**引言**

本文首先总结了什么是元学习、元学习的基本方法，以及元学习和传统机器学习之间的区别；然后基于总结的内容提出了自己对元学习的看法，并提出了一些疑惑；其次，解释了One Shot learning, Siamese networks, Triplet Loss，并给出了它们的用途；再者，基于Keras实现了它们的。

**然后，通过上述学到的知识在 Omniglot数据集 上做了相关实验**。

1. **元学习**
2. **解释**

元学习即Meta Learning，其目的是使机器学会如何学习。根据老师给的资料，何为学会学习：机器已经在过去的100个任务上进行了学习，现在我们希望，机器能够基于过去100个任务学习的经验，在第101个新任务到来之时，机器能够更快地学习。机器之所以能够学习地更快并不依赖于在旧任务中已获取的“知识”，而是机器学到了如何去更好获取知识的方法，并将这一方法应用于新任务当中，从而较快地提升学习效率。

**自己的理解：**在我看来，元学习应该就是像我们学习各类编程语言，我们最开始学习了C/C++；然后，基于我们学习上述两种语言的基础，我们可以更快的学会Python。但是更快的学习Python并不依赖于我们在学习C语言时获得的语法知识，而是我们学会了如何去编程。这里，学习每一类编程语言就像可以获得一个f，在学习完一定种类的语言之后，我们就可以获得一个F，当我们再去学习其他语言时，我们可以通过已经学到的F的经验去更快的获得f。

1. **元学习的基本方法**
2. Metric-based

即基于度量的元学习。

**理解：**以图片分类为例，基于度量的元学习即是去计算需要预测的图片与已有类别的图片之间的距离，距离越小的样本之间越相似，可将它们分为一类。这个方法其实是很好理解的，即通过距离来表述两张图片的相似程度。

1. Model-based

基于模型的元学习。

**理解：**这类方法应该是针对不同任务得到一个通用的学习参数θ，面对不同的任务时，可以改变这个参数去适应不同的任务。而参数θ的值可以通过元学习器去学习，在面对不同任务的时候，根据损失函数通过梯度下降的方法不断地更新θ值，使这个模型不断向能解决这个任务的模型靠近，当θ值最终收敛时，我们认为元学习器学习到了一个较好的参数θ，让模型自适应地解决相应任务。

1. Optimization-based

基于优化的元学习。

**理解：**元学习的样本数量较小，此时学习器可能出现过拟合的现象。为了避免这些现象对学习器性能的影响，使用一种优化方案，让学习器优化更新初始化参数，使其在面对相应任务的时候，能够仅用少量的样本较快收敛，达到快速学习解决任务的目的。

1. **元学习和传统机器学习之间的区别**

Machine Learning ≈ 根据资料找一个函数 f 的能力；Meta Learning ≈ 根据资料找一个找一个函数 f 的函数F 的能力。

**理解：**传统机器学习是由人来设计一套学习算法，然后将大量数据输入这个算法，通过训练得到算法里的参数，并用参数拟合出一个函数。然后将测试数据输入这个算法之中，经过这个算法之后会输出一个结果，而这个结果就是类别。

而元学习是指从数据中学到一个F，这个F实际上是f的函数；针对不同的任务，我们可以根据F得到相应的f。在我看来，元学习就是这样。

1. **疑惑**
2. 元学习的解释部分

老师所给的解释以及网上给的解释都说是，“机器已经在过去的100个任务上进行了学习”，然而有如下问题：①这一百个任务到底是什么，是要与我想得到的任务结果类似的任务还是说可以是完全不相关的任务？②假设我们已经对100个任务进行了学习，在第101个任务上我们显然要用到我们的学习结果，如果没有理解错的话，应该就是使用这个F，问题是：怎么用？也即是我如何把F应用于我想要生成的f。

1. 有关元学习的基本方法

我没有理解最后一种方法。第一种很好解释，我们可以通过度量距离来表征两个数据之间的相似程度，从而达到分类的目的。第二种方法就是通过调节参数θ去适应不同的目标。但是对于第三种方法，难道它不就只是一种避免出现过拟合或者可以实现更快收敛的方法而已吗，怎么也可以作为一种元学习的方法？也就是这个F是如何产生的？

1. 元学习的输入

以图片分类为例。传统深度学习的输入就是图片，然而针对元学习，我们不仅会使用到输入数据集，而且还会用到F。F怎么输入网络？

1. **One Shot learning, Siamese networks and Triplet Loss**
2. **One Shot learning**

One Shot learning即是单样本学习。

**自己的理解：**就是说假设给的训练集中有N个类别，每一个类别都只有一个样本，这样的情况叫做One Shot learning。而这也可以引申到K-shot learning,也就是对于每一个类别有K个训练样本。以老师给的文章中的例子来说：在猫狗分类下加入猴子的分类，这个时候我们只需要一张猴子的图片以及它的标签，放入网络中。在之后的识别过程中，我们用测试数据与数据集中的数据来计算距离，距离近的就归为一类！但是，怎么计算距离就可以引出我们所用到的Siamese networks。

1. **Siamese networks**

Siamese networks即是孪生网络。

**理解：**输入图片通过两个网络，但是这里的这两个网络实际上是相同的。相同的架构，相同的权重，实际上是相同的网络，但用于两个不同的输入。然后使用输出确定它们是否相似。

1. **Triplet Loss**

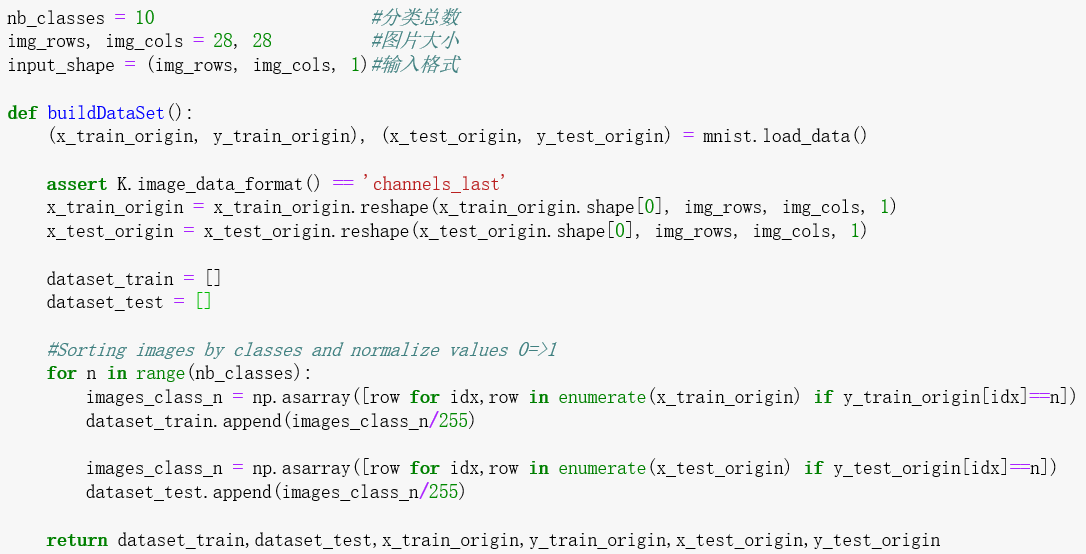
**自己的理解：**首先来看这个Triplet，它其实就是表示三种图片，第一个是起始图片，称为Anchor；第二个是与起始图片相似的图片**(问题：这里到底应该是同一类图片还是相似的图片？因为教案上给的MNIST数据集的三个数字是9，4，7；而给的资料中的图片是狗，狗，猫；前者显然是相似，后者显然是同一类别)**

即是Positive。第三个则是来自不同类别的图片，即Negative。

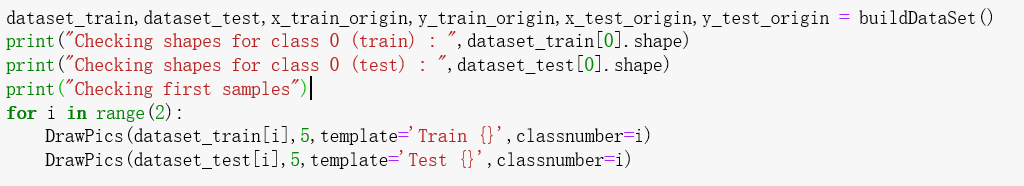
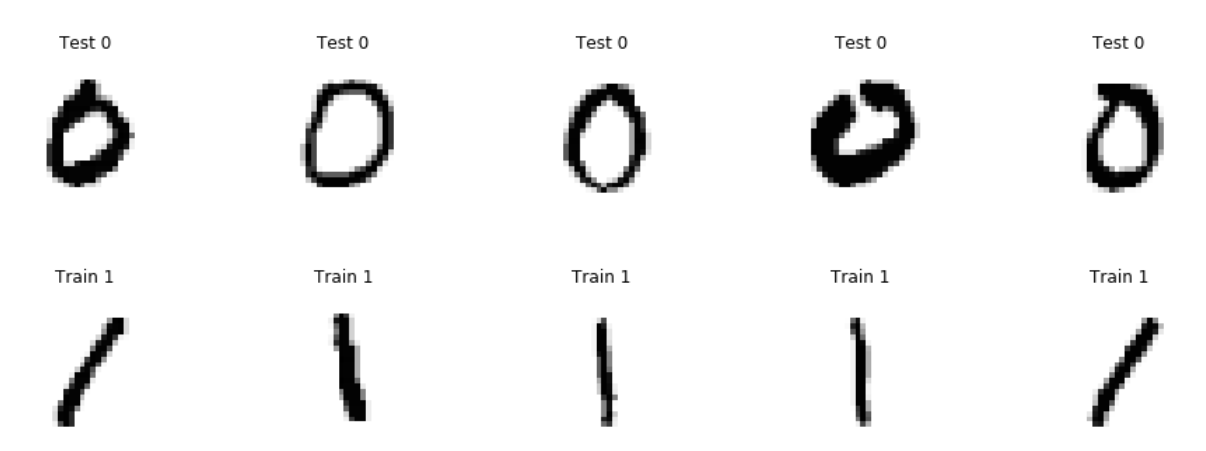
定义：L = max（d（A，P）-d（A， N）+ margin，0）。其中d（A，P）表示Anchor和Positive的距离，d（A， N）即是Anchor和Negative的距离。为了防止网络通过为所有内容输出零来学习满足方程式的简单解决方案，用一个 margin 参数强制在 AP 和 AN 之间留出一个余量。

1. **实现**
2. **数据准备**

这里用专门的数据准备函数buildDataSet()来准备数据。其思路是，先加载MNIST数据集，然后将其转换为卷积神经网络的格式；然后再准备此处所需的数据集和测试集dataset\_train，dataset\_test。这两个数据集的准备其实就是将原有数据集分成0-9共十类数据，然后放入这两个数据集中(放入之前先进行归一化操作)。其代码如下：

1. **查看所得到的数据**

这里对得到的dataset\_train，dataset\_test数据集进行了显示，可以看到数据集中一共有5923个’0’，测试集中共有980个’0’。其代码如下：

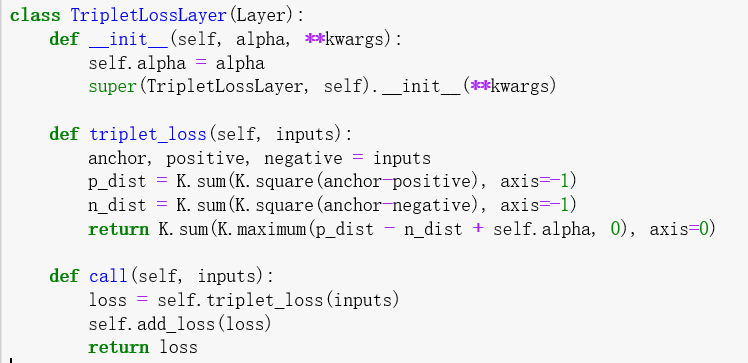
这里有一个DrawPics函数，其实它就是为了展示图片专门写的一个调用函数。其部分结果如下：

1. **神经网络构建**

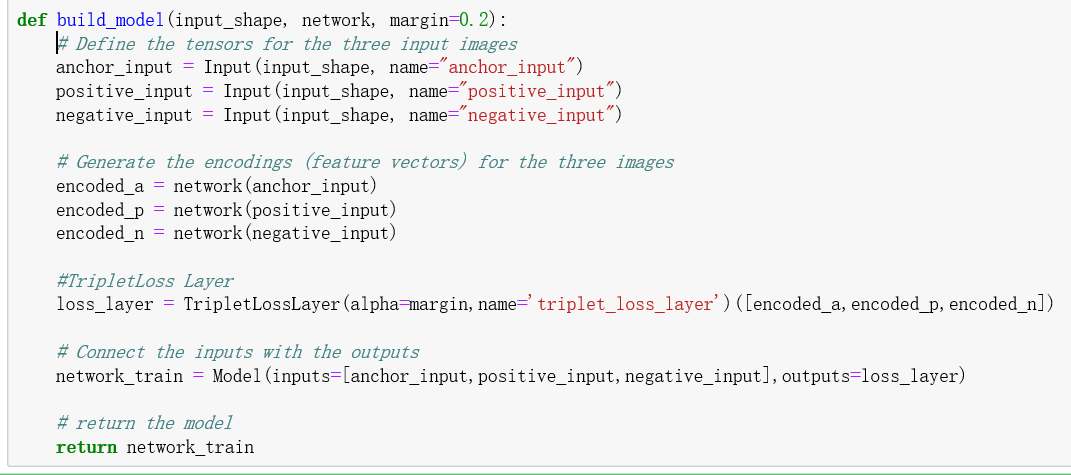
这里是建立相应的神经网络。其实就是将网络定义为一个简单的卷积神经网络。运行到这里，就有一个疑惑：可以看到这个网络和卷积神经网络几乎上没有区别，但是我们不是要建立孪生网络么?难道是将这个网络定义好了建立两个模型？不过孪生网络的参数不应该是相同的吗？(继续做下去，看下面是怎么解决的)神经网络构建代码如下：

1. **Triplet Loss层的实现**

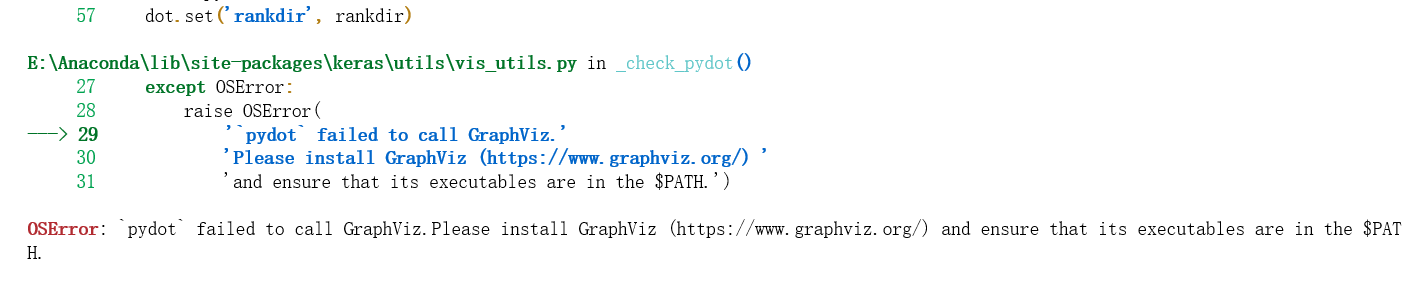
这里定义了Triplet Loss的计算方法，也就是刚才提到的公式：L = max（d（A，P）-d（A，N）+ margin，0）。如下图所示，这里的距离K.square(anchor-positive)，如果没想错的话，应该是计算的欧氏距离。然后，计算式子K.sum(K.maximum(p\_dist - n\_dist + self.alpha, 0), axis=0)就是根据公式L而来，其中alpha就是margin。其解释在上面。

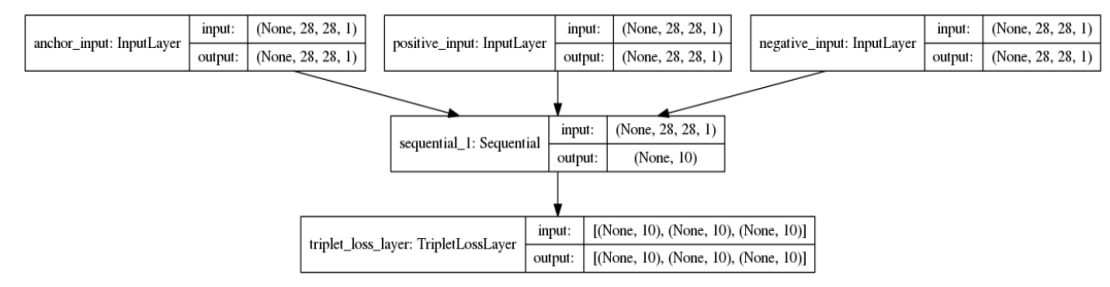
1. **建立模型**

这里针对Anchor、Positive、Negative建立模型。不过到这里还是没有搞清楚是怎么将参数设置成相同的。其实现如下：

1. **将模型结构显示出来**

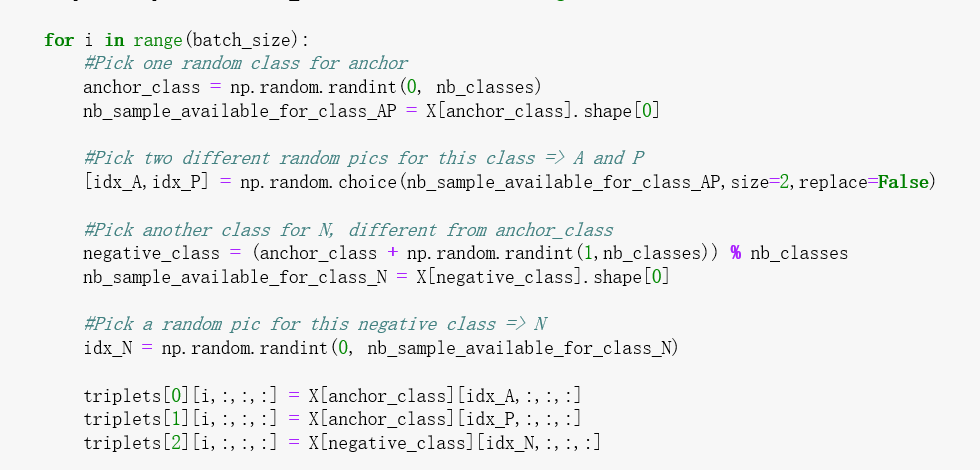
这里运行时出现了问题，其截图如下。出现这个问题的原因：作者所用到的pydot已经不再更新，应该用pydotplus。先用pip install pydotplus安装好pydotplus，然后再将E:\Anaconda\Lib\site-packages\keras\util\vis\_utils.py中所有的pydot都改成pydotplus。这个时候再次运行就会出现下图所示的错误，这个错误是提示无法启动GraphViz。解决方案是去官网上下载对应版本的GraphViz，安装好之后再将其路径添加在系统环境变量之下。

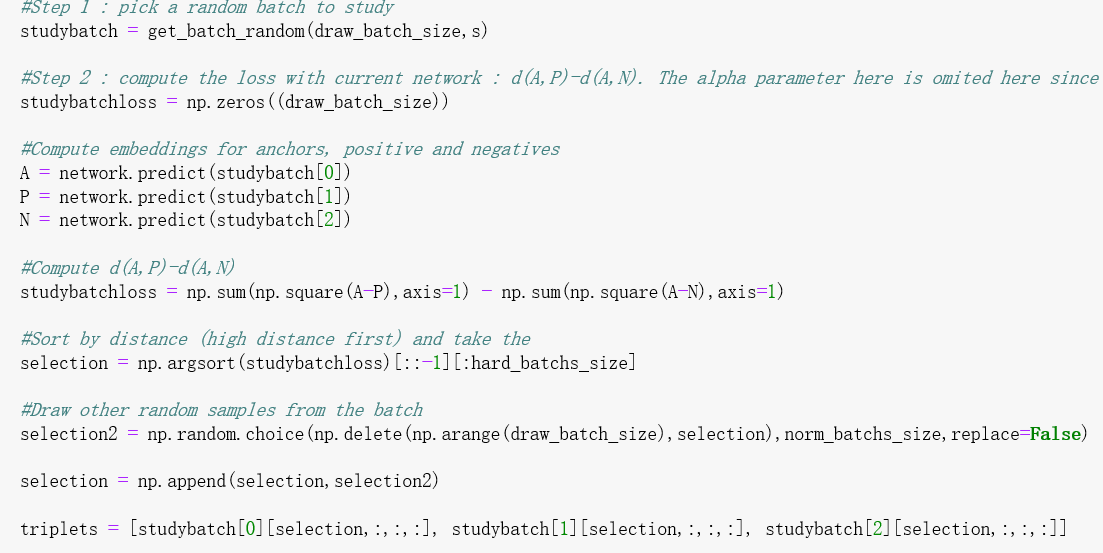
解决之后显示的模型结构如下：

****然后这里还有一个问题network\_train.load\_weights('mnist-160k\_weights.h5')，mnist-160k\_weights.h5应该是作者写好的，我们在这里加载不了，这样对后面的影响是：再后面进行训练时，会从头开始训练，而不像作者文件中所给的那个位置开始。

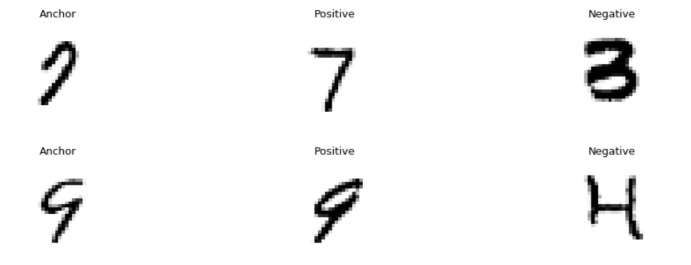
1. **选择三元组**

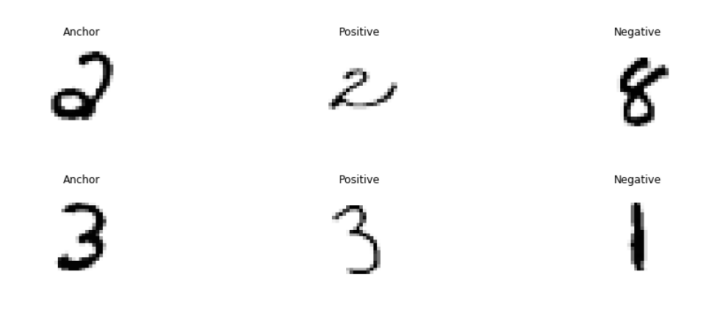
这里即是选择Anchor、Positive、Negative三个类型的图片。根据损失的定义，可以将三元组分为三类：①简易三元组：损失为 0 的三元组， d（A，P）+ margin <d（A，N）；②硬三元组：d(A,N)<d(A,P)；③半硬三元组：d（A，P）<d（A，N）<d（A，P）+ margin。这里用了两种方法选择三元组：第一种是用的简易三元组，即随机抽取，其代码如下：

第二种半硬性的三元组，其中一半的三元组是随机生成的，另一半则是根据d(A,N)<d(A,P)的损失函数进行选择的。其代码如下所示：

1. **查看选到的三元组**

这里对所选到的三元组进行显示，其结果如下。首先是随机选择得到的结果：

然后是半硬性三元组的选择结果：



可以看到，由于有随机成分，因此运行之后就和原来的结果不同。

1. **评估方法**

我们的模型是计算两张图之间的差别，首先来说，即使是两张图片来自于同一个类，它们之间依旧会有“距离”，这个时候，我们就不能通过简单的判断两张图有没有距离(实际上，两张图没有距离的话则应该是同一张图)来说明是否是同一类的图片，此时我们应当设定一个阈值，然后去计算两张图片的距离，如果距离大于这个阈值，我们就可以说，它们不是同一个类的，如果小于该阈值，就可以说是同一个类的。显然，这个阈值的设定是非常重要的，设定过低在我看来应该会增加计算量(而且猜测也可能会出现过拟合的情况)，过高显然会降低正确率。

1. ROC曲线

这里用ROC曲线来进行评估。先来看一下什么是ROC曲线：ROC曲线全称为受试者工作特征曲线 ，它是根据一系列不同的二分类方式（分界值或决定阈），以真阳性率（敏感性）为纵坐标，假阳性率（1-特异性）为横坐标绘制的曲线。

**自己对ROC曲线的理解：**以本实验为例。我们是计算两张图片的距离，再用距离去与阈值相比较，然后得出结论。这个时候，就会出现以下四种情况： ①是同一类，也被分为同一类(设为TP)

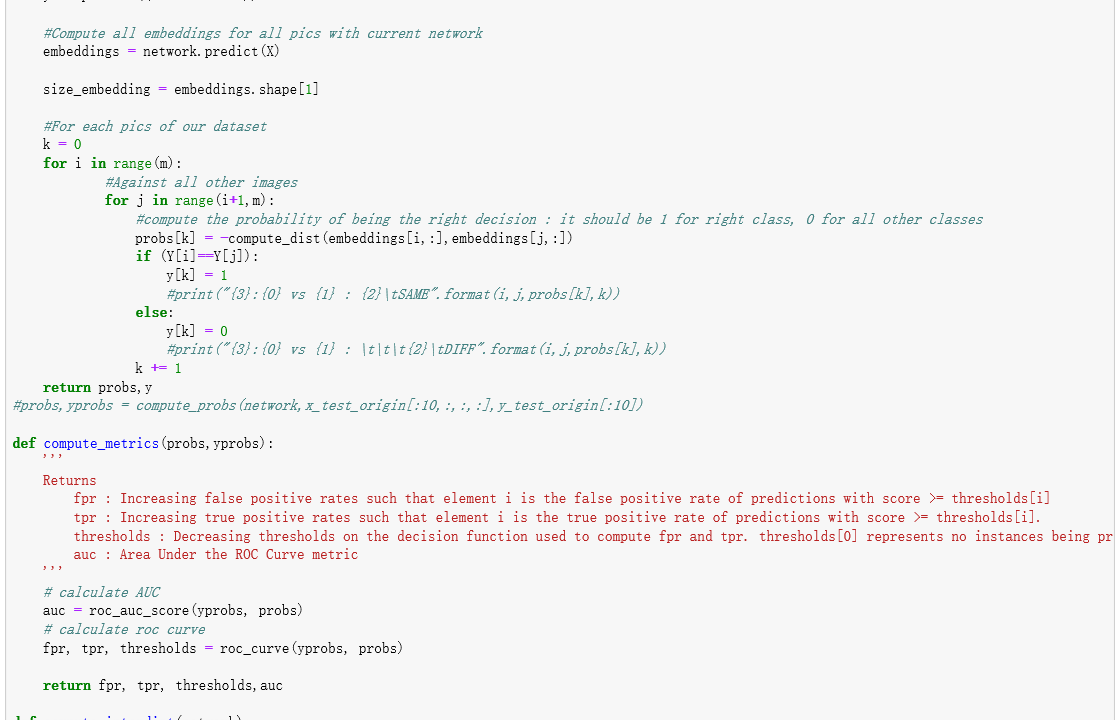
②是同一类，但被分为不同类(设为FN)

③不是同一类，但被分为同一类(设为FP)

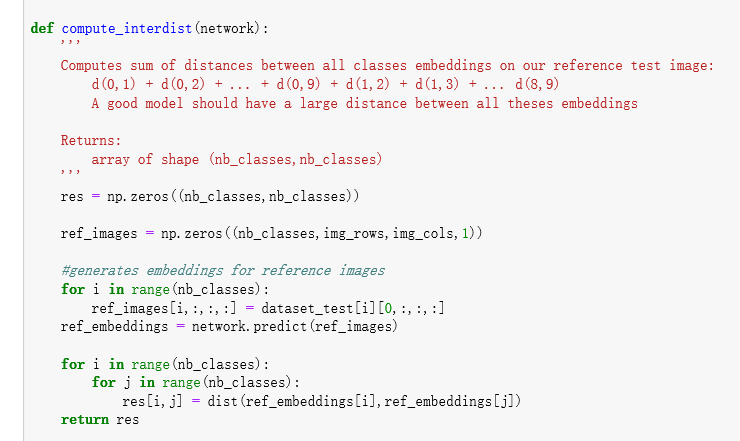
④不是同一类，也被分为不同类(设为TN)

然后就可以计算得到查出率：TPR = TP/(TP + FN)，以及误检率：FPR = FP / (FP+TN)。整个ROC曲线就是以FPR为横坐标，TPR为纵坐标画的曲线。再次思考：这其实是个二分类问题，要不然我就分对了，要不然我就分错了。这个时候，如果我采用随机的分类，那么正确和错误的概率都是百分之五十，因此有了中间那条y=x的虚线。越靠近左上角的情况查出率越高，差错率越低，即(0,1)点是最好的点。最后，可以有一个结论，曲线下面的面积AUC越大越好！！！

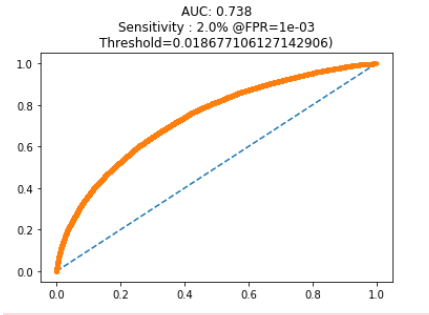
**ROC曲线理解的难点：**为什么右上角的点TPR、FPR都这么高？这个是因为当把阈值调的很高时TPR = TP/(TP + FN)中的FN就变少了，FPR = FP / (FP+TN)中的TN就变小了，因此会出现TPR、FPR都这么高的情况。

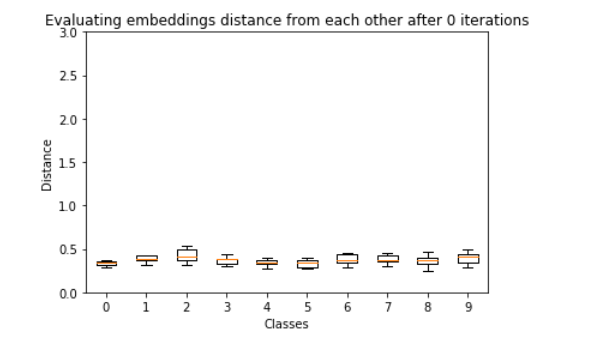
1. 定义计算AUC的函数

然后定义计算所有图片之间的距离：

1. **绘制ROC曲线**

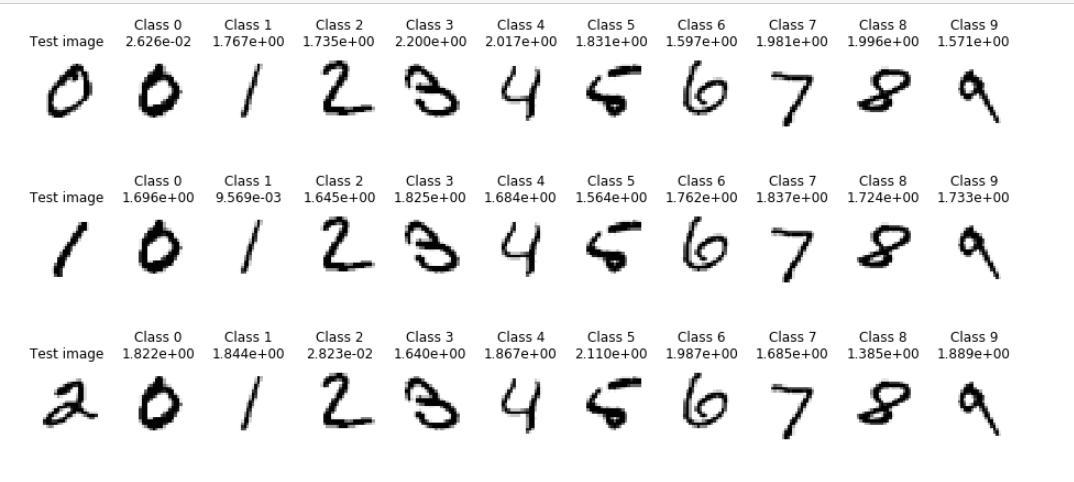
如图所示，首先是未经过训练就得到的ROC曲线图，可以看到其AUC很低。

不过这里AUC其实高于作者的测试(而且还高了0.1左右)，还有Sensitivity也改变了。这里，又用**箱线图**来表征类间距离：训练期间要查看的另一个指标是每个班级彼此之间的距离有多远。确切地说，我们应该评估整个数据集，为了检查网络是否对所有类别都收敛良好。



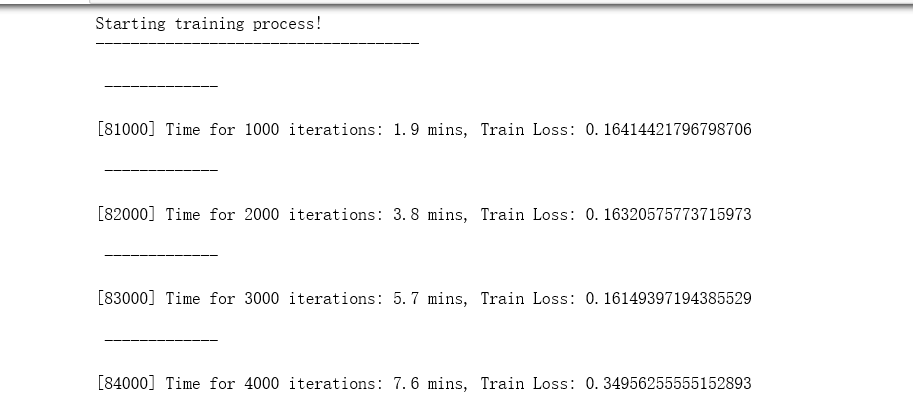
1. **结果分析**

可以看到，输入的图片去与所有类别计算距离，结果如下图所示。如图，当测试图片是0的时候，其与class 0的距离最近，就可以把该图片分到0类中，因此可以说分类正确。又如1，其与class 1的距离最近，等等。最后，经过训练之后又可以看到ROC曲线，此时其AUC的值接近1，说明这个分类已经很成功了。

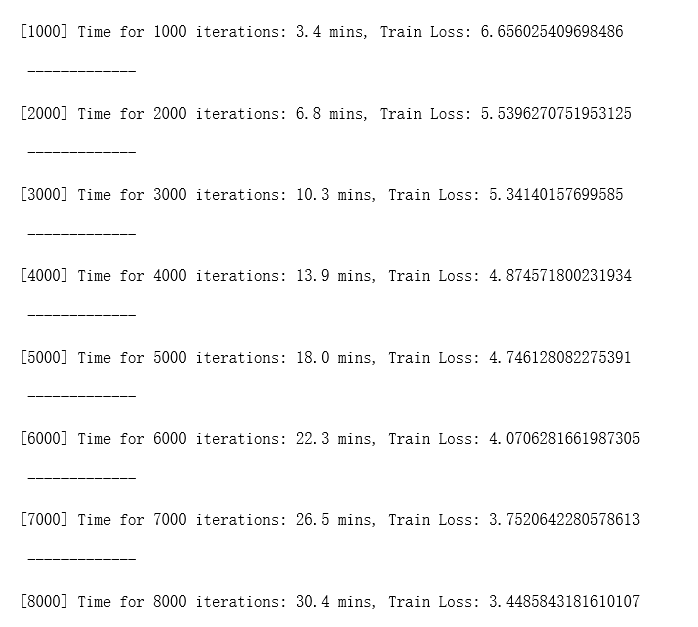


1. **遇到问题**

到上面一步为止都没有出现太大问题。即可以计算图片之间的距离，距离近的就可以进行分类。

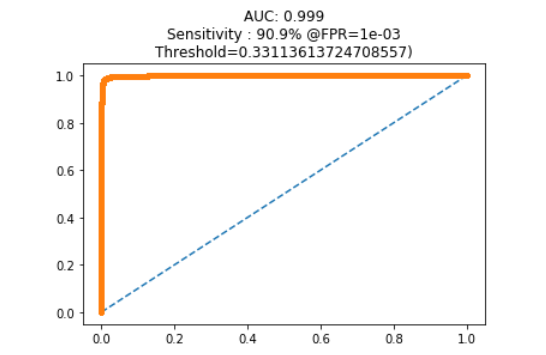
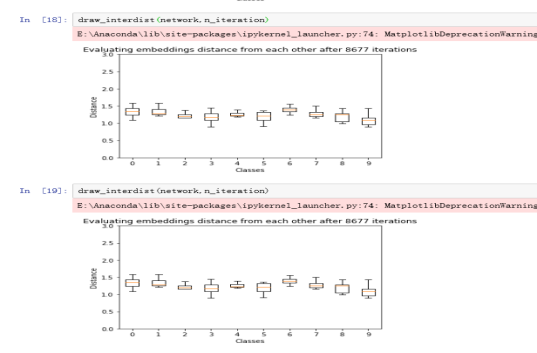
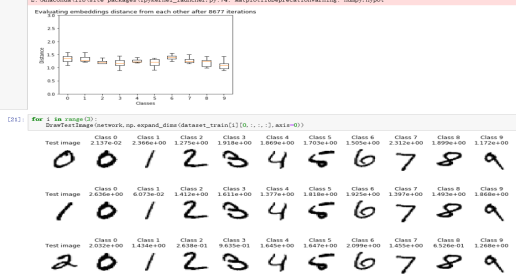
 但是，这里出现了运行不下去的情况，不过不是不能运行，而是运行花费的时间成本太大了，而且电脑不支持。运行一段时间后，电脑滚烫不说，还开始卡顿。初步分析了其原因：就是前面作者加载了自己的一个模型里面的参数，因此作者的训练实际上是从已经训过的模型出发的，会出现如下开始的情况：

而我出现的开始情况是：

可以看到其loss也是逐渐下降的，只是要训练到如作者所示的那个程度，还需要很久，而电脑已然不支持。

问题：不知道出现问题的原因是不是正如我分析的那样？

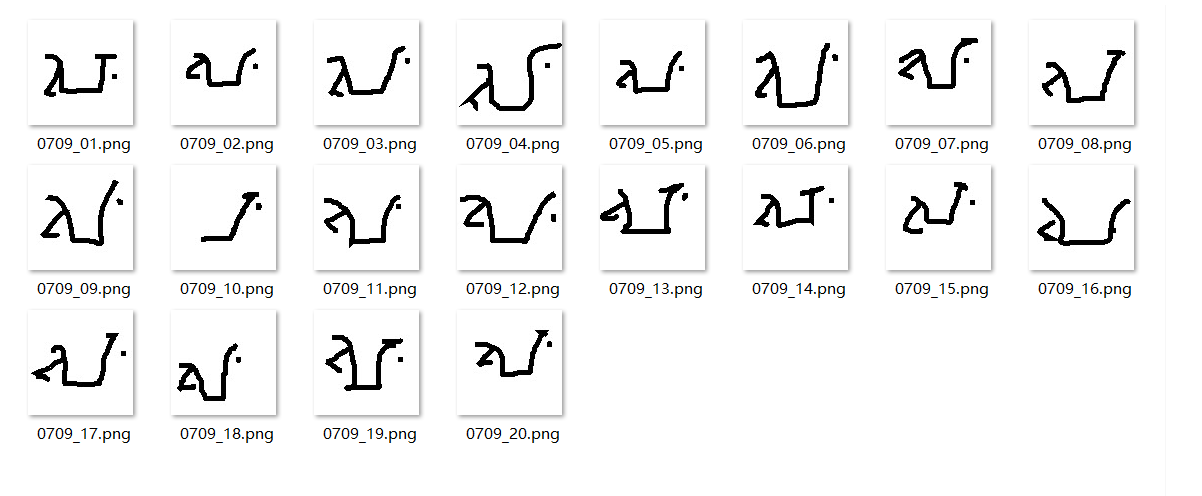
不过，即使我没有训练完整个代码，也出现了如下图所示的结果，其AUC也已经接近1：



可以看到上述结果，即使没有训练完成，所得到的AUC为0.999，已经是很高的数值了，而Sensitivity比之作者所做的实现低了百分之六，而阈值也已经接近。问题：为什么会出现这样的情况？

1. **Omniglot数据集上做的相关实验**
2. **数据准备**
3. **下载数据集**

这里omniglot数据集是从https://github.com/brendenlake/omniglot上下载下来的。下载下来的压缩包中只用到了images\_background.zip和images\_evaluation.zip，也即是训练集和测试集。解压之后看到其中一类数据如下：

omniglot数据集拥有50种，1623类手写字符，对于每类字符仅有20个样本。其中某类字符的数据样本如上所示。

这里，一般把样本分为30类训练样本，剩下20类作为评估。

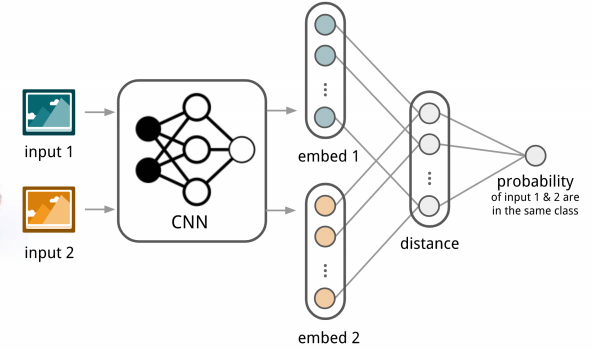
1. **数据预处理**

这里对数据进行预处理，需将这些文件全部转换成pickle流数据。Pickle文件的目的是将数据压到一个文件中永久保存。这样在取用时，只需将该文件中的数据取出，而不用每次都重新进过各种语句，处理得到对象数据。具体处理过程如下：

这里遇到一个问题：就是处理文件的时候，使用args = parser.parse\_args()方法无法获取用户输入的路径参数，也就索性直接在代码里加入了放置文件的目录。而且最奇怪的是：一共需要输入两个参数，一个是文件所在的目录，另一个是pickle文件要保存的目录，但是只能读取到后者，读取不到前者，因此可以看到代码里面只是直接将前者的路径加进去了。

1. **网络构建**

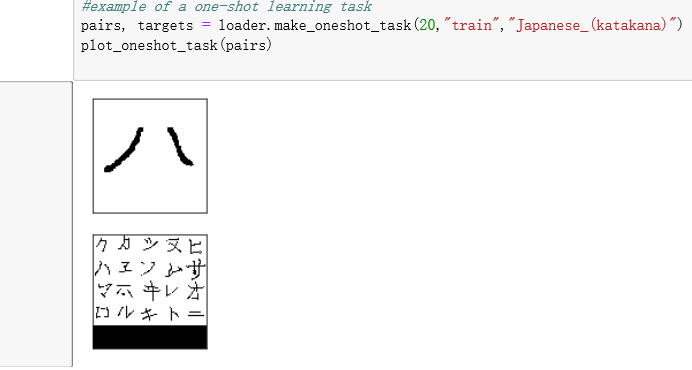
我觉得构建孪生网络，老师给的资料中的网络不够“孪生”！到此处我依旧没有明白他是怎么做到使得两个网络的参数相同之类的。这里直接构建孪生网络，即就像课件中给到的，如下图所示：

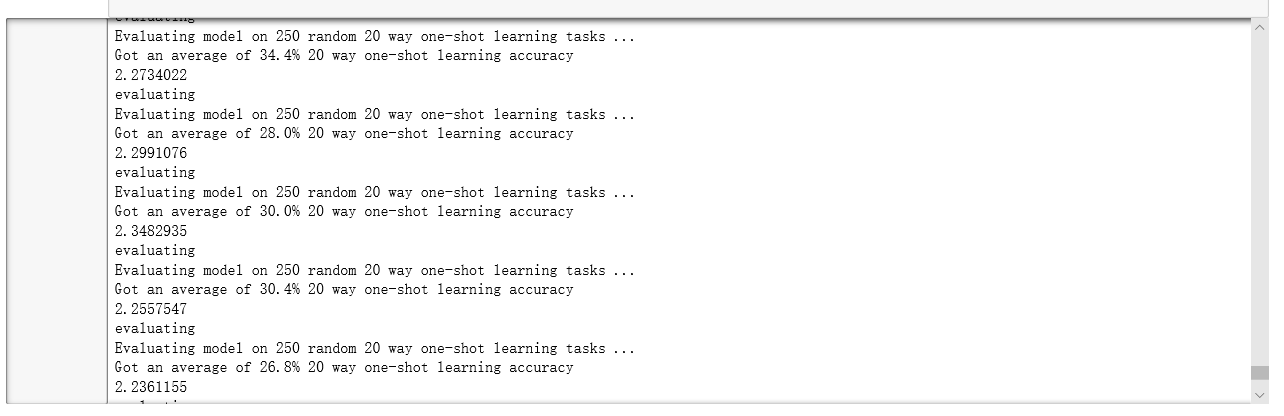


直接构建两个左右两个输入：

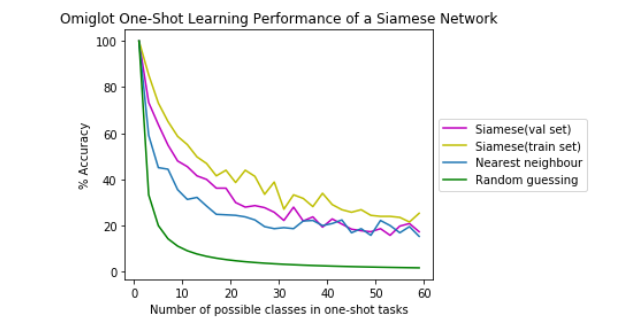
两个孪生网络都是卷积神经网络。这两个孪生网络每个的架构如下：64通道的10×10卷积核，relu->max pool->128通道的7×7卷积核，relu->max pool->128通道的4×4卷积核，relu->max pool->256通道的4×4卷积核。

1. **构造在Omniglot数据集上单样本学习的数据**

 输出其中一个例子如下：

1. **训练**
2. **结果**

其结果如下图所示：



1. **结果分析**

可以看到，结果如上图所示。最下面那一根线是随机猜测，黄色的线是训练集中的正确率，紫色的线是测试集的正确率。至于为什么会这么低，是因为训练次数不够，查阅网上的资料，一般都进行了几十万次的训练，这里只训练了几百次。不过几百次的训练就已经花了两个多小时，然后就停止了训练。在网上看到的最终结果差不多是如下图所示：

