1. **实验目的**
2. **学习pytorch的基本用法**

**（2）学习如何使用 PyTorch 读取并处理数据集**

**（3）学会使用pytorch构建卷积神经网络 CNN**

1. **实验内容及步骤**

手写体数字识别是一项非常经典的分类问题，要求算法可以识别手写体数字的图片，并

且输出图片中的数字类别。图片由 28\*28 个灰度像素组成，总共有 784 个特征。

卷积层是由若干卷积单元组成，每个卷积单元的参数都是通过反向传播算法最佳化得到

的。卷积运算的目的是提取输入的不同特征，第一层卷积层可能只能提取一些低级的特征如

边缘、线条和角等层级，更多层的网路能从低级特征中迭代提取更复杂的特征。

后面会跟上Relu（线性整流层），来作为激活函数。它可以用来增强判定函数和整个网

络的非线性特征，而本身并不会改变卷积层。虽然也可以使用其他的激活函数，但是 Relu 因为对计算友好，可以提高训练速度，所以更受青睐。

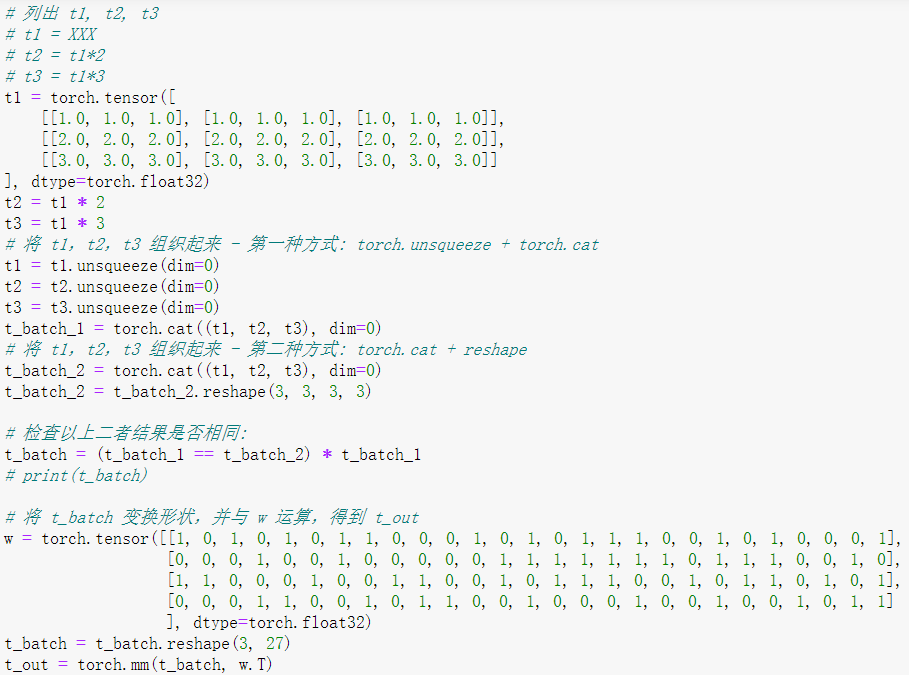
在 Relu 后面，还会再接上一层池化层。池化（Pooling）是卷积神经网络中另一个重要的概念，它实际上是一种形式的降采样。有多种不同形式的非线性池化函数，而其中“最大池化（Max pooling）”是最为常见的。它是将输入的图像划分为若干个矩形区域，对每个子区域输出最大值。直觉上，这种机制能够有效地原因在于，在发现一个特征之后，它的精确位置远不及它和其他特征的相对位置的关系重要。池化层会不断地减小数据的空间大小，因此参数的数量和计算量也会下降，这在一定程度上也控制了过拟合。

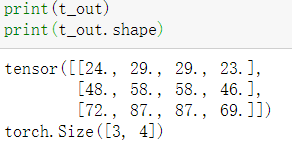
上面三个模块一般以固定组合的方式，出现在神经网络的前面，用来对图像提取特征。

1. **实验结论和问题解答**

# **1. Tensor 综合练习**

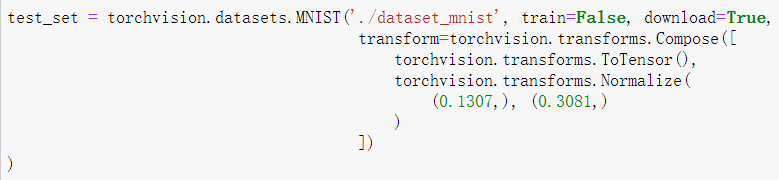
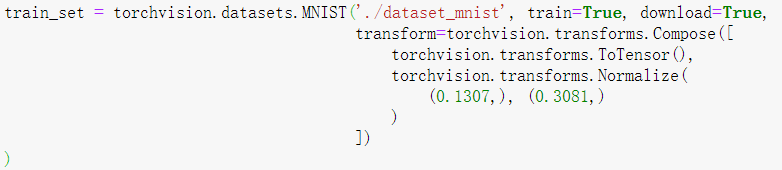
请问运算结果是什么形状的张量？每一个维度分别对应什么？请在实验报告中回答这些问题，并展示本段代码和运行结果，进行说明。





**答：两个维度， t\_out的每一行代表batch中一张图片的全连接的4个神经元的输出，每一行的每一列代表全连接的一个神经元的输出。**

**2. 读取并处理数据集 MNIST。**（请完成对 train\_set 相关代码的解释说明，并补全 test\_set 的代码。）



**root根（字符串）**–存在MNIST/processed/training.pt和MNIST/processed/test.pt的数据集的根目录。就是MNIST所在的文件夹。

**train（bool，可选）**–如果为True，则从training.pt创建数据集，否则从test.pt创建数据集。train/test模式切换。

**download（bool，可选）**–如果为true，则从internet下载数据集并将其放在根目录中。如果数据集已下载，则不会再次下载。

**transform（可调用，可选）**–接受PIL图像并返回已转换版本的函数/转换。Eg，变换，随机裁剪。

**transform** = torchvision.transforms.ToTensor(),将图片转化成取值[0,1]的Tensor用于网络处理。torchvision.transforms.Normalize()数据集归一化。

1. **定义一个神经网络**

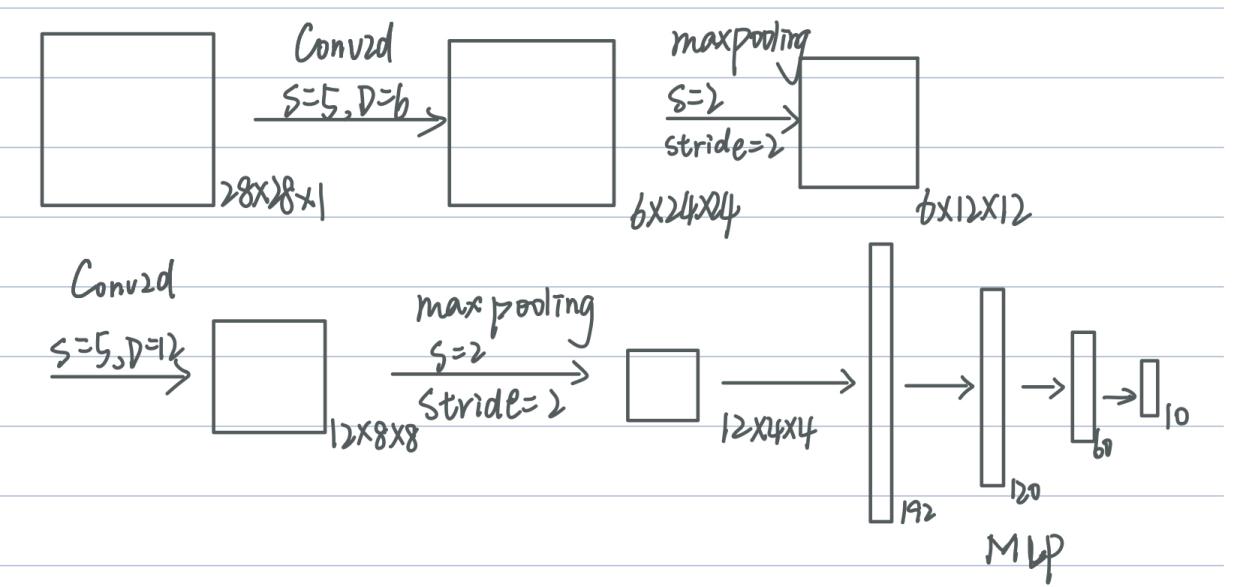
　　假设我们将使用 MNIST 数据集，每张图片为一个 28\*28 的灰度图。在即将定义的神经网络中，每个卷积层之后我们都会进行池化操作 (pooling)，池化的 kernel 大小和 stride 均取 2（即每次将对 2\*2 的元素进行 max pooling）。 池化层由于不包含可训练的参数，因此不作为 Network1 的类属性（class attribute），而是会在之后写入 forward() 函数里。

　　请大家阅读以下代码，在实验报告中：

　　 1) 画出对应的**网络结构**（包含各层权重的形状）；

　　 2) 注明当一张图片输入进网络后**各层输入、输出值的形状；**（由于不考虑批量化处理图片，因此卷积层激活值以**三维张量**表示，全连接层激活值以**一维张量**表示即可。）

　　 3) 补全 **“self.fc1”**的代码。（[提示]：需要计算 fc1 层的输入激活向量的长度。）



IMG_256

请在实验报告中展示补全后的 Network2.forward()，并说明这样写的原因。（在图中已展示）



把这一部分代码（从"network = Network2()"开始）多跑几次你会发现，每次 "scores"、"predict\_class" 等都会发生变化，请思考这是为什么？另外，尽管"scores"每一次都不同，但其大致取值却都差不多，这又是为 什么？请在实验报告中说明。

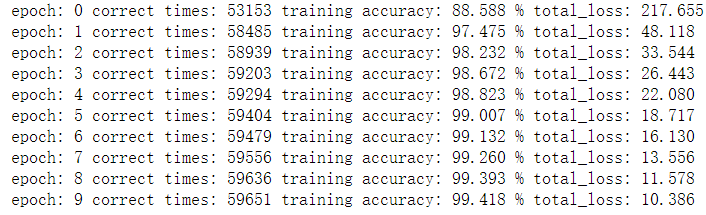
答：

1. **每次 "scores"、"predict\_class" 等都会发生变化是因为网络的参数初始化不同，训练所得的结果也不同。**
2. **网络参数初始化是按照某种分布（一般为高斯分布）随机初始化的，所以尽管"scores"每一次都不同，但其大致取值却都差不多。**
3. **训练神经网络。**

请大家补全以下代码，在实验报告中展示相关代码并做出说明，训练结果请截图保存。



**解释在注释当中。**



测试结果：

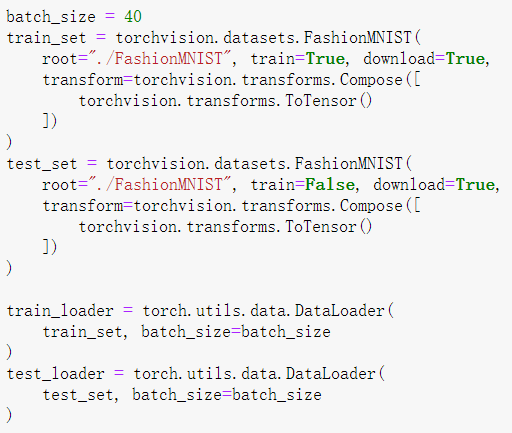
IMG_256

1. **知识整合及应用。**

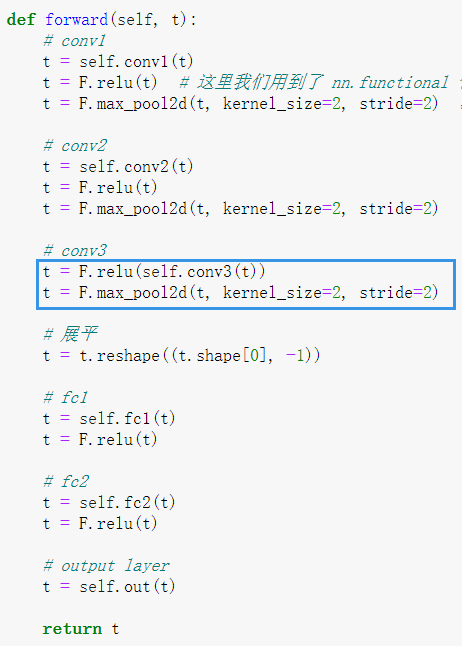
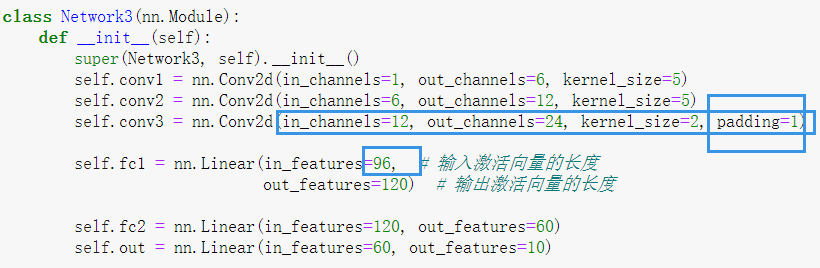
## 自定义神经网络在 Fashion-MNIST 上的训练与测试。自行设计一个7层神经网络（含三个卷积层和三个全连接层，卷积层可以尝试将 padding 设置为非零值），并Fashion-MNIST 数据集上进行训练和测试。

目标：测试集上准确率高于 87%。

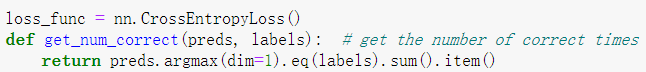
1. 读取和处理数据集



1. 自定义神经网络



3) 　损失函数和准确率函数

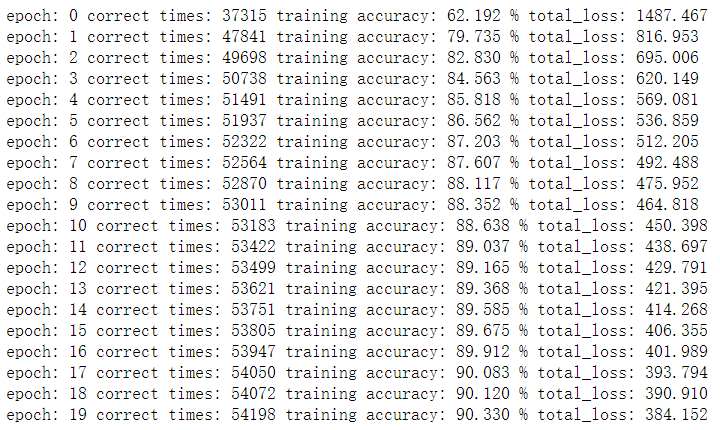


4)　优化算法

IMG_256

1. 训练



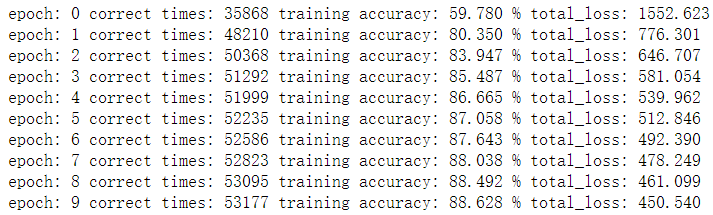


IMG_256

我做了三个调整：

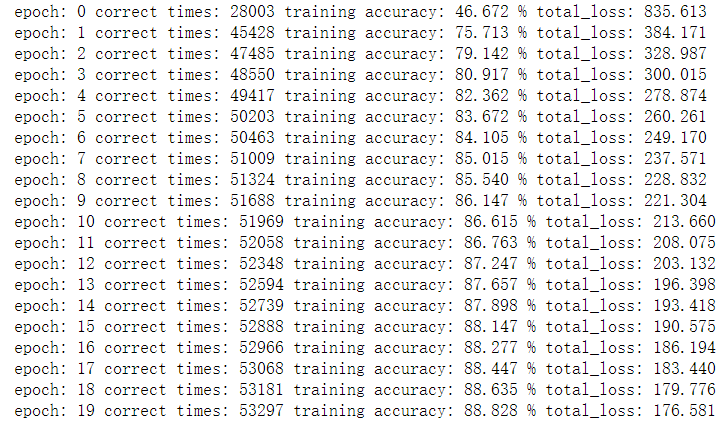
1. Total\_loss一直在下降，没有收敛，所以加大了epoch的数量，效果有所提升，但并不明显。
2. 学习率的调整，效果也比较一般。
3. Batch\_Size为100和为40的情况，在这个数据集下，Batch\_Size为40的更符合。

Epoch = 10



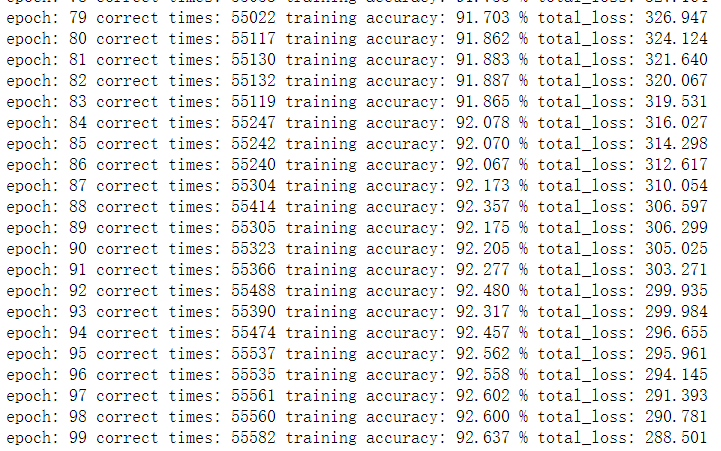
IMG_256

Batch\_size = 100





Epoch = 100,a = 0.01

IMG_257

1. **实验体会**

通过本次实验，与第一次用numpy实现全连接网络的对比，我深刻地感觉到了pytorch封装的好用。使用pytorch这个深度学习框架，可以更快速和有效的实现神经网络架构。Pytorch具有强大的 GPU 支持的张量计算功能，能够高效实现神经网络中的各种张量运算，它能够自动实现反向传播过程中的求导运算。不用去担心底层实现出错，可以将精力集中于网络本身。并且，其中也有很多其他研究者的成果，可以在深度学习框架中自由的使用。CNN 是神经网络中专门用来处理图像的组件，对于 mnist 这种小型数据集来说，卷积层可以非常有效的解决这类问题。同时还了解到了Fashion-MNIST这个数据集，并且自己搭建了一个网络运用在了这个数据集上。同时，我还更加熟练地学会了隐藏层的计算，在每一次卷积和padding之后得出之间的训练神经元的个数。我还学会了去调整一些参数以及超参数去使得神经网络的效果更加理想。并且我更加理解了参数的初始化的重要性。通过这次实验，从理论课堂到代码操作实际，能够搭建出各种各样的问题去解决很多相关的问题了。