

# 人神作业 2—— CNN 实验报告

姓名：范国杰

班级：计 64

学号：2015011619

## 目录

一、	实验要求 .....	3
二、	实现思路 .....	4
1、	loss 使用 softmax 交叉熵 .....	4
2、	relu、linear 全连接使用和 mlp 相同的做法 .....	4
3、	卷积实现思路 .....	4
	scipy.signal.convolve2d+for 循环 .....	4
	im2col .....	4
	经典图像卷积处理——快速卷积 .....	4
4、	参数更新实现思路 .....	8
	grad_b: .....	8
	grad_w: .....	8
5、	池化实现思路 .....	8
6、	池化梯度实现发现 .....	8
三、	实验结果 .....	9
1、	loss 和 accuracy rate 随 epoch 的变化 .....	9
2、	4 个指定类别 .....	9
3、	训练的最后结果 .....	10
四、	对比 MLP .....	11
1、	运行速度 .....	11
2、	准确率 .....	11
3、	tradeoff .....	11
五、	总结 .....	12

## 图表目录

Figure 1	带 padding 的 lena .....	5
Figure 2	把 9 张图片加权相加 .....	7
Figure 3	grad_w 标准公式 .....	8
Figure 4	局部梯度递推公式 .....	8
Figure 5	loss 和 accuracy rate 随 epoch 的变化 .....	9
Figure 6	标签 3 .....	10
Figure 7	标签 2 .....	10
Figure 8	标签 1 .....	10
Figure 9	标签 4 .....	10
Figure 10	训练的最后结果 .....	10

## 一、实验要求

- 1、使用 numpy 实现 cnn 对 mnist 进行分类
- 2、将 loss 和准确率随 epoch 变化输出
- 3、输出 4 个不同的指定类别的图片
- 4、对比 mlp 和 cnn

## 二、实现思路

### 1、loss 使用 softmax 交叉熵

softmax 公式： $\exp(x_i) / \sum_j \exp(x_j)$

softmax 可以视为多分类的 sigmoid 实现。从定义上满足了归一性，本身可以作为预测的结果。

交叉熵的公式 -  $\sum y_i \log(p_i)$ ,  $y_i$  是第  $i$  个数据的标签， $p_i$  是对第  $i$  个标签的预测概率，标签是 onehot 编码的，预测概率也是一个向量，当二者越接近的时候，值越小，因而可以作为损失函数使用。

二者公式看起来都比较复杂，但求导的时候会发现只有留下  $y_i - p_i$  这一项，因而计算简单

### 2、relu、linear 全连接使用和 mlp 相同的做法

这一部分在上一份报告里面已经详细论述了，这里不做细致讨论

### 3、卷积实现思路

#### scipy.signal.convolve2d+for 循环

开始的时候使用 `scipy.signal.convolve2d+for` 循环的方案，有结果，但训练一个 epoch 大概要  $18s \times 600 = 3h$  的时间，我最长让它训练了 13 个 epoch，能有 95 的准确率，但速度慢的惊人。

#### im2col

网上的方案大多用 `im2col`，但网上的实现感觉都太繁琐了。同学说用这个方案一个 epoch 可以达到最快 35s。是上面的两个数量级

#### 经典图像卷积处理——快速卷积

经过同学的启发，上学期听了数字图像处理的课，想起来可以用经典图像处理的方法来做卷积，以  $3 \times 3$  的卷积核为例，见下图

a11	a12	a13
a21	a22	a23
a31	a32	a33

第一步：我这里的卷积用的都是 same，因而先生成带有 padding 的图像

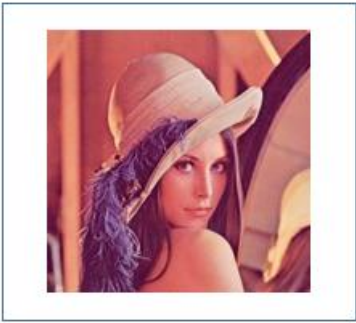
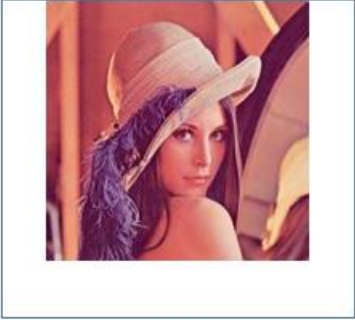
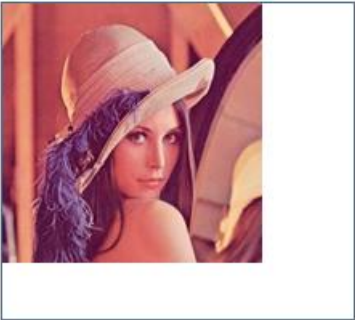


Figure 1 带 padding 的 lena

第二步：根据卷积核相应元素，反方向移动图片，说的比较抽象，见下表

a11	The Lena image shifted one pixel to the top-left corner within its padding border.
a12	The Lena image shifted one pixel to the top-right corner within its padding border.
a13	The Lena image shifted one pixel to the bottom-left corner within its padding border.

a21	
a22	
a23	
a31	

a32	
a33	

第三步 :把对应系数和相应的图片相乘, 而后全部加起来, 这样得到的就是 full 模式的卷积。  
我们现在只要 same, 故只把中间部分取出来。

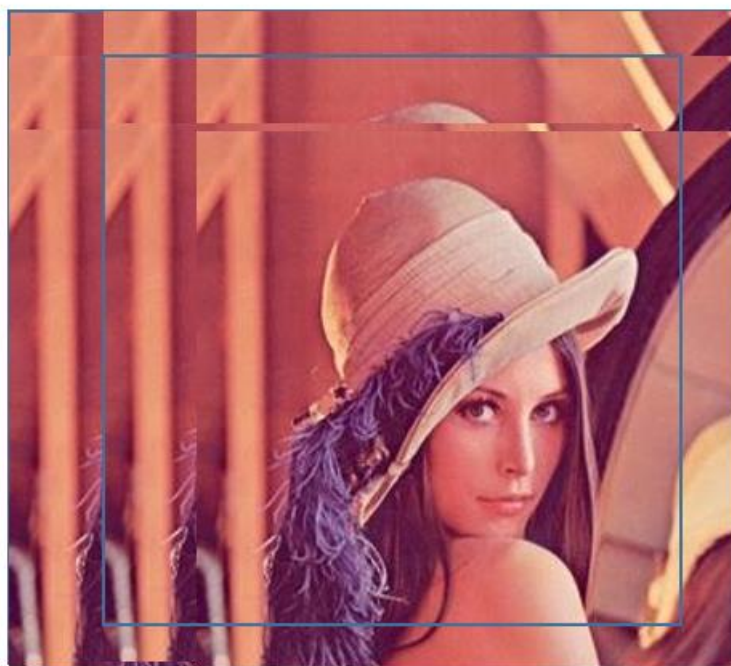


Figure 2 把 9 张图片加权相加

如此计算, 一个卷积操作只要把 9 张图片加起来的的工作量, 大大减小了计算量。

一个 epoch 用时 28s, 更快, 并且卷积代码共 3 行, 其中两行在写 for 循环, 可以说是非常的简洁了

## 4、参数更新实现思路

这里只说明卷积层的参数更新：

### grad\_b:

这个比较简单，把梯度矩阵除了输入层的 channel 以外其他维度全部相加即可

值得注意的是，我把 grad\_b 恒定为 0，也就是在卷积时没有 b 的影响，在 20 个 epoch 也可以达到 99% 的准确率了，猜测这里它的影响很有限

### grad\_w:

引用课件上的结论

$$\frac{\partial E^{(n)}}{\partial w^{(l)}} = y^{(l)} *_{\text{valid}} \text{rot180}(\delta^{(l+1)})$$

Figure 3 grad\_w 标准公式

局部梯度的递推公式也引用课件上的例子：

$$\delta^l = \left( \delta^{(l+1)} *_{\text{full}} w^{(l)} \right) \bullet \left( f'(u^{(l)}) \right)$$

Figure 4 局部梯度递推公式

但我们这里把 relu 单独作为一层提取出来了，因而局部梯度只需要被乘数一项即可

## 5、池化实现思路

没考虑 padding ,个人觉得池化的时候 padding 没什么用，按 0 处理了，平均池化很直观，用切片就可以实现

## 6、池化梯度实现发现

理论上说，池化梯度应该是输入的梯度除以池化的大小。但实验中发现，如果是乘以池化核的大小收敛速度大大加快

就算迭代了 300 个 epoch，用乘的方法能到 99% 左右，可是用除以的方法只有将近 98，差距还是有的

和助教讨论的结果是，这一层的 lr 可能偏小了，但我这里的 lr 都是全局的，如果是 caffe 就可以对每一层的 lr 都给一个值来

但至少，事实证明池化层这里乘除一个常数对网络模型没有本质影响，对收敛速度会有差别



### 三、实验结果

#### 1、loss 和 accuracy rate 随 epoch 的变化

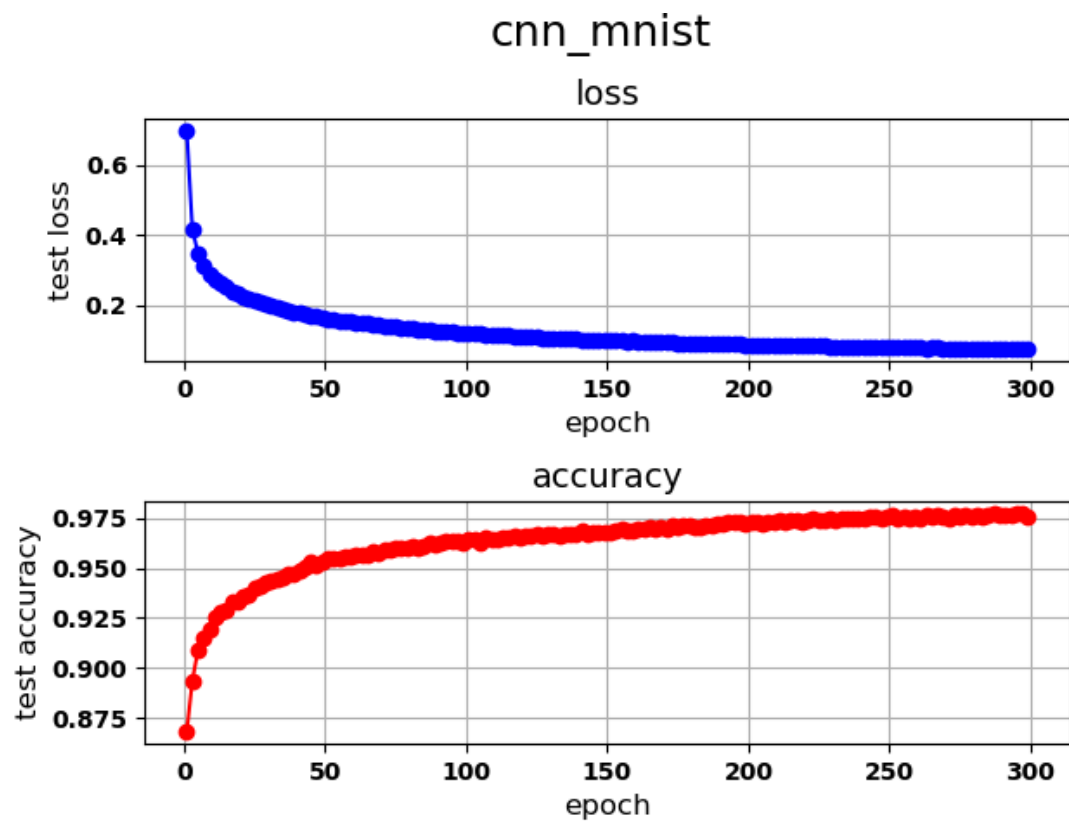


Figure 5 loss 和 accuracy rate 随 epoch 的变化

#### 2、4 个指定类别

```
categories=[1,2,3,4]
```

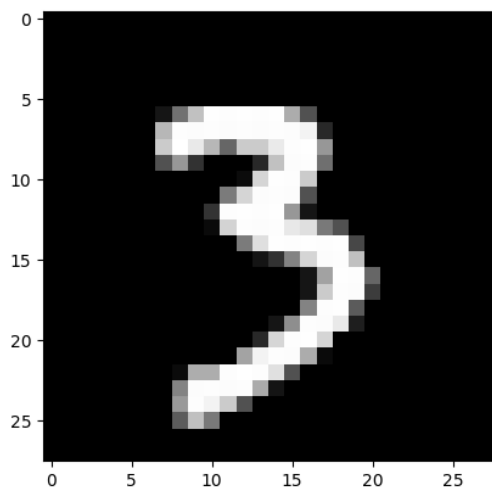


Figure 6 标签 3

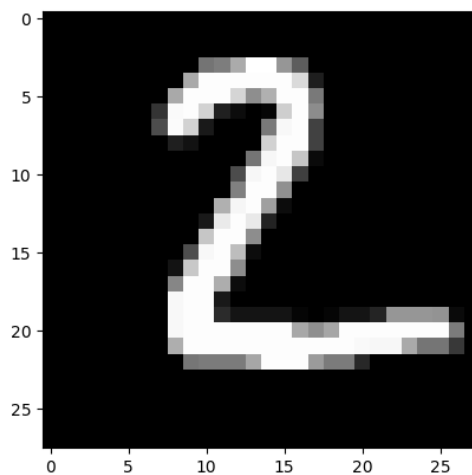


Figure 7 标签 2

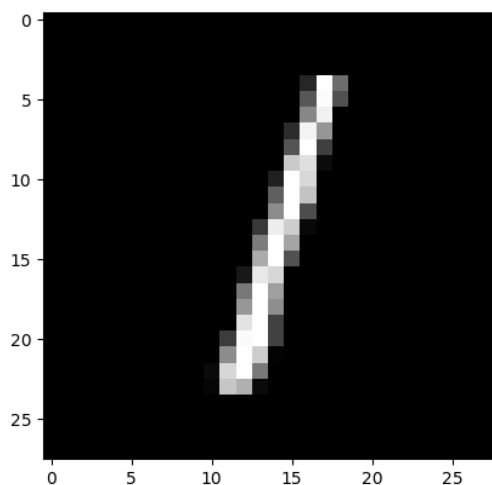


Figure 8 标签 1

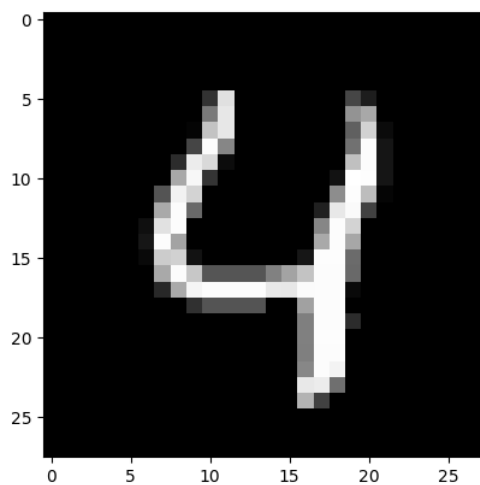


Figure 9 标签 4

### 3、训练的最后结果

```
17:31:13.240 Training @ 299 epoch...
17:31:19.265 Training iter 20, batch loss 0.0309, batch acc 0.9913
17:31:25.007 Training iter 40, batch loss 0.0293, batch acc 0.9898
17:31:30.948 Training iter 60, batch loss 0.0330, batch acc 0.9890
17:31:36.898 Training iter 80, batch loss 0.0370, batch acc 0.9897
17:31:43.229 Training iter 100, batch loss 0.0317, batch acc 0.9900
17:31:48.152 Training iter 120, batch loss 0.0329, batch acc 0.9897
17:31:52.556 Training iter 140, batch loss 0.0295, batch acc 0.9918
17:31:58.015 Training iter 160, batch loss 0.0423, batch acc 0.9872
17:32:03.458 Training iter 180, batch loss 0.0340, batch acc 0.9892
17:32:09.559 Training iter 200, batch loss 0.0325, batch acc 0.9900
```

Figure 10 训练的最后结果

## 四、对比 MLP

### 1、运行速度

cnn 网络显然比 mlp 一层两层深，计算量由于卷积的存在也更大，速度自然更慢。但不得不说，这慢的的不是一点两点

### 2、准确率

cnn 网络更“高级”，而且深度也更深，自然效果要更好，我训练了 200~300 个，只到 99%左右，网上说，最好的有 99.8%。但这显然已经不本质了，因为有些错误就算是人也很容易认错

### 3、tradeoff

现实情况是需要权衡利弊的。mlp 可以轻松到 98%+，上了 cnn，参数合理 99%+，需要调参加上训练本身时间就更长。

结论：cnn 是经典的图像处理方法，但对于像 mnist 这种简单的图像，mlp 就足够好了。像素级别大的可以降低像素训练效果也不会太差，毕竟在图像中的冗余信息还是太多了。

## 五、总结

手写完 CNN，感觉整个人都升华了。特别是看到它能跑的时候，还是非常的开心的。尽管中间有不少坎坷，但结果是满意的，也因此学习了不少编程技巧。比如 softmax 的时候要先统一减去一个最大值，不然可能会越界等神奇事情发生。计算 log 的时候，输入量用 np.clip 限制一下，虽然正常计算是不会出问题的，但特殊情况会有奇怪的事情发生干扰 debug。

实现过程有和同学讨论和向 Internet 取经，如实回答不是全部自己拍脑袋想出来的，但总体框架肯定都是自己手写出来的。

终于可以用 TensorFlow 了，开心一下。