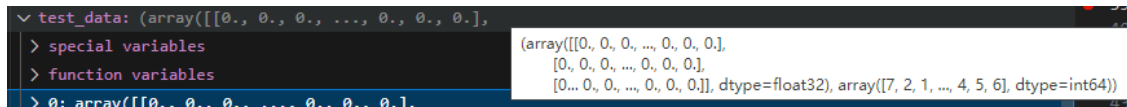


1 目标

构建一个自己手写的数据集，作为测试集，测试神经网络的性能。

2 步骤

1. 数据处理：将图片转化为28*28的大小（同时进行灰度化，归一化等）
2. 格式转换：转为mnist数据集的格式。这一步将图片转为下面这样，第一个



```
> test_data: (array([[0., 0., 0., ..., 0., 0., 0.],  
> special variables (array([[0., 0., 0., ..., 0., 0., 0.],  
> function variables [0., 0., 0., ..., 0., 0., 0.],  
> 0: array([[0., 0., 0., ..., 0., 0., 0.], [0... 0., 0., ..., 0., 0., 0.]], dtype=float32), array([7, 2, 1, ..., 4, 5, 6], dtype=int64))
```

3. 读取并测试：将原本的 `mnist_loader.py` 改为加载自己的测试集，并运行 `test1.py` 进行测试

3 数据处理

灰度化、转化为28*28的大小、归一化并反相（因为输入为白底黑字，若为黑底白字就不需要）：

```
1 img = Image.open(os.path.join(root, filename)).convert('L')  
2 resized_img = img.resize((28, 28))  
3 normalized_arr = 1-arr.astype(np.float32) / 255
```

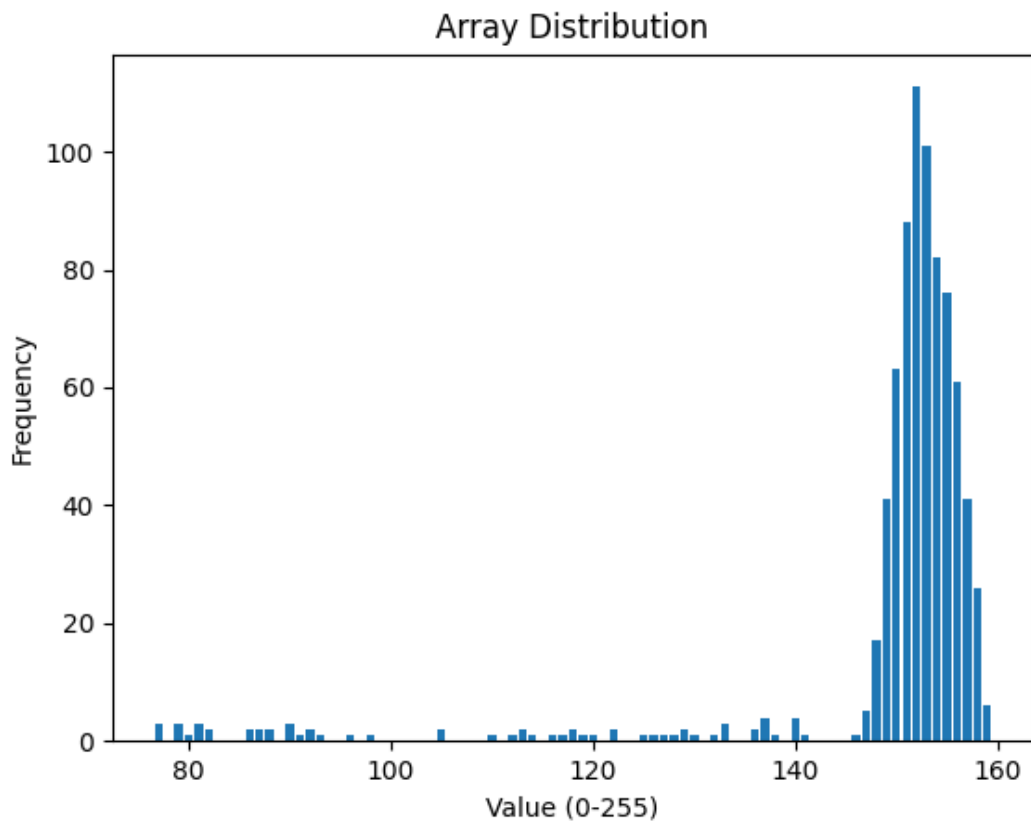
不过这样生成的图片是灰度的，而不是纯黑白的。实验证明，准确率比较低，因此仿照官方测试集进行处理。

对于准确率来说，数据处理的怎么样很重要。如果数据处理的时候只进行灰度化，识别结果会比较差，准确率最高为18%。如果加一点对比度，结果会有点提升，能到23%左右，但是拉不开区别。

若进行了纯黑纯白化，即通过设置阈值，在阈值以上的全部设置为255，阈值以下的全部设置为0，那么图片将变为黑底白字的只有两个极端的图片，如下图，此时识别结果就会好很多，最高能达到40%。

```
1 arr = np.array(resized_img)  
2  
3 # 将数据拉平成一维数组  
4 arr_1 = arr.flatten()  
5  
6 # 计算每个元素的出现次数  
7 counts = np.bincount(arr_1)  
8 # 找出出现次数最多的元素  
9 max_count = max(counts)  
10 most_frequent_elements = np.where(counts == max_count)[0][0]  
11 # 设置阈值和区间  
12 intervals=40  
13 threshold_1 = most_frequent_elements-int(intervals/2)  
14 threshold_2 = most_frequent_elements+int(intervals/2)  
15  
16 # 调整阈值，将之绝对化为0和255，输出画面可变为纯黑白  
17 arr[(arr > threshold_1) & (arr < threshold_2)] = 255  
18 arr[arr < threshold_1] = 0
```

至于为何设置阈值和区间，通过下图的像素分布可以看出，有很大一部分的像素深度位于一个区间，而这个区间则是背景像素所在的区间（背景像素占比最多）。

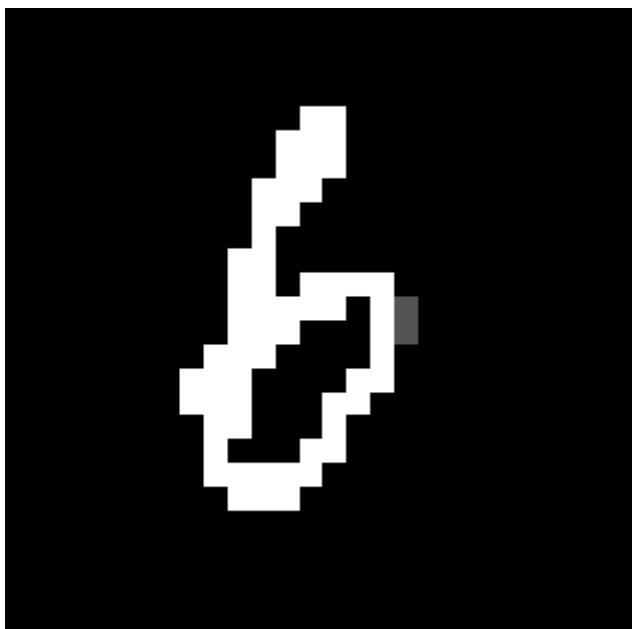


如果简单设置一个阈值，则可能出现如下结果：

```
1 threshold=127
2 arr[arr > threshold] = 255
3 arr[arr < threshold] = 0
```



同样这张图片，在前面可以被调整成下面这样，差别很大。



4 格式转换

输出原本数据集的数组形状可以得知，原本的数据集形状是（10000，784），这意味着要把我们从图片转化而来的（28，28）数组展平，而标签数据已经通过文件夹名字获取（这里的前提是，我们将各个数字放在其对应的文件夹中，完成分类，例如数字1就放在文件夹“1”中）。

```
1 # 图片数据，归一化数据拉平成一维数组
2 flattened_arr = normalized_arr.flatten()
3 array_data.append(flattened_arr)
4
5 # 标签数据
6 label = get_label(root)
7 label_data.append(label)
8
9 label_data_out=np.array(label_data, dtype=np.int64)
10 array_data_out=np.array(array_data, dtype=np.float32)
```

然后我们将这些数据保存成pkl:

```
1 def save_as_pkl_gz(data_list, file_path):
2     with gzip.open(file_path, 'wb') as f:
3         pickle.dump(data_list, f)
4
5 def save_images(images_data, label_data, directory):
6     os.makedirs(directory, exist_ok=True)
7     for i, (image, label) in enumerate(zip(images_data, label_data)):
8         sub_dir = os.path.join(directory, label)
9         os.makedirs(sub_dir, exist_ok=True)
10        image.save(f'{sub_dir}/image_{label}_{i}.png')
```

5 读取并测试

本书例子里面，mnist数据集的格式是mnist.pkl，里面直接把训练集和测试集都包括进去了

```
1 training_data, validation_data, test_data = pickle.load(f, encoding="latin1")
```

我们要做的就是：转换我们自己的图片数据为pkl，并作为test_data。

在mnist_loader.py 修改这个函数代码，将测试集替换成我们自己的：

```
1 def load_data():
2     f = gzip.open('mnist.pkl.gz', 'rb')
3     training_data, validation_data, test_data = pickle.load(f,
4         encoding="latin1")
5     f.close()
6
7     f_mydata = gzip.open('mydata.pkl.gz', 'rb')
8     test_data = pickle.load(f_mydata, encoding="latin1")
9     f_mydata.close()
10
11     return (training_data, validation_data, test_data)
```

最终，在使用官方训练集，我们自己的测试集进行测试的情况下，结果如下。其中0和9的识别率降低，推测与写法以及7和9易混淆有关。

```
1 [(4, 0), (5, 0), (2, 0), (7, 0), (0, 0), (7, 0), (9, 0), (0, 0), (4, 0),
2  (1, 1), (6, 1), (6, 1), (1, 1), (1, 1), (5, 1), (1, 1), (5, 1), (5, 1),
3  (1, 2), (2, 2), (2, 2), (2, 2), (2, 2), (1, 2), (2, 2), (6, 2), (1, 2), (8,
4  2),
5  (5, 3), (3, 3), (3, 3), (3, 3), (3, 3), (5, 3), (5, 3), (4, 3), (1, 3),
6  (4, 4), (5, 4), (0, 4), (4, 4), (4, 4), (9, 4), (4, 4), (6, 4), (9, 4), (8,
7  4),
8  (1, 5), (5, 5), (1, 5), (5, 5), (5, 5), (5, 5), (5, 5), (1, 5), (5, 5), (5,
9  5),
10 (1, 6), (6, 6), (1, 6), (5, 6), (6, 6), (6, 6), (6, 6), (6, 6), (4, 6), (5,
11 6),
12 (1, 7), (2, 7), (2, 7), (7, 7), (1, 7), (1, 7), (7, 7), (7, 7), (5, 7), (1,
13 7),
14 (6, 8), (8, 8), (8, 8), (5, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8, 8), (8,
15 8),
16 (1, 9), (1, 9), (2, 9), (7, 9), (5, 9), (7, 9), (5, 9), (7, 9)]
17 0 : 22.22%
18 1 : 44.44%
19 2 : 50.0%
20 3 : 44.44%
21 4 : 40.0%
22 5 : 70.0%
23 6 : 50.0%
24 7 : 30.0%
25 8 : 80.0%
26 9 : 0.0%
27 Epoch 2 : 42 / 96 43.75% 43.75%
```

