**基于计算机视觉的人工智能学习**

牛远卓，王浩哲

2012.07.10

注意：这是以前的一个报告，我在此基础上又加了点东西。王浩哲是我当时所在队伍的小队成员。

**一、项目结构**

Train.py

Model.py

Evaluate.py

Test.py

**二、实验过程**

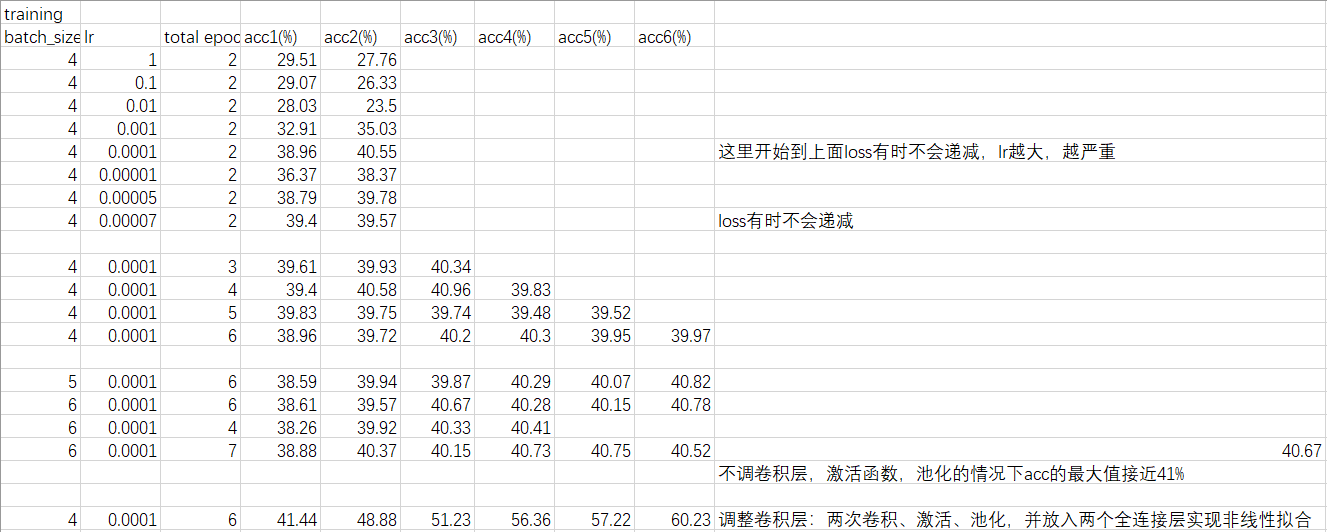
**（1）Cnn卷积神经网络**

先在pycharm上装了pytorch。

老师提供初始lenet5神经网络代码。Cnn卷积神经网络结构如图。



我们在老师给的代码上做修改，先将数据放在gpu上跑，这样可以快一点。然后调参数，比如lr,total epoch,batch\_size,意思分别为反向传播时对梯度的权重，总共训练几次和每次训练几个数据。最后调整卷积层，按照lenet5的方式改，提高训练正确率。结果如下。

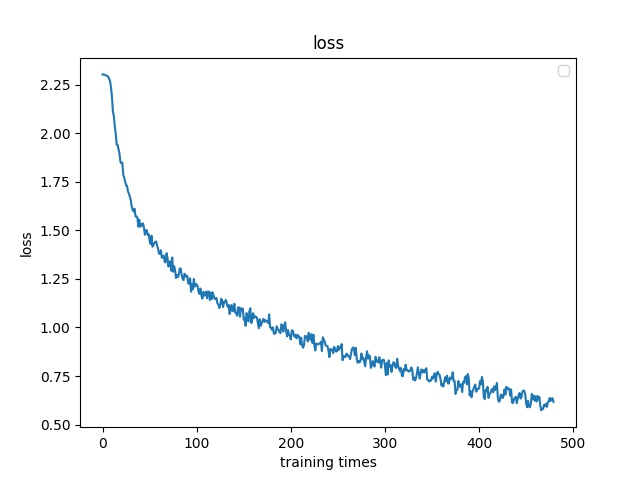


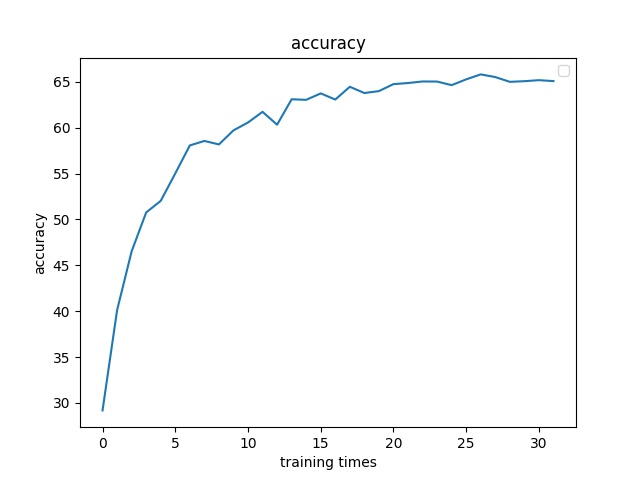
**结果分析**

训练次数为32

batch\_size=32

最高准确率在65%左右

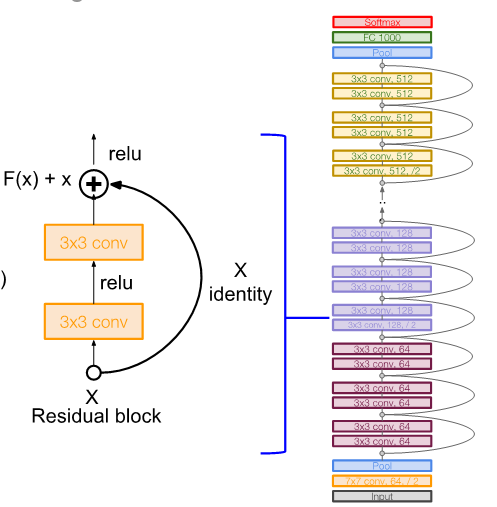
****

****

**（2）Resnet残差网络**

**结构介绍**

随着网络层数 的加深，出现了训练集准确率下降的现象，我们们可以确定这不是由于Overfit过拟合造成的(过拟合的情况训练集应该准确率很高)；所以作者针对这个问题提出了一种全新的网络，叫深度残差网络，它允许网络尽可能的加深，其中引入了全新的结构如左图。



其中ResNet提出了两种mapping：一种是identity mapping，指的就是左图中”弯弯的曲线”，另一种residual mapping，指的就是除了”弯弯的曲线“那部分，所以最后的输出是y=F(x)+x

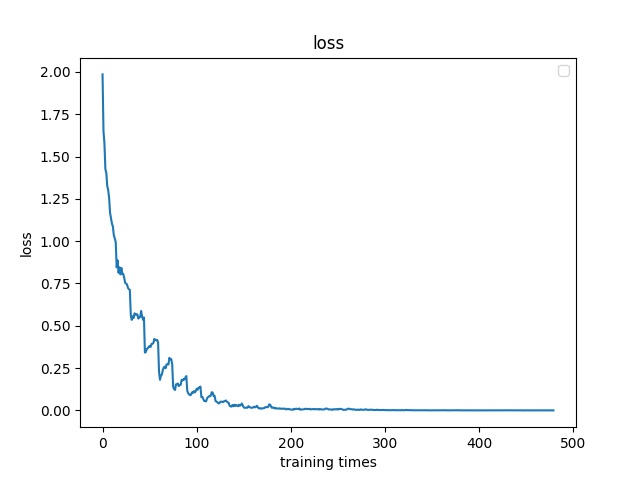
identity mapping顾名思义，就是指本身，也就是公式中的x，而residual mapping指的是“差”，也就是y−x，所以残差指的就是F(x)部分。

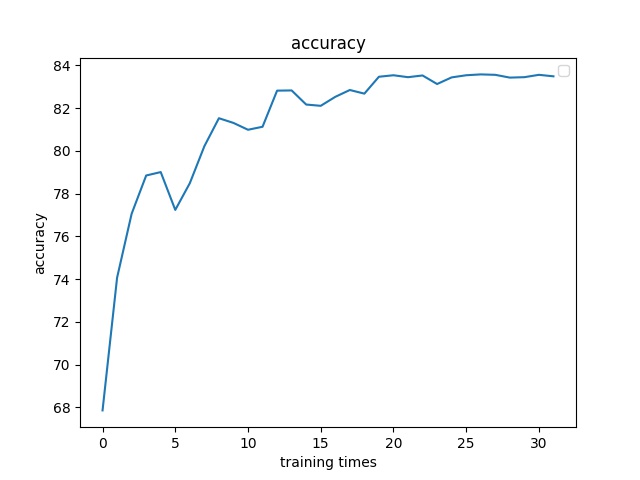
**模型结果**

训练次数为32

batch\_size=32

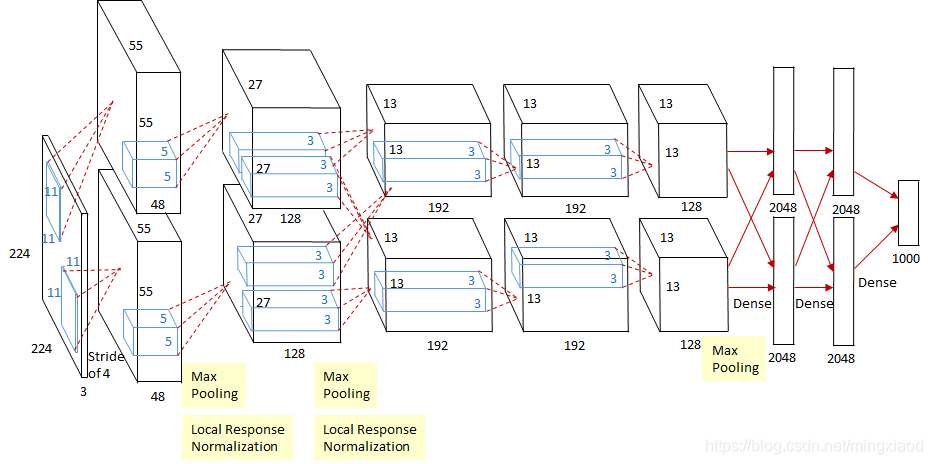
最高准确率在**84%**左右

****

****

**（3）Alexnet神经网络**

**结构介绍**

****

举第一层为例

输入的图片大小为:224\*224\*3（或者是227\*227\*3）

第一个卷积层为:11\*11\*96即尺寸为11\*11,有96个卷积核,步长为4,卷积层后跟ReLU,因此输出的尺寸为 224/4=56,去掉边缘为55,因此其输出的每个feature map 为 55\*55\*96,同时后面跟LRN层,尺寸不变.

最大池化层,核大小为3\*3,步长为2,因此feature map的大小为:27\*27\*96.由于当年硬件限制，不能一下处理这么庞大的数据。于是分成27\*27\*48\*2的两路处理。

**网上的摘抄**

**新的技术点**：

**五个卷积层+三个全连接层**

（1）成功使用ReLU作为CNN的激活函数，并验证其效果在较深的网络超过了Sigmoid函数，成功解决了Sigmoid在网络较深时的梯度弥散（消失）现象。虽然ReLU激活函数在很久之前就被提出了，但是直到AlexNet的出现才将其发扬光大。

（2）训练时使用Dropout随机忽略一部分神经元，以避免模型过拟合。在AlexNet中主要是最后的几个全连接层使用了Dropout。

（3）在CNN中使用重叠的最大池化。以前CNN中普遍使用平均池化，AlexNet全部使用最大池化，避免平均池化的模糊化效果。并且AlexNet中提出让步长比池化核的尺寸小（eg：size=[3,3], stride=2.）,这样池化层的输出之间会有重叠和覆盖，提升了特征的丰富性，减少了信息的丢失（最大池化：可以保留最显著的特征）。

（4）提出了LRN层，对局部神经元的活动创建竞争机制，使得其中响应比较大值变得更大，并抑制其他反馈较小的神经元，增强了模型的泛化能力。

（5）使用CUDA加速深度卷积网络的训练，利用GPU强大的并行计算能力，处理神经网络训练时大量的矩阵运算，双卡通信。

（6）数据增强：Data Augmentation

- 方法1：随机裁剪，训练：随机的从256256的原始图像中截取224224大小的区域（以及水平翻转的镜像），相当于增加了（256-224）的平方的2倍=2048倍的数据量。

测试：取图片的四个角加中间共5个位置，并进行左右翻转，一共获得10张图片，对它们进行预测并对10次结果求平均。

- 方法2：对RGB数据进行PCA处理，并对主成分做一个标准差为0.1的高斯扰动，增加一些噪声，这个Trick可以让错误率再下降1%。

AlexNet输入的图片尺寸为224224，第一层使用了较大的卷积核尺寸1111，步长为4，有96个卷积核；紧接着一个LRN层；然后是一个33的最大池化层，步长为2。这之后的卷积核都比较小，都是55或者33的大小，并且步长都为1，即会扫描全图所有像素；而最大池化层依然保持为33，并且步长为2，出现在第一个和第二个以及最后一个卷积层之后。

思考：Dropout 为什么奏效？

Dropout 背后的原理与模型集成类似。由于 Dropout 层的作用，关闭的不同神经元集呈现一种不同的架构，并行训练所有这些不同架构，赋予每个子集权重，权重的总和为 1。如果 Dropout 连接了 n 个神经元，则子集架构的数量是 2^n。因此，预测是对所有这些模型的集成取平均。Dropout 有用的另一个原因是：由于神经元是随机选择的，因此它们更有可能避免产生共适应（co-adaptation），从而产生独立于其他神经元的有意义的特征。

Dropout率的选择：

（1）经过交叉验证，隐含节点dropout率等于0.5的时候效果最好，原因是0.5的时候dropout随机生成的网络结构最多。

（2）dropout也可以被用作一种添加噪声的方法，直接对input进行操作。输入层设为更接近1的数。使得输入变化不会太大（0.8）。

**三、我们的改进**

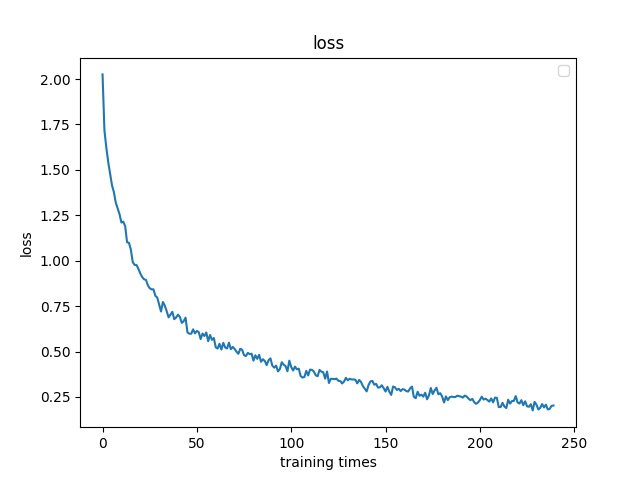
加了水平翻转的数据增强和learning rate 的动态地随训练轮数的增加有所降低以找到更精确的lossfunction的最低值的方法后。

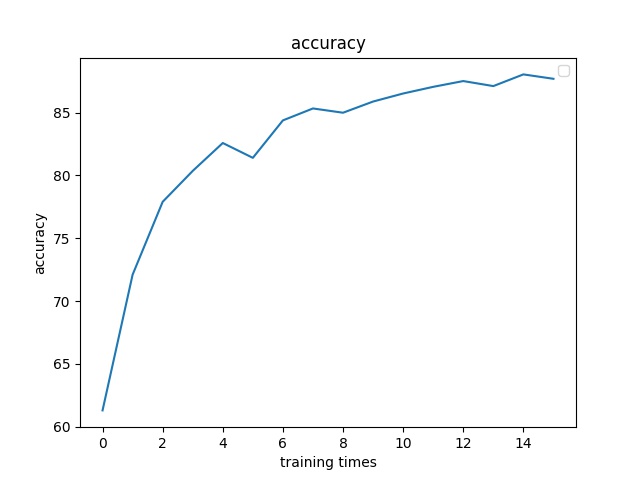
**结果分析**

训练次数为32（不用为2的幂次）

batch\_size=16

最高准确率在88%左右





**总结与感想（这以上都是以前做的）**

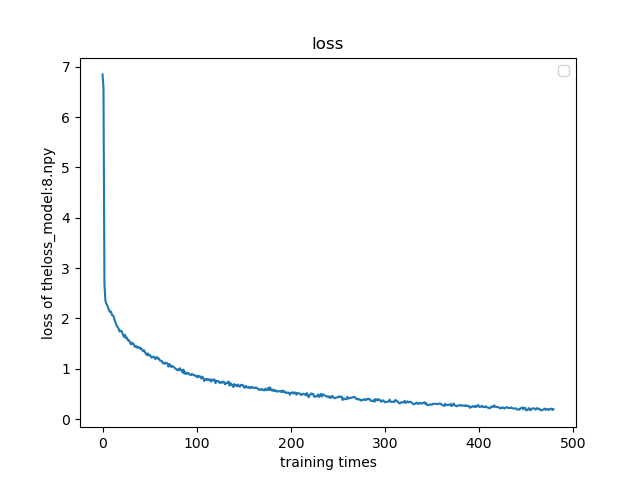
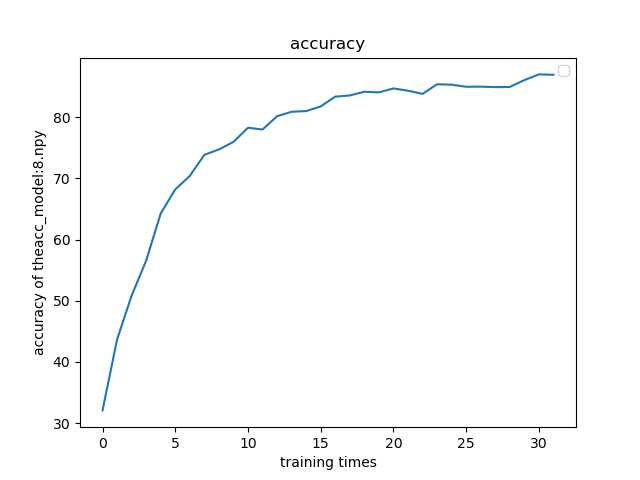
这次初入机器学习模块，总体感受极佳。 小组做了lenet5，resnet残差神经网络，alexnet卷积神经网络，并且给相应代码加了些数据增强和不一样的lr的动态变化，实现了对cifer10数据集的从正确率65%到88%的蜕变。我们的正确率依旧不尽如人意，但是老师在答辩时指出了我们有待改进的地方——可能alexnet代码写错了。这给了迷茫的我们指明了道路。老师教得易懂，专业性没的说，而且看得出来我们老师是对这份职业有超过养家糊口的热情的，这是超过许多老师的素养的。这门课本身也非常吸引人，因为神经卷积网络可以对成百万上千万的图片进行处理。但是数据量大时，对硬件gpu要求高，过了这次夏令营的租用的gpu试用期以后跑程序存在困难，希望以后硬件能同步改善吧。

**2022/4/20最新进展**

由于最终用的alexnet对cifer10数据集的正确率停留在88%，我希望能看是不是因为我的alexnet代码出错了。但是，由于这是几个月之后的事了，我遇到了一些问题。我在重新训练时，发现自己的准确路一直停留在10%，这很不正常。于是在排除了版本和数据集问题后，一通debug，发现backward()根本没调用。则会很奇怪，因为做ai\_master时，是可以的。这有可能是pytorch版本的改动问题。所以说我查了一下我的pytorch版本下应该用哪个包来backward。最后发现torch.functional. cross\_entropy就可以达到我想要的效果。



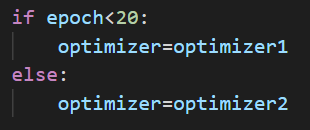
之后准备按照网上的alexnet描述重新回顾，对我的原有代码进行纠错。我在数据增强时，将输入图片大小resize成224，并且randomcrop的大小也是224.。之后，我在网上找了一段alexnet的代码以epoch为32，batchsize32重新训练，结果如下。



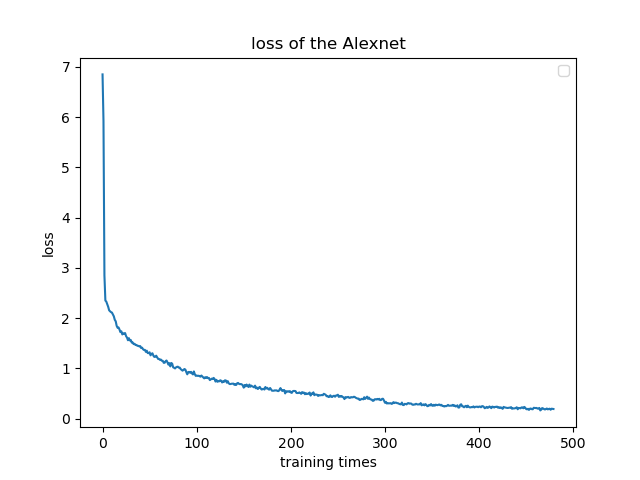
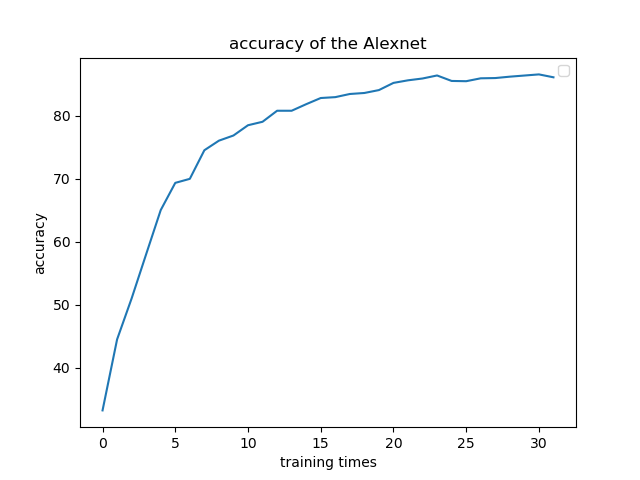
最后正确率是86%，比原来还低。于是我准备在此基础上寻找提高准确率的方法。

我发现原来的代码根本没有在训练到一半时更改learning rate。根据现有的准确率的图，我们发现，在epoch从20到23这一部分是掉的，我合理怀疑这附近可以减小learning rate以达到更好的效果。



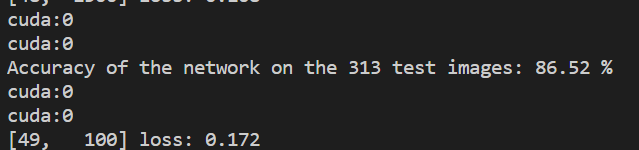


改过后准确率还是86%，哈哈



我又按照ImageNet Classification with Deep Convolutional Neural Networks这篇论文中的对alexnet的描述重新调整了一下各项参数。也按照128的batchsize跑cifer10，但是我发现这样每次只能跑400次（原来batchsize为32时可以跑1600次）。这导致了我的效果很不好，就不展示了。（而且一直没遇到overfitting的问题。）

于是，我又将batchsize调回32，epoch调成64试了一下。



暂时最高时86.5%。（由于时间太久而我要去做别的事，没到64次，就没输出可视化图片）