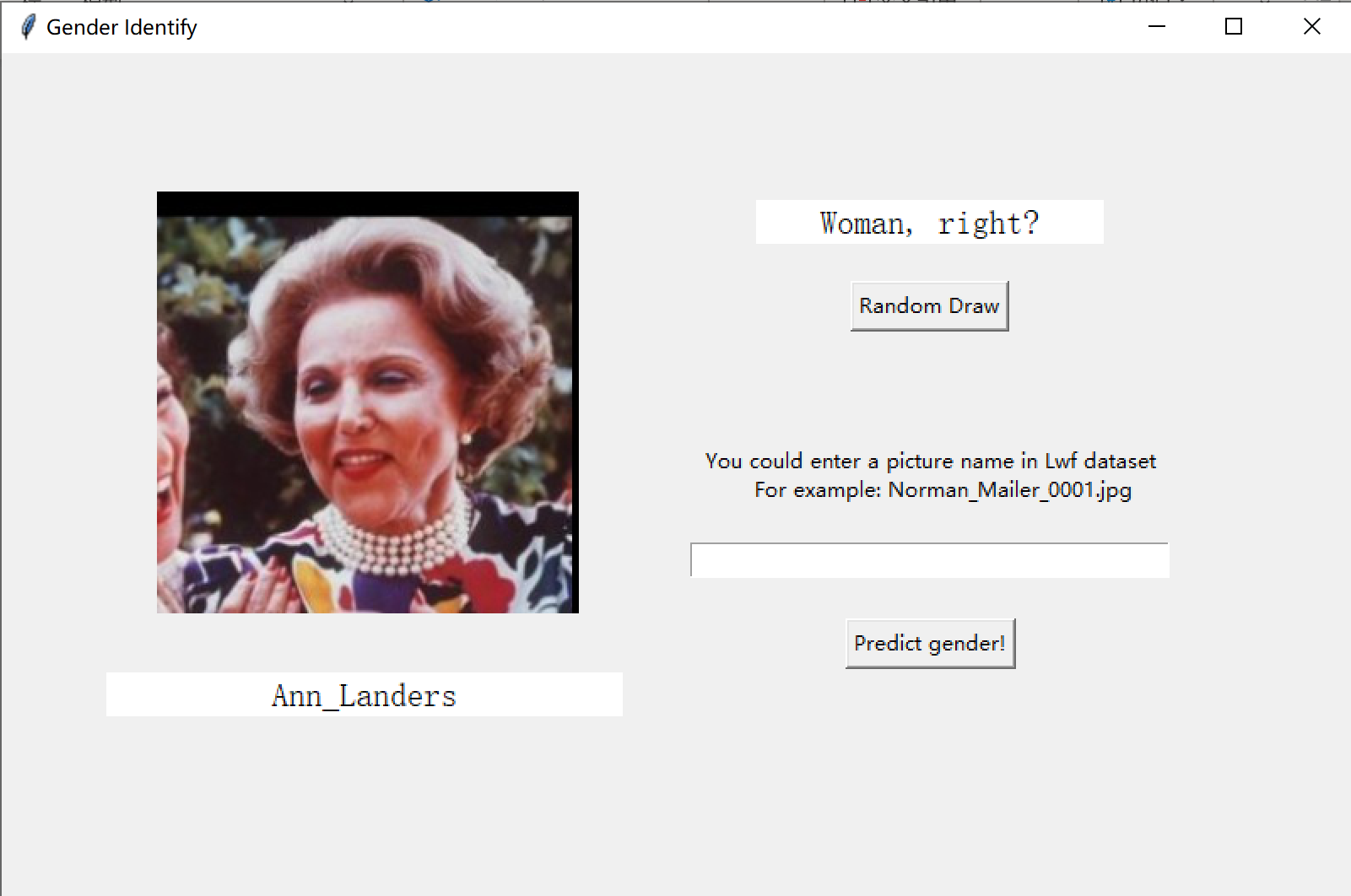
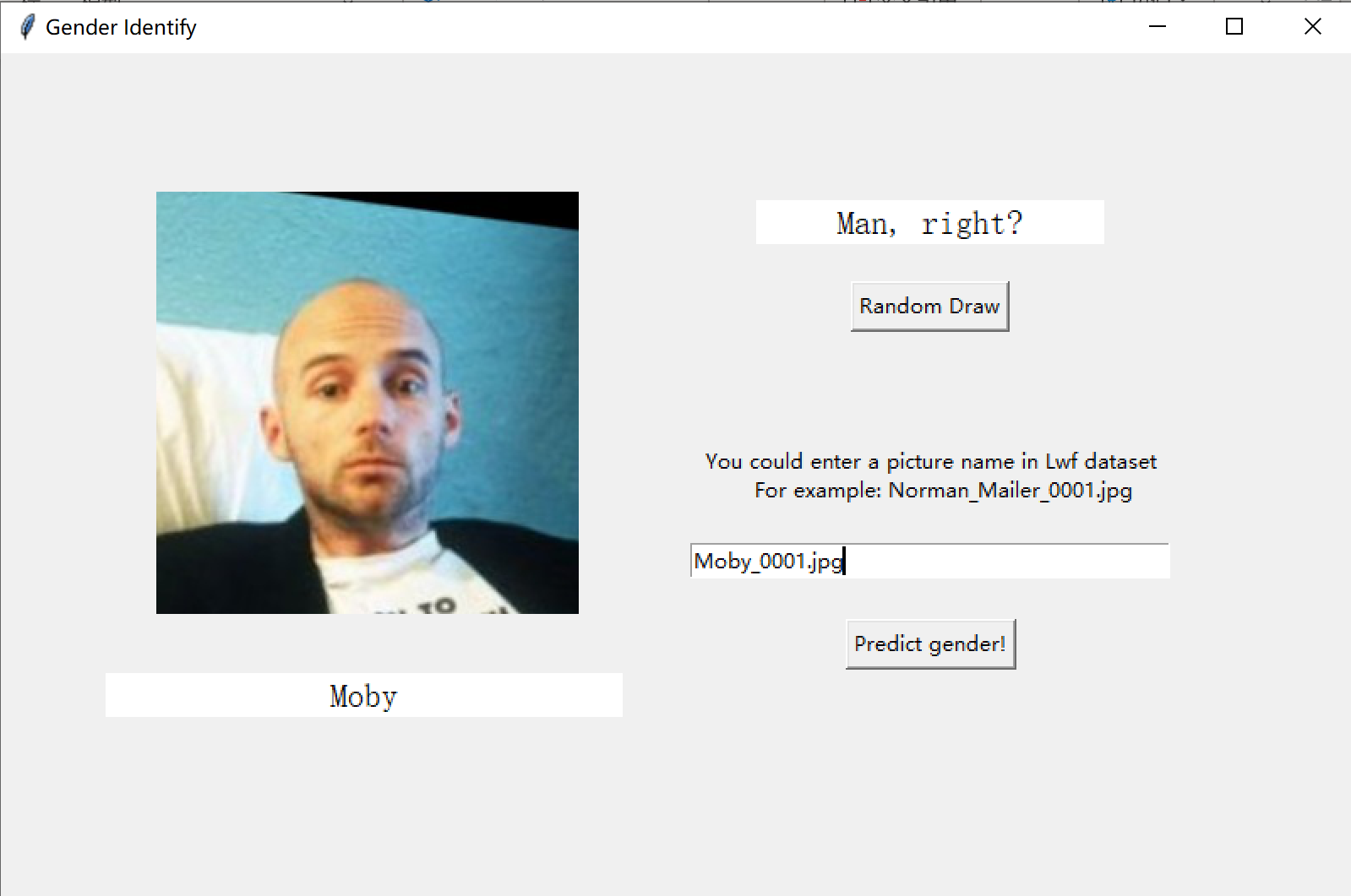
train Loss: 0.0249 Acc: 0.9904

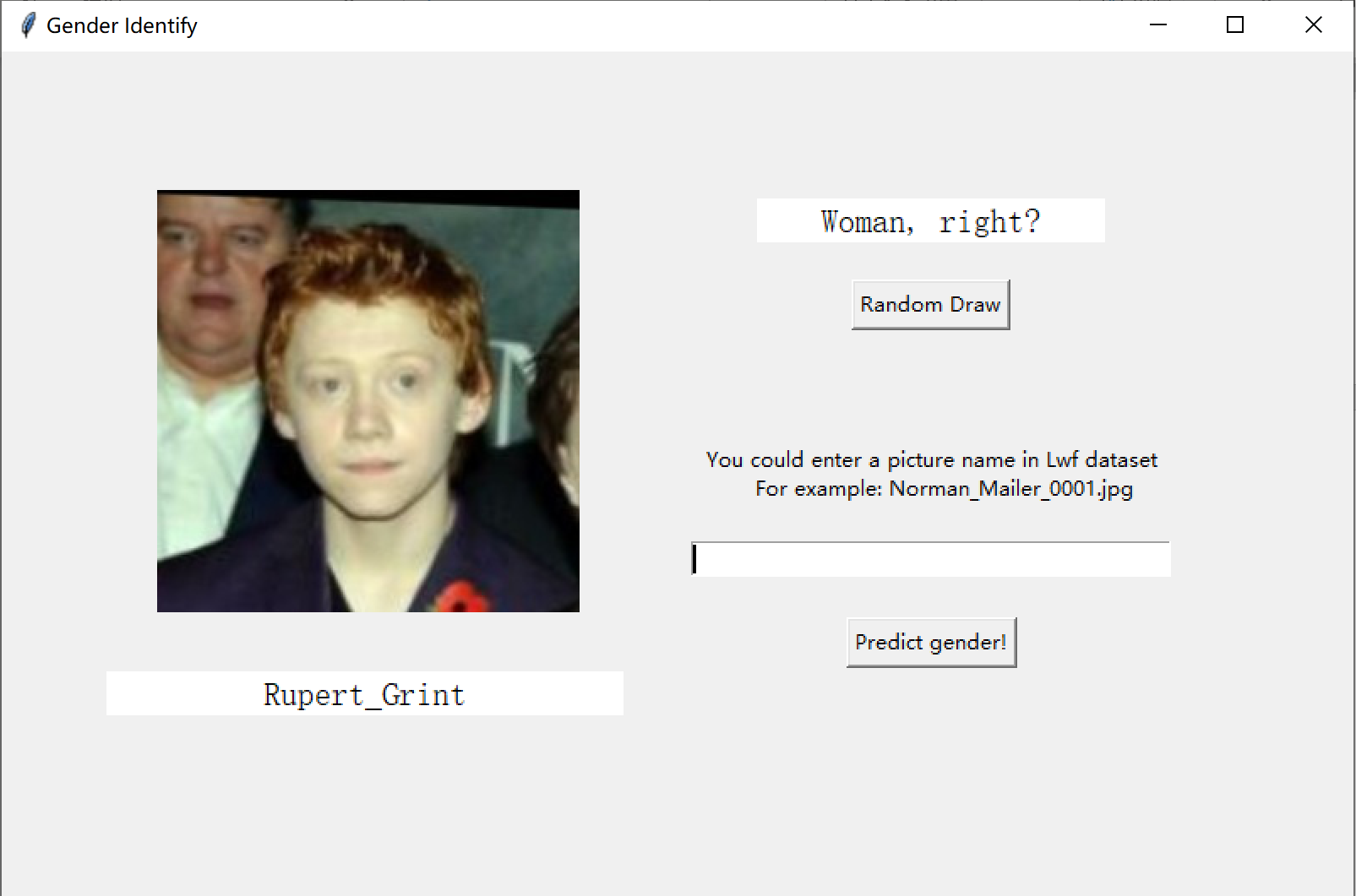
test Loss: 0.2527 Acc: 0.9417

Best Acc: 0.943962

运行时间很长，这20轮下来大约有5个小时，这点是应当改进的。

GUI界面：

也可以通过输入：

偶尔也会有错误：

# 五、源代码

*# LwfDataset2.py  
# 继承Dataset类，用于加载数据***import** os  
**import** random  
**from** torch.utils.data **import** Dataset  
**from** PIL **import** Image  
  
**'''生成包含所有名字的文件'''**f\_f = open(**"female\_names.txt"**) *# 打开女名文件*f\_name = f\_f.readlines() *# 将女名读入list*f\_f.close() *# 关闭文件*m\_f = open(**"male\_names.txt"**) *# 打开男名文件*m\_name = m\_f.readlines() *# 将男名读入list*m\_f.close() *# 关闭文件*f = open(**"names.txt"**, **'w'**)  
**for** name **in** f\_name:  
 f.write(name)  
**for** name **in** m\_name: *# 临时修改* f.write(name)  
f.close()  
  
  
**class** LwfDataset(Dataset):  
  
 **def** \_\_init\_\_(self, root\_dir, names\_file, transform=**None**, train=**True**, test=**False**):  
 *"""* **:param** *root\_dir: 所在文件夹* **:param** *names\_file: 包含名字的文件* **:param** *transform: 图片变换* **:param** *train: 是否是训练集* **:param** *test: 是否是测试集  
 """* **'''参数'''** self.root\_dir = root\_dir  
 self.names\_file = names\_file  
 self.transform = transform  
 self.train = train  
 self.test = test  
  
 **'''其它属性'''** self.names\_list = [] *# 名字列表* self.images\_list = [] *# 图片列表* **"""从包含名字的文件中获取名字列表"""  
 if not** os.path.isfile(self.names\_file): *# 确保文件存在* print(self.names\_file + **'does not exist!'**)  
 file = open(self.names\_file)  
 **for** name\_file **in** file:  
 fi = name\_file[:-1] *# 去掉最后的回车* self.names\_list.append(fi)  
 file.close()  
 random.shuffle(self.names\_list) *# 打乱名字列表* **"""把图片读入图片列表"""  
 for** name **in** self.names\_list:  
 image\_path = self.root\_dir + **'/'** + name[:-9] + **'/'** + name  
 **if** image\_path != **'lfw\_funneled//'**: *# 防止读到空行* image = Image.open(image\_path) *# 用自带的Image读取图片* **if** self.transform:  
 image = self.transform(image) *# 图片变换* self.images\_list.append(image) *# 添加图片到列表， 此时名字列表和图片列表均随机，且一一对应* **"""进行数据集的划分"""** self.size = self.\_\_len\_\_()  
 **if** self.train:  
 self.images\_list = self.images\_list[:int(0.4 \* self.size)] *# 训练集* self.names\_list = self.names\_list[:int(0.4 \* self.size)]  
 self.size = int(self.size \* 0.5)  
 **elif** self.test:  
 self.images\_list = self.images\_list[int(0.4 \* self.size):] *# 测试集* self.names\_list = self.names\_list[int(0.4 \* self.size):]  
 self.size = int(self.size \* 0.1)  
  
 **def** \_\_getitem\_\_(self, index):  
 *"""* **:param** *index: 图片下标* **:return***: 图片数据，包括图片本身和标签  
 """* image = self.images\_list[index]  
  
 **if** self.names\_list[index] + **'\n' in** f\_name:  
 label = 0  
 **elif** self.names\_list[index] + **'\n' in** m\_name:  
 label = 1  
 **else**:  
 print(**"No such name!"**)  
 label = 2  
  
 **return** image, label  
  
 **def** \_\_len\_\_(self):  
 *"""* **:return***: 数据集中所有图片个数  
 """* **return** len(self.images\_list)

*# YutongNet2.py  
# 搭建网络***from** torch **import** nn  
  
  
**class** YutongNet(nn.Module):  
 **def** \_\_init\_\_(self):  
 super().\_\_init\_\_() *# 父类初始化* self.layer = nn.Sequential(  
 nn.Conv2d(in\_channels=3, out\_channels=64,  
 kernel\_size=(5, 5), stride=(1, 1), padding=**'same'**),  
 nn.ReLU(),  
 nn.MaxPool2d(kernel\_size=(2, 2), stride=2),  
  
 nn.Dropout(0.5),  
 nn.Conv2d(in\_channels=64, out\_channels=128,  
 kernel\_size=(3, 3), stride=(1, 1), padding=**'same'**),  
 nn.ReLU(),  
 nn.MaxPool2d(kernel\_size=(2, 2), stride=2),  
  
 nn.Dropout(0.25),  
 nn.Flatten(),  
 nn.Linear(175232, 64),  
  
 nn.Dropout(0.25),  
 nn.Linear(64, 2),  
 )  
  
 **def** forward(self, x):  
  
 x = self.layer(x)  
 **return** x

*# Gender\_identify3.3.py  
# 主程序***import** torch  
**from** torch.optim **import** lr\_scheduler  
**from** torchvision **import** transforms  
**from** torch.utils.data **import** DataLoader  
**from** torch **import** nn, optim  
  
**from** LwfDataset2 **import** LwfDataset  
**from** YutongNet2 **import** YutongNet  
  
**'''设置超参数'''**BATCH\_SIZE = 64 *# 每次选取64张*LR = 0.1  
EPOCH = 20 *# 训练轮数*device = torch.device(**'cuda' if** torch.cuda.is\_available() **else 'cpu'**) *# 并没有英伟达的显卡qwq***'''构建训练集和测试集'''**train\_data = LwfDataset(root\_dir=**'lfw\_funneled'**,  
 names\_file=**'names.txt'**,  
 transform=transforms.Compose([  
 *# transforms.Resize((128, 128)),  
 # transforms.RandomHorizontalFlip(p=0.5),  
 # transforms.Pad(10),  
 # transforms.RandomCrop((100, 100)),* transforms.CenterCrop(150), *# 裁剪到大脸* transforms.ToTensor(),  
 transforms.Normalize(  
 mean=[0.5, 0.5, 0.5],  
 std=[0.5, 0.5, 0.5]),  
 ]), train=**True**, test=**False**)  
  
test\_data = LwfDataset(root\_dir=**'lfw\_funneled'**,  
 names\_file=**'names.txt'**,  
 transform=transforms.Compose([  
 *# transforms.Resize((128, 128)),  
 # transforms.RandomHorizontalFlip(p=0.5),  
 # transforms.Pad(10),  
 # transforms.RandomCrop((100, 100)),* transforms.CenterCrop(150), *# 裁剪到大脸* transforms.ToTensor(),  
 transforms.Normalize(  
 mean=[0.5, 0.5, 0.5],  
 std=[0.5, 0.5, 0.5]),  
 ]), train=**False**, test=**True**)  
  
**'''加载训练集和测试集'''**print(**"Loading data..."**)  
train\_loader = DataLoader(train\_data, batch\_size=BATCH\_SIZE,  
 shuffle=**True**, num\_workers=1)  
test\_loader = DataLoader(test\_data, batch\_size=BATCH\_SIZE,  
 shuffle=**True**, num\_workers=1)  
  
*# 网络模型实例化*model = YutongNet().to(device)  
  
*# 交叉熵损失函数*loss\_func = nn.CrossEntropyLoss()  
  
*# 随机梯度下降法*optimizer\_ft = optim.SGD(model.parameters(), lr=LR)  
  
*# 定义每5个epoch，学习率变为之前的0.1*exp\_lr\_scheduler = lr\_scheduler.StepLR(optimizer\_ft, step\_size=5)  
  
  
**def** train\_model(m\_model, criterion, optimizer):  
 *"""  
 训练神经网络* **:param** *m\_model: 模型* **:param** *criterion: 损失函数* **:param** *optimizer: 优化器* **:return***: 训练后的模型  
 """* m\_model.train()  
  
 running\_loss = 0.0  
 running\_corrects = 0  
  
 **for** inputs, labels **in** train\_loader:  
 inputs = inputs.to(device)  
 labels = labels.to(device)  
  
 optimizer.zero\_grad()  
 outputs = m\_model(inputs)  
 \_, preds = torch.max(outputs, 1)  
 loss = criterion(outputs, labels)  
  
 loss.backward()  
 optimizer.step()  
  
 running\_loss += loss.item() \* inputs.size(0)  
 running\_corrects += torch.sum(preds == labels.data)  
  
 epoch\_loss = running\_loss / len(train\_data)  
 m\_epoch\_acc = float(running\_corrects) / len(train\_data)  
  
 print(**'train Loss: {:.4f} Acc: {:.4f}'**.format(  
 epoch\_loss, m\_epoch\_acc))  
  
 **return** m\_model  
  
  
**def** test\_model(m\_model, criterion):  
 *"""  
 测试神经网络* **:param** *m\_model: 模型* **:param** *criterion: 损失函数* **:return***: 测试准确率  
 """* m\_model.eval()  
 running\_loss = 0.0  
 running\_corrects = 0  
 **with** torch.no\_grad():  
 **for** inputs, labels **in** test\_loader:  
 inputs = inputs.to(device)  
 labels = labels.to(device)  
  
 outputs = m\_model(inputs)  
 \_, preds = torch.max(outputs, 1)  
 loss = criterion(outputs, labels)  
  
 running\_loss += loss.item() \* inputs.size(0)  
 running\_corrects += torch.sum(preds == labels.data)  
  
 epoch\_loss = running\_loss / len(test\_data)  
 m\_epoch\_acc = float(running\_corrects) / len(test\_data)  
  
 print(**'test Loss: {:.4f} Acc: {:.4f}'**.format(  
 epoch\_loss, m\_epoch\_acc))  
 **return** m\_epoch\_acc  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
  
 best\_acc = 0  
 **for** epoch **in** range(EPOCH):  
 print(**'-'** \* 10)  
 print(**'Epoch {}/{}'**.format(epoch + 1, EPOCH))  
  
 model = train\_model(model, loss\_func, optimizer\_ft)  
 epoch\_acc = test\_model(model, loss\_func)  
 **if** epoch\_acc > best\_acc: *# 保存训练好的模型，以供GUI使用* torch.save(model, **"best\_model.pth"**)  
 best\_acc = epoch\_acc **if** epoch\_acc > best\_acc **else** best\_acc *# 最佳正确率* print(**'Best Acc: {:4f}'**.format(best\_acc))

*# Gender\_identify\_gui.py  
# 图形化界面***import** torch  
**from** tkinter **import** \*  
**from** random **import** choice  
**from** PIL **import** Image, ImageTk  
**from** torch.utils.data **import** Dataset, DataLoader  
**from** torchvision **import** transforms  
**import** os  
  
**from** YutongNet2 **import** YutongNet  
  
*# 获取名字的集合*f = open(**"names.txt"**)  
names = f.readlines()  
f.close()  
  
*# 分别获取男女名单，用于标注label*f\_f = open(**"female\_names.txt"**) *# 打开女名文件*f\_name = f\_f.readlines() *# 将女名读入list*f\_f.close() *# 关闭文件*m\_f = open(**"male\_names.txt"**) *# 打开男名文件*m\_name = m\_f.readlines() *# 将男名读入list*m\_f.close() *# 关闭文件***class** SimpleDataset(Dataset):  
 **def** \_\_init\_\_(self, name\_file, transform=**None**):  
 self.name\_file = name\_file  
 self.transform = transform  
  
 file\_path = **'lfw\_funneled/'** + self.name\_file[:-9] + **'/'** + self.name\_file  
 image = Image.open(file\_path)  
 **if** self.transform:  
 self.image = self.transform(image)  
  
 **def** \_\_getitem\_\_(self, index):  
  
 **if** self.name\_file + **'\n' in** f\_name:  
 label = 0  
 **else**:  
 label = 1  
  
 **return** self.image, label  
  
 **def** \_\_len\_\_(self):  
 **return** 1  
  
  
**def** show(p\_name):  
  
 **global** p\_img, p\_photo, p\_img\_label  
  
 p\_name\_path = **'lfw\_funneled/'** + p\_name[:-9] + **'/'** + p\_name  
 **if not** os.path.isfile(p\_name\_path):  
 print(p\_name + **'does not exist!'**)  
 **else**:  
 p\_img = Image.open(p\_name\_path)  
 p\_photo = ImageTk.PhotoImage(p\_img)  
 p\_img\_label = Label(root, image=p\_photo)  
 p\_img\_label.place(x=90, y=80)  
  
 p\_name\_label = Label(root, text=p\_name[:-9], font=20, bg=**'white'**, width=30)  
 p\_name\_label.place(anchor=CENTER, x=215, y=380)  
  
 simple\_data = SimpleDataset(name\_file=p\_name,  
 transform=transforms.Compose([  
 *# transforms.Resize((128, 128)),  
 # transforms.RandomHorizontalFlip(p=0.5),  
 # transforms.Pad(10),  
 # transforms.RandomCrop((100, 100)),* transforms.CenterCrop(150),  
 transforms.ToTensor(),  
 transforms.Normalize(  
 mean=[0.5, 0.5, 0.5],  
 std=[0.5, 0.5, 0.5])]))  
 simple\_loader = DataLoader(simple\_data, batch\_size=1)  
  
 **for** ph, target **in** simple\_loader:  
 ph = ph.to(device)  
 target = target.to(device)  
  
 **with** torch.no\_grad():  
 outputs = model(ph)  
  
 **if** outputs.argmax(1) == 1:  
 guess = **'Man, right?'  
 else**:  
 guess = **'Woman, right?'** guess\_label = Label(root, text=guess, font=25, width=20, bg=**'white'**)  
 guess\_label.place(anchor=CENTER, x=550, y=100)  
  
  
**def** rand\_show():  
 *# 该函数用于Random Draw被点击时* rand\_name = choice(names)[:-1]  
 show(rand\_name)  
  
  
**def** appoint\_show():  
 pic = t.get()  
 **if** pic + **'\n' not in** names:  
 error\_label = Label(root, text=**"No such picture!"**, font=25, width=20, bg=**'white'**)  
 error\_label.place(anchor=CENTER, x=550, y=100)  
 **else**:  
 show(pic)  
  
  
*# 加载模型*device = torch.device(**'cuda' if** torch.cuda.is\_available() **else 'cpu'**)  
model = YutongNet().to(device)  
model = torch.load(**"best\_model.pth"**)  
model.eval()  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 *# 初始化GUI* root = Tk()  
 root.title(**"Gender Identify"**)  
 root.geometry(**'800x500'**)  
  
 ran\_label = Button(root, text=**'Random Draw'**, command=rand\_show)  
 ran\_label.place(anchor=CENTER, x=550, y=150)  
  
 name = choice(names)[:-1]  
 name\_path = **'lfw\_funneled/'** + name[:-9] + **'/'** + name  
 p\_img = Image.open(name\_path)  
 p\_photo = ImageTk.PhotoImage(p\_img)  
 p\_img\_label = Label(root, image=p\_photo)  
 p\_img\_label.place(x=90, y=80)  
  
 hint\_label = Label(root, text=**'You could enter a picture name in Lwf dataset\n\  
 For example: Norman\_Mailer\_0001.jpg'**, anchor=W)  
 hint\_label.place(anchor=CENTER, x=550, y=250)  
  
 t = Entry(root, width=40)  
 t.place(anchor=CENTER, x=550, y=300)  
  
 t\_label = Button(root, text=**'Predict gender!'**, command=appoint\_show)  
 t\_label.place(anchor=CENTER, x=550, y=350)  
  
 root.mainloop()

# 六、总结

从这次作业中我的收获非常大。

知识方面，为了完成这次作业，我学会去找视频自学、查看官方文档、在不同的网站上搜索问题。技能方面，无疑我的编程水平有所提升，我渐渐发现此前所学的知识应该如何应用，在pycharm上学会规范自己的代码，领会python的高级与强大；我开始懂得，写好代码只是第一步，不断的调试才是真正的开始。

这次作业也让我思考追求。如果我仅要求自己的模型正确率超过50%，那么我开始后两天就已经可以提交作业了，但我希望看见一个完整的结果，于是磕了两周。见到最后的结果令我心满意足。我也知道很多事情，比如这个作业，攻克了它才知道没有它看上去那么难。

# 七、参考网站

<https://www.runoob.com/python/python-func-property.html>

<https://www.runoob.com/python/python-func-eval.html>

<https://docs.python.org/zh-cn/3/howto/functional.html>

<https://docs.python.org/3/howto/functional.html>

<https://www.python.org/dev/peps/pep-0008/>

<https://docs.python.org/3/howto/functional.html>

<https://www.python.org/>

<https://baike.baidu.com/item/%E5%8D%B7%E7%A7%AF%E7%A5%9E%E7%BB%8F%E7%BD%91%E7%BB%9C/17541100?fr=aladdin>

<https://baike.baidu.com/item/k%E8%BF%91%E9%82%BB%E7%AE%97%E6%B3%95/9512781?fr=aladdin>

<https://www.zhihu.com/question/49136398>

<https://mirror.tuna.tsinghua.edu.cn/help/anaconda/>

<https://www.jianshu.com/p/273f36cfac2d>

<https://www.py.cn/tools/anaconda/15646.html>

<https://www.zhihu.com/question/344311341>

<https://blog.csdn.net/hehedadaq/article/details/107270906>

<https://blog.csdn.net/bl136823/article/details/88813033>

<https://blog.csdn.net/miracleoa/article/details/106039388>

<https://blog.csdn.net/t20134297/article/details/104691192/>

<https://blog.csdn.net/ShirleyPlumLee/article/details/119787344>

<https://pytorch-cn.readthedocs.io/zh/latest/notes/autograd/>

<https://pytorch.apachecn.org/docs/1.4/5.html>

<https://www.jianshu.com/p/2d9927a70594>

<https://www.bilibili.com/video/BV1hE411t7RN?from=search&seid=14222437770201507474>

<https://blog.csdn.net/qq_31187803/article/details/79601643>

<https://www.jb51.net/article/181954.htm>

<https://blog.csdn.net/cjw12581/article/details/105057655>

<https://blog.csdn.net/chary8088/article/details/79032223>

<https://www.runoob.com/numpy/numpy-tutorial.html>

<https://blog.csdn.net/tumantou2hao/article/details/88293288>

<https://search.bilibili.com/all?keyword=%E9%80%BB%E8%BE%91%E5%9B%9E%E5%BD%92&from_source=webtop_search&spm_id_from=333.851>

<https://www.bilibili.com/video/BV1PJ411676g?from=search&seid=6919711676656568749>

<https://www.bilibili.com/video/BV1Ma411F7Y4?from=search&seid=11591455095474788979>

<https://blog.csdn.net/kieven2008/article/details/81582591>

<https://blog.csdn.net/weixin_43343072/article/details/86715023>

<https://www.runoob.com/python/python-dictionary.html>

<https://blog.csdn.net/qq_41185868/article/details/82915063>

<https://www.runoob.com/python/python-strings.html>

<https://m.py.cn/jishu/jichu/12838.html>

<https://blog.csdn.net/qq_36783142/article/details/105286841>

<https://www.cnblogs.com/wangbin2188/p/7339169.html>

<https://blog.csdn.net/qq_30708445/article/details/89479773>

<https://blog.csdn.net/huang9012/article/details/17417133>

<https://blog.csdn.net/qq_16525279/article/details/80245350>

<https://blog.csdn.net/AC_shuai/article/details/81165085?utm_medium=distribute.pc_relevant_t0.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-1.control&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-1.control>

<https://blog.csdn.net/qq_30708445/article/details/89479773>

<https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html>

<https://blog.csdn.net/wangguan9527/article/details/82845409>

<https://www.cnpython.com/qa/116161>

<https://blog.csdn.net/weixin_44553473/article/details/109747689>

<https://blog.csdn.net/qq_44702847/article/details/103253494>

<https://blog.csdn.net/u010801994/article/details/81121078>

<https://www.jianshu.com/p/a68cc19d0f9d>

<https://stackoverflow.com/questions/44604689/pycharm-error-process-finished-with-exit-code-1073740791-0xc0000409>

<https://www.pianshen.com/article/4118922495/>

<http://www.bubuko.com/infodetail-3767354.html>

<https://blog.csdn.net/weixin_39990138/article/details/110976262>

<https://www.bilibili.com/video/BV1Vx411j7pJ?from=search&seid=1516929581209019166>

<https://www.bilibili.com/video/BV1oa411c7mS/?spm_id_from=autoNext>

<https://blog.csdn.net/qq_36118564/article/details/90324471>

<https://blog.csdn.net/tonydz0523/article/details/79075077>

<https://blog.csdn.net/lizzy05/article/details/90521909>

<https://blog.csdn.net/weixin_35076078/article/details/115738462>

<https://blog.csdn.net/wohu1104/article/details/107743290>

<https://pytorch.apachecn.org/docs/1.4/4.html>

<https://pytorch.apachecn.org/docs/1.4/blitz/tensor_tutorial.html>

<https://pytorch.apachecn.org/docs/1.4/blitz/neural_networks_tutorial.html>

<https://pytorch.apachecn.org/docs/1.4/5.html>

<https://www.runoob.com/python/python-func-enumerate.html>

<https://blog.csdn.net/jzwong/article/details/104272600>

<https://www.jianshu.com/c/4e22723d3fe2>

<https://blog.csdn.net/weixin_44920757/article/details/116517029>

<https://blog.csdn.net/zzc15806/article/details/84875070>

<https://blog.csdn.net/weixin_39606575/article/details/110837635>

<https://www.cnblogs.com/-xuewuzhijing-/p/12971924.html>

<https://blog.csdn.net/ftimes/article/details/105202039>

<https://blog.csdn.net/qq_40564301/article/details/113065439>

<https://blog.csdn.net/qq_40564301/article/details/112886578>

<https://blog.csdn.net/cumudi0723/article/details/107799347>

<https://blog.csdn.net/as480133937/article/details/93137001>

<https://bbs.csdn.net/topics/370007558>

<https://stackoverflow.com/questions/62924547/typeerror-pic-should-be-pil-image-or-ndarray-got-class-bool>

<https://blog.csdn.net/qq_40178291/article/details/101108327>

<https://blog.csdn.net/wacebb/article/details/108003306>

<https://blog.csdn.net/weixin_42101177/article/details/104222253>

<https://blog.csdn.net/vitaminc4/article/details/78202780>

<https://blog.csdn.net/liuchenbaidu/article/details/107759873>

<https://blog.csdn.net/m0_37693335/article/details/78039512>

<https://blog.csdn.net/zkp_987/article/details/81748098>

<https://www.cnblogs.com/ranjiewen/p/10128046.html>

<https://blog.csdn.net/iammelon/article/details/89761279>

<https://blog.csdn.net/shaoyou223/article/details/84561523>

<https://blog.csdn.net/yangdashi888/article/details/82223672>

<https://blog.csdn.net/zfhsfdhdfajhsr/article/details/115054702>

<https://blog.csdn.net/weixin_34020067/article/details/112201810>

<https://hellozhaozheng.github.io/z_post/Python-tqdm/>

<https://www.cnblogs.com/sddai/p/14648966.html>

<https://blog.csdn.net/Carlsummer/article/details/119673334>

<https://blog.csdn.net/yangwangnndd/article/details/95385628>

<https://blog.csdn.net/yangdashi888/article/details/82223672>

<https://github.com/gzshan/dogsVScats>

<https://blog.csdn.net/weixin_43593330/article/details/107483671>

<https://blog.csdn.net/weixin_44928646/article/details/105965540>

<https://www.csdn.net/tags/MtjaQg3sMjM2NDgtYmxvZwO0O0OO0O0O.html>

<https://blog.csdn.net/a8039974/article/details/80287421?utm_term=skimage%E7%81%B0%E5%BA%A6%E5%A4%84%E7%90%86&utm_medium=distribute.pc_aggpage_search_result.none-task-blog-2~all~sobaiduweb~default-0-80287421&spm=3001.4430>

<https://www.nuomiphp.com/eplan/28735.html>

<https://www.cnblogs.com/wojianxin/p/12638205.html>

<https://blog.csdn.net/ttdxtt/article/details/112347283?utm_medium=distribute.pc_relevant_t0.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-1.control&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-1.control>

<https://www.runoob.com/python/file-readlines.html>

<https://blog.csdn.net/qq_42031142/article/details/107494484>

<https://blog.csdn.net/weixin_37746009/article/details/94367162>

<https://www.cnblogs.com/fujian-code/p/8919670.html>

<https://www.cnpython.com/qa/57865>

<https://blog.csdn.net/xm_csdn/article/details/74640416>

<https://www.pianshen.com/article/81561097962/>

<https://blog.csdn.net/sazass/article/details/109725458>

<https://blog.csdn.net/ttdxtt/article/details/112347283>

<https://blog.csdn.net/The_dream1/article/details/118066252>

<https://stackoverflow.com/questions/68105949/typeerror-tensor-got-an-unexpected-keyword-argument-names>

<https://blog.csdn.net/dss_dssssd/article/details/82980222>

<https://ptorch.com/news/57.html>

<https://blog.csdn.net/weixin_41529093/article/details/111410513>

<https://www.5axxw.com/questions/content/sfmy18>

<https://blog.csdn.net/weixin_42018112/article/details/90084419>

<https://blog.csdn.net/qq_42079689/article/details/102873766>

<https://www.cnblogs.com/douzujun/p/13366939.html>

<https://blog.csdn.net/daydaydreamer/article/details/102638624>

<https://download.csdn.net/download/weixin_38740391/12860321?utm_medium=distribute.pc_relevant_t0.none-task-download-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-1.essearch_pc_relevant&depth_1-utm_source=distribute.pc_relevant_t0.none-task-download-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-1.essearch_pc_relevant>

<https://www.cnblogs.com/Archer-Fang/p/10645473.html>

<https://www.jianshu.com/p/c6eabe6f69b1>

<https://blog.csdn.net/jining11/article/details/89114502>

<https://blog.csdn.net/qq_38863413/article/details/104108808>

<https://stackoverflow.com/questions/67104073/runtimeerror-expected-scalar-type-double-but-found-float>

<https://www.cnblogs.com/wangling1820/p/12533473.html>

<https://blog.csdn.net/weixin_41514525/article/details/110129655>

<https://blog.csdn.net/moshiyaofei/article/details/89703161>

<https://www.csdn.net/tags/Mtzagg4sOTE1MS1ibG9n.html>

<https://blog.csdn.net/qq_22764813/article/details/103404462>

<https://www.jianshu.com/p/1ae863c1e66d>

<https://blog.csdn.net/weixin_39736913/article/details/111437867>

<https://blog.csdn.net/weixin_39605894/article/details/110446035>

<https://blog.csdn.net/zjh12312311/article/details/107293265/>

<https://stackoverflow.com/questions/66074684/runtimeerror-expected-scalar-type-double-but-found-float-in-pytorch-cnn-train>

<https://blog.csdn.net/qq_23385943/article/details/106040650>

<https://stackoverflow.com/questions/39808529/redeclared-s-defined-above-without-usage>

<https://stackoverflow.com/questions/55738418/redeclared-variable-defined-above-without-usage-in-python>

<https://www.656463.com/wenda/mysygdsmdydzxsm_448>

<https://blog.csdn.net/weixin_44274975/article/details/88366540>

<https://blog.csdn.net/u013632755/article/details/78739448>

<https://stackoverflow.com/questions/43865347/why-does-pycharm-warn-about-redeclared-variable-defined-above-without-usage>

<https://www.cnpython.com/qa/227758>

<https://www.py4u.net/discuss/179699>

<https://blog.csdn.net/moshiyaofei/article/details/89703161>

<https://blog.csdn.net/weixin_39785081/article/details/111167415>

<https://blog.csdn.net/weixin_41514525/article/details/110129655>

<https://blog.csdn.net/xrfxhh/article/details/115336041>

<https://blog.csdn.net/kangshuangzhu/article/details/108125843>

<https://stackoverflow.com/questions/67878735/how-to-change-the-dtype-of-data-in-tf-data-dataset>

<https://stackoverflow.com/questions/35725513/tensorflow-cast-a-float64-tensor-to-float32>

<https://blog.csdn.net/dcrmg/article/details/79747814>

<https://blog.csdn.net/qq_37555071/article/details/107553416>

<https://blog.csdn.net/oldmao_2001/article/details/102661974>

<https://blog.csdn.net/weixin_37763870/article/details/105818681>

<https://www.cnblogs.com/ocean1100/p/9494640.html>

<https://blog.csdn.net/u012986684/article/details/79179640>

<https://blog.csdn.net/qq_34886403/article/details/82558399>

<https://www.cnblogs.com/lliuye/p/9471231.html>

<https://blog.csdn.net/jinxiaonian11/article/details/83105439>

<https://blog.csdn.net/u012513618/article/details/78454652?utm_medium=distribute.pc_relevant_t0.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-1.control&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-1.control>

<https://www.jianshu.com/p/22c50ded4cf7>

<https://zhuanlan.zhihu.com/p/29409502>

<https://blog.csdn.net/jzwong/article/details/104272600>

<https://blog.csdn.net/qq_35027690/article/details/103742697>

<https://blog.csdn.net/qq_18649781/article/details/92120175>

<https://www.cnblogs.com/oliyoung/p/transforms_Normalize.html>

<https://www.jianshu.com/p/d84c00b2eb91>

<https://discuss.pytorch.org/t/understanding-transform-normalize/21730>

<https://blog.csdn.net/u014453898/article/details/90700904>

<https://bbs.pinggu.org/thread-136298-1-1.html>

<https://www.csdn.net/tags/MtjaEg2sMDY5MjMtYmxvZwO0O0OO0O0O.html>

<https://blog.csdn.net/f9547064141/article/details/107216647>

<https://blog.csdn.net/qq_24407657/article/details/103992170>

<https://blog.csdn.net/u013231020/article/details/106025521/>

<https://pytorch-cn.readthedocs.io/zh/latest/package_references/torch-nn/>

<https://blog.csdn.net/qq_34243930/article/details/107231539>

<https://blog.csdn.net/qq_26369907/article/details/88366147>

<https://blog.csdn.net/qq_44554428/article/details/104415818>

<https://blog.csdn.net/sscc_learning/article/details/79814146?utm_medium=distribute.pc_relevant_t0.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-1.control&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-1.control>

<https://www.pianshen.com/article/6365429555/>

<https://blog.csdn.net/xys430381_1/article/details/80587963>

<https://blog.csdn.net/yangfengling1023/article/details/82911306>

<https://blog.csdn.net/qq_42079689/article/details/102873766>

<https://blog.csdn.net/daydaydreamer/article/details/102638624>

<https://www.cnblogs.com/douzujun/p/13366939.html>

<https://www.jianshu.com/p/c6eabe6f69b1>

<https://zhuanlan.zhihu.com/p/152198144>

<http://www.digtime.cn/articles/159/pytorch-zhong-nn-linear-han-shu-jie-du>

<https://blog.csdn.net/qq_36387683/article/details/107638184>

<https://blog.csdn.net/woo233/article/details/117746044>

<https://zhenkai.blog.csdn.net/article/details/88778371?utm_medium=distribute.pc_relevant_t0.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-1.control&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-1.control>

<https://blog.csdn.net/w372845589/article/details/84303498>

<https://www.zhihu.com/question/330980636>

<https://blog.csdn.net/qq_15557299/article/details/104307690>

<https://www.thinbug.com/q/51132429>

<https://forums.fast.ai/t/train-loss-at-training-is-much-more-than-the-actual-train-loss/45610>

<https://www.programcreek.com/python/?CodeExample=train+one+epoch>

<https://stackoverflow.com/questions/65336491/why-do-i-get-test-loss-0-and-accuracy-1-0-constant-for-all-epochs>

<https://blog.csdn.net/liboshi123/article/details/116078509>

<https://zhuanlan.zhihu.com/p/98563580>

<https://www.pianshen.com/article/264438391/>

<https://github.com/Machine-Learning-For-Research/Dogs-Cats/issues>

<https://github.com/Machine-Learning-For-Research/Dogs-Cats>

<https://blog.csdn.net/weixin_43201920/article/details/106010743>

<https://www.cnblogs.com/denny402/p/6931338.html>

<https://zhuanlan.zhihu.com/p/149087752>

<https://zhuanlan.zhihu.com/p/85373781>

<https://www.cnblogs.com/54hys/p/10199359.html>

<https://blog.csdn.net/weixin_42575020/article/details/82944291>

<https://www.cnblogs.com/-wenli/p/11234878.html>

<https://www.zhihu.com/question/32246256>

<https://www.jianshu.com/p/d25c7f0c36d8>

<https://blog.csdn.net/zzh_hn/article/details/103854858>

<https://blog.csdn.net/weixin_34293059/article/details/85947681>

<https://blog.csdn.net/lxj1229022827/article/details/45557349>

<https://www.jianshu.com/p/6ae3809ad515>

<https://blog.csdn.net/kane7csdn/article/details/86475918>

<https://www.cnblogs.com/wt714/p/12545129.html>

<https://blog.csdn.net/zxc120389574/article/details/105827843/>

<https://blog.csdn.net/weixin_44936889/article/details/103625112>

<https://blog.csdn.net/sinat_33563325/article/details/79868109>

<https://www.cnblogs.com/rtyxxy/p/12687768.html>

<https://stackoverflow.com/questions/44604689/pycharm-error-process-finished-with-exit-code-1073740791-0xc0000409>

<https://blog.csdn.net/weixin_44928646/article/details/105965540>

<https://ask.csdn.net/questions/768763>

<https://blog.csdn.net/weixin_30819163/article/details/101711305>

<https://blog.csdn.net/weixin_30819163/article/details/101711305>

<https://blog.csdn.net/bitcarmanlee/article/details/82320853>

<https://www.zhihu.com/question/23765351>

<https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm1d.html#torch.nn.BatchNorm1d>

<https://pytorch.org/docs/stable/generated/torch.nn.Softmax.html?highlight=softmax#torch.nn.Softmax>

<https://www.jb51.net/article/214280.htm>

<https://blog.csdn.net/qq_38262266/article/details/103411808>

<https://blog.csdn.net/qq_34600424/article/details/112770024>

<https://blog.csdn.net/a1456123a/article/details/118878682>

<https://www.cnblogs.com/ttdeveloping/p/10668621.html>

<https://blog.csdn.net/weixin_30036289/article/details/112199379?utm_term=cnn%E8%BF%9B%E8%A1%8C%E5%9B%BE%E5%83%8F%E4%BA%8C%E5%88%86%E7%B1%BB&utm_medium=distribute.pc_aggpage_search_result.none-task-blog-2~all~sobaiduweb~default-2-112199379&spm=3001.4430>

<https://blog.csdn.net/lxdssg/article/details/81233837>

<https://zhuanlan.zhihu.com/p/114869268>

<https://www.cnblogs.com/cyssmile/p/13570042.html>

<https://pytorch.org/docs/stable/generated/torch.nn.Sigmoid.html?highlight=sigmoid#torch.nn.Sigmoid>

<https://blog.csdn.net/qq_40608730/article/details/107134830?utm_source=app&app_version=4.14.1>

<https://bbs.csdn.net/topics/392064652>

<https://blog.csdn.net/lx127372/article/details/105647199>

<https://zhuanlan.zhihu.com/p/107687473>

<https://www.lmlphp.com/user/3659/article/item/29184/>

<https://blog.csdn.net/weixin_40499753/article/details/100056919>

<https://zhuanlan.zhihu.com/p/82448855>

<https://www.cnblogs.com/hellcat/articles/7220040.html>

<https://www.jianshu.com/p/d55bc23501af>

<https://www.cnblogs.com/kk17/p/9693462.html>

<https://www.shangmayuan.com/a/51f9e627034d41818edecbc8.html>

<https://blog.csdn.net/qq_23262411/article/details/100175943>

<https://ask.csdn.net/questions/7407432?spm=1005.2026.3001.5622&utm_source=app>

<https://wanghao.blog.csdn.net/article/details/117353112>

<https://zhuanlan.zhihu.com/p/98666173>

<https://www.pianshen.com/article/106041404/>

<https://segmentfault.com/q/1010000019514460/a-1020000019530845>

<https://www.zhihu.com/question/335713101>

<https://www.cnblogs.com/Allen-rg/p/9008951.html>

<https://blog.csdn.net/weixin_44020747/article/details/111881496>

<https://www.cnblogs.com/ssyfj/p/13964588.html>

<https://www.jianshu.com/p/9b6bde60f3e9>

<https://blog.csdn.net/dss_dssssd/article/details/82980222>

<https://blog.csdn.net/t20134297/article/details/105490361>

<https://zhuanlan.zhihu.com/p/95680105>

<https://zhuanlan.zhihu.com/p/71772321>

<https://www.pythonheidong.com/blog/article/397831/cfa9659ce6e98677eabf/>

<https://liumin.blog.csdn.net/article/details/85725195>

<https://zhuanlan.zhihu.com/p/54527197>

<https://blog.csdn.net/weixin_30239339/article/details/95840248>

<https://zhuanlan.zhihu.com/p/38056115>

<https://www.jianshu.com/p/6ba95579082c>

<https://zhuanlan.zhihu.com/p/39667289>

<https://pytorch-cn.readthedocs.io/zh/latest/torchvision/torchvision-utils/>

<https://zhuanlan.zhihu.com/p/60753993>

<https://blog.csdn.net/qq_27825451/article/details/96856217>

<https://blog.csdn.net/sz793919425/article/details/84305669>

<https://ptorch.com/news/77.html>

<https://blog.csdn.net/qq_38863413/article/details/103316931>

<https://www.jianshu.com/p/91844c5bca78>

<https://www.runoob.com/python/python-func-sum.html>

<https://blog.csdn.net/joson1234567890/article/details/78639014>

<https://blog.csdn.net/white_156/article/details/90579031>

<https://stackoverflow.com/questions/60780128/typeerror-photoimage-object-is-not-iterable>

<https://www.cnblogs.com/semishigure/p/8124397.html>

<https://www.cnblogs.com/ruo-li-suo-yi/p/7425307.html>

<https://blog.csdn.net/xoofly/article/details/89735573>

<http://www.weixueyuan.net/a/562.html>

<https://www.runoob.com/python/python-tk-label.html>

<https://blog.csdn.net/qq_34108714/article/details/90106018>

<https://bbs.csdn.net/topics/370105452>

<https://www.cnpython.com/qa/460429>

<https://www.imooc.com/wenda/detail/609369>

<https://www.runoob.com/python/python-tk-button.html>

<https://blog.csdn.net/L_0000/article/details/88911344>

<https://www.zhihu.com/question/23542885>

<https://blog.csdn.net/wangfei8348/article/details/51744968>

<https://www.py.cn/faq/python/12525.html>

<https://www.cnpython.com/qa/231554>

<https://blog.csdn.net/weixin_39704314/article/details/110551170>

<https://blog.csdn.net/chenf1999/article/details/104333709>

<https://www.jb51.net/article/206733.htm>

<https://www.pythonheidong.com/blog/article/283309/3aebdfeace9e1876f7ba/>

<https://blog.csdn.net/bobo_jiang/article/details/82597548>

<https://www.zhihu.com/question/315003098/answer/623927403>

<https://stats.stackexchange.com/questions/418868/100-accuracy-on-training-high-accuracy-on-testing-as-well-what-does-this-mean>

<https://datascience.stackexchange.com/questions/43313/why-100-accuracy-on-test-data-is-not-good>

<https://www.baidu.com/s?wd=cnn%20test%E5%87%86%E7%A1%AE%E7%8E%87%E4%B8%8D%E5%8F%98&pn=10&oq=cnn%20test%E5%87%86%E7%A1%AE%E7%8E%87%E4%B8%8D%E5%8F%98&ie=utf-8&fenlei=256&rsv_idx=1&rsv_pq=f0e43c77000ee00a&rsv_t=e01eQBN9zP1rKz7gDOIJV146LI8Rf4Sw2Ym4z4mCvf5m3zToIEmx1ZZYCgQ&rsv_page=1>

<https://www.it610.com/article/1279740186304987136.htm>

<https://github.com/keras-team/keras/issues/1597>

<https://www.javaroad.cn/questions/305512>

<https://www.cnblogs.com/ymjyqsx/p/11379726.html>

<https://datascience.stackexchange.com/questions/72222/multi-class-cnn-model-predicting-only-one-class-but-still-the-accuracy-is-high>

<https://www.adoclib.com/blog/keras-cnn-prediction-output-is-only-predicting-within-a-range-of-numbers.html>

<https://www.cnpython.com/qa/220794>

<https://www.zhihu.com/question/311931241>

<https://stackoverflow.com/questions/56027737/why-does-cnn-only-predict-one-class>

<https://stackoverflow.com/questions/41488279/neural-network-always-predicts-the-same-class>

<https://www.csdn.net/tags/NtjaYg0sMjYzMDktYmxvZwO0O0OO0O0O.html>

<https://blog.csdn.net/u012370185/article/details/90243551>

<https://www.csdn.net/tags/OtTacg0sNjUyOTctYmxvZwO0O0OO0O0O.html>

<https://blog.csdn.net/qq_20979341/article/details/105806933>

<http://wenda.chinahadoop.cn/question/5333>

<https://www.cnblogs.com/xbhog/p/11745744.html>

<https://www.cnblogs.com/kongzhagen/p/6145429.html>

<https://www.runoob.com/python/python-tkinter-entry.html>