# 手写体识别

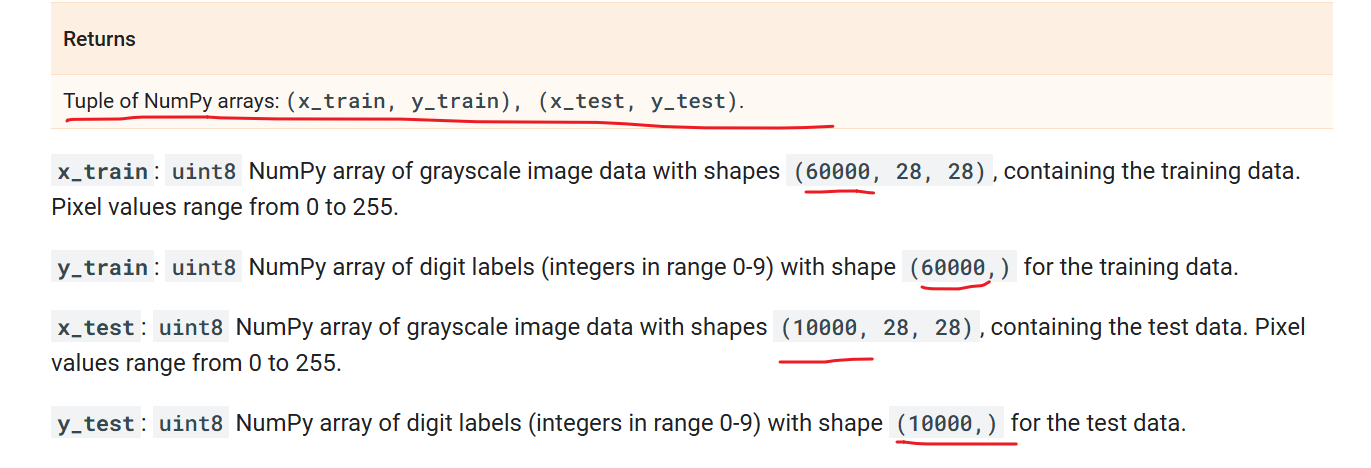
# 数据处理

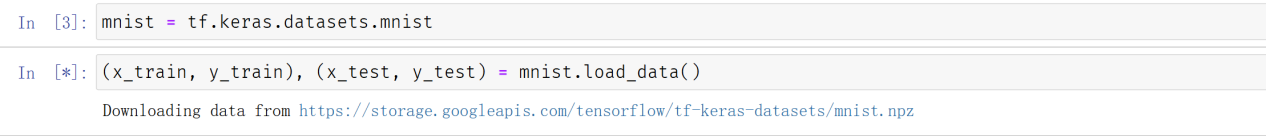
对于手写数据而言，网上开源的数据集有MNIST，首先我们先看看数据集中有哪些，是否符合我们的预期？

我们选用tensorflow作为基本网络架构。tf的版本问题兼容问题比pytorch多很多，所以为了便于复现，先打印出当前tf的版本。



tf官方提供了专门的api来下载数据mnist，我们查看官方文档API的返回值，按照他给的示例，静候下载结束。



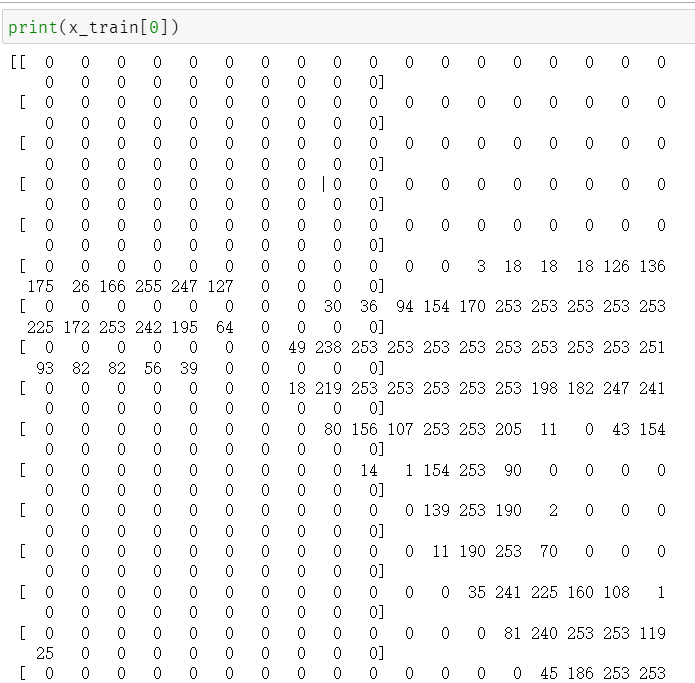


使用matplotllib库显示其中的一张看看先，选择色彩为‘Blues’（具体查看imshow中的色彩表），标题为它的类别。



可以看出来x\_train[i]的标签为y\_train[i]，对于x\_test和y\_test同理。

为后续方便进行预处理，先打印x\_train[0]看看



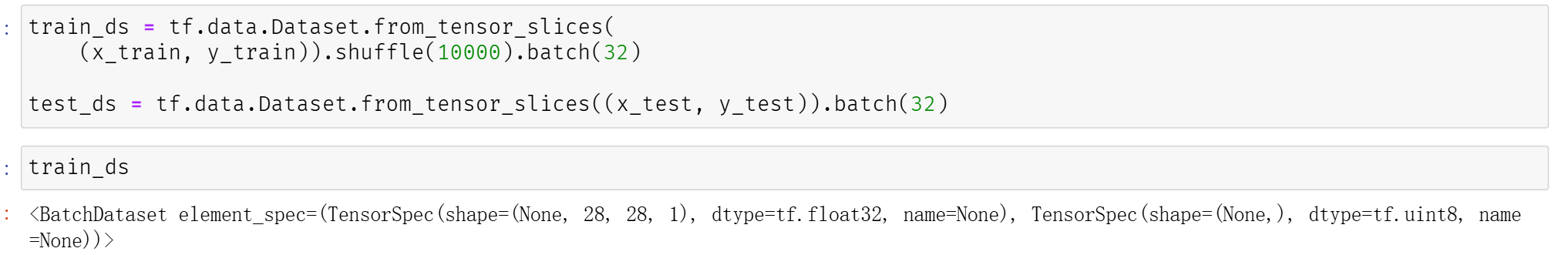
看出来这是一个二维数组，里面最大值为255，最小值为0。我们进行简单标准化，放缩到[0,1]区间。



由于tf架构的网络对于图像数据集的输入为(batch\_size, height, width, channels),而x\_train,x\_test都是三维数据（数量，height，width），所以我们需要增加维度，mnist提供的是灰度图，通道数为1，所以扩充后形状为（数量，height，width，1）



然后我们对数据集按batch进行划分，并且数据无时序关系，我们可以打乱数据，从而提高随机性。这里我们设置打乱时缓冲区大小为10000，batch大小为32。

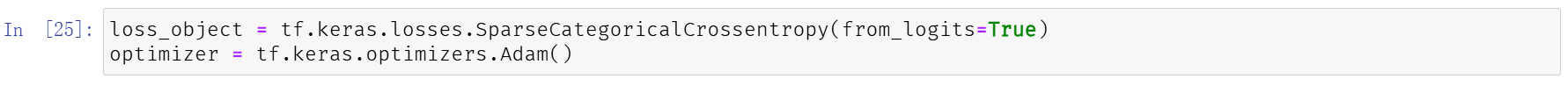


# 模型搭建

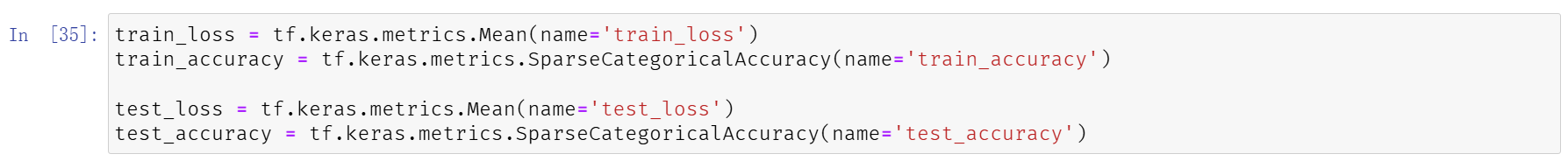
数据处理完后，我们构建模型，首先核心肯定是卷积神经网络，CNN在识别图像上效果较好，同时由于我们是分类任务，所以需要连接几层DNN。这里我没有进行池化操作，因为只进行了一个卷积操作，并不大，使用池化反而丢失参数。



接着我们选取损失函数和优化器。由于我们的任务是多分类任务（分类数量大于2），所以我们有两种方式，使用one\_hot编码将标签从1列变成10列，然后再使用categorical\_crossentrop损失函数，或者我们直接使用tf提供的sparse\_categorical\_crossentropy损失函数，虽然原理是一样的，但是提供了更方便的结果，将参数`from\_logits`设置为True，可以跳过one-hot编码。优化器选择方面直接使用Adam即可。



接下来为了可视化，我们选择打印损失值（loss）和准确率（accuracy）的趋势。



定义如何训练模型：

1. 获得预测值
2. 以损失函数为指标，计算预测值和真实值的差距
3. 梯度传播
4. 优化参数



和训练模型差不多，我们省去反向传播和参数优化过程即可。



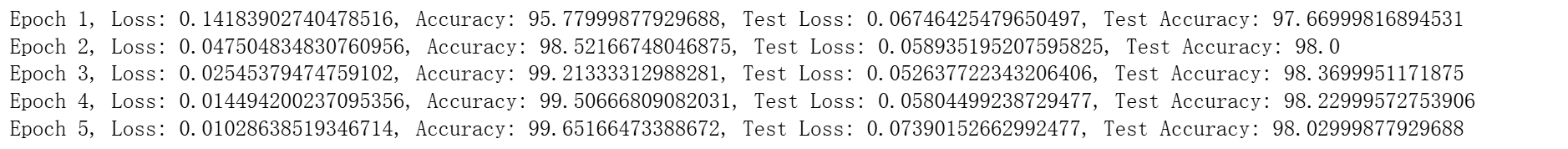
开始训练模型，步骤大概为：

1. 梯度清空
2. 训练
3. 测试

我们设置epoch为5，进行简单训练。



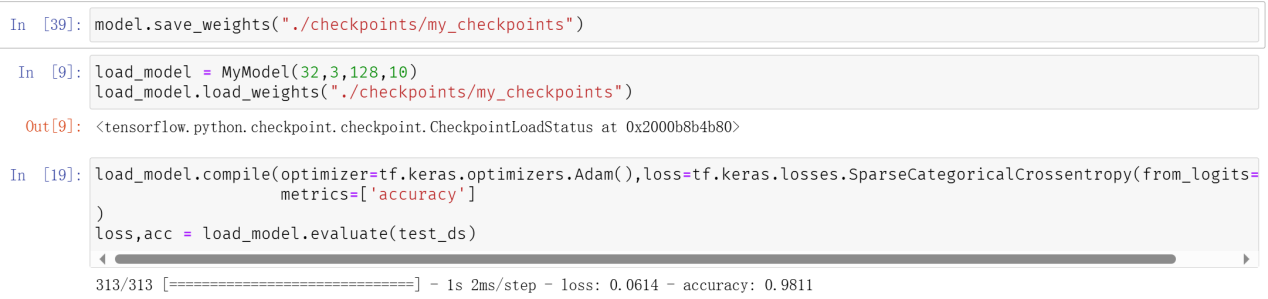
可以看到5轮训练，结果就非常不错，这是因为mnist数据集比较简单。但是不管简不简单，如果后续我们上传的图片识别能成功，说明就是优质数据集。



我们接下来保存我们训练的模型，tensorflow提供两种基本方法：

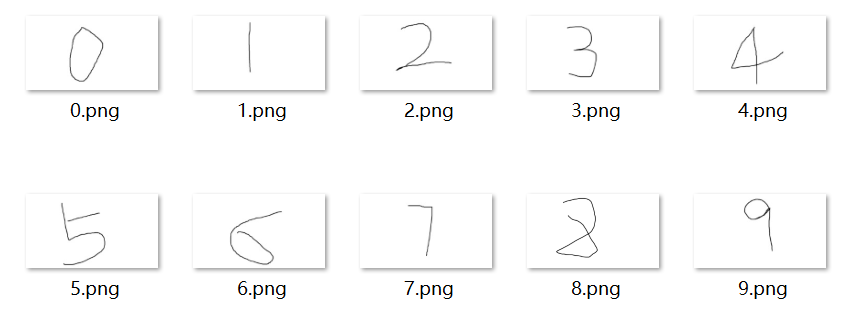
1. 直接保存模型，加载前无需重构模型结构
2. 保存模型参数，加载前需重构模型结构，同时需要编译损失函数和优化器

由于我是用构建子类的方法（pytorch中常用方法），这种方法直接保存模型比较麻烦，弄了半天搞不定，我就使用方法2。使用evaluate方式可以看到完成了载入

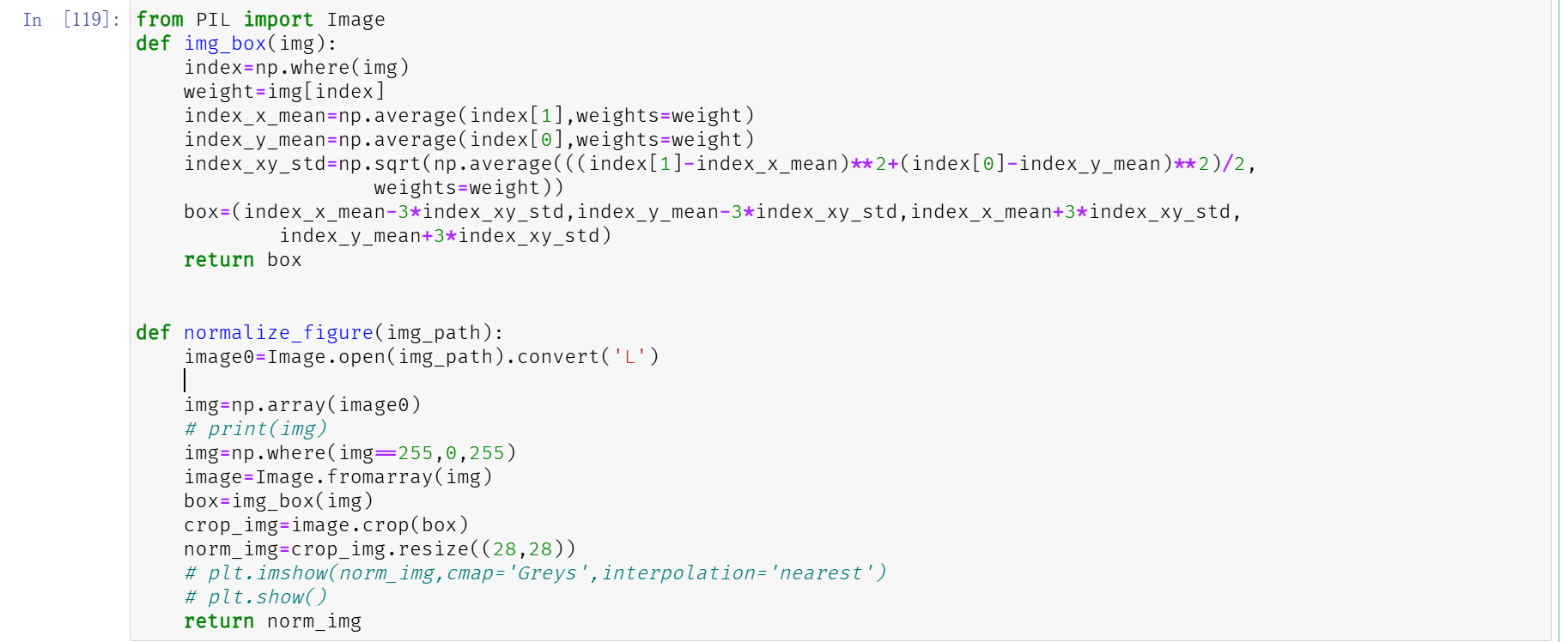


接下来我们的任务就是传入手写字体，进行图像预处理，再让模型预测即可。

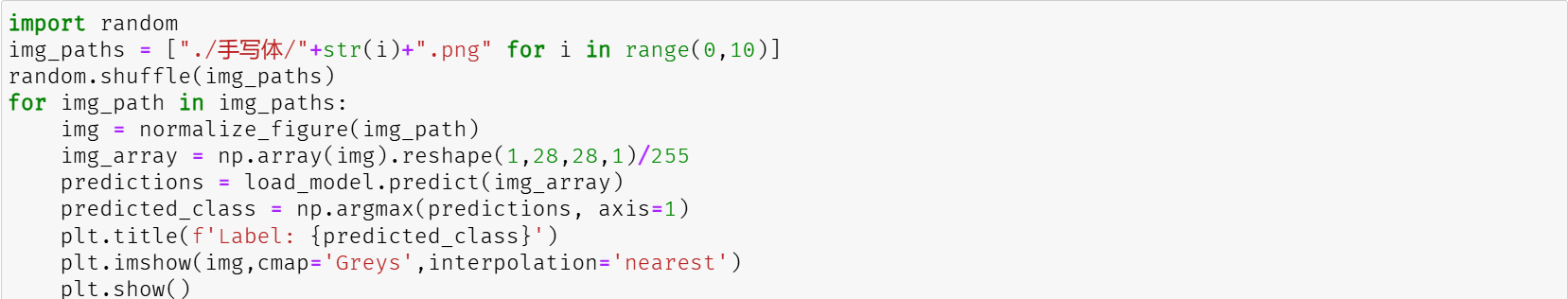
我们先制作一个手写字体，我是使用”画图”软件手写得到的



首先我们传入的图片一般是RGB格式，我们需要转化成灰度图，同时我们传入的图片通常背景冗余导致数据过大，而mnist数据集的尺寸为28\*28，为避免缩放图片时导致的数据丢失，我们先进行重要区域识别、裁剪，再进行缩放。



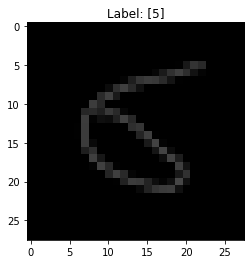
解决这个问题后，我们需要明白模型的输入参数为(batchsize,height,width,channel)，经过预处理后的图片为(28,28)，我们将batchsize设置为1（只有一张图片，如果图片比较多的话，可以考虑其他值），channel设置为1（灰度图通道数为1）所以我们使用reshape修改为(1,28,28,1）。最后简单标准化就行。然后转入模型，进行预测后，取概率最大的类别为标签，打印类别和图像（预处理后的图片）。



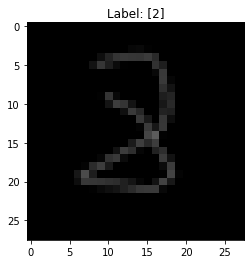
1/1 [==============================] - 0s 11ms/step



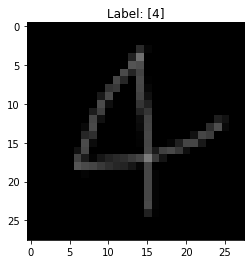
1/1 [==============================] - 0s 12ms/step



1/1 [==============================] - 0s 12ms/step



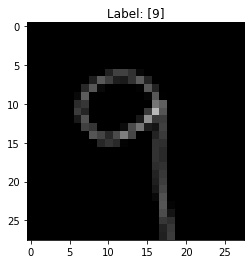
1/1 [==============================] - 0s 12ms/step



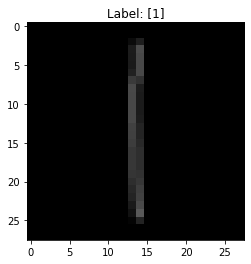
1/1 [==============================] - 0s 12ms/step



1/1 [==============================] - 0s 12ms/step



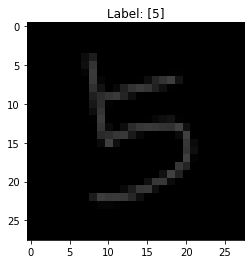
1/1 [==============================] - 0s 13ms/step



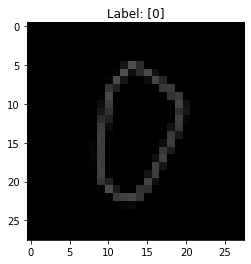
1/1 [==============================] - 0s 12ms/step



1/1 [==============================] - 0s 12ms/step



1/1 [==============================] - 0s 11ms/step



总结

整个实验大致用时3个小时，因为我对图像处理只是粗略的知道一点，所以大部分时间都浪费在上传图片的怎么处理上，同时tensorflow的保存方式没有torch那么方便，这里我查了半天，实验了半天也不行，就用保存参数的简单方法，但是如果要封装成系统的话，这么做肯定不妥当。

所有函数的使用基本上是查tensorflow的中文api文档完成的。示例解释的蛮清楚的。

对比实验

我们选取纯神经网络DNN作为对比。







Epoch 1, Time: 20.58 sec, Loss: 0.0032643997110426426, Accuracy: 99.90833282470703, Test Loss: 0.0789511650800705, Test Accuracy: 98.43999481201172

Epoch 2, Time: 19.96 sec, Loss: 0.0023209815844893456, Accuracy: 99.92500305175781, Test Loss: 0.08609671145677567, Test Accuracy: 98.36000061035156

Epoch 3, Time: 20.24 sec, Loss: 0.003263623220846057, Accuracy: 99.89666748046875, Test Loss: 0.09426737576723099, Test Accuracy: 98.41999816894531

Epoch 4, Time: 20.82 sec, Loss: 0.001772475428879261, Accuracy: 99.93666076660156, Test Loss: 0.10845891386270523, Test Accuracy: 98.22000122070312

Epoch 5, Time: 23.96 sec, Loss: 0.0024886683095246553, Accuracy: 99.92166900634766, Test Loss: 0.10686993598937988, Test Accuracy: 98.4000015258789

Epoch 6, Time: 26.19 sec, Loss: 0.0018211592687293887, Accuracy: 99.94666290283203, Test Loss: 0.09373657405376434, Test Accuracy: 98.40999603271484

Epoch 7, Time: 31.88 sec, Loss: 0.0024322790559381247, Accuracy: 99.94000244140625, Test Loss: 0.08785195648670197, Test Accuracy: 98.44999694824219

Epoch 8, Time: 24.63 sec, Loss: 0.0006504395278170705, Accuracy: 99.98500061035156, Test Loss: 0.1068781167268753, Test Accuracy: 98.29000091552734

Epoch 9, Time: 20.83 sec, Loss: 0.0018585361540317535, Accuracy: 99.93333435058594, Test Loss: 0.09135966747999191, Test Accuracy: 98.50999450683594

Epoch 10, Time: 22.48 sec, Loss: 0.0006255367770791054, Accuracy: 99.9800033569336, Test Loss: 0.11527257412672043, Test Accuracy: 98.27999877929688

结果很高。原因和最开头提到的一致，mnist数据集过于简单了。我们可以简单分析一下：

在mnist中出现的图片大小为28\*28。