目录

python大作业实验报告

前言

在本次大作业中，按笔者完成作业的时间顺序，笔者尝试了了进行性别识别的三种不同方法，依次为

- k-Nearest Neighbor (KNN)
- Logistic Regression
- Convolutional Neural Network (CNN)

从结果来看，该顺序也是识别准确率的递增顺序，依次为

- KNN $\sim 80\%$
- L-R $\sim 86\%$
- CNN $\sim 95\%$

KNN算法

0.环境

anaconda基础环境：`Python 3.8.8 64-bit ('base': conda)`

1.DataLoader

定义`class DataLoader`用于导入经随机打乱并划分为训练、验证和测试的图像数据及标签，其中图像经过黑白处理。

2.KNN算法的实现

2.1 手写**KNN**

KNN算法需要程序学习参数K的取值，笔者主要参考助教给出的示例代码，首先定义KNN类对象`class KNeighborsClassifier`，其中封装了手写的训练模型以及用模型预测的方法：

```
def train(self, data_x, data_y, k) # data_x为图像数据，data_y为标签
def predict(self, data_x, data_y) # 确定待测图像矩阵与训练集图像矩阵距离最近的K个出现最多的那个标签
```

其中`train`单纯将训练集图像和标签转换为`KNeighborsClassifier`类的属性，`predict`则计算需要预测图像的矩阵与类中存储的矩阵“距离”最近的是哪些，并认为其中占多数的标签值就是预测的标签值。

之后设定K上下限并采用循环并迭代最优的K：

```
K_min = 1
K_max = 15
```

```
for k in range(K_min, K_max+1, 2):  # 通过循环尝试不同的超参数，自动找到较优的超参数
    knn = KNeighborsClassifier()  # 初始化KNN类对象
    knn.train(train_images, train_labels, k)  # 使用训练集训练
    result = knn.predict(validation_images, validation_labels)  # 使用验证集预测
```

得到最佳的K参数后同样调用predict方法对测试集预测。

2.2调用**sklearn**的**KNN**

直接import sklearn中的`KNN`类对象：

```
from sklearn.neighbors import KNeighborsClassifier
```

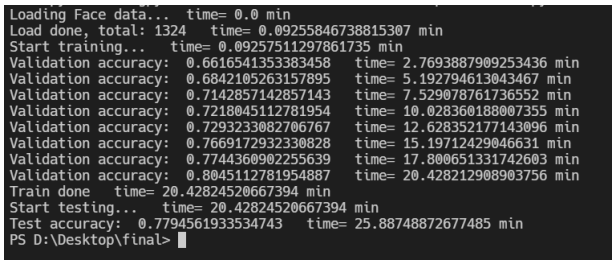
随后调用库里的方法训练：

```
knn.fit(train_images, train_labels)  # 使用训练集训练
accuracy = 100*knn.score(validation_images, validation_labels)  # 使用验证集预测
if accuracy > accuracy_max:  # 保存较优的模型
    accuracy_max = accuracy
    best_knn = knn
```

3.结果

3.1手写**KNN**结果

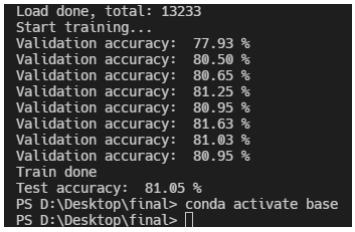
调用`KNeighborsClassifier`类的`predict`方法对测试集进行预测，结果如下：



算法性能差，选1/10的数据运行，仍需25min以上的运行时间，且准确率不足80%。

3.2调库**KNN**结果

调用`KNeighborsClassifier`类的`score`方法对测试集进行预测，结果如下：



程序在1min以内运行完毕，准确率相比手写的略有提高。

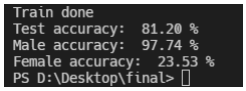
4.结果分析及总结

KNN算法是笔者第一个实现的算法，其想法是朴素的：计算未知图片矩阵与已知图片的距离，距离近的认为标签更有可能一致，本质上仅仅是考察两幅图片像不像，考虑到数据集包括了白种人、黄种人、黑种人等多种种族数据，也存在人脸姿态、光照、年龄等多种干扰[1]，可见这种算法本身是没有什么道理的。

由于所给的数据集中男性和女性并不一样多——约为10:3，从结果上看，`KNN`算法的准确率接近较为接近男性所占数据集的比例。笔者添加代码

```
accuracy_male = 100 * best_knn.score(test_images_male, test_labels_male)
print('Male accuracy: ', ('%.2f' % accuracy_male), '%')
accuracy_female = 100 * best_knn.score(test_images_female,
test_labels_female)
print('Female accuracy: ', ('%.2f' % accuracy_female), '%')
```

分别测试了男性和女性的准确率，结果如下



可见由于数据集中男性更多，自然成为K-neighbors的概率更大，识别时程序在80%的情况下给出了男性的回答，故而当被测试图像为女性时几乎都是错误的。

总体来说，`KNN`算法的优势是简单直观，在程序的编写上笔者算是调库调得最少的一个算法。缺点是准确率低。

Logistic Regression算法

0.环境

anaconda基础环境：`Python 3.8.8 64-bit ('base': conda)`

1.DataLoader

同`KNN`算法。

2.调用**sklearn**的**Logistic Regression**

笔者选用了线性拟合方法，对于训练用图片的输入，以图片的像素矩阵为自变量，标签值为因变量构建图像，并学习寻找其中的高维超平面，本质上是最小二乘问题的高维拓展。

首先实例化sklearn中的`Logistic Regression`对象：

```
lr = LogisticRegression(penalty='l1', C=c, multi_class='ovr',
solver='liblinear')
```

根据官网设置了对象的参数[2]，其中选择的线性优化算法仅支持 `l1` 的惩罚机制，它可以更有效地避免过拟合问题[3]。

随后通过训练集训练，验证集验证：

```
lr.fit(train_images, train_labels)  # 使用训练集训练
accuracy = 100 * lr.score(validation_images, validation_labels) # 使用验证集验证
```

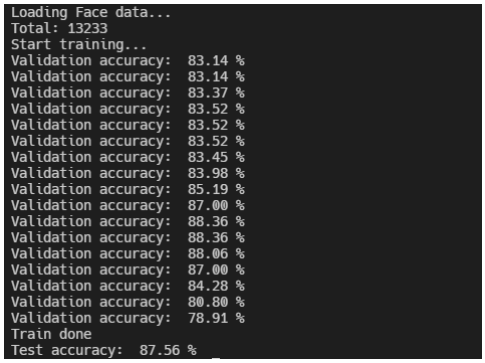
并以验证集准确率高的迭代最佳的`Logistic Regression`对象：

```
if accuracy > accuracy_max:  # 保存较优的模型
    accuracy_max = accuracy
    best_lr = lr
```

在训练过程中不断调小参数`C`，目的是加强模型对过拟合的抑制作用。

3.结果

用测试集测试，同样调用`score`方法，结果如下：



4.结果分析及总结

`Logistic Regression`相较`KNN`的准确率有相当的提升。

`Logistic Regression`算法存在过拟合的问题，这可以从结果中的`Validation accuracy`先上升后下降看出来。

运行时间上，该算法用时约为15min。总体来说，`Logistic Regression`相较于`KNN`算法提高了准确率，代价是提高了运行时间。

CNN算法

与前两种算法主要依靠助教提供的示例不同，`CNN`算法的实现主要是笔者通过自学和摸索做到的，相较前两种算法更为复杂，为提高准确率而使用的 'tricks' 也更多。

0.环境

GPU版的tensorflow2.0环境：`Python 3.6.13 64-bit ('tf-gpu': conda)`

1.TensorDataLoader

在这里笔者做出了相当多改变，试图提高准确率。

1.1输入的变化

tensorflow2.0中的模型要求图像集的输入格式为四阶张量，每一维的大小依次代表图像数量，图像行数，图像列数，色彩通道数（对于彩色图为3），因此在之前`DataLoader`的基础上，还要通过`np.array`将输入的列表数据转换为矩阵，并进行`reshape`。

1.2扩充训练数据

`CNN`算法的过拟合现象非常严重，笔者采用的避免方法其一就是扩充训练用数据集，方法是读入`train_data`后将其第三维度的元素值倒序排列，即

```
self.train_data_flip = self.train_data[:, :, ::-1, :]  # 左右翻转图片列元素倒序排列
```

即对于图像的每一个像素，行不变，列进行对称，相当于左右翻转图像，使得训练集得到了扩充。

1.3压缩图片

同时，`CNN`对计算机性能要求大，笔者将图片进行了压缩处理，压缩后图片的尺寸约为50×50，使得运行的时间效率大大提高。

1.4先进行人脸识别

注意到数据集的照片存在背景的干扰，这是不小的影响因素。笔者搜集资料，在程序读入图片时使其仅包含人脸。

方法是实例化`cv2`中的人脸识别器对象，通过其获取人脸区域坐标范围，再进行图像裁剪：

```
# 创建人脸检测器放在同目录
face =
cv2.CascadeClassifier("./Common/haarcascades/haarcascade_frontalface_default.
xml")
# 将image转为灰度图像，存放中gray中
gray = cv2.cvtColor(image_cv, cv2.COLOR_BGR2GRAY)
# 检测图像中的人脸
faces = face.detectMultiScale(gray, 1.1, 5, cv2.CASCADE_SCALE_IMAGE)
```

裁剪前后的图像效果对比示意图如下。

在笔者提交的代码文件中，为了方便起见，笔者在`Common`文件夹中添加了`image-generator.py`文件，将人脸居中后的图像存储在`Dataset\Image-haired_colored`文件夹中，之后程序直接从该文件夹中读入图像。其实这与读入原始图像再进行人脸居中并没有什么区别，只是节约了进行人脸居中的时间。

经过测试，这样可以提高准确率约2%。

2.CNN网络结构

使用`keras.model.summary()`方法打印如下：

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 44, 44, 64)        1792
_________________________________________________________________
batch_normalization (BatchNo (None, 44, 44, 64)        256
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 22, 22, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 20, 20, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 10, 10, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 128)         0
```

```
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 2, 2, 128)         147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 1, 1, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 128)               16512
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 64)                4160
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 130
=================================================================
Total params: 305,986
Trainable params: 305,858
Non-trainable params: 128
_________________________________________________________________
```

该网络结构主要参考mnist与fashion-mnist问题中常见的网络结构[4]，输入图片的尺寸改为46×46，可以大幅提高程序运行速度。网络的深度较之也有所增加，同时在最后一层的之前两层增添了`Dropout`，可以防止过拟合。

3.方法

笔者定义模型类`class Model(object)`

其中封装了

- 成员：类型为`tf.keras.Sequential()`的`model`；
- 方法
  - `__init__`

    进行初始化

    - 初始化自身成员
    - 确定输入图片的尺寸
  - `train`

    进行模型的训练

    - 选择优化器，通过`self.model.compile`。

      笔者选用的优化器是Adam，不过默认的Adam优化器学习率太大，会导致训练后期出现过拟合现象，所以将其学习率改为默认的1/10：

      ```
      adam = tf.keras.optimizers.Adam(learning_rate=0.0001)
      ```

    - 训练模型，通过`self.model.fit`

为避免过拟合，笔者定义了召回类`class myCallback(tf.keras.callbacks.Callback)`，旨在控制训练进程。其主要思想是**patience**以及保留最优模型，包括：

验证集准确率设置的`patience[0]`是3，即随着训练epoch的增加，若验证集的准确率不增反降的次数达到三次，便停止训练；

训练集与验证集差异过大的`patience[1]`是8，随着训练epoch的增加，训练集的准确率甚至能达99%以上，这时往往会导致其与验证集的准确率差异过大，因而需要停止训练，否则过拟合程度愈发严重。

保留最优模型通过`self.model.get_weights()`获得模型权重，之后`self.model.set_weights(self.best_weights)`。

- `test`

  进行模型的测试，通过`self.model.evaluate`。

- `show`

  显示界面，其中笔者采用了两种方法，在界面设计有更详细的介绍。

4.界面设计

4.1界面**1**

笔者在网上查到了一种较为简单的方法——`gradio`[5]，只需调用`gradio`库：

```
import gradio as gr
```

随后在训练完模型后加入三行代码即可

```
inputs = gr.inputs.Image()
outputs = gr.outputs.Label(num_top_classes=2)
gr.Interface(fn=predict, inputs=inputs,
outputs=outputs).launch(inbrowser=True)
```

其中`inputs`读入由用户输入的图片信息，`outputs`向用户展示标签值（男或女），`gr.Interface`中`fn`直接调用`predict`对图片进行预测，`launch`中设定`inbrowser`为True，一个网页便被自动打开，可以从本地上传图片，分析结果在页面给出展示。

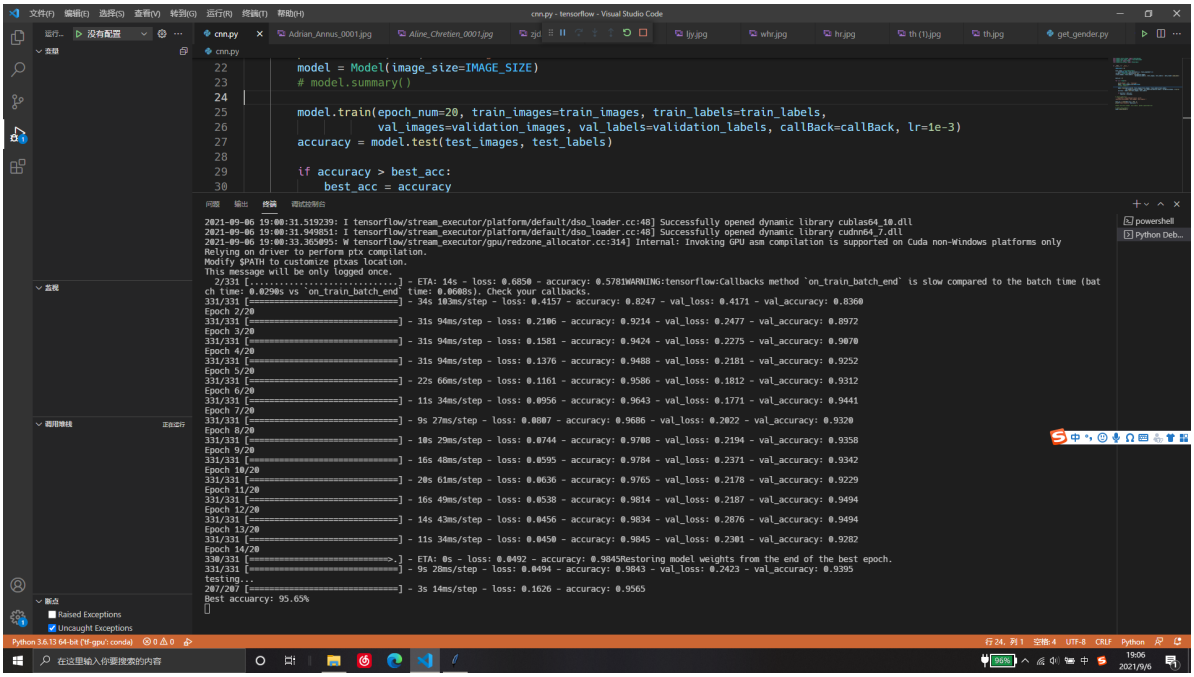
该界面外观优美，上传图片可直接拖拽，十分方便，但经笔者的抽样测试发现准确率远小于终端显示的95%，且调库后封装程度高，难以debug，故令手写另一种界面。

4.2界面**2**

定义界面类`class GetGender()`调用`tkinter`库进行编写，该界面需要用户手动输入图片的相对路径。

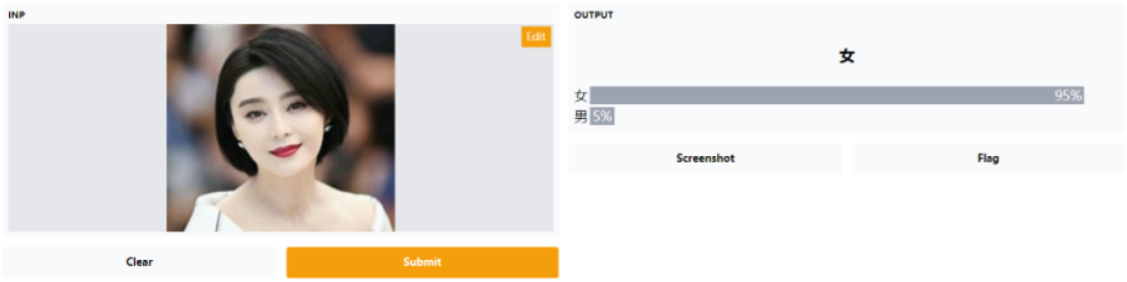
5.结果

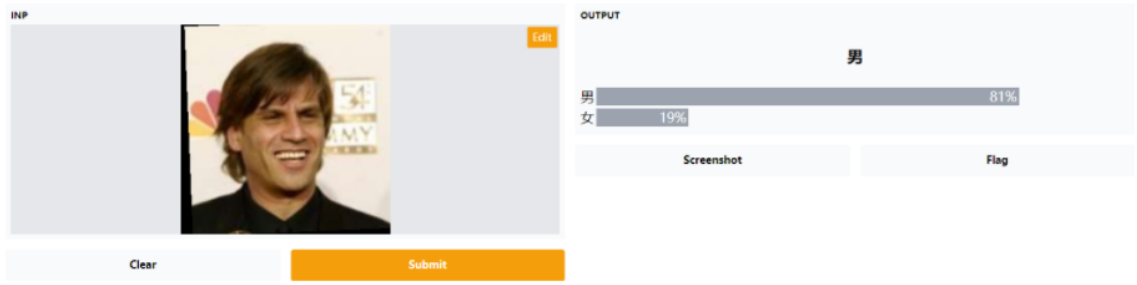
5.1程序运行结果



5.2界面展示

5.2.2界面1





5.2.2界面2

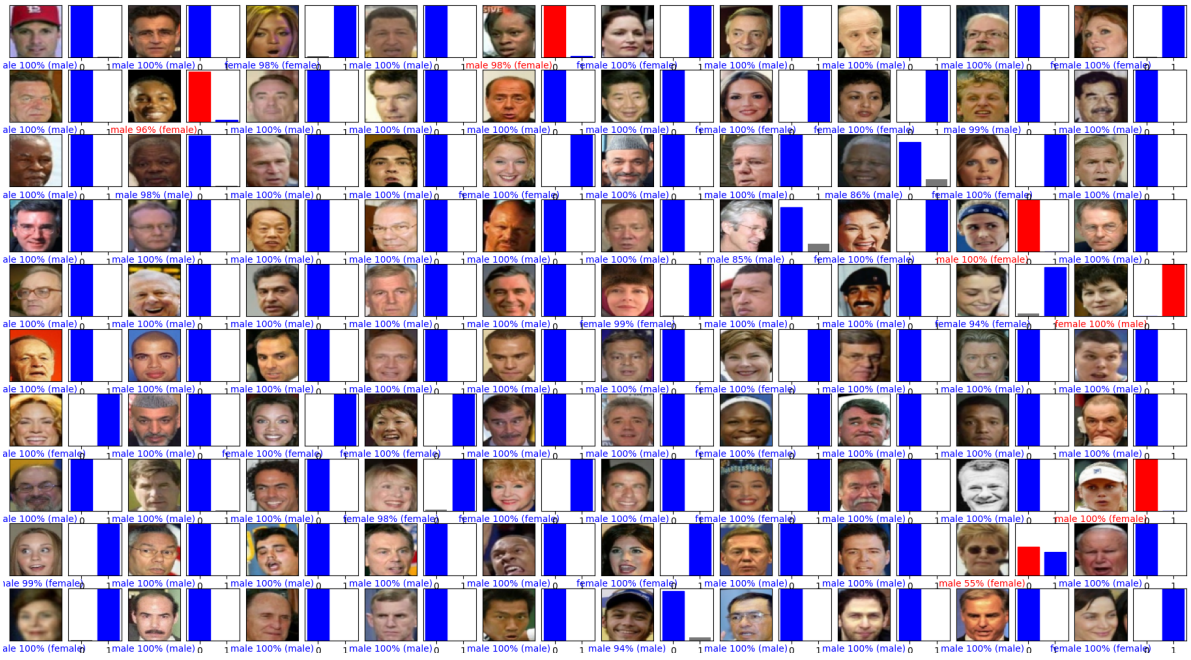
5.3结果分析

笔者从某一次的训练中查看了识别男性的准确率和识别女性的准确率，结果如下

```
Best accuarcy: 94.09%
Male testing...
162/162 [==============================] - 5s 33ms/step - loss: 0.0896 -
accuracy: 0.9700
Female testing...
46/46 [==============================] - 2s 46ms/step - loss: 0.7399 -
accuracy: 0.8381
```

可见即使使用`cnn`算法，女性准确率仍低于男性，这应该也与数据集中不均衡的男女比有关，

笔者编写`ResultDrawer`类抽取100张图片进行结果打印，其中错误的六张有五张为女性错识别为男性，也印证了这一点。



观察它识别错的几幅图像，笔者作为一个人类也很难认对，我想考虑到这些因素，性别识别能做到的最高准确率远不像mnist问题能达到接近100%，所以总体来说，`cnn`算法的结果令人满意，之后再寻求新的方法，想必也用处不大了，故笔者的探索到此为止。

源代码说明

源代码在`code`文件夹，其中的两个文件夹对应的编译环境由文件夹名给出。

程序中的类均单独存放在`Common`文件下，此外其中还有`haarcascades`的人脸识别器文件。

`base`文件夹中

- `KNN.py`
- `KNN-SKLearn.py`
- `LogisticRegression-SKLearn.py`

可直接运行。

`tensorflow`文件夹中

- `cnn.py`

可直接运行。

文献引用

---

1. 计算机程序设计基础（2） (tsinghua.edu.cn)
2. sklearn.linear_model.LogisticRegression — scikit-learn 0.24.2 documentation
3. Python sklearn逻辑回归(Logistic Regression,LR)参数_weixin_50304531的博客-CSDN博客
4. 卷积神经网络（Convolutional Neural Network，CNN） | TensorFlow Core (google.cn)
5. Gradio: 让机器学习算法秒变小程序 - 知乎 (zhihu.com)