数字识别(Digit Recognizer)

作者：朱瑾煜（2401111689），潘远（2410305335），李佩达（2410301207），
谢昊宸（2400010622），罗彪（2410117113），李萌恩（2110306222）

1 引言

图像识别是利用计算机对图像进行处理、分析和理解，以识别不同模式的目标和场景的技术，是计算机视觉领域的一个重要分支。

手写数字识别是图像识别的一个基本应用。

在这个课题中，我们将基于 Kaggle 上 “Digit Recognizer” 项目的 42000 份手写数字图像样本来训练模型，运用卷积神经网络（CNN）等方法，对预测集上 28000 份图像的标签（数字）进行预测。

2 材料和方法

2.1 数据集

1. **Kaggle 训练集：** 包括 42000 个样本。原图像为 28×28 像素的单通道图（每个像素只有介于 0~255 的整数灰度值）。训练集数据中已按 $ x= i \times 28 + j$ 的方法展平为 784 个特征（称作 `pixel` x），分别位于 2~785 列。第 1 列为 `label`, 是 0~9 之间的数字，代表图像表示的手写数字。在初步处理中，进一步将 Kaggle 训练集按 $85/15$ 的比例分为本地使用的训练集和测试集。
2. **Kaggle 测试集：** 28000 个样本。大体与训练集相同，但是没有 `label` 列，因此无法用于本地的验证。
3. **提交的样本：** 2 列。第一列为测试集样本对应序号，第二列为预测的 `label` 结果。

2.2 软件

使用的编程语言为 Python（3.10 及以上版本），所用模块主要包括 numpy、pandas、matplotlib、scikit-learn、tensorflow、keras。

2.3 基本算法策略

2.3.1 卷积神经网络（CNN）

CNN 本身即受启发于对生物视觉系统的研究，可以通过卷积核进行局部感知，能很好地提取到图像的局部特征，非常适合于此类图像分类问题。

但是 CNN 对于大规模、多样化的训练数据的需求较高。如果训练数据集较小或不平衡，可能会导致过拟合或模型泛化能力不足。

2.2.2 AdaBoost 策略

AdaBoost 策略通过对错误分类的样本投入更大的关注（给予更大的权重）来训练多个弱学习器，提升分类准确度。相较于原本的弱学习机，使用 AdaBoost 策略可以显著地提升精度。

但它往往对噪声敏感，可能对异常值赋予过高的权重，这种情况下容易发生过拟合。

2.2.3 堆叠（stacking）

堆叠（或堆栈）是一种集成学习方法，它通过将多个弱学习器集成为元学习器，组合它们的预测结果来得到最终预测。堆叠可以显著提高模型的精确度，训练过程中涉及多个弱学习器的交叉验证。

2.3 代码可用性

所有代码均以 Jupyter Notebook 的形式提供在附件中。

3 初步结果

3.1 数据探索性分析（EDA）

3.1.1 数据概况

首先，我们展示数据形状和数据缺失情况。

```
print("数据形状:", X_train.shape)
print("数据形状:", Y_train.shape)
print("数据形状:", X_test.shape)
print("数据形状:", Y_test.shape)
```

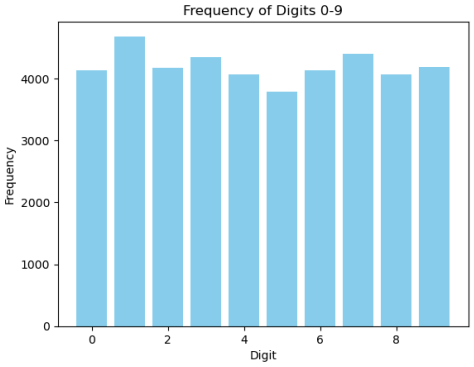
输出结果为

```
数据形状: (42000, 784)
数据形状: (42000,)
数据形状: (42000, 784)
数据形状: (42000,)
```

可以看到，样本未出现整体或部分的缺失。

然后，我们统计训练集中各个标签（数字）的出现频次，并绘制柱状图：

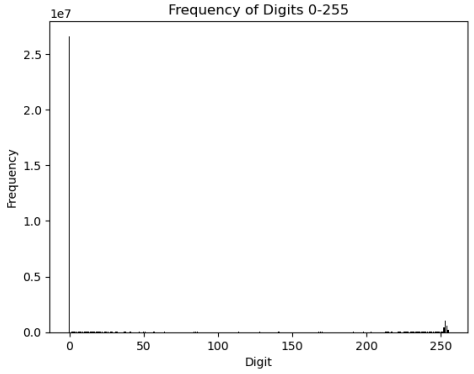
```
0: 4132
1: 4684
2: 4177
3: 4351
4: 4072
5: 3795
6: 4137
7: 4401
8: 4063
9: 4188
```



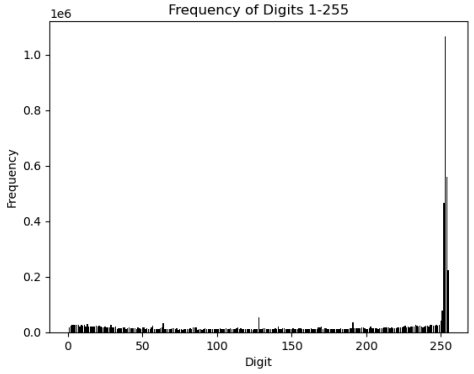
可以看出各个数字的出现频次分布比较均匀。无需过多处理。

3.1.2 图片像素的灰度分布

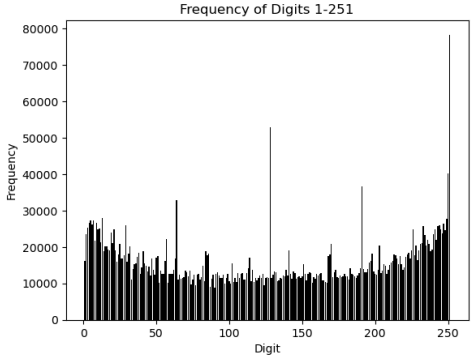
我们再查看一下图片的像素的灰度分布。



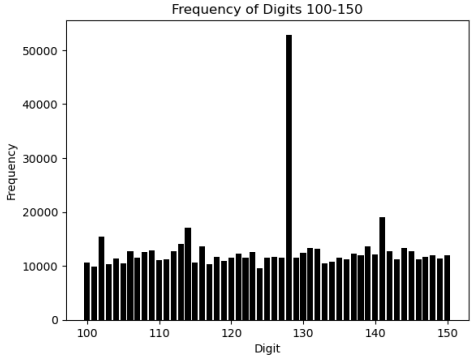
灰度值为 0 的点明显多出几个数量级，影响直方图的观感，下面不考虑这些灰度值为0的点进一步绘图观察。



灰度值 ≥252 的点明显又比其他点多出一个数量级，我们进一步绘图。



100到150中间又存在一个明显突出的值，进一步绘制。



总结:在 0 和 255 附近的灰度值占绝大多数，即从整体上看，图像基本由黑点和亮度极高的亮点构成，是非常“明锐”的。

抽取部分样本，恢复原图，可以看见数字边缘分明，也验证了我们的预期。

从最后一张直方图可以看出，灰度值为128的点数量也相对偏多，这或许提示我们在后续数据处理（如锐化）时可能可以将128作为一个分隔点。

整体来看，这是一个“相当好”的数据集，好到甚至可以无需预处理而直接应用。

基于此，李佩达同学进行了早期探索。

3.2 初步探索

李同学分别运用CNN、AdaBoost+CNN、数据分割后AdaBoost+CNN等方法构建了模型。

3.2.1 CNN

```
1116/1116 ━━━━━━━━━━━━━━━━━━━━ 9s 8ms/step - accuracy: 0.9981 - loss: 0.0095 - val_accuracy: 0.9890 - val_loss: 0.043
Epoch 13/20
...
Epoch 20/20
1116/1116 ━━━━━━━━━━━━━━━━━━━━ 10s 9ms/step - accuracy: 0.9980 - loss: 0.0051 - val_accuracy: 0.9906 - val_loss: 0.04
197/197 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9906 - loss: 0.0462
Test accuracy: 0.9906
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

与预期一致，CNN 算法取得了相当高的准确度。

3.2.2 AdaBoost + CNN

```
...
197/197 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
197/197 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Train Accuracy: 100.00000000%
Test Accuracy: 98.85714286%
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

令人意外的是，加入 AdaBoost 算法框架后出现了过拟合，准确度反而下降了。

3.2.3 数据分割后使用 AdaBoost+CNN

```
Fit called with sample_weight: None
...
197/197 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
197/197 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
197/197 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
Test Accuracy: 89.35%
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

准确度进一步严重降低，可能是因为更严重的过拟合。方法的优化反而引发了更大的问题，单纯堆加高效方法未必能提高模型总体效果。

CNN 作为学习器显得过强，可能使 AdaBoost 的增强效果不明显。更麻烦的是，AdaBoost 可能拘泥于强化学习噪声值，反而减弱了拟合效果。

3.2.4 堆叠（`Digit Recognizer堆栈.ipynb`）

我们使用 2.2.3 中介绍的堆叠方法，训练了 5 个 CNN 基学习器，每个基学习器使用一半的训练集。随后，将它们用 Keras 内置的 `StackingClassifier` 进行堆叠。测试结果表明，堆叠模型取得了目前最好的 accuracy。

```
[19]
# 进行预测并计算准确率
y_pred_train = stacking_clf.predict(X_train_reshaped)
y_pred_test = stacking_clf.predict(X_test_reshaped)
train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)

print("Train Accuracy: {:.8f}%".format(train_accuracy * 100))
print("Test Accuracy: {:.8f}%".format(test_accuracy * 100))
```

MagicPython

```
1116/1116 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step
1116/1116 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
1116/1116 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step
1116/1116 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step
1116/1116 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step
197/197 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
197/197 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
197/197 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
197/197 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
197/197 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Train Accuracy: 99.94117647%
Test Accuracy: 99.25396825%
```

根据上述探索，接下来的优化方向主要是：

- 锐化数据，降噪
- 做数据增强

之后的所有优化均建立在堆叠模型的基础上。

4 模型优化

4.1 锐化

理论上改变卷积核的大小亦可以达到锐化的目的，但也可能带来意想不到的风险（过拟合或梯度消失）。本着“Garbage In，Garbage Out”的考量，决定提前对数据进行处理，这样代码的可解释性也更强。

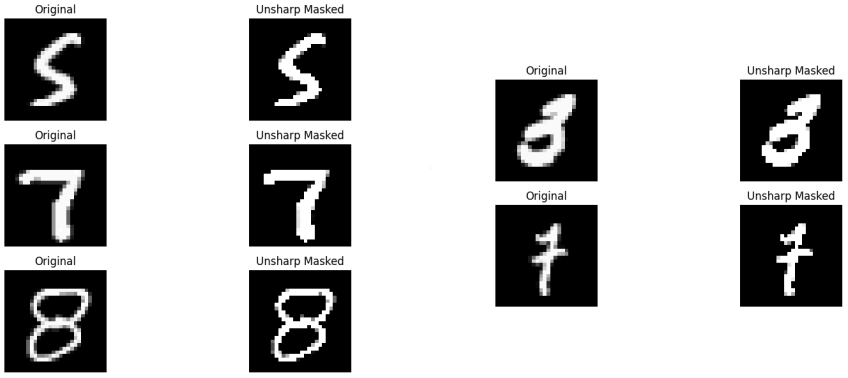
4.1.1 使用Unsharp-Mask(USM)

USM通过放大原始图像与模糊化图像的差异来增强图像的边缘和细节。

```
# 重新调整数据形状
X_train_reshaped = X_train.reshape((-1, 28, 28))
X_test_reshaped = X_test.reshape((-1, 28, 28))
z_test_reshaped = z_test.reshape((-1, 28, 28))

# 转换图像数据类型为 CV_8UC1
X_train_reshaped = X_train_reshaped.astype(np.uint8)
X_test_reshaped = X_test_reshaped.astype(np.uint8)
z_test_reshaped = z_test_reshaped.astype(np.uint8)

def unsharp_mask(image, kernel_size=(5, 5), sigma=1.0, amount=5.0, threshold=0):
    blurred = cv2.GaussianBlur(image, kernel_size, sigma)
    sharpened = float(amount + 1) * image - float(amount) * blurred
    sharpened = np.maximum(sharpened, np.zeros(sharpened.shape))
```

```
    sharpened = np.minimum(sharpened, 255 * np.ones(sharpened.shape))
    sharpened = sharpened.round().astype(np.uint8)
    return sharpened

# 对所有图像进行unsharp_mask锐化处理
X_train_sharpened = np.array([unsharp_mask(image) for image in X_train_reshaped])
X_test_sharpened = np.array([unsharp_mask(image) for image in X_test_reshaped])
z_test_sharpened = np.array([unsharp_mask(image) for image in z_test_reshaped])
```

锐化后的结果（示例）如下：



成效较为显著。

4.1.2 寻找一个函数，将灰度值映射到0和255两个端值附近（Functional_Sharpen_CNN.ipynb）

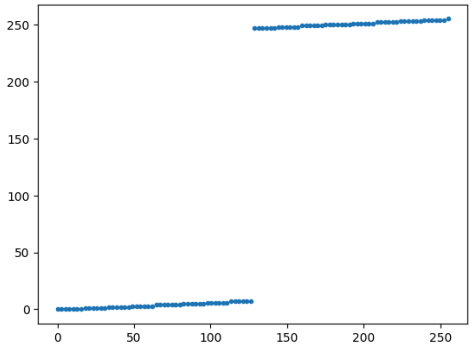
在前期的 EDA 中我们发现，像素的灰度大量地集中在 0，255 左右，且除去上述两类，剩下的灰度数值在128附近形成峰值。我们希望构造一个锐化函数，将 127 及其以下的值映射至 0 附近，将 128 及其以上的值映射至 255 附近，这样或许能保证在不丢失过多特征的情况下同时实现锐化。

```
import math
def sharpen_1(x):
    if x<=127:
        return x//16
    else:
        return (255*15+x)//16

dataX_sharpen=[[] for _ in range(42000)]

for i in range(42000):
    dataX_sharpen[i]=np.array(list(map(sharpen_1,dataX[i])))

dataX_sharpen=np.array(dataX_sharpen)
print(dataX_sharpen[0])

dataX_sharpen.shape
```

在上述代码中，我们使用的函数是：

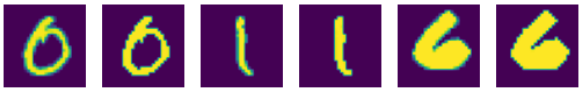
$$y=\begin{cases}\lfloor \frac{x}{16} \rfloor & 0<x\leq 127 \\ \lfloor \frac{255\times 15+x}{16} \rfloor & 128\leq x<255\end{cases}$$

它的映射效果如图：



可以看到它几乎将所有灰度值映射到了0和255附近。

从图片上可以看到，模糊的地方基本消失：



我们测试一下这种锐化后CNN的表现：

（采用函数映射锐化的CNN）：

```
Epoch 20/20
1116/1116 ━━━━━━━━━━━━━━━━━━━━ 18s 16ms/step - accuracy: 0.9854 - loss: 0.0534 - val_accuracy: 0.9905 - val_loss: 0.0383
197/197 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9910 - loss: 0.0493
Test accuracy: 0.9905
```

发现在测试集的正确率上几乎与之前无异。以下是一些反思与改进。

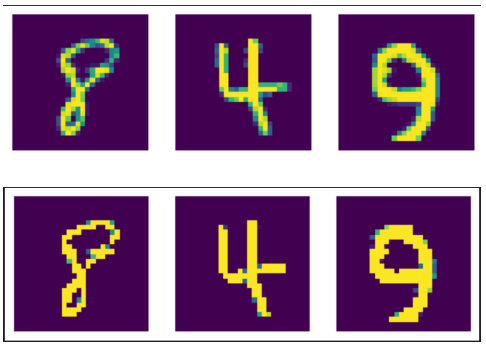
4.1.3 卷积核锐化（`Convolution_Sharpen_CNN.ipynb`）

考虑到上一种锐化方法可能由于过分直接而减少了图像的部分特征，对训练器的训练效果可能也会有部分下降，这里又考虑一种用卷积核锐化的方式。

注意到使用 3*3 卷积核 `[[-1,-1,-1],[-1,9,-1],[-1,-1,-1]]` 对二维图片进行滚动可以有效突出中心值，从而达到突出图片特征的效果，并且这种锐化方式能更多的保留图形原本的特征点，因此考虑用这一卷积核对训练集作预处理，再训练卷积神经网络。这里举对测试集图片进行锐化的代码为例：

```
sharpen_kernel = np.array([[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]]) # 这样的卷积核可以强化图像的中心特征

# 假设 dataX_2D 是包含彩色图像的数组，形状为 (28000, 高, 宽, 3)
test_X_2D_convolution = [[] for _ in range(28000)]

for i in range(28000):
    # 确保每张图像是 uint8 类型，且是彩色图像
    image = test_X_2D[i].astype(np.uint8)

    # 对彩色图像应用锐化滤波
    test_X_2D_convolution[i] = cv2.filter2D(image, -1, sharpen_kernel)
test_X_2D_convolution=np.array(test_X_2D_convolution)
```

对比一下处理后的图片，有一定效果，并且保留了图形的绝大部分轮廓。



我们测试一下采用这种方法锐化后的 CNN 准确率：

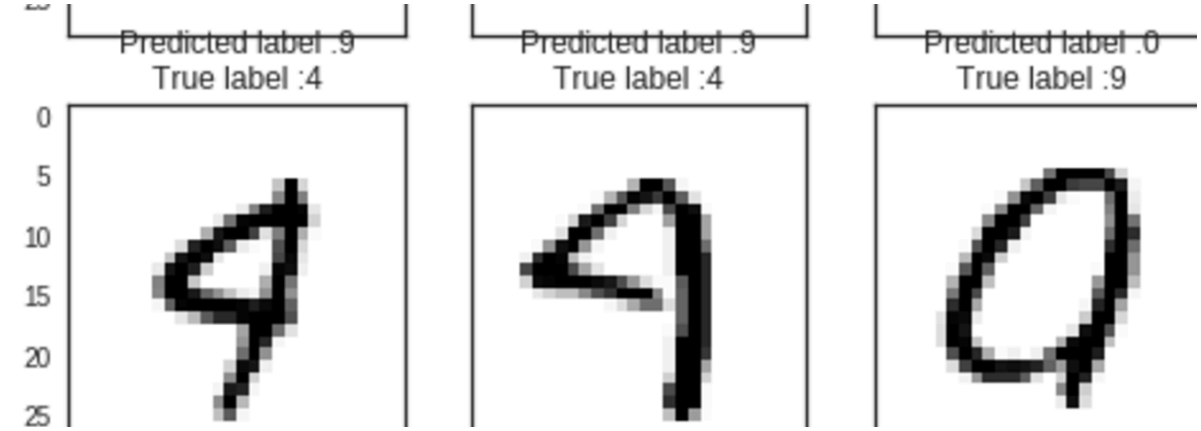
```
Epoch 20/20
1116/1116 ━━━━━━━━━━━━━━━━━━━━ 14s 12ms/step - accuracy: 0.9854 - loss: 0.0533 - val_accuracy: 0.9938 - val_loss: 0.0208
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9915 - loss: 0.0235
Test accuracy: 0.9938
```

相较之前准确率提升了 0.3 个百分点，已是比较显著的提升。

相比于前一种简单直接的锐化方式，采用卷积核锐化能更大限度的保留图像原本的特征点，并且也有不错的特征增强效果，使得后续 CNN 模型在拟合时能够更好获取图像的特征，从而得到更高的 accuracy。

4.2 数据增强

我们发现，在训练集中存在一定量的数字，可能是由于书写潦草导致，即使是人工也难以准确辨认（如下图）。为了增强对这种难以分辨的情况的训练，我们使用了 Keras 的 `ImageDataGenerator` 功能扩大数据集，比如通过适当平移、小幅度旋转、缩放增加数据数量。同时进行小幅度的旋转会有益于提高模型的泛化能力。



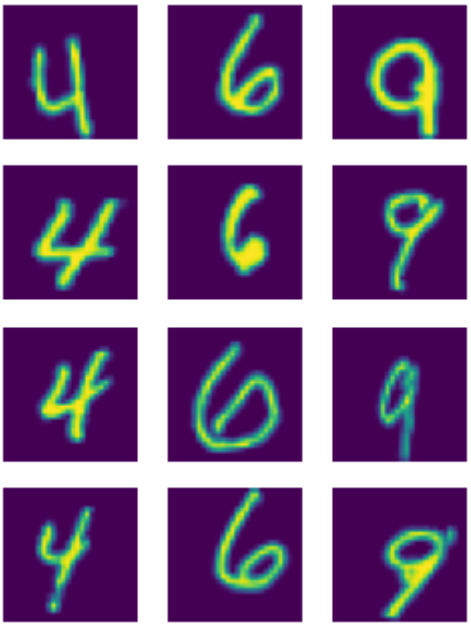
4.2.1 直接进行数据增强（`image_augmentation_with_analysis.ipynb`）

```
# 数据增强
datagen = ImageDataGenerator(
    rotation_range=10,  # 随机旋转范围
    width_shift_range=0.1,  # 水平移动
    height_shift_range=0.1,  # 垂直移动
    shear_range=0.1,  # 剪切变换程度
    zoom_range=0.1,  # 放大/缩小
    horizontal_flip=False,  # 水平翻转
    fill_mode="nearest",  # 填充模式
)
```

选取了 4、6、9 三个可能难以辨认的数据，指定 `batch_size = 4` 进行增强，使它们的样本容量增加 4 倍。

```
def augment_specific_digit(digit, X_train_reshaped, y_train, batch_size=4):
    y_train_digit = np.array(np.where(y_train == digit)).T
    X_train_reshaped_digit = X_train_reshaped[np.where(y_train == digit)]
    print(y_train_digit.shape)
    allcount = y_train_digit.shape[0]
    count = 0

    datagen.fit(X_train_reshaped_digit)

    # datagen生成数据，并加入训练集
    for X_batch, y_batch in datagen.flow(X_train_reshaped_digit, y_train_digit,
batch_size):
        count += 1
        if count % 100 == 0:
            print("Processing: ", count, "/", allcount)
        X_train_reshaped = np.concatenate((X_train_reshaped, X_batch))
        y_train = np.concatenate((y_train, [digit] * batch_size))
        if count >= allcount:
            print("Processing: ", count, "/", allcount)
            break

    return X_train_reshaped, y_train
```

部分增强后的数字如图所示。



使用增强后的数据集进行本地测试，所得 accuracy 较增强前反而有所降低。

- 评估Stacking模型：

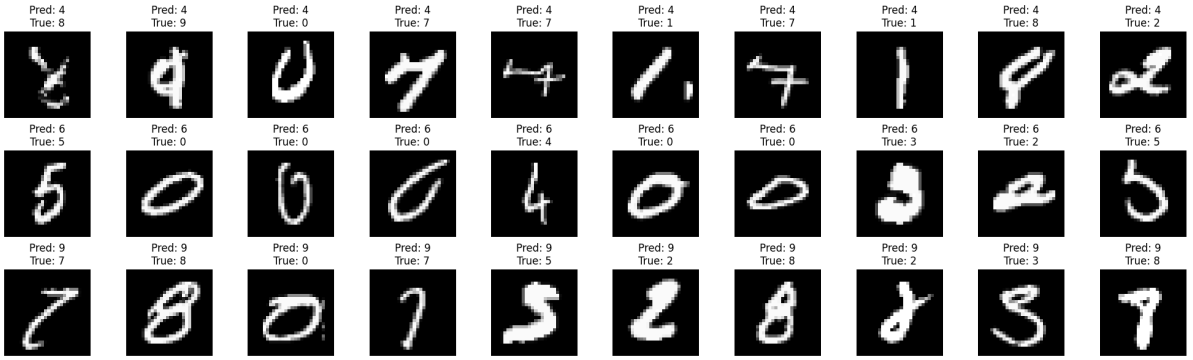
```
# 进行预测并计算准确率
y_pred_train = stacking_clf.predict(X_train_reshaped)
y_pred_test = stacking_clf.predict(X_test_reshaped)
train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)

print("Train Accuracy: {:.8f}%".format(train_accuracy * 100))
print("Test Accuracy: {:.8f}%".format(test_accuracy * 100))
```

✓ 36.0s

```
2433/2433 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step
2433/2433 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step
2433/2433 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step
2433/2433 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step
2433/2433 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step
197/197 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
197/197 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
197/197 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
197/197 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
197/197 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
Train Accuracy: 99.14071029%
Test Accuracy: 97.12698413%
```

我们推测这可能是对 4、6、9 的过拟合导致的。对预测结果进行分析，发现当模型将数据预测为 4、6、9 时，出现的错误较多、accuracy 较低，如下图，说明模型有较大的概率将其他的数字预测为 4、6、9，即模型对 4、6、9 过拟合。这验证了我们的猜想。

```
Train accuracy for guessing 4:  0.9968699281242754
Train accuracy for guessing 6:  0.9900918047469771
Train accuracy for guessing 9:  0.9760610552901774
Test accuracy for guessing 4:  0.9780564263322884
Test accuracy for guessing 6:  0.9214175654853621
Test accuracy for guessing 9:  0.8440860215053764
Train accuracy for labeling 4:  0.9990124891083357
Train accuracy for labeling 6:  0.9995478948855608
Train accuracy for labeling 9:  0.999493984032385
Test accuracy for labeling 4:  0.9920508744038156
Test accuracy for labeling 6:  1.0
Test accuracy for labeling 9:  0.9984101748807631
```



4.2.2 卷积核锐化 + 数据增强

考虑在 4.1.3 采用卷积核锐化的基础上再采用 `ImageDataGenerator` 进行数据增强：

对数据增强后的模型进行测试：

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9945 - loss: 0.0193
Test accuracy: 0.9950
```

正确率又有了小幅度的提升，在 Kaggle 测试平台上面提交我们的预测结果：

这个模型在测试集上的正确率达到了 99.321%。

以上是使用了卷积核锐化进行特征增强，以及ImageDataGenerator进行随机旋转平移数据增强手段的结果。

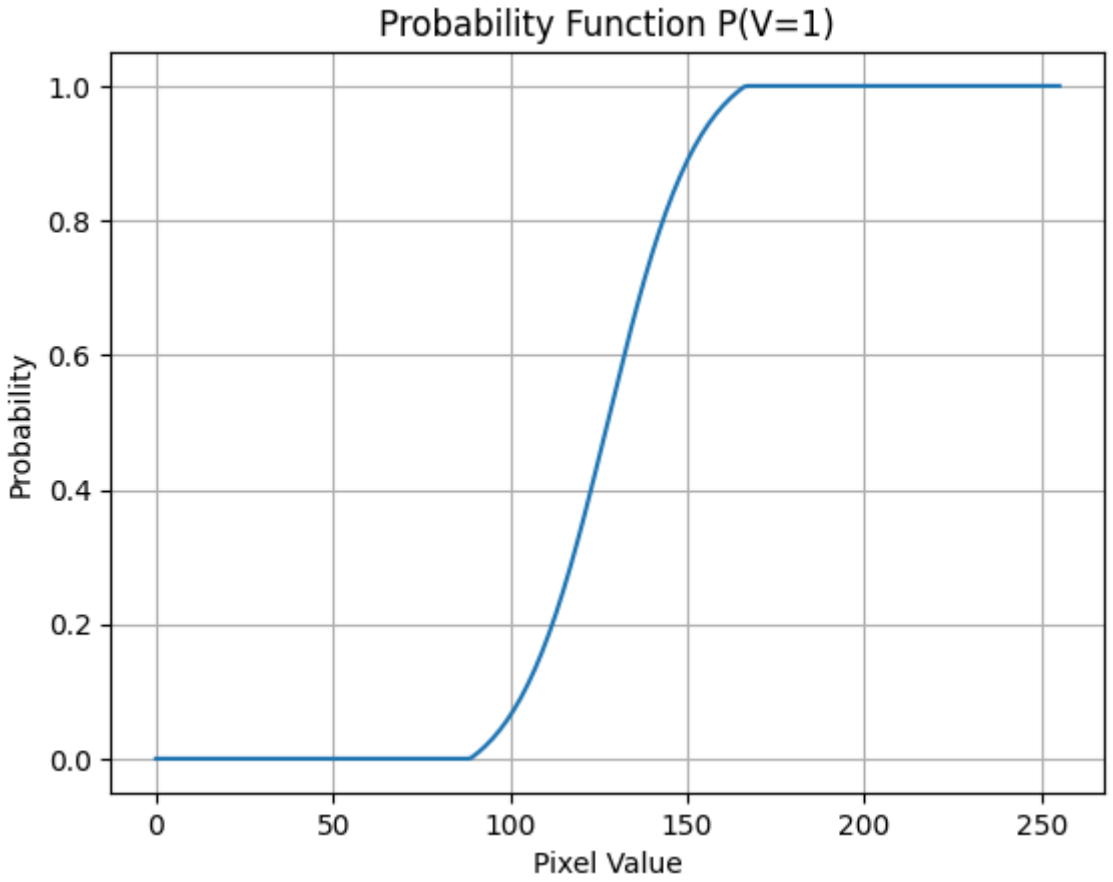
4.3 Random Boolean Enhancement (`boolean_enhancement.ipynb`)

受到 4.1 中映射函数锐化和 4.2 中数据增强思想的启发，我们提出了一种结合了锐化和数据增强思想的优化方法，命名为 Random Boolean Enhancement。

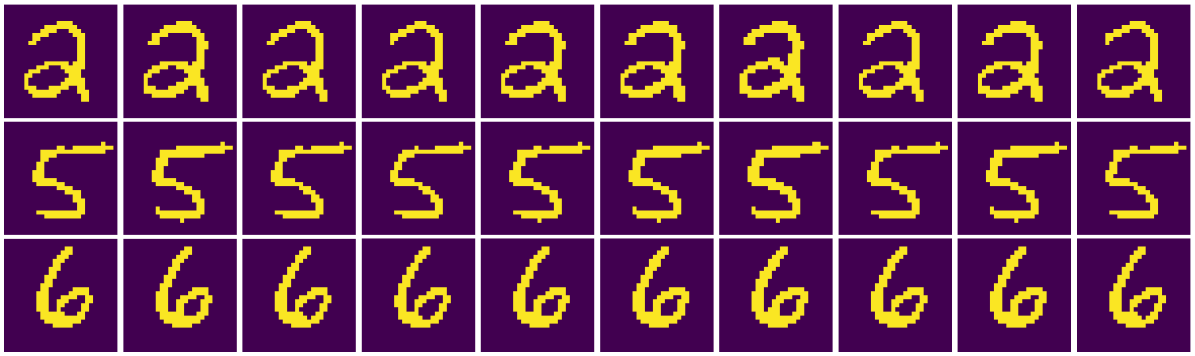
该方法的主要内容是将图片中的每个像素点都转变为 0 或 255，进而压缩到 0 或 1（False 或 True）。由于这种简化必然损失信息，因此考虑根据像素点原有的灰度值，对像素点赋予一定的转变为 0 或 1 的概率。这个概率基于 Sigmoid 函数（以此达到锐化目的），但进行了一定的 “抹平” 处理，如下：

```python
def probability_function(x):
    alpha = 0.05
    p = (1 + 2 * alpha) * (1 / (1 + np.exp(20 * (-x/255 + 1/2)))) - alpha
    p = np.clip(p, 0, 1)
    return p
```

$$
\begin{aligned}
&P(\text{transformed value} = 1 | \text{pixel value} = x) \\
&= \max\{0, \min\{1, \left[1.1(1 + \exp(20(-\frac{x}{255} + 0.5)) - 1\right]^{-1}\}\}
\end{aligned}
$$



下图分别展示了一部分 Random Boolean Enhancement 后的结果和各个随机变换叠加后的结果。



此过程随机重复 10 次，得到一个 10 倍大小的训练集。预测测试集时，也将测试集扩大一定倍数（本地测试 10 倍，Kaggle 测试集 50 倍），对于每个原始图片，使用投票的方法取多数预测。该方法在 Kaggle 上提交，accuracy 为 99.375%，这在到此为止的模型中是效果最好的。

扩大倍数过程中，使用课上介绍的 numpy 向量化运算技术可以有效提高效率。然而，由于 Keras 并不原生支持 boolean 矩阵的卷积运算，亦没有相关的激活函数和损失函数，因此最后还是作为 float 类型处理了，训练神经网络的时间消耗不减反增，这是此方法的一大瓶颈。同时，我们有理由推测，继续扩大训练集规模将进一步提升模型预测的精度。事实上，在此前的尝试中，继续扩大训练集到 50 倍，仅基学习器就达到了 99.9% 的 test accuracy，但由于消耗时间过长，又由于时间有限，不得不取了规模较小的 10 倍。

4.4 其他特征提取（`other_features_extraction.ipynb`）

经过反思之后，我们在思考排除掉一些人类手写带来的误差，为什么一张对于人类来说再简单不过的手写照片，对于机器来说会是一个天大的难题呢？我们又能否利用人类识别数字的经验，来解决问题呢？

经过思考我们提出了以下思路：

- 寻找一个程序识别手写数字图像中的圆，通过引入圆圈数目这个新特征，提升识别效果。
- 寻找一个程序识别手写数字中的直线，通过直线的数目来提升识别效果。
- 通过识别数组图像的长宽，来提升机器的识别效果。
- 由于不同数字的笔画数目不一样，通过计算数字的所有像素格数值的加和，提升识别效果。

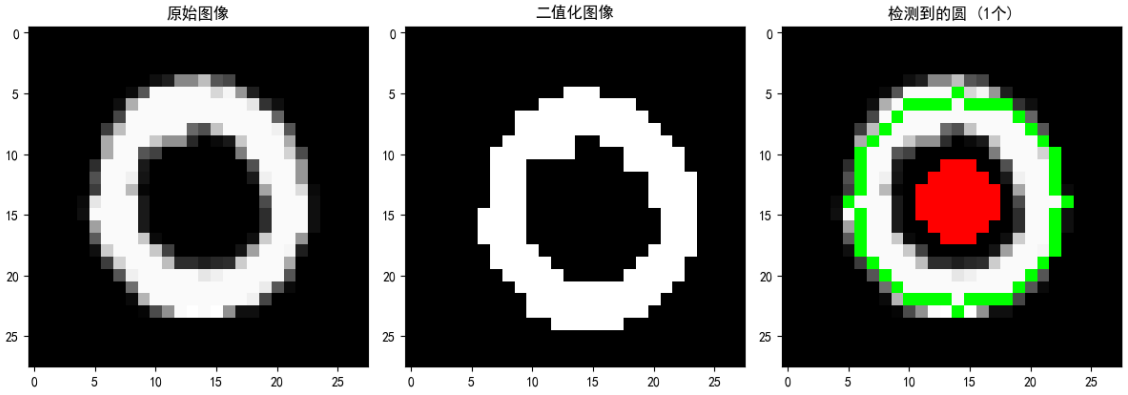
4.4.1 利用霍夫曼变换写寻找手写数字图像中的圆圈个数

```
def detect_circles(image):
    # 1. 图像预处理操作，这里省去
    # 2. 形态学操作改进
    # 3. 多尺度高斯模糊 ——这里其实很重要，要想能识别出大小不同的圆，多尺度的高斯模糊参数功不可没
    # 4. 改进的圆形检测参数，过于冗长，是不断试错的出的结果
    # 5. 合并重复的圆
    if circles_list:
        circles_array = np.array(circles_list)
        # 根据圆心位置合并相近的圆
```

```
        final_circles = []
        used = set()
        for i, (x1, y1, r1) in enumerate(circles_array):
            if i in used:
                continue
            current_circle = [x1, y1, r1]
            used.add(i)
            # 检查其他圆是否与当前圆重叠
            for j, (x2, y2, r2) in enumerate(circles_array):
                if j in used:
                    continue
                # 计算圆心距离
                dist = np.sqrt((x1-x2)**2 + (y1-y2)**2)
                if dist < max(r1, r2):
                    used.add(j)
            final_circles.append(current_circle)
        return len(final_circles)
    return 0
# 存储每张图片中的圆圈个数
# 处理所有图像
```

程序的识别效果如下：

- 成功识别出0



- 测试了一下程序的准确性，如下图片程序成功识别出了图像中的26个（全部）圆

- 但是程序也有一些问题，比如说：8里面的两个圆没办法同时识别出来，下面是计算机程序识别出含有1个圆的样例

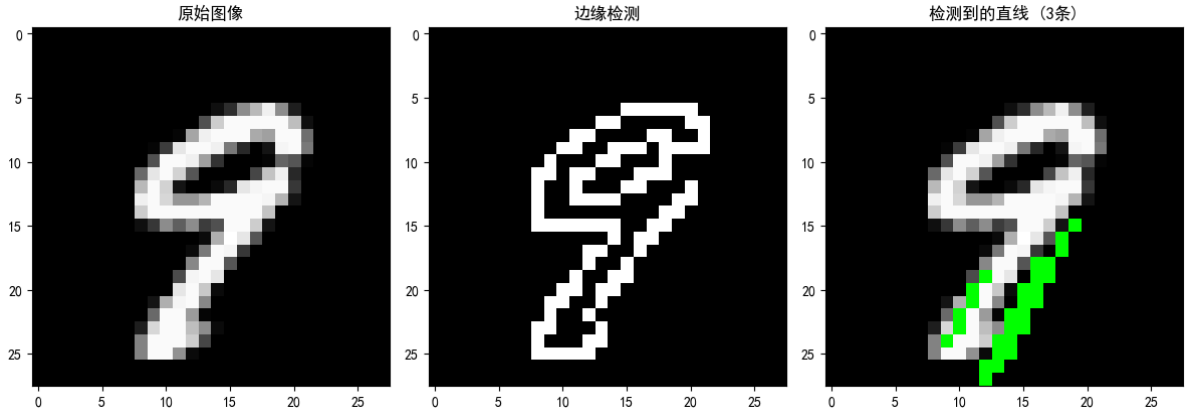
- 经过思考，这很可能涉及到高斯模糊和边缘检测算子的配合。高斯模糊的正向作用是降噪，但是过于强大的高斯模糊，很有可能成功模糊掉了数字8里面的小圆，所以边缘检测算子和高斯模糊的参数可以进一步优化以取得更加优秀的识别结果。

4.4.2 利用霍夫曼变换写寻找手写数字图像中的直线个数

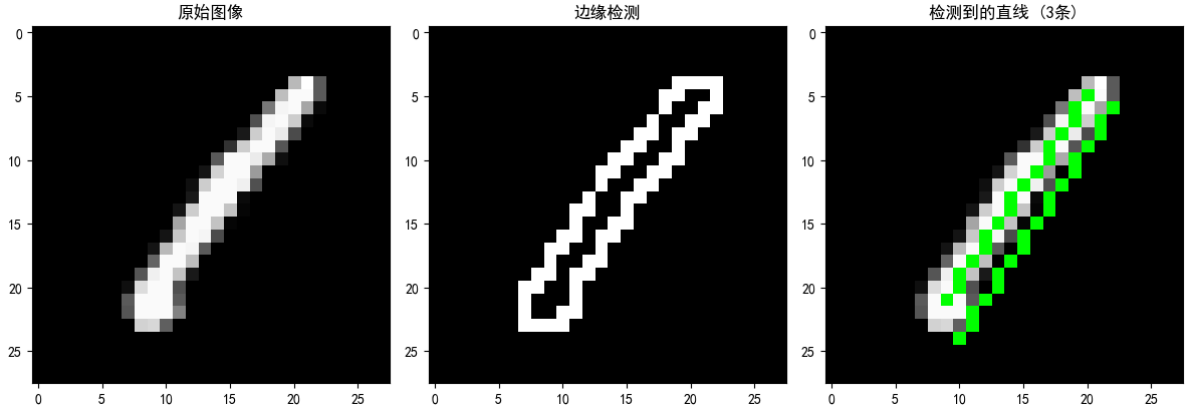
```
# 存储每张图片中的直线个数和斜率
def detect_lines_in_image(image):
    # 1. 图像预处理
    # 确保图像类型正确并归一化
    # 增强对比度
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    gray = clahe.apply(gray)
    # 二值化处理
    # 应用形态学操作
    # 高斯模糊
    blurred = cv2.GaussianBlur(binary, (3, 3), 0)
    # 2. 边缘检测 - 使用多个Canny阈值
    edges1 = cv2.Canny(blurred, 20, 60)
    edges2 = cv2.Canny(blurred, 30, 90)
    edges = cv2.bitwise_or(edges1, edges2)
    # 3. 使用概率霍夫变换检测直线 - 多次检测合并结果
    all_lines = []
    # 使用不同参数进行多次检测
    params = [
        {'threshold': 20, 'minLineLength': 8, 'maxLineGap': 3},
        {'threshold': 25, 'minLineLength': 10, 'maxLineGap': 5},
        {'threshold': 15, 'minLineLength': 5, 'maxLineGap': 2}
    ]
    for param in params:
        lines = cv2.HoughLinesP(
            edges,
```

```
            rho=1,
            theta=np.pi/180,
            threshold=param['threshold'],
            minLineLength=param['minLineLength'],
            maxLineGap=param['maxLineGap']
        )
        if lines is not None:
            all_lines.extend(lines)
    # 4. 过滤和合并相似的线段
    if len(all_lines) > 0:
        filtered_lines = []
        all_lines = np.array(all_lines)
        # 计算每条线的长度和角度
        for line in all_lines:
            x1, y1, x2, y2 = line[0]
            length = np.sqrt((x2-x1)**2 + (y2-y1)**2)
            angle = np.arctan2(y2-y1, x2-x1) * 180 / np.pi
            # 只保留长度大于阈值的线段
            if length > 5:
                filtered_lines.append(line)
        return filtered_lines
    return None
# 处理所有图像
# 转换为数组
# 输出统计信息
# 可视化函数
```

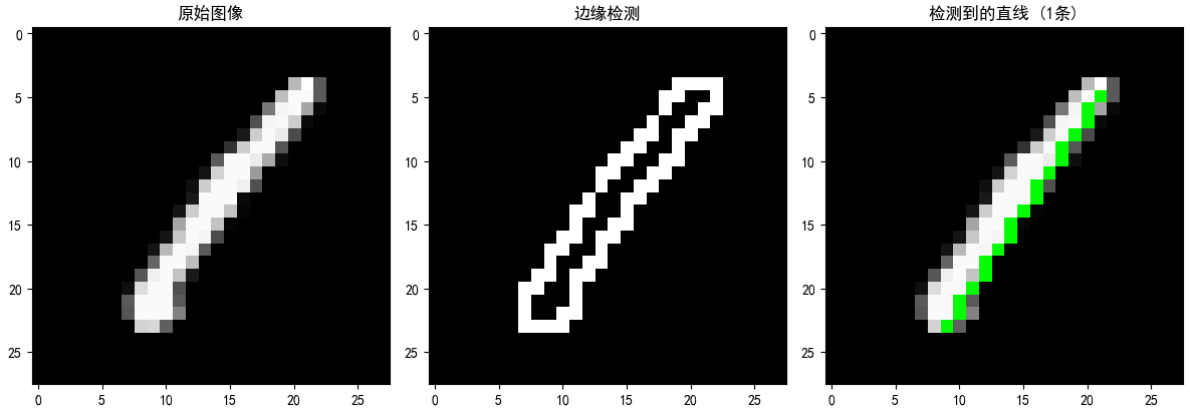
图像识别效果展示：



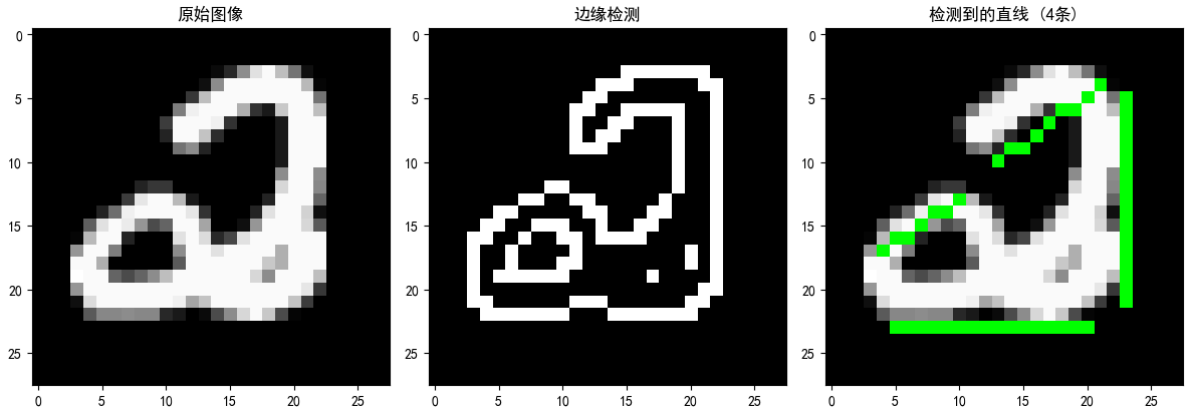
同样地这个程序也存在一定的问题：无法融合相临近的直线，如下图所示：



改进措施：利用斜率和中点距离识别相临近的直线，下面是识别效果：



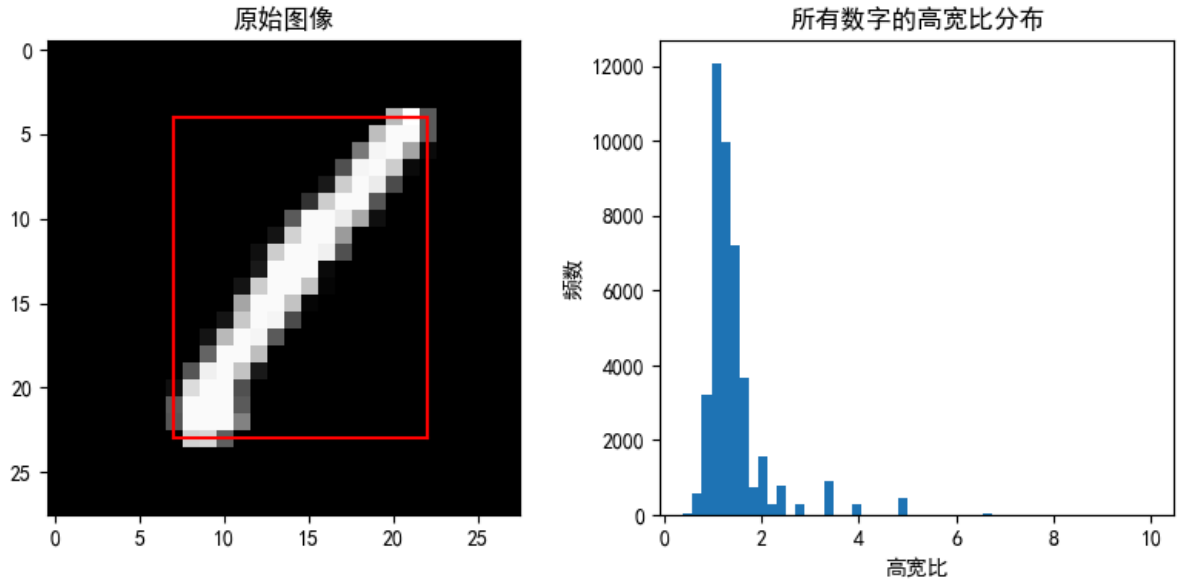
但是还是有个别图像的识别效果不尽如人意，如下图所示：

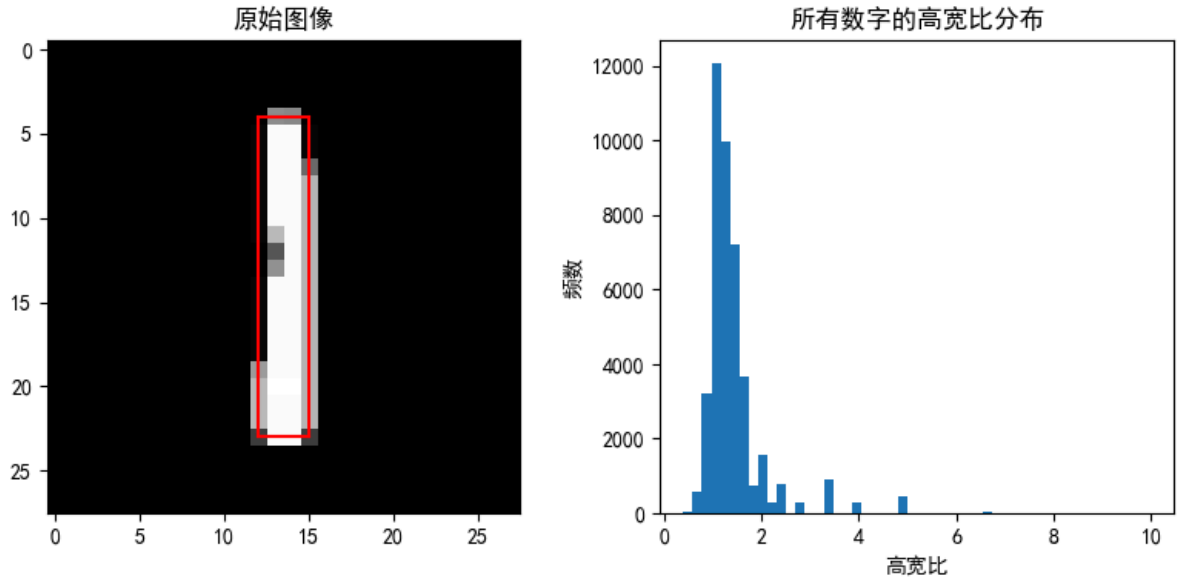


4.4.3 识别图像的长宽

```
def get_digit_dimensions(image):
    rows = np.where(np.any(image > 20, axis=1))[0]
    cols = np.where(np.any(image > 20, axis=0))[0]
    if len(rows) == 0 or len(cols) == 0:
        return 0, 0  # 如果没有检测到数字，返回0
    # 计算高度和宽度
    height = rows.max() - rows.min() + 1
    width = cols.max() - cols.min() + 1
    return height, width
# 存储所有图像的高度和宽度
heights = []
widths = []
height_width_ratios = []  # 高宽比
# 处理所有图像
for image in train_sample_reshape:
    h, w = get_digit_dimensions(image)
    heights.append(h)
    widths.append(w)
    # 计算高宽比（避免除以零）
    ratio = h / w if w != 0 else 0
    height_width_ratios.append(ratio)
```

这个程序不涉及到识别，但是我们很明显能够发现一个问题，如下图所示

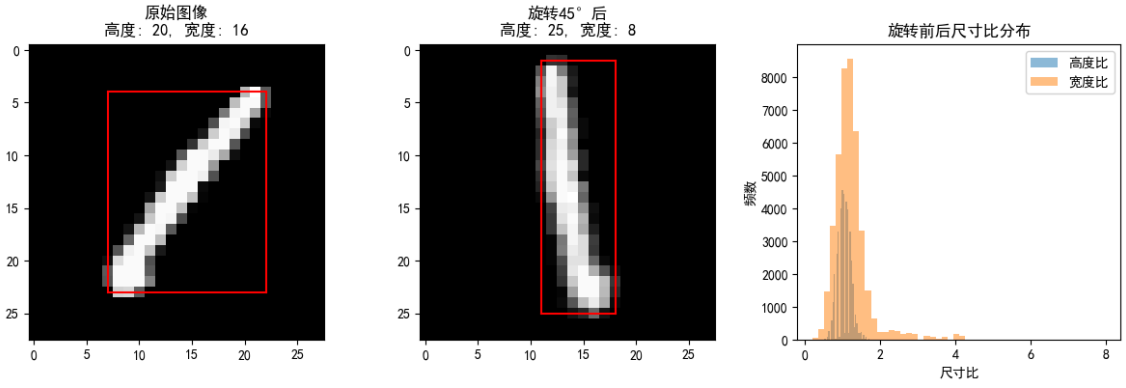




很明显，两个 1 的长宽比差异过大。

解决方案：

- 计算逆时针逆时针旋转45°之后的长和宽，效果如下



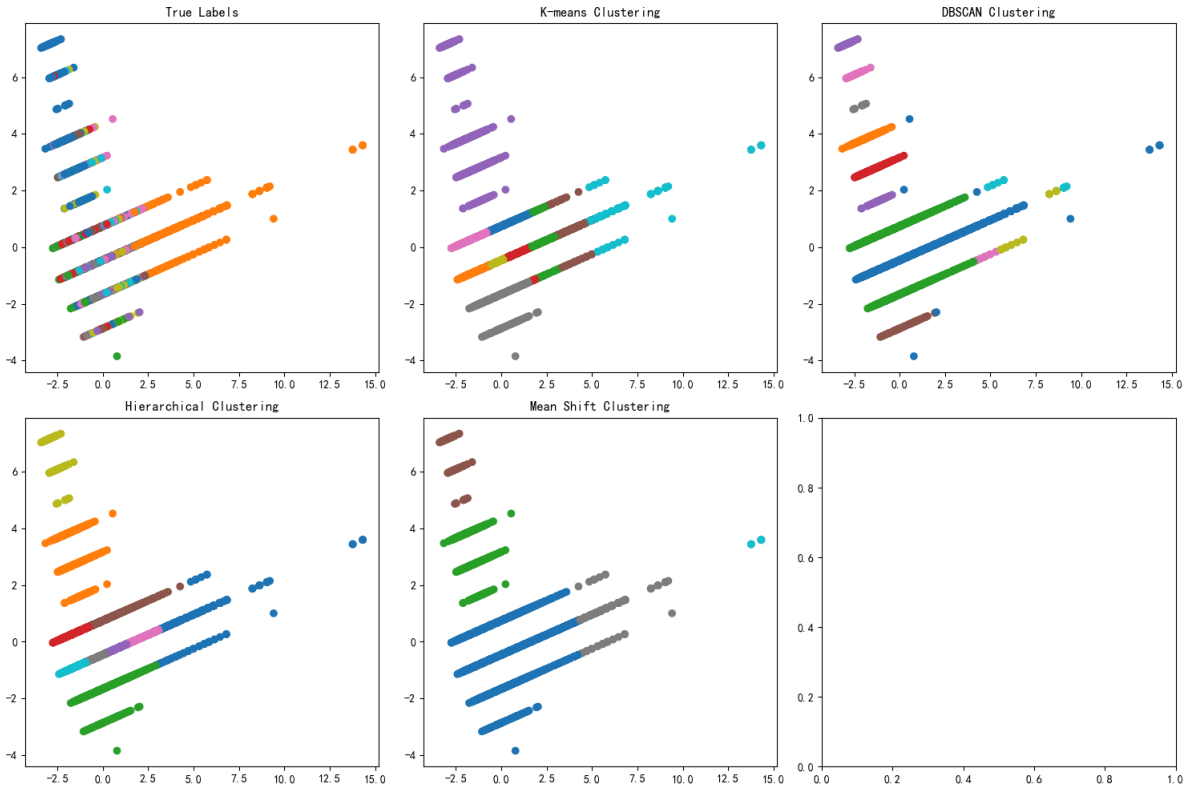
- 后续用旋转前后的长宽比之和以及旋转前后的长宽比乘积进行建模

4.4.4 识别图像所有像素格数值加和

这一步骤发现由于不同人之间的书写差异过于巨大，同一数字的像素数值加和过于巨大，反而不同数字之间差异很小，因此舍弃此特征。

4.4.5 建模

首先尝试了用新建立的特征进行聚类，尝试了K-means，层次聚类，密度聚类，Mean Shift四种聚类方法，想观察一下我们的新特征的效果。可视化如下。



- 很明显，综合聚类可视化图像，以及 ARI、轮廓系数，聚类效果根本不行，仅仅依靠这四个特征是不可能实现很好的图像识别效果。
- 此时我们可以选择继续优化上述参数，但是碍于对于霍夫曼变换的认识不足、时间及算力有限，没能继续进行下去。
- 我们同样也可以选择其他模型，例如“优化”我们的 CNN。

CNN 融合新建立的四个特征进行建模：

- 在全连接层，我们将四个特征融合到了 CNN 模型之中，但是结果不尽如人意，Kaggle 上提交的正确率仅有98.48%。
- 随后我们将未引入其他特征的 CNN 和引入四个特征的 CNN 进行了 Stacking，Kaggle 上提交的正确率为98.89%。

4.4.6 总结

这部分工作为识别图像提供了新思路——**利用人类识别图像的经验**（图像中的圆圈个数，直线个数，长和宽）。尽管效果不尽如人意，但可能蕴含着对计算机视觉的更深入理解。对霍夫曼变换的更加深入认识能够帮助我们获得更加优异的参数，取得一个满意的结果。可能的改进措施有：

- 调整参数，尤其是识别圆圈和直线的程序，这部分效果不尽如人意。
- 进行特征工程：
    - 将不同的参数交互到一起，取得更有利于机器学习的参数，但是这部分依赖于经验。

    - 进行特征 PCA 降维，筛选重要的特征。
- 优化建模的流程：
    - 换一种 CNN 结构的风格，比如使用 ResNet 风格的结构。
    - 可以试一下使用多层感知机处理自己加入 CNN 的四个特征。

5 讨论

总结

对一个图像识别的任务而言，CNN是一种优选策略。我们以CNN为思考的基点，进行了数据处理，通过如下手段实现了的性能优化。

1. 数据处理：
    - 图像锐化。采用了多种方式对输入数据进行图像锐化。例如采用类 sigmoid 函数进行中间灰度值的转化，以提高图像中重点像素的识别率，从而提升性能。同时也从CNN的卷积层着手，在卷积阶段提升原始信息保留率。
    - 数据增强。分析尝试阶段结果，使用 Keras 工具对特定类型数据进行定向的数据扩增，从而提升分类器的准确性和泛化能力。
    - 经过以上测试，我们最终得到了 Random Boolean Enhancement ，一种结合锐化和数据扩增的数据处理方式，作为我们小组最优的数据处理方式。
2. 特征提取：
    - 尝试提取数据集中更丰富的信息，来辅助机器对手写数字的识别。例如基于霍夫曼变换，提取图形中的圆圈或直线的特征。甚至分析图像长宽，灰度高于阀值像素点总数等信息。
3. 模型优化：
    - 尝试使用Adaboost方式进行训练，但遇到了以下问题。一，所谓"弱分类器"分类精度本就较高，加重下一步的稍强分类器对于错误的过度关注，加之错分类图片质量较低（人都难以分辨），使得模型对于此类噪声更加敏感，导致模型鲁棒性降低。二，模型过拟合现象严重。这两个问题可能是相互联系的。
    - 使用堆栈方法，获得了目前小组最好的模型。

反思与展望：

我们通过卓有成效的工作，使最好评分达到了99.375%。在参考kaggle平台上的部分报告后，认为有如下原因限制了正确率的进一步提升：

- 硬件条件有限：CNN模型中卷积层的大小以及学习率等参数有进一步优化空间，但限于算力有限，难以优化。
- 对做特征工程的经验不足：小组成员通过不同的手段完成了特征工程，虽然都展示出来优秀的性能，但是最终没能通过统一的思路将不同的特征组合起来，以实现更加优异的模型性能。
- 训练集较小：尽管采用了多种数据增强手段，42000个训练集相比28000个预测集还是过小。kaggle上的报告也指出若想达到99.8%的正确率，需要导入MNIST的70000个训练集训练模型，或者直接用KNN在原MNIST数据集上直接获取答案。
- 部分图片书写过于潦草：在检查错误样本时，发现部分图片（在4.2中有所展示）混淆性过大，人类也难以分辨。

在本次期中项目中，通过在模型选择、超参调优、特征工程方面的不懈努力和执着探索，我们取得了优秀的模型效果，对于计算机视觉有了更加深刻的理解。在未来，希望在硬件加持下，在对模型本身、对特征工程有了更加深刻的理解后，我们能够取得更加优异的成果。

小组成员和分工

- 朱瑾煜（2401111689）：完成了 Random Boolean Enhancement 的设计和实现（4.3）、数据增强（4.2.1），撰写了报告中 4.3、4.2.1 和 2.3 部分，整合组员代码并对实验报告进行大幅修改。
- 李佩达（2410301207）：完成了CNN，AdaBoost，stacking的实现，锐化算法的调试挖掘，unsharp_mask代码的调优实现，数据增强部分代码的实现。
- 谢昊宸（2400010622）：完成了数据探索性分析（EDA）部分，设计了锐化方案（4.1.2）与（4.1.3），并撰写报告中的相应内容。
- 罗 彪（2410117113）：撰写报告初始框架和部分内容。
- 李萌恩（2110306222）：完成了其他特征提取部分（图像中的圆圈个数，直线个数，长和宽）的代码设计和报告的撰写（4.4）。
- 潘 远（2410305335）：为同学的想法提供了部分支持复现，协助完成了部分报告的撰写。在本项目中主要处于学习地位。