人工神经网络 HW2 实验报告

计 52 路橙 2015010137

Part 1：综述

本次实验中，我实现了 CNN 神经网络的构建，完成了 CNN 卷积层和池化层的实现，并再次对 MNIST 数据集进行训练和测试，最终极大程度优化了代码的速度性能，且最高测试准确率可达 0.985。

关于参数的选择上，我调节的参数有如下几种：

- 卷积层的 kernel_size
- 卷积层的 channel_out
- 卷积层的权值初始化方差（init_std）
- Learning rate 的初始选择及动态调整的策略
- Weight decay
- Momentum

通过调节如上参数，发现了一些参数调节的初步规律，尝试做出了一些解释。

此外，我在实现内部函数时，避免了所有的 for-loop，使得训练效率大幅提高，在我自己的笔记本上（Intel Core i7 4710HQ，主频 2.5GHz， 四核）**只需要 1.5h** 便可以完成原始代码初始规模的网络的训练（kernel_size=3， channel_out=4， init_std=1，learning_rate=0.01，weight_decay=0，momentum=0.09）。

Part 2：训练性能加速方法（CPU）

① 卷积层：

1. Im2col 的实现：

下图展示了在 batch_size=1 的情况下，如何将输入的 4 维矩阵卷积转化为 2 维矩阵乘法，从而避免 for-loop，利用 numpy 的特性大幅度加速计算性能。

具体地，分别实现对 filter 的 im2col、对 input 的 im2col，对 output 的 col2im：

(1)Filter 的 im2col 直接调用 numpy.reshape 函数即可；

(2)Input 的 im2col，需要将输入的 4 维矩阵进行“可重复的切片”，并 reshape 成与 filter 相适应的维度。

此处，利用 *numpy.lib.stride_tricks.as_strided* 将矩阵元素进行数组下标的重排，实现对矩阵的切片并重排。

(3)在进行矩阵乘操作后，需要对 ouput 进行 reshape，切换到正确的输出维度。此处需要注意的是，reshape 需要结合 transpose 一同使用，从而使元素正确归位。

因此，利用 numpy 的 reshape、transpose、stride_tricks.as_strided 可实现快速卷积，且避免了所有的 for-loop，大大提高了卷积层的计算效率。

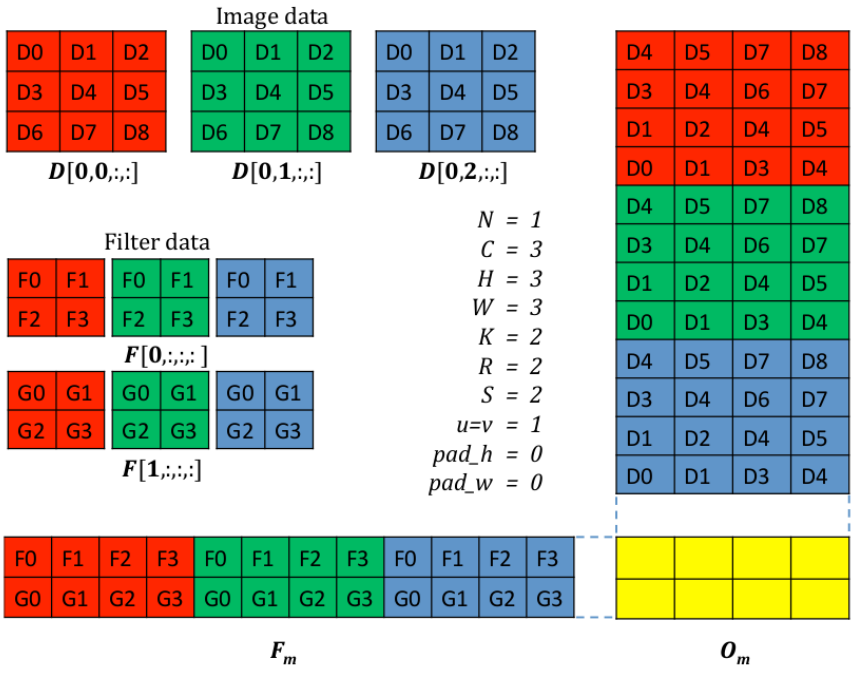


Figure 1. im2col 原理（图片来源知乎）

2. 卷积运算的实现：

利用 numpy 的 rot90 函数可以实现旋转 180 度，由于矩阵的卷积等价于其中某一矩阵旋转 180 度后，采用卷积层的 forward 运算，因此 forward 运算中的 im2col 可以被复用，从而在卷积层中 backward 计算局部梯度时，可以将局部梯度旋转 180 度后采用与 forward 计算相同的算法，避免了 for-loop 逐 channel 求卷积。

② **池化层：**

1. Forward 计算平均池化：

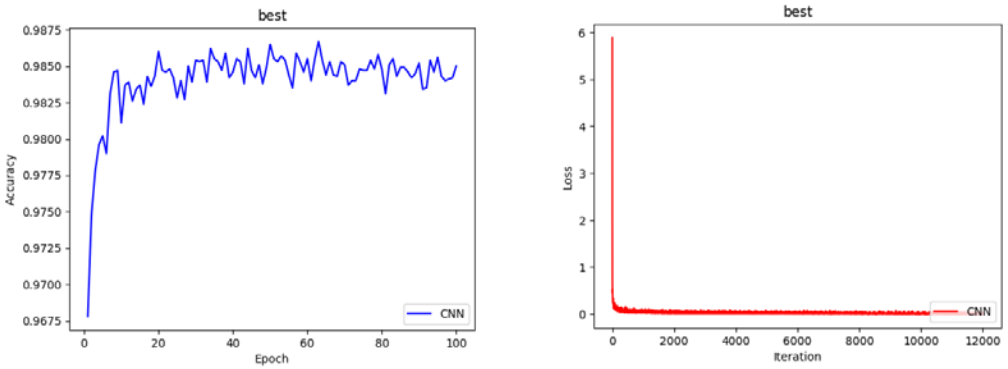
利用 im2col，添加 stride=kernel_size，对 input 进行裁剪切片后，对列调用 numpy.mean，最终进行 reshape，从而避免了逐 batch、逐 channel 地进行平均。

2. Backward 计算 upsample：

利用 *numpy.kron*，对每个元素进行超采样，从而避免了 for-loop。

Part 3：训练过程及超参数的选择

① **最优结果：**



最优结果最终波动较大，猜测是由于 CNN 对梯度变化敏感，初始的 Learning Rate 在后期对于当时的模型参数而言较大，应该采取动态调整的策略。

最终在 test 上的准确率稳定在 0.985 左右。

② **参数的调整过程：**

主要参数优化过程见 **Part 5.**

在调节过程中，发现几个超参数调整特点：

- Kernel_size 增大时，准确率和稳定性会急剧增高。
- Channel_out 增大时，准确率和稳定性会急剧增高。
- Learning Rate 初始化时较合适或 Momentum 初始化较大时，可以在第一个 epoch 结束后就能达到 0.93 左右的准确率，这也说明 Relu 的效果很好，收敛很快。
- 初始时设置时，网络的规模决定了最终调节的上界，而 Learning Rate 和 Momentum 的搭配决定了离这个上界有多近。

- Kernel_size 和 channel_out 是 CNN 独特有的网络规模相关的参数，稍微调整便可对网络的性能产生极大影响，从网络收敛速度到准确率都有明显影响。
- 尽管 Relu+CNN 可以很快地收敛到一个较好的结果，但如果初始时 Learning Rate 和 Momentum 的值搭配不当，很有可能出现较普通（收敛到 0.96 左右）的结果。

经实验发现，双层 CNN 的一组较优参数为：

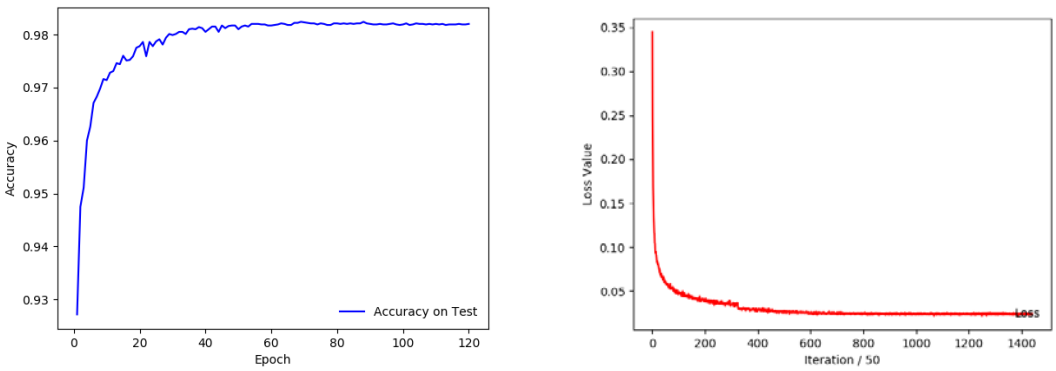
- **Learning Rate:0.01，且动态调整（详见 Part 4）**
- **Weight decay:0**
- **Momentum:0.9**
- **Batch size:100**
- **Init_std:1**
- **Kernel_size：3**
- **Channel_out: 16**

以上参数可以在不到 5 个 epoch 即可获得 0.98 以上的准确率，最终结果为 0.990.

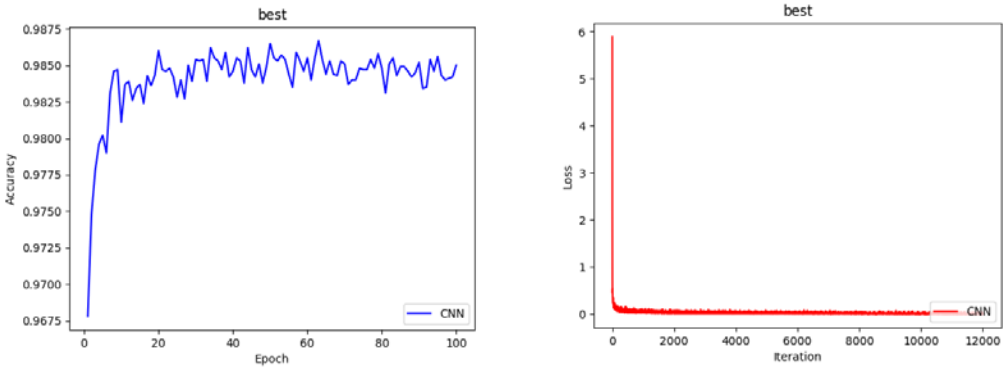
Part 4：CNN 与 MLP 的区别比较：

① 最优结果比较：

MLP 单层+Relu 最优结果：



双层 CNN 最优结果：



在训练过程中，发现二者最优结果的训练过程中有如下区别：

- CNN 的参数个数主要由 kernel_size 和 channel_size 决定，对于 kernel_size=3，channel_size=4（即原始代码默认参数）的情况下，CNN 的参数个数为：

  $(1 \times 4 \times 3^2 + 4) + (4 \times 4 \times 3^2 + 4) + (196 * 10 + 10) = 2158$

  而 MLP 在隐含层大小很小（100 * 50）时，参数个数为：

  $(784 \times 100 + 100) + (100 * 50 + 50) + (50 * 10 + 10) = 84060$

  CNN 参数个数占 MLP 参数的百分比为 2158/84060 = **2.53%**

  这说明，MLP 中很多参数实际上都是有很大相关性的，即 MLP 的参数用高维数组描述了一个低维特征，其中有很多维度都互相有关联，并不是本质的刻画。而 CNN 用远少于 MLP 参数个数的参数刻画了同样的特征，甚至准确率高于 MLP，这一方面可以说明原理层面上的改变比参数的调整更重要，另一方面也可以说明 CNN 在图像处理领域的强大。

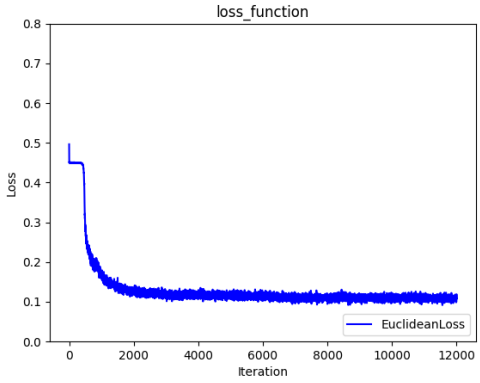
- CNN 训练速度极慢，在阿里云的高性能计算云平台上进行计算时，CNN 需要 3h 左右才可以基本收敛，但 MLP 在十几分钟之内便可收敛。

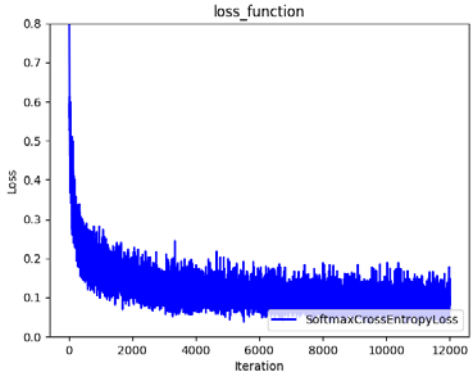
  一方面，由于 CNN 需要的计算量较大，使得 CNN 的计算较慢，即使在参数量远小于 MLP 的情况下，依然需要很长的训练时间；另一方面，由于 MNIST 数据量像素较小，任务较简单，用 CNN 来操作仿佛“杀鸡用宰牛刀”，无法明显突出二者的准确率等性能上的差距。

- CNN 收敛速度较快，基本在 10 个 epoch 以内便可收敛至 0.98 左右的准确率，这一点 MLP 差之千里。这也体现出 CNN 效果优秀的特点。
- 在准确率上，二者最优结果有较相近的准确率，CNN 的准确率略高于 MLP。

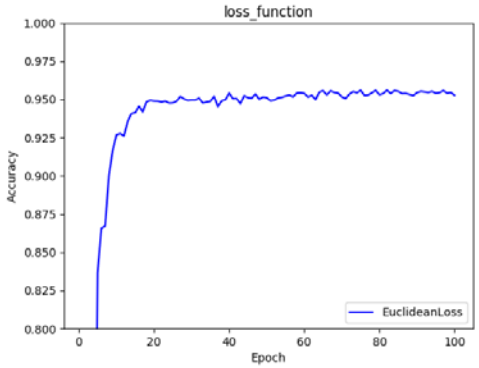
② **Loss 函数比较：**

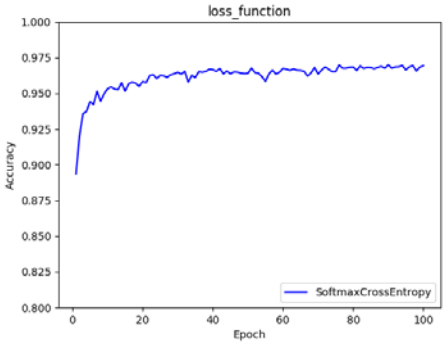
CNN 采用两种不同的 Loss 函数时，loss 的变化如下图：





Accuracy 如下：

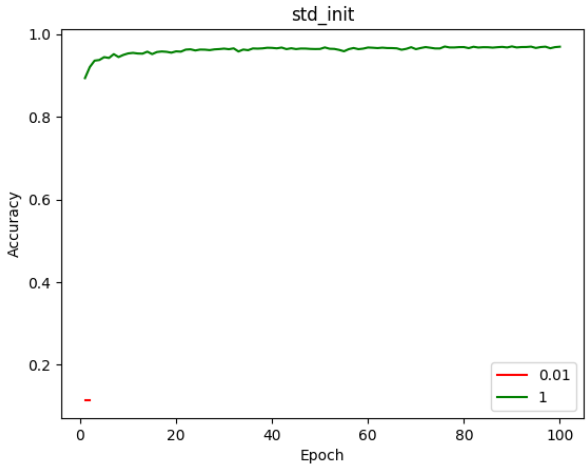




由图可见，softmaxCrossEntropy 中 loss 的抖动极大，但 accuracy 却在稳步上升，并超过前者。因此，CNN 中 softmaxCrossEntropy 的 loss 函数对 CNN 的性能提升起到了关键作用。

③ **Init_std 比较：**

在训练 CNN 时发现，若 std_init 调整太小（0.01），网络会无法开始训练，只有当 std_init 较大时（1 左右）网络才会很快收敛。（见下图）



而在 MLP 的训练过程中，std_init 调整较小时才更容易收敛至最好的结果。

④ **Learning Rate 比较：**

CNN 在 Learning Rate 较大时极容易产生准确率的很大浮动，即绕过最优点（详见 Part 5 中关于 Learning Rate 的部分）；

但 MLP 需要较大的 Learning Rate 才可以启动训练，加速收敛。若 Learning Rate 较小，则收敛速度会较慢。

Part 5：网络训练参数优化：

① **Learning Rate 的选择：**

1. 理论层面分析 Learning Rate 大小对训练结果的影响：

学习率决定了权值更新的速度，设置得太大会使结果越过最优值，太小会使下降速度过慢甚至原地徘徊。

2. Learning Rate 调整的实验过程以及最终采纳的方法：

(1) 设定固定的 Learning Rate：

通过打印 loss 下降的曲线可以看出，loss 在下降到一定范围后便不再下降，因此固定的 Learning Rate 在训练到后期时便会无法更精细地调整网络。因此，我采取了如下动态设置的策略。

(2) 动态设置 Learning Rate：

学习率较大时可快速收敛，较小时可精细地优化，因此我采用动态地减小学习

率的方式进行动态调参。通过对最近 10 次 loss 的观测来判断是否训练趋于稳定，若训练趋于稳定则将 Learning Rate 下降。

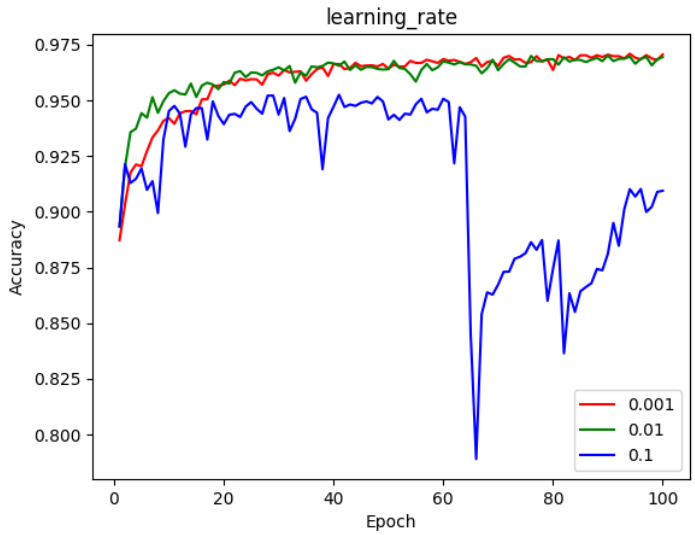
具体地，每次观测 loss 是最近 1 次 epoch 在测试集上的输出。把最近 10 次记录的 loss 存一个列表里，并把最新一次的 loss 与这 10 次的平均做差，若绝对值小于 0.001，则将此时的 Learning Rate 乘以 0.5，并把列表清空。（保证不会连续减小 Learning Rate，两次减小至少间隔 10 个 epoch）

3. Learning Rate 初始值对 CNN 训练结果的影响：

注：以下实验在如下参数下进行：

- **Weight decay:0**
- **Momentum:0.9**
- **Batch size:100**
- **Init_std:1**
- **Kernel_size：3**
- **Channel_out: 4**

下图显示的是不同初始值的 Learning Rate 对测试集上准确率的影响。



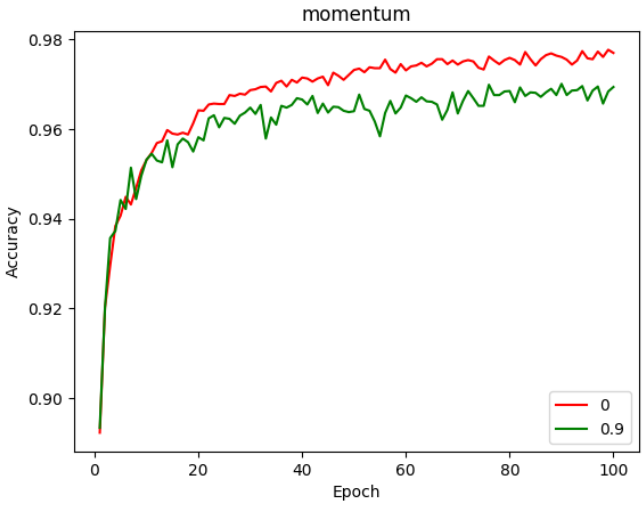
由图可见，当 Learning Rate 初始值较大时，CNN 会产生很大的浮动，即很容易绕过最优点从而跌入其他区域。因此，**CNN 对梯度的敏感性很强，在调整 lr 时需要很小心地、很细致地调整，且这样的调整会在模型上有很大的反映。**

② **Momentum 的选择：**

注：以下实验在如下参数下进行：

- **Weight decay:0**
- **Learning Rate:0.01**

- **Batch size:100**
- **Init_std:1**
- **Kernel_size：3**
- **Channel_out: 4**



由上对比可见，Momentum 较大时，可以加速收敛，使得在前几个 epoch 有较高的准确率；

但由于上面的讨论可知 CNN 对梯度的敏感性很强，当训练后期有 Momentum 的干预时，CNN 很可能无法“转弯”到最优点，而是开始在最优点徘徊，导致准确率较低。

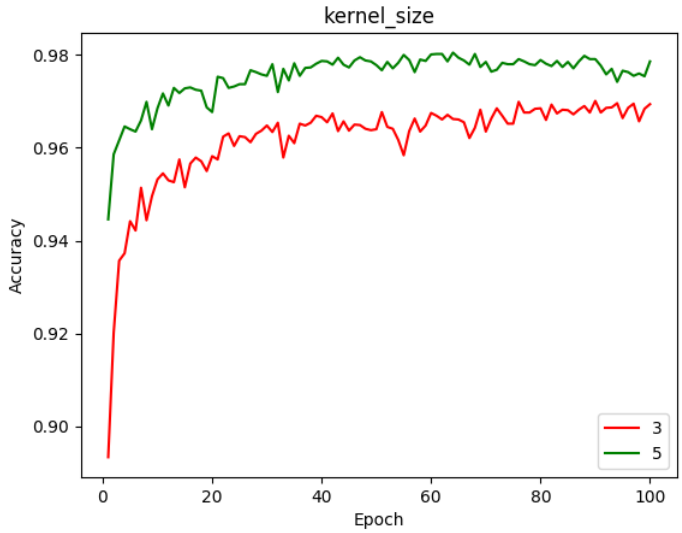
因此，CNN 在训练时，初期添加 Momentum，后期去掉 Momentum，是一种较合理的训练方法。

③ **Kernel_size 的选择：**

注：以下实验在如下参数下进行：

- **Weight decay:0**
- **Momentum:0.9**
- **Batch size:100**
- **Init_std:1**
- **Learning Rate:0.01**
- **Channel_out: 4**

下图显示不同 kernel_size 对训练准确率的影响：

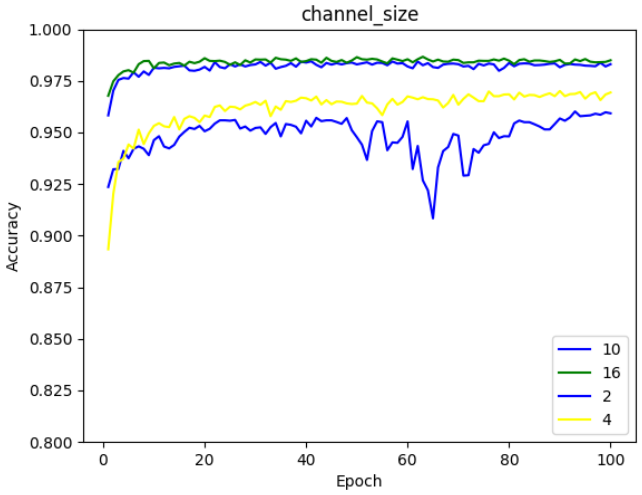


由此可知，当 **Kernel Size** 增大时，**CNN** 局部感知的区域适当增大，从而可以使每个局部感知元学习到更多的局部信息，从而可以大幅度提高训练的准确率。

④ **Channel Size** 的选择：

注：以下实验在如下参数下进行：

- **Weight decay:0**
- **Momentum:0.9**
- **Batch size:100**
- **Init_std:1**
- **Learning Rate:0.01**
- **Kernel_size: 3**



由上图可见，当 **channel_size** 增大时，训练的准确率会大幅度提升，并且训练的稳定性也会变高。此外，收敛的速度也明显增快。但美中不足的是，由于网络规模增大，训练

时间需要成倍地增长。

Part 6：神经网络隐含层输出的可视化

如图，下图为最优结果训练 100 个 epoch 后，第一层卷积层在经过 Relu 后的输出结果可视化后的图形。可见，网络第一层基本学习到了数字的形状、局部特征（局部亮度变化的地方）。