

研究生课程考试成绩单

(试卷封面)

院 系	计算机科学与工程学院	专业	计算机科学与技术			
学生姓名	卓旭	学号	212138			
课程名称	数字图像处理进展					
授课时间	2021 年 12 月至 2021 年 12 月	周学时	3	学分	1	
简要评语						
考核论题	非局部均值滤波去噪算法的实现及对比实验					
总评成绩 (含平时成绩)						
备注						

任课教师签名: _____

日期:

- 注: 1. 以论文或大作业为考核方式的课程必须填此表, 综合考试可不填。“简要评语”栏缺填无效。
2. 任课教师填写后与试卷一起送院系研究生秘书处。
3. 学位课总评成绩以百分制计分。

(本页空白以备双面打印)

非局部均值滤波去噪算法的实现及对比实验

——《数字图像处理进展》课程报告

212138 卓旭

摘要

非局部均值 (Non-Local Means, NLM) 算法[1]是一种简单有效的图像去噪算法。其通过待去噪像素周围邻域与图像其他像素邻域的相似度度量加权, 综合得出去噪结果。本报告主要研究了 NLM 算法, 主要工作内容和目的有:

- 再次讨论了 NLM 算法的原理, 加深了理解;
- 使用 C++ 实现了 CPU 上执行的 NLM 算法, 说明了实际实现中的问题;
- 使用 CUDA 实现了 GPU 上执行的并行加速的 NLM 算法, 借此机会入门了 CUDA 编程;
- 介绍了使用积分图对 NLM 算法进行优化的策略;
- 将 NLM 算法与传统方法: 高斯低通滤波和深度学习方法: DnCNN, 在多种类型的噪声和图像上进行了对比实验。

本报告的所有实验相关代码后续均开源到 <https://github.com/z0gSh1u/nlm-cuda>。

一、图像去噪问题与 NLM 算法的原理

图像的噪声是一种常见的图像降质原因，因此图像去噪问题是图像处理领域一直在研究的基础问题。常用的降质过程描述是：

$$\mathbf{I} = \mathbf{I}_0 + \mathbf{n}$$

其中 \mathbf{I} 是观测到的实际图像， \mathbf{I}_0 是不含噪声的图像， \mathbf{n} 是加性噪声。

常用的高斯低通、均值滤波等去噪方法是基于噪声主要占据图像频谱的高频部分的事实，削减高频，从而降低噪声水平。NLM 算法则基于另一种进路：待去噪像素的一定邻域，应能在图像其他位置找到相似邻域，借助它们的信息可以更好地恢复待去噪像素。

NLM 算法[1]的具体做法的表达式如下：

$$NL[v](i) = \sum_{j \in \mathcal{W}_i} w(i, j) v(j)$$

其中 v 是待去噪图像， $w(i, j)$ 是像素 j 对待去噪像素 i 的贡献权重， \mathcal{W}_i 表示以 i 为中心的一定窗口， $w(i, j) \in [0, 1]$ 且 $\sum_j w(i, j) = 1$ 。原论文提供的权重 $w(i, j)$ 的计算方式如下：

$$w(i, j) = \frac{1}{Z(i)} \exp \left(- \frac{\|v(\mathcal{P}_i) - v(\mathcal{P}_j)\|_2^2}{h^2} \right)$$

$Z(i)$ 是归一化配分：

$$Z(i) = \sum_j \exp \left(- \frac{\|v(\mathcal{P}_i) - v(\mathcal{P}_j)\|_2^2}{h^2} \right)$$

其中 \mathcal{P}_i 以 i 为中心的一定邻域， h 是控制去噪平滑程度的参数， $\|v(\mathcal{P}_i) - v(\mathcal{P}_j)\|_2^2$ 是邻域内像素的欧氏距离和的平方。

NLM 算法涉及到的“窗口”较多，本文的命名与记号为：将较小的“窗口”称为邻域(Patch)，将较大的“窗口”称为窗口(Window)。待去噪像素 i 的邻域 \mathcal{P}_i 称为 *SourcePatch*；比较像素 j 的邻域 \mathcal{P}_j 称为 *ComparePatch*； \mathcal{P}_i 和 \mathcal{P}_j 拥有一样的半径 r_p ，即边长为 $2r_p + 1$ ；搜索窗口 \mathcal{W}_i 称为 *SearchWindow*，半径记为 r_w 。 \mathcal{P}_j 在 \mathcal{W}_i 中滑动。示意图如下：

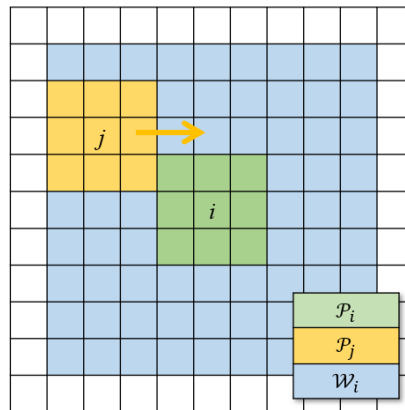


图 1 – NLM 算法示意图

完成上述计算流程后，即可得到由一系列比较像素 j 加权得到的去噪后像素的值。

二、NLM 算法的 C++ CPU 实现

我首先使用 C++ 实现了无特殊优化的 NLM 算法。本部分代码在 `nlm-cpu/nlm-cpu.cpp`。本实验编写的程序处理目标均为单通道、像素 8 bit 深的单通道 RAW 图像，RAW 图像内按小端序、行扫描顺序保存各像素值，无其他数据。图像的宽高、输入输出路径、NLM 算法相关参数需要由配置文件提供（见 `NLMConfig.json`），执行程序时将配置文件路径作为命令行参数。

首先，读入原图像，并将其转换为 float 型数据格式以便后续运算。常见做法将各像素值除以 255 归一化到 $[0, 1]$ 后进行运算。但经尝试，本程序中保留在 $[0, 255]$ 区间上能取得更好的处理效果[2]，猜测可能原因是归一化后计算距离、权重时存在浮点误差累积的问题，归一化操作见 `uint8ImageToFloat` 函数。

与均值滤波等操作类似的，由于原图周围一圈的像素无法作为中心放下 *SourcePatch*，为使得它们也得到去噪处理，需要先将原图向外补出一圈（上下左右各 r_p 像素）。本文采用镜像对称法填补，比起补黑边效果要好。相关函数是 `padSourceImageSymmetric`，示意图如下，带绿色边框的是原图，红色边框的是四个翅膀的填补，蓝色边框的是四角填补（依据行方向的翅膀）：

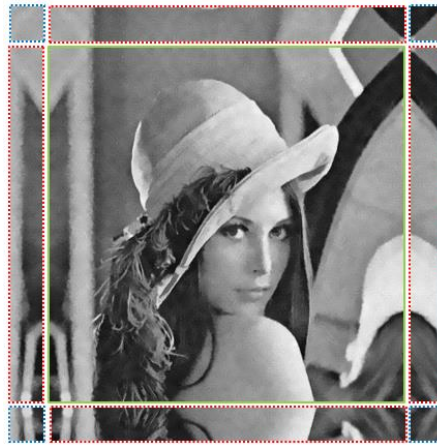


图 2 – 镜像对称法填补图像示意图

接下来，创建与原图等大的目标图像（结果图像），对其进行所谓的“反向映射”：遍历其中各个像素，按照第一节的 NLM 算法具体过程计算该像素的处理后值，最终即得到去噪后的图像。相关函数为 `NLMDenoise`。其中有一些实现细节有必要说明：

① 由图 1 可知某时刻会有 $\mathcal{P}_i = \mathcal{P}_j$ ，此时未归一化权重 $w(i, j)Z(i) = 1$ 。该权重远远大于其他比较像素得到的权重，这将导致原始像素被极大保留，去噪无法进行。本文的处理方式是暂时跳过，最后将 $i = j$ 时的权重设置为其他权重中的最大值。

② 在计算权重项中邻域内像素的欧氏距离和的平方 $\|v(\mathcal{P}_i) - v(\mathcal{P}_j)\|_2^2$ 时，将其除以了邻域尺寸 $(2r_p + 1) \times (2r_p + 1)$ ，以控制指数的数量级对 r_p 的稳定性。

计算完成后，将结果图像重新拉伸到 $[0, 255]$ 灰度级并量化到 8 bit 位深，相关函数为 `floatImageToUint8`。

使用 CPU 上运行的 NLM 算法对注入了零均值、标准差为 $\sigma = 25$ 的高斯噪声的 512×512 的 Lena 灰度图像进行去噪处理，参数为 $2r_p + 1 = 5$ ， $2r_w + 1 = 13$ ， $h = 18.0$ 。结果图如下：

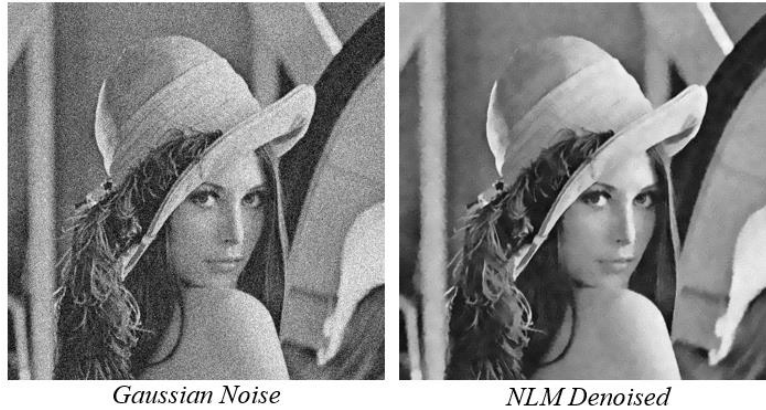


图 3 – NLM 算法的处理结果示意图

在我的计算机（i5 7300HQ）上运行耗时为 112 秒，可见在没有进行多线程并行等优化的情况下，NLM 算法的时间开销较大。

三、NLM 算法的 CUDA GPU 实现

不难发现，NLM 算法中，不同像素之间的去噪过程没有相关性，具备较好的并行执行可能性，而 GPU 的通用计算能力正适合处理此类任务。NVIDIA 公司的 CUDA 技术允许我们使用类似于 C++语言的方式进行 CPU-GPU 混合编程[3]。本部分的相关代码在 nlm-cuda/nlm-cuda.cu。

首先，对 CUDA 的架构和其中的基本概念进行简单说明：

- 宿主机被称为 Host，GPU 设备被称为 Device，二者之间使用 `cudaMemcpy` 函数互传数据；
- 在 Device 上运行的函数需使用 `__global__` 标记，被称为 Kernel 函数；
- CUDA 软件架构从小到大由 Thread、Block 和 Grid 组成，它们之间的组成关系如下图所示；
- Kernel 函数会被映射到 Thread 上执行，可同时运行的最大 Thread 数量由 GPU 的计算能力（CC）决定。暂时无法调度上 GPU 运行的 Block 则串行排队。

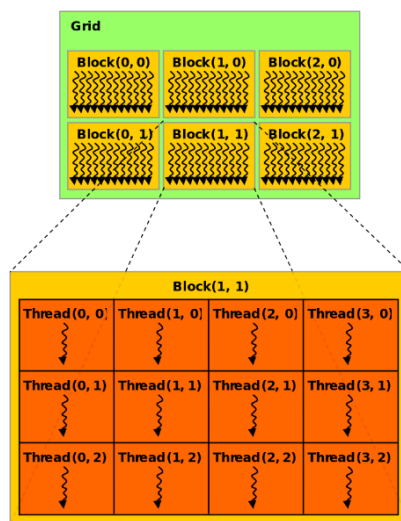


图 4 – CUDA 软件架构示意图 [https://en.wikipedia.org/wiki/Thread_block_(CUDA_programming)]

在图像处理领域，我们通常希望一个 Thread 对应一个像素的处理。因此，可按如下方式组织 Thread 与 Block:

```
dim3 BlockThread(16, 16); // 每个Block安排16×16个Thread
// 总Block数如下计算，以确保每个像素都分配到Thread，padH和padW为宽高
dim3 GridBlock((padH + BlockThread.x - 1) / BlockThread.x,
               (padW + BlockThread.y - 1) / BlockThread.y);
// Kernel函数调用时需传入并发配置信息
NLMDenoise<<<GridBlock, BlockThread>>>(/* 其他参数 */);
```

如此，每个 Thread 运行 Kernel 函数时即可按如下方式获知自己处理的像素坐标:

```
// 通过本Thread的Block号和Thread号即可求得映射到的像素坐标
int i = (blockIdx.x * blockDim.x) + threadIdx.x;
int j = (blockIdx.y * blockDim.y) + threadIdx.y;
// 由于求Block数时向上取整，可能出现多余映射，要排除
if (i >= srcH || j >= srcW) return;
```

对于更细节的 CUDA 编程相关知识超出了本报告的讨论范围，此处略去不述。

除了 NLMDenoise 被改写为 CUDA Kernel 函数外，实现的其他环节与 CPU 版一致。执行前，图像被拷贝到显存中；执行完成后，拷贝回了内存。同样对第二节中的 Lena 图按相同配置进行 NLM 算法处理，显卡为 GTX 1050 移动版，获得了相同的处理结果，但执行时间降低到 140 毫秒，仅为 CPU 执行的 0.125%。

需要说明的是，当我们使用了一些第三方库时，Debug 与 Release 编译模式下得到的程序执行效率会有很大差异。但二、三节我实现的 NLM 算法没有依赖任何第三方库，均为原生代码编写，且计时部分仅为 NLMDenoise 函数，故尽管编译模式为 Debug，比较仍然是正确和公平的。

四、NLM 算法的积分图优化法

本节，将介绍一种使用积分图像（Integral Image，前缀和图像）对 NLM 算法进行

原理层次的加速的优化策略[4]。该方法没有在前面的程序中实现，故仅做简要介绍。

考虑二、三节实现的 NLM 算法的时间开销，若待处理图像尺寸为 $H \times W$ ，则时间复杂度为：

$$O(HW)O((2r_w + 1)^2)O((2r_p + 1)^2) = O(HWr_w^2 r_p^2)$$

（对每个像素点在每个 *SearchWindow* 内求每组 Patch 之间的权重）。

积分图像的基本思想是，对每个像素点 (x, y) ，提前求出 $(0, 0), (x, 0), (x, y), (0, y)$ 组成的矩形的像素和 $S(x, y)$ 。则对于下示意图，可方便地求得任一矩形区域 $ABCD$ 的像素和 $sum(ABCD) = S(D) - S(B) - S(C) + S(A)$ ：

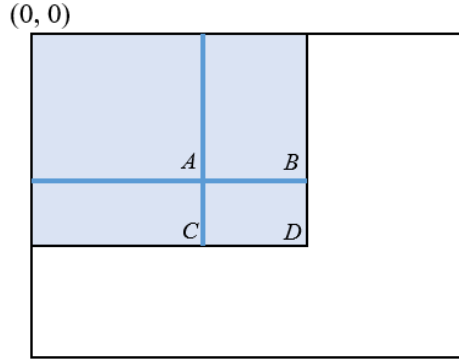


图 5 – 积分图像示意图

倘若我们按照下式提前构造一组关于“距离”的积分图像：

$$S_t(x) = \sum_{0 \leq z \leq x} \|v(z) - v(z + t)\|_2^2$$

其中像素 x 为积分图变量；像素 z 为求和变量； $0 \leq z \leq x$ 表示 z 落在原点与 x 确定的矩形区域内（ \leq 表示偏序小于）； t 为平移向量，取遍 $[-r_p, +r_p] \times [-r_p, +r_p]$ 张成的二维空间的整数点。

那么有：

$$\begin{aligned} & \|v(\mathcal{P}_i) - v(\mathcal{P}_j)\|_2^2 \\ &= S_t(x_1 + r_p, x_2 + r_p) - S_t(x_1 - r_p, x_2 + r_p) - S_t(x_1 + r_p, x_2 - r_p) \\ &+ S_t(x_1 - r_p, x_2 - r_p) \end{aligned}$$

其中 $j = i + t$, $i = (x_1, x_2)$ 。

这是一个常量时间的操作，而积分图的构建可以在 $O(HW)$ 时间递推得到。综上，NLM 的时间复杂度优化为：

$$O(HW) + O(HW)O((2r_w + 1)^2)O(1) = O(HWr_w^2)$$

可以看到，效率的提升是很显著的。但是积分图的递推构建步骤不适合并行完成，故比起全程并行的 CUDA 实现，实际能快多少还有待实验。

五、NLM 算法与其他去噪方法的对比实验

本节，我将实现的 NLM 算法与另外两种去噪算法：传统的高斯低通滤波器和深度

学习的 DnCNN [5]，在三种性质的图像：Lena 灰度图像、医学 CT 图像、彩色大尺寸自然图像上，注入了三种类型的噪声：高斯噪声、泊松噪声、椒盐噪声，对比了三种算法各自的去噪性能表现。进一步地，分析了 NLM 算法的参数调节的纲领性准则，以及几种去噪算法各自的优缺点。

首先，对本节用于对比 NLM 的两种去噪算法进行简单介绍。

高斯低通滤波器

基于噪声主要占据图像频谱的高频部分的事实，削减高频可以降低噪声水平。高斯低通滤波器的传递函数是高斯函数。零均值的二维高斯函数表达式为：

$$h(x,y)=\frac{1}{2\pi\sigma^2}\exp(-\frac{x^2+y^2}{2\sigma^2})$$

其中 σ 是高斯分布的标准差。这是一个连续函数，在进行卷积时需要将其离散化成一定尺寸的卷积核，如下图所示：

0.0947	0.1183	0.0947
0.1183	<u>0.1478</u>	0.1183
0.0947	0.1183	0.0947

图 6 – 一个 3×3 的 $\sigma=1.5$ 时的归一化的高斯核

随后使用该核对图像进行卷积操作即可。

本报告中的高斯低通滤波使用 ImageJ 软件自带的功能实现，其只需 σ 参数，核的半径是按照下式自动计算的：

$$r=\min(\lceil\sigma\sqrt{-2\log a}\rceil+1,50)$$

其中 a 为精度控制参数，默认取 0.01。按该式计算的卷积核尺寸能保留绝大部分的高斯函数能量。

DnCNN

DnCNN 是我们课上讲过的一种经典的基于深度学习的图像去噪模型。其网络结构如下图所示：

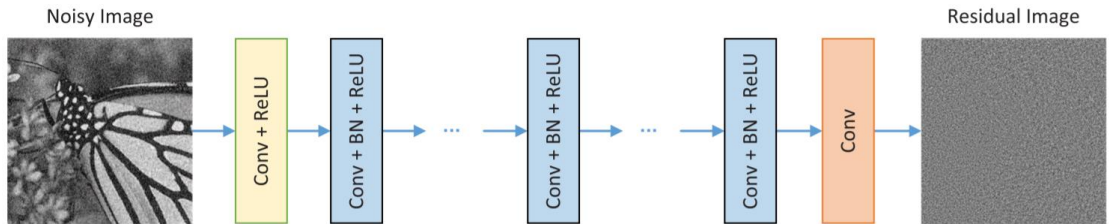


图 7 – DnCNN 的网络结构

相比于直接学习出去噪后的图像，DnCNN 选择学习其中的噪声，然后将噪声从原图中减去以实现去噪。这是因为待去噪图像的景物千变万化，而噪声模式相对单一，更

好学习。本报告中的 DnCNN 相关实验使用官方的 MATLAB 代码和预训练模型完成。

接下来，介绍本节的三幅实验对象图像。这些图像均可在 `experiment/GroundTruth` 和 `experiment/Noisy` 目录下找到。

Lena 灰度图像

图像尺寸为 512×512 。选择该图像的动机是，这是一幅常用的数字图像处理基准图像，其中的发丝等细节也能较好地考察去噪算法的性能。



图 8 – Lena 灰度图像

医学 CT 图像 [6]

图像尺寸为 512×512 ，像素 8 bit 位深，可视为加窗后图像。选择该图像的动机是，比起自然图像，医学图像对细节的保留要求更高，且性质与自然图像很不同，如不存在景深等。



图 9 – 医学 CT 图像

彩色大尺寸自然图像 [7]

图像尺寸为 2040×1356 ，为三通道彩色图像。选择该图像的动机是，考察各算法在彩色图像去噪上的效果，以及大尺寸图像上执行的性能。对于该图，NLM 算法分别处理 RGB 三个通道，并将结果重新融合为彩色图像。



图 10 – 彩色大尺寸自然图像

接下来，介绍本节注入的三种噪声。

高斯噪声

高斯噪声是符合高斯函数的噪声。本实验中使用 ImageJ 软件[8]注入，均值为 0，标准差为 25。特定高斯噪声的产生可使用如下公式进行：

$$y = \sigma x + \mu$$

其中 $x \sim N(0,1)$ （服从标准正态分布）， $y \sim N(\mu, \sigma^2)$ ，而标准正态分布可从均匀分布中借助 Box-Muller 算法产生。

泊松噪声

泊松噪声是符合泊松分布的噪声。泊松分布的概率分布为：

$$P(X = k) = \frac{\lambda^k}{k!} e^{-\lambda} \quad k = 0, 1, \dots$$

其中 λ 被称为平均发生次数。严格来说，泊松噪声与像素值有关，不是加性噪声，不符合第一节提出的降质模型。

本实验中使用 ImageJ 软件的 poisson-noise 插件[9]注入泊松噪声。其内部构造的随机过程用代码表示如下：

```
int poissonValue(float pixVal) { // pixVal 为像素值
    double L = exp(-pixVal);
    int k = 0; double p = 1;
    do {
        k++; p *= rand(); // rand 函数返回[0, 1]之间的随机数
    } while (p >= L);
    return (k - 1);
}
```

椒盐噪声

椒盐噪声表现为白点（盐噪声，最高灰度）或黑点（胡椒噪声，最低灰度）。本实验中使用 ImageJ 软件注入，密度约为 16×16 的区域内有 13 个噪点。

下面是各去噪算法执行时的具体参数说明：

方法	图像	参数
NLM	Lena 灰度图像	$r_{\mathcal{P}} = 2, r_{\mathcal{W}} = 6, h = 18.0,$ 但对泊松噪声 $h = 8.0$
	医学 CT 图像	
	大尺寸彩色自然图像	$r_{\mathcal{P}} = 2, r_{\mathcal{W}} = 8, h = 15.0,$ 但对泊松噪声 $h = 5.0$
高斯低通滤波	Lena 灰度图像	$\sigma = 1.3$
	医学 CT 图像	$\sigma = 1.5$
	大尺寸彩色自然图像	$\sigma = 1.0$

（下页续）

(续上页)

方法	图像	参数
DnCNN	Lena 灰度图像	盲去噪, 无参数
	医学 CT 图像	
	大尺寸彩色自然图像	

这些参数的选择是根据实验测试得到的效果相对较好的参数。对于 NLM 算法, 参数选择的纲领性准则如下:

- ① 大部分图像中相似模式的 Patch 尺寸都不大, r_p 的取值不应过大;
- ② r_w 是影响 NLM 算法性能的关键参数, 在原论文中 w 选取为整张图像, 实际操作中只搜索当前像素点周围一定区域, 对于较大尺寸的图像 r_w 应适当扩大;
- ③ h 越大, 图像越平滑, 在去噪效果可接受的范围内 h 应尽可能小, 以更多地保留图像的原始细节。

NLM 算法需要调节的参数较多, 合适的参数选择比较繁琐。有一些文献专门研究了 NLM 算法的参数调节[10]。而本报告中的高斯低通滤波器只需要一个参数, 简单了许多。深度学习方法 DnCNN 甚至不需要外部参数, 因为其模型针对的是盲去噪任务, 即不知噪声具体性质和水平情况下的去噪。

使用峰值信噪比 (PSNR) 和结构相似度 (SSIM) 作为定量评价指标。

峰值信噪比 (PSNR)

PSNR 是一种普遍且广泛使用的图像相似程度评价指标。其计算公式如下:

$$\text{PSNR}(\mathbf{I}_1, \mathbf{I}_2) = 10 \log_{10} \frac{(2^n - 1)^2}{\text{MSE}(\mathbf{I}_1, \mathbf{I}_2)}$$

单位为分贝 (dB)。 n 为图像的灰度级比特数; \mathbf{I}_1 和 \mathbf{I}_2 表示用来比较的两张图像; MSE 表示两张图像的均方误差。两张图像越相似, 则 PSNR 值越高。

结构相似性 (SSIM)

SSIM 也是一种用于衡量两张图像相似程度的数值指标, 它主要考虑了两张图像之间的三种比较量: 亮度、对比度、结构。其计算公式如下:

$$\text{SSIM}(\mathbf{I}_1, \mathbf{I}_2) = \frac{(2\mu_{\mathbf{I}_1\mathbf{I}_2} + c_1)(2\sigma_{\mathbf{I}_1\mathbf{I}_2} + c_2)}{(\mu_{\mathbf{I}_1}^2 + \mu_{\mathbf{I}_2}^2 + c_1)(\sigma_{\mathbf{I}_1}^2 + \sigma_{\mathbf{I}_2}^2 + c_2)}$$

其中 μ 表示均值; σ^2 表示方差; $\sigma_{\mathbf{I}_1\mathbf{I}_2}$ 表示两张图像的协方差; $c_1 = (0.01L)^2$ 和 $c_2 = (0.03L)^2$ 是为了控制数值稳定性引入的小量, $L = 2^n - 1$, n 为图像的灰度级比特数。两张图像越相似, 则 SSIM 值越高。

下面给出实验的定量结果, 最优结果加粗展示:

(下页续)

图像：Lena 灰度图像（PSNR/SSIM）

	NLM	高斯低通滤波	DnCNN
高斯噪声	26.32/0.74	29.02/0.76	32.25/0.87
泊松噪声	23.42/0.84	30.68/0.85	35.77/0.92
椒盐噪声	23.31/0.55	27.95/0.72	20.81/0.42

图像：医学 CT 图像（PSNR/SSIM）

	NLM	高斯低通滤波	DnCNN
高斯噪声	26.47/0.56	23.90/0.47	24.98/0.50
泊松噪声	29.82/0.75	27.15/0.66	30.58/0.79
椒盐噪声	19.56/0.39	24.36/0.48	18.86/0.40

图像：彩色大尺寸自然图像（PSNR）

	NLM	高斯低通滤波	DnCNN
高斯噪声	27.79	26.45	28.81
泊松噪声	31.81	28.82	32.80
椒盐噪声	19.61	26.70	28.54

从中可以得到如下几点结论：

① 由于 DnCNN 的训练数据为注入高斯噪声的自然图像，因此其对自然图像的处理结果很好，例如人像图 Lena 和彩色照片。但对训练时未见过的医学 CT 图像处理效果，就没有和传统方法有很大差异了。这体现出深度学习相关方法对数据集的强依赖性，以及对陌生数据的不可靠性；

② 对于灰度图中的椒盐噪声，各算法的处理效果都不够令人满意。当确定噪声类型是椒盐噪声时，中值滤波始终是最优秀的去噪方法。而彩色图像中，得益于深度学习模型的泛化能力，DnCNN 对椒盐噪声的处理效果较好；














③ 撇开椒盐噪声不谈，NLM 算法在结构模式相对单一、能较好找到类似结构的医学 CT 图像中的处理效果是比较优秀的，但在情况相对复杂的图像中凭借粗暴的 L2 范数作为相似度度量不够可靠，且效果尤其依赖于参数调节；

④ 虽然理论上泊松噪声的结构最为复杂，但各类去噪算法都取得了不错的结果，基础地去除泊松噪声似乎不是一个很困难的议题。

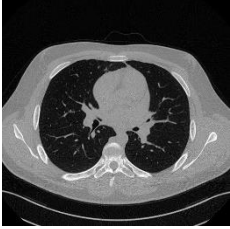
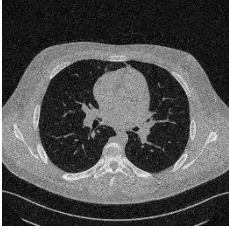
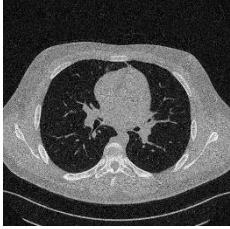
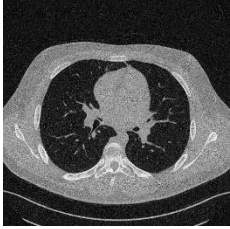
另外，虽然 DnCNN 的处理效果整体较好，但其运算时间较高。以彩色大尺寸自然图像为例，耗时 109 秒才得出最终结果。而 CUDA 实现的 NLM 算法只使用了 1980 毫秒 $\times 3 \approx 5.9$ 秒，如果单纯使用简单的高斯低通滤波，耗时更是仅有百毫秒级。几者的运算效率之间的巨大差异启示我们需要进行时间性能和效果的权衡，根据实际应用场景选择合适的处理方法。

下面给出实验的定性结果：

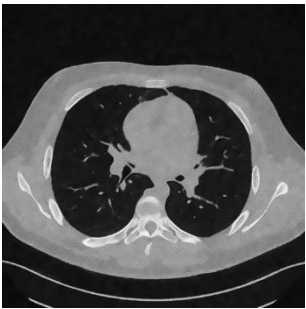
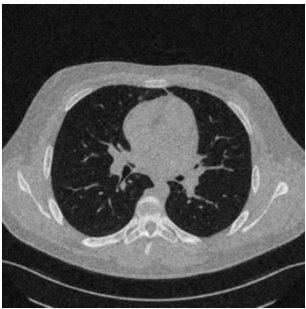
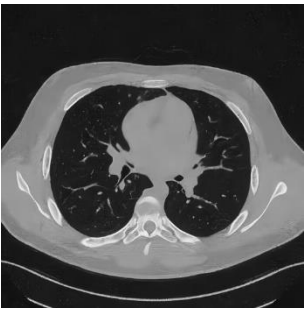
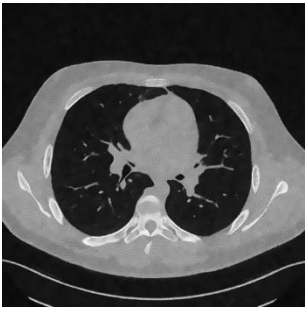
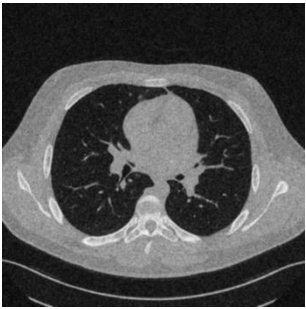
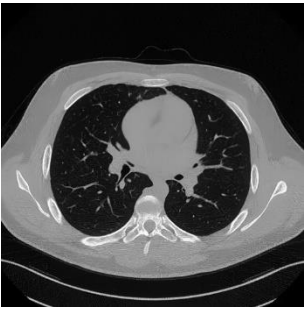
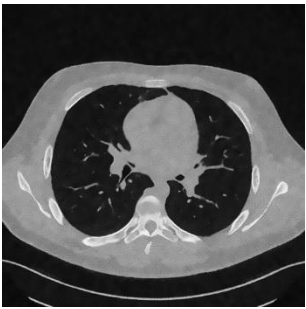
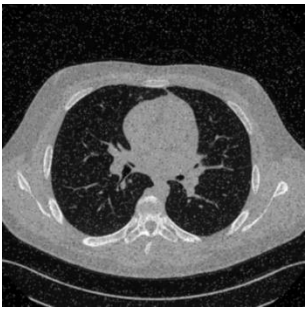
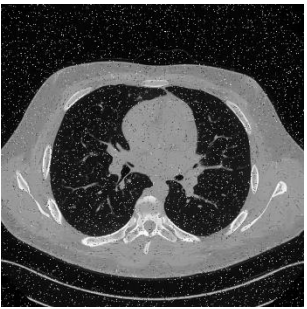
图像：Lena 灰度图像

原图		高斯噪声图	泊松噪声图	椒盐噪声图
				
	NLM	高斯低通滤波		DnCNN
高斯噪声				
泊松噪声				
椒盐噪声				

图像：医学 CT 图像


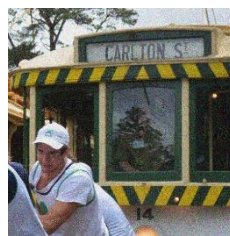
原图	高斯噪声图	泊松噪声图	椒盐噪声图
			

(下页续)







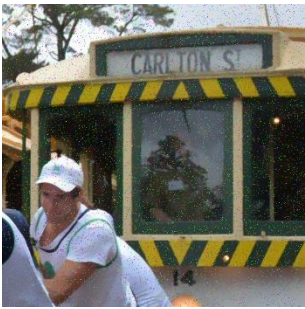


	NLM	高斯低通滤波	DnCNN
高斯 噪声			
泊松 噪声			
椒盐 噪声			

由于彩色大尺寸自然图像尺寸较大，因此统一选取了 512×512 的一块区域展示，以更好地展现结果。

图像：彩色大尺寸自然图像

原图	高斯噪声图	泊松噪声图	椒盐噪声图
			

(下页续)

	NLM	高斯低通滤波	DnCNN
高斯 噪声			
泊松 噪声			
椒盐 噪声			

上述实验的结果图像文件大小较大，没有打包在提交内容中。如有需要，可在东大网盘上下载后解压到 `experiment` 文件夹下：

<https://pan.seu.edu.cn:443/link/8A2873BB068F54B3073F5CF0C5CFFF3C>

有效期限：2027-01-08

访问密码：GDUF

从中可以得到如下结论：

- ① Lena 图像中，DnCNN 的处理结果在视觉效果上确实显著优于其他算法，且和原图相当相似；
- ② 医学 CT 图像中，NLM 算法对肺内一些细节的保留程度不够，特别是心脏的表面结构上损失了较多细节，但由于细节部分占比不多，数值指标仍然较高。这启示我们 PSNR 和 SSIM 两种常用的数值指标不一定能可靠地反映视觉效果，需要结合具体图像查看，或是探索其他指标；
- ③ 在自然图像的处理中，DnCNN 的处理结果在视觉效果上确实显著由于其他算法。

六、总结与体会

本课程报告中，我重新研读并阐述了非局部均值（NLM）去噪算法的原理与流程，并使用 C++ 和 CUDA 实现了 CPU 与 GPU 两种版本的 NLM 算法程序。在此过程中，我的编程能力得到了锻炼，并且进一步认识到“纸上得来终觉浅，绝知此事要躬行”的道理。论文中的算法很多时候只是纲领性的思路说明，在实际实现中有很多细节需要具体考虑。另外，体会到了并行计算对程序加速的巨大作用。

课程报告还介绍了一种使用差值积分图的原理层面的 NLM 加速方法。这启示我们除了要有复现他人的算法的强大的工程动手能力以外，还要有能够静下心来思考，发现现有算法中的问题，并融会贯通其他已有方法的思想，以达到对算法的优化。相比起纯粹堆叠大量的硬件，有时候一点儿原理上的改动能起到更强的优化效果。

课程报告最后使用多种方法在多种噪声下的多种图像上进行了去噪效果的对比实验，进一步学习了各种噪声的产生和具体性质，体会到了深度学习算法的强大和不足，以及传统算法的重要性。在工程实践中，天平两端的矛盾与妥协、权衡与协调是时刻在发生的，要学会根据具体需求选用合适的算法。

参考资料

- [1] Antoni Buades, Bartomeu Coll, Jean-Michel Morel. A non-local algorithm for image denoising. CVPR'05[C].
- [2] lu01873626242. 非局部均值去噪（NL-means）[R]. <https://www.cnblogs.com/luo-peng/p/4785922.html>.
- [3] Bhaumik Vaidya. Hands-On GPU-Accelerated Computer Vision with OpenCV and CUDA. Packt. 2018.
- [4] Jacques Froment. Parameter-Free Fast Pixelwise Non-Local Means Denoising[R]. 2014. Image Processing On Line.
- [5] Kai Zhang, Wangmeng Zuo, Yunjin Chen, et al. Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising[J]. IEEE Transactions on Image Processing. 2017.
- [6] AAPM. Low Dose CT Grand Challenge[R]. <https://www.aapm.org/grandchallenge/lowdosect/>.
- [7] Radu Timofte, Eirikur Agustsson, Shuhang Gu, et al. DIVERse 2K resolution high quality images as used for the challenges[R]. <https://data.vision.ee.ethz.ch/cvl/DIV2K/>.
- [8] ImageJ. ImageJ[R]. <https://imagej.net/>.
- [9] Ignazio Gallo. Add Poisson Noise[R]. <https://imagej.nih.gov/ij/plugins/poisson-noise.html>.
- [10] Vincent Duval, Jean-François Aujol, Yann Gousseau. On the parameter choice for the Non-Local Means[R]. 2010. France HAL.