FAST AND SOMEWHAT ACCURATE ALGORITHMS

MARIO BARELA, SU CHEN, EMILY GUNAWAN, MORGAN SCHREFFLER, MINHO SONG, DOYEOB YEO, AND MENTOR: CHAI WAH WU (IBM), TEAM 4

1. Introduction

In applications such as image processing, computer vision or image compression, often times accuracy and precision are less important than processing speed as the input data is noisy and the decision making process is robust against minor perturbations. For instance, the human visual system (HVS) makes pattern recognition decisions even though the data is blurry, noisy or incomplete and lossy image compression is based on the premise that we cannot distinguish minor differences in images. In this project we study the tradeoff between accuracy and system complexity as measured by processing speed and hardware complexity.

emily: This paragraph above is exactly what Chai wrote in the IMA page, so we probably need to paraphrase it.

We seek to produce algorithms using look-up-tables (LUT) for speeding up filter operations (such as edge-detection, sharpening, and blurring filters). In order for any LUT-based algorithm to be useful, the size of the LUTs need to be reasonably small. To meet this aim, we use the following strategies.

(1) a bit-truncation technique,
(2) convolution decompositon of a kernel 3×3 matrices (rank one),
(3) approximate convolution decompositon of a kernel 3×3 matrices (rank two),
(4) singular value decomposition,
(5) and taking advantage of the observation that edge-detection algorithm shows higher tolerance to bit-truncation.

2. Background

An h-by-w-pixel image file can be thought of as a three-dimensional array

$$\text{IMAGE} = [p_{ijk}]_{h\times w\times 3}$$

whose entries are integers between 0 and 255. Given pixel (i, j), the three entries p_{ijk}, $k = 1, 2, 3$, determine the color of the pixel in some colorspace. When the image is grayscale, let $\text{IMAGE} = [p_{ij}]_{h\times w}$, and let the entries indicate the pixel's intensity (0 for black, 255 for white). For the sake of simplicity, we will only consider grayscale images in this paper.

2.1. **Filtering images using kernel convolutions.** When filtering an image, first choose a matrix $K = [k_{ij}]_{n\times n}$ with double-precision entries, where n is an odd number greater than 1. The matrix K is called the *filter kernel* or *convolution kernel*. In this paper we usually only consider 3×3 and 5×5 kernels.

Date: August 22, 2015.

Next, we compute the *matrix convolution* $F = K * \text{IMAGE}$, a $\max\{h, n\} \times \max\{w, n\}$ matrix whose ijth entry is

$$f_{ij} = \begin{cases} \displaystyle\sum_{q=(1-n)/2}^{(n-1)/2} \sum_{r=(1-n)/2}^{(n-1)/2} k_{i+q,j+r} p_{i-q,j-r} & \text{if } 1 \leq i+q \leq h,\ 1 \leq j+r \leq w \\ 0 & \text{otherwise.} \end{cases}$$

To eliminate the piecewise nature of determining f_{ij}, one can first "pad" IMAGE with a frame of zeros that is $\frac{n-1}{2}$ entries thick. Observe that this definition of matrix convolution coincides with the usual discrete convolution, but with two-dimensional indexing instead of one-dimensional. In particular, it is associative and distributive.

2.2. **Truncation methods.**

2.3. **Measures of errors.** We measure the accuracy of our algorithms using the following four measures.

Definition 2.1. Let $A = [a_{ij}]_{h \times w}$ and $B = [b_{ij}]_{h \times w}$ be two images. Define

$$\begin{aligned} \ell^2\text{-difference}(A, B) &:= \sqrt{\frac{\sum_{i,j} (a_{ij} - b_{ij})^2}{h \times w}}, \text{ and} \\ \ell^\infty\text{-difference}(A, B) &:= \max_{i,j} |a_{ij} - b_{ij}|. \end{aligned}$$

Per [HG08], PSNR (in dB) is defined as

$$\begin{aligned} \text{PSNR}(A, B) &= 10 \cdot \log_{10} \left(\tfrac{\text{MAX}_B^2}{\text{MSE}} \right) \\ &= 20 \cdot \log_{10} \left(\tfrac{\text{MAX}_B}{\sqrt{\text{MSE}}} \right) \end{aligned}$$

where

$$\text{MSE} = (\ell^2\text{-difference}(A, B))^2 \qquad \text{and} \qquad \text{MAX}_B = 255.$$

The *Image Euclidean Distance (IMED)* is given by

$$d_{\text{IMED}}^2(A, B) = \frac{1}{2\pi} \sum_{i,k=1}^{h} \sum_{j,\ell=1}^{w} \exp\left(\frac{-(k^2 + \ell^2)}{2} \right) \big(a_{ij} - b_{ij}\big)\big(a_{i+k,j+\ell} - b_{i+k,j+\ell}\big).$$

Remark 2.2. In general, MAX_B is the maximum *possible* pixel value of an image. Since we are assuming each pixel is an 8-bit integer, MAX_B will be 255 for our purposes.

Remark 2.3. PSNR can be easily calculated by using MATLAB command psnr(A, B). PSNR is the abbreviation for Peak Signal-to-Noise Ratio. In brief, it shows us how different two images are. As PSNR is higher, it becomes harder to recognize the difference between original image and filtered image. Typical values for the PSNR in lossy images are between 30 and 50 dB, provided the bit depth is 8 bits.

Remark 2.4. See [FWZ05] for an analysis of why IMED might be a reasonable image distance. Another reasonable image distance may be $\frac{d_{\text{IMED}}}{\sqrt{h \cdot w}}$.

3. Results

3.1. **Look-up tables (LUTs).** As in [WSLQE06], the usage of look up tables provides a way to reduce the implementation time caused by direct algebraic computations. By storing the values of convolution of all possible square matrices with the kermel matrix K, the look up table serves as a dictionary whose values can be retrieved readily.

$$\begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix} * \begin{bmatrix} k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 \\ k_7 & k_8 & k_9 \end{bmatrix} \longrightarrow \begin{bmatrix} y_1 & y_2 & y_3 \\ y_4 & y_5 & y_6 \\ y_7 & y_8 & y_9 \end{bmatrix}$$

The potential challenge for LUTs lies in their size which can be so large that storing them in the computer memory becomes impractical. As an example, a 3×3 kernel K with all entries non-zero will require $S = (2^8)^9 = 2^{72}$ bytes storage, which is far beyond the storing capability of modern computers.

3.2. **Bit Truncation.** One remedy to the problem of LUT size is to use truncated versions of the LUTs. Namely, we drop a prescribed number of bits from the entries in the square matrix that will be convolved with the kernel. As an illustration, a binary number 11111011 will become 11111000 after we drop the last two digits, i.e. replacing them by 0's. In this case, the original set $\{0, 1, \ldots, 255\}$ is turned into the set $\{0, 4, 8, 12, \ldots, 252\}$ after the truncation by two bits.

Definition 3.1. Given an $n \times n$ filter kernel K, we say that $T = [t_{ij}]_{n \times n}$ is a *truncation matrix* if its entries are all integers from 0 to the image's bit depth. When performing a kernel convolution on an $n \times n$ block IMAGE_B of IMAGE, T indicates that we truncate the last t_{ij} bits of the ijth entry of IMAGE_B when applying K. Given this notation, it is easy to see that if $T_{\text{tot}} := \sum_{i,j=1}^{n} t_{ij}$, the size of the LUT associated with K is reduced by a factor of $2^{T_{\text{tot}}}$ when truncated according to T.

Observe that truncation trades LUT size for algorithmic accuracy. Indeed, given some integer $0 \leq x \leq 255$ with $x \equiv 0 \pmod{2^{t_{ij}}}$, if the ijth entry of IMAGE_B is in the interval $[x, x + 2^{t_{ij}} - 1]$, it will still be entered into the LUT as x. If t_{ij} is close to the bit depth of IMAGE, or if pixel values tend to be closer to the right endpoints of these intervals than to the left, this could potentially result in some fairly large error.

Although the above truncation is able to reduce the size of LUTs substantially, it is still not enough to generate LUTs which are acceptable in practice, especially for LUTs constructed for low pass filters with 5 by 5 or even larger kernels. To deal with this issue, we introduce the decomposition of the kernel so that the LUTs for each decomposition matrix is small enough to be generated and stored in the memory.

3.3. **A rank 1 decomposition.** We first examine the simple case when $rank(K) = 1$ where $K \in \mathbb{R}^{n \times n}$ and n is an odd number. It is well known that every rank 1 matrix can be decomposed as

$$K = b \cdot c^T$$

for some $b, c \in \mathbb{R}^{n \times 1}$. Denoting the matrix convolution by $*$, the following relationship can be verified directly:

$$K = b \cdot c^T = b * c^T$$

So is the equation

$$\text{IMAGE} * K = \text{IMAGE} * (b \cdot c^T) = K * (B * C) = (K * B) * C \tag{1}$$

Here $B = (0, \ldots, b, \ldots, 0)$ and $C = (0, \ldots, c, \ldots, 0)$ where 0 represents $n \times 1$ column vectors and b,c are in the center. We need to point out that the above relationship may not be true for general kernel which is not rank one. This decomposition allows us to build two LUTs for the kernel K, each of which is a table of the column vector b or c since we can ignore all 0's. The size of LUTs are reduced significantly in this way.

As an example, let us consider the following low pass filter kernel

$$K = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

It is apparent that $rank(K) = 1$ and an obvious decomposition of K is given by

$$K = \frac{1}{4}\begin{bmatrix} 0 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 0 \end{bmatrix} * \frac{1}{4}\begin{bmatrix} 0 & 0 & 0 \\ 1 & 2 & 1 \\ 0 & 0 & 0 \end{bmatrix} \triangleq B * C$$

The symmetric decomposition of K ($C = B^T$), simplifies the LUT further: only one LUT is required, since

$$\text{IMAGE} * C = \text{IMAGE}^T * B$$

The normalizing constant $\frac{1}{4}$ rules out the possibility of overflow and the maximum size (no truncation) of the LUT is $(2^8)^3 = 2^{24}$ bytes, or 16MB. Truncation will further reduce the size to magnitude of kilobyte. Numerical test is given in subsection 8.3.

3.4. **Singular value decomposition.** Notice that the relationship (1) does not hold for general matrix K even if K can be decomposed as $K = B \cdot C$. However, it is well known that any matrix can be written as the sum of a series of rank 1 matrices, i.e. we have the following singular value decomposition (SVD):

$$K = \sum_{i=1}^{r} \sigma_i u_i \cdot v_i^T$$

Here u_i,$v_i \in \mathbb{R}^{n\times 1}$ are the left and right singular vectors respectively and σ_m are singular values. Moreover, u_i's and v_i's compose of orthonormal matrices and r is the rank of K. Now we are able to make use of the LUTs for rank 1 decomposition as in the previous section. As a result, at most $2r$ LUTs are required, each of which has a maximum size of $(2^8)^n = 2^{8n}$ bytes. For example, a 3 by 3 kernel with rank 2 needs at most 4 LUTs with maximum size of 16Mb for each. The benefit of SVD is that for most kernels in practice, their rank (usually 1 or 2) is far less than their dimension so that very few LUTs are required even if the kernel has large dimensions. The other benefit we obtain from SVD lies in the fact that all the summands can be dealt with independently so that the power of parallel computation can be exploited.

3.5. **Numerical test.** In this section, we implement the ideas in 8.1 and 8.2 and perform two numerical tests.

Scheme	LUT Size	PSNR (dB)
Direct	N/A	N/A
LUT with T_1	512KB	41.8459
LUT with T_2	16KB	30.4442

TABLE 1. Error and size of LUT for direct computation and LUTs with different truncation schemes.

3.5.1. *A numerical test for a low-pass filter.* We first examine the kernel

$$K_1 = \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix} = \frac{1}{4}\begin{bmatrix}0 & 1 & 0\\ 0 & 2 & 0\\ 0 & 1 & 0\end{bmatrix} * \frac{1}{4}\begin{bmatrix}0 & 0 & 0\\ 1 & 2 & 1\\ 0 & 0 & 0\end{bmatrix} \triangleq B * C$$

As mentioned above, one LUT need to be generated for kernel K_1. Some of the filtered images together with the original one (267×188) are given in figure 2. The truncation scheme T_1 and T_2 are given by

$$T_1 = \begin{bmatrix}0 & 2 & 0\\ 0 & 1 & 0\\ 0 & 2 & 0\end{bmatrix}, \qquad T_2 = \begin{bmatrix}0 & 4 & 0\\ 0 & 2 & 0\\ 0 & 4 & 0\end{bmatrix}$$

where the numbers in matrices T_1 and T_2 are those of the bits to be truncated. It is apparent in figure 1 that the blurred effect with truncated LUTs for darker parts of the image is close to that from from direct application of the filter. There are visible artifacts for lighter parts of the image in the filtered images with truncated LUTs, which is an indication of a loss of precision after truncation. To measure the error introduced by our approximation, we will exploit the peak signal-to-noise ratio (PSNR) introduced in section 2.3. A comparison is summarized in Table 1. The PSNRs for LUTs with T_1 and T_2 indicates that both schemes provides satisfactory approximation of the original filter. It is quite notable that even with truncation T_2, the PSNR is still above 30 with the size of LUT reduced to only 16Kb.

We also run an optimization to examine which truncation scheme will give the best approximation of the original kernel, provided that the size of LUT is fixed. Some of the results are summarized in Table 2. It is not surprising that there is a symmetry in the best truncation scheme thanks to the symmetry of our original kernel K_1 .

3.5.2. *A numerical test for a high-pass filter.* The second numerical test involves the edge detection kernel (high pass kernel)

$$K_2 = \frac{1}{8}\begin{bmatrix}-1 & -1 & -1\\ -1 & 8 & -1\\ -1 & -1 & -1\end{bmatrix}$$

What this kernel does is to expose the edges of the pictures with white color and the non-edges tend to be black. Evidently the rank of K_2 is 2 and we resort to the singular value decomposition approach for reduction of the look up table size. The decomposition is given by

$$K_2 = U\Sigma V^T$$

(A) Original image

(B) Low pass, direct

(C) Low pass, LUT with T_1

(D) Low pass, LUT with T_2

Figure 1. Comparison of filtered images. Figure A: The original image. Figure B: filtered image by direct applying kernel $K1$. Figure C: filtered image by using LUT with truncation scheme T_1. Figure D: filtered image by using LUT with truncation scheme T_2

LUT Size	Best Truncation	PSNR (dB)
16Kb	$(0, b, 0)$ with $b = (4, 2, 4)^T$	30.4442
32Kb	$(0, b, 0)$ with $b = (3, 3, 3)^T$	33.3275
64Kb	$(0, b, 0)$ with $b = (3, 2, 3)^T$	35.5597
256Kb	$(0, b, 0)$ with $b = (2, 2, 2)^T$	39.4329

Table 2. Best truncation scheme for fixed-size LUT

where

$$U = \begin{bmatrix} 0.0971 & 0.7004 & -0.7071 \\ -0.9905 & 0.1374 & 0 \\ 0.0971 & 0.7004 & 0.7071 \end{bmatrix}, \qquad \Sigma = \begin{bmatrix} 1.0245 & 0 & 0 \\ 0 & 0.2745 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

and

$$V = \begin{bmatrix} 0.0971 & -0.7004 & -0.7071 \\ -0.9905 & -0.1374 & 0 \\ 0.0971 & -0.7004 & 0.7071 \end{bmatrix}$$

Four LUTs are needed since we need to deal with $u_1 = (0.0971, -0.9905, 0.0971)$, $u_2 = (0.7004, 0.1374, 0.7004)$, $v_1 = (0.0971, -0.9905, 0.0971)$, $v_2 = (-0.7004, -0.1374, -0.7004)$ separately. The increase in the number of LUTs causes the size of LUT to rise. Another factor that contributes to the increase in the LUT size is the singular vectors which may stretch the original values of each pixel. As in the low pass filter, we still use the two truncation schemes T_1 and T_2 and the image we use here has the size of 1024×768. The filtered images with direct computation and truncated LUTs are shown in figure 2 while the errors in PSNR and the size of the LUTs are listed in Table 3. Note that the size of LUTs are significantly larger than that of the rank 1 kernel K_1. We observe that the size with SVD is approximately 8 times, rather than 4, of the previous LUT for rank 1 case. The extra increase in the size is due to the stretching effect of singular vectors. However, by truncating more numbers of bits, such as the T_2 scheme, we can draw similar conclusions, as in the case of low pass filter, that our truncated LUTs with the SVD scheme provides satisfactory result while the size of the LUTs are kept acceptable in practice.

(A) Original image

(B) Low pass, direct

(C) Low pass, LUT with T_1

(D) Low pass, LUT with T_2

FIGURE 2. Comparison of filtered images. Figure A: The original image. Figure B: filtered image by direct applying kernel $K2$. Figure C: filtered image by using LUT with truncation scheme T_1. Figure D: filtered image by using LUT with truncation scheme T_2

Scheme	Total LUT Size	PSNR (dB)
Direct	N/A	N/A
LUT with T_1	3.8MB	40.9868
LUT with T_2	128.8KB	30.4436

TABLE 3. Error and size of LUT for direct computation and LUTs with different truncation schemes.

3.5.3. *A numerical test for the Sobel magnitude edge-detection filter.* We describe a numerical test using a similar method as in section 3.5.1. The Sobel gradient magnitude is an edge-detection operation [SF68] which can be computed using $\mathbf{G} = \sqrt{{\mathbf{G}_x}^2 + {\mathbf{G}_y}^2}$ where

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \text{IMAGE} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \text{IMAGE}.$$

Recall that $*$ denotes the matrix convolution.

Let

$$K_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

and

$$K_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 1 & 2 & 1 \\ 0 & 0 & 0 \end{bmatrix}.$$

Let T_j denote the truncation matrices $\begin{bmatrix} 0 & j & 0 \\ 0 & j & 0 \\ 0 & j & 0 \end{bmatrix}$ and $\begin{bmatrix} 0 & 0 & 0 \\ j & j & j \\ 0 & 0 & 0 \end{bmatrix}$ where the numbers in matrices T_k are those of the bits to be truncated (see Definition 3.1).

Because of the symmetry in the entries of K_x and K_y, we can use the same look-up table, say, $\text{LUT}(K_x)_T$, for both. Let $\mathbf{G}(T)$ denote the Sobel magnitude operation using the truncated look-up table $\text{LUT}(K_x)_T$.

To measure the error introduced by our approximations, we use the peak signal-to-noise ratio (PSNR) and the two norms ℓ_2 and ℓ_∞ introduced in section 2.3. The comparison summarized in Table 4 is done for a suite of representative images. The PSNRs and norms show that truncation up to 5 digits provide satisfactory approximations to the actual Sobel magnitude operation.

The sizes of the LUTs for different truncation matrices T_j are also listed in Table 4. The filtered images with direct computation and truncated LUTs are shown in figure 3.

3.6. **A Decomposition Approximation.** Recall that a rank 1 decomposition of our kernel K had the form

$$K = b * c^T$$

where $c, b \in \mathbb{R}^n$. This resulted in two LUTs of size 2^{24} bytes instead of the full LUT of size 2^{72} bytes. This drastically reduces the size of the LUT before any truncation but it requires

truncation	LUT sizes	PSNR	ℓ_2	ℓ_∞
T_0	16MB and 64KB	inf	0	0
T_1	2MB and 16KB	52.1,54.9	$4.310^{-7}, 1.410^{-6}$	0.00551,0.0086
T_2	256KB and 4KB	44.7,47.2	$1.210^{-6}, 2.710^{-6}$	0.0144, 0.0210
T_3	32KB and 1KB	38.3,40	$2.610^{-6}, 5.410^{-6}$	0.0310, 0.0423
T_4	4KB and 256 bytes	32.4, 33.5	$6.110^{-6}, 1.110^{-5}$	0.0698, 0.0876
T_5	512 bytes and 64 bytes	26.1, 27.7	$1.310^{-5}, 2.810^{-5}$	0.1443, 0.1714

TABLE 4. Ranges of PSNR and norms for a suite of images for measuring the difference between $\mathbf{G}(T_j)$ and $\mathbf{G}$. See Section 3.5.3

us to have a rank 1 kernel K. What can we do for a filter that is not rank 1?
Question: Can we decompose the filter K in a better way?
Consider the decomposition of our (3×3) filter K into the convolution:

$$K = A * B$$

where A and B are of the form

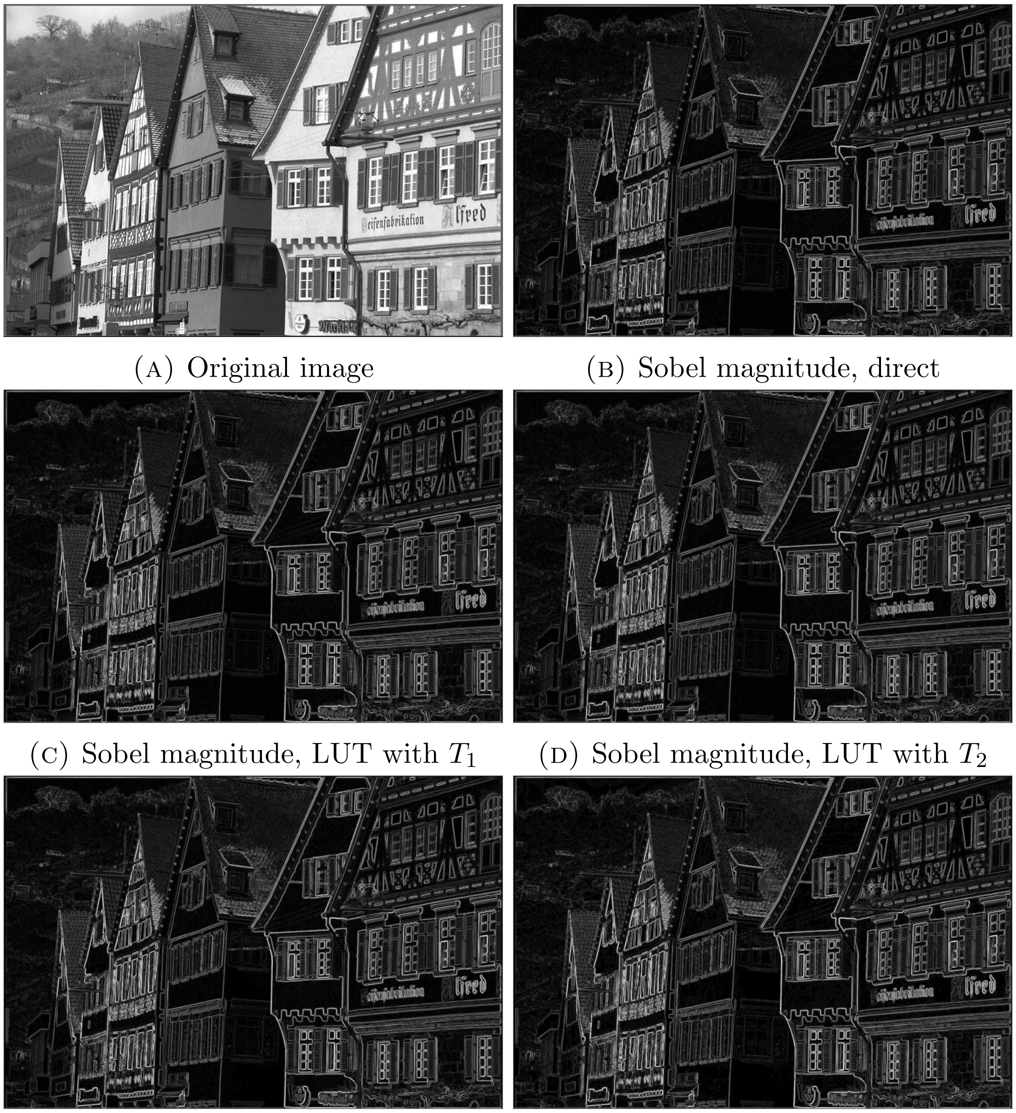
$$A = \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 0 \end{bmatrix}, B = \begin{bmatrix} g & h & i \\ j & k & l \\ 0 & 0 & 0 \end{bmatrix}$$

The main reason we would like a decomposition of this form is that due to the number of 0 entries in A and B we can automatically reduce the size of our full LUT of 2^{72} bytes to two LUTs of size 2^{48} bytes. In general this decomposition is not possible but we may be able to find an approximate decomposition by posing this as a minimization problem. The problem is

$$\min \|K - A * B\|_F$$

Notice that that are other ways we can arrange the variables $a, b, .., l$ in the matrices A and B. There are in fact 9 choose 6 possibilities for each of the two matrices. This gives us a total of $C(9,6)^2 = 84^2 = 7056$ possible minimization problems to solve. This seems like quite a lot of computation but this problem will actually be part of pre-processing. It is important to note the the minimizing values $(a, b, c, ..., k, l)$ may be 0 and so we may further reduce LUT size. For example, the decomposition may lead to A having only 2 nonzero entries and B 4 nonzero entries. This would lead to two LUTs of size 2^{16} and 2^{32} bytes respectively.

We can actually change the structure of the matrices A and B above to have a variable amount of nonzero entries in them. This will allow us to force more entries in the matrices to be zero resulting in smaller LUTs.
Note: After solving all possible minimization problems for a particular decomposition we will have a vector f_{min} and a matrix x_{min} containing all the min function values and minimizers. To get the "best" decomposition we look at the smallest function value. Since this will be an approximate decomposition we should look at all function values under a threshold and find the corresponding minimizers with the most number of zeros. For example there may be two function min values smaller than $\min(f_{min}) + 10^{-2}$ with minimizers $x_1 = (1, 3, 0, 4, 2, 1)$ and $x_2 = (0, 0, 0, 1, 2, 0)$. In this example we should choose the minimizer x_2 because it has



(A) Original image

(B) Sobel magnitude, direct

(C) Sobel magnitude, LUT with T_1

(D) Sobel magnitude, LUT with T_2

(E) Sobel magnitude, LUT with T_3

(F) Sobel magnitude, LUT with T_4

FIGURE 3. Comparison of results of the Sobel magnitude operation with various truncation matrices. See Section 3.5.3.

more zeros in it. It should also be noted that minimizers may have very small values that may be truncated.

3.7. **Decomposing a Kernel Matrix by Rows.** Let K be an $n \times n$ filter kernel. For $k = 1, \ldots, n$, let e_k be the $n \times 1$ matrix whose k1th entry is 1 and whose other entries are 0, and let r_k be the $1 \times n$ matrix whose entries are the kth row of K. Note that K may always be decomposed as $K = \sum_{k=1}^{n} r_k * e_k$. Taking advantage of the associativity of $*$, it follows that $K * \text{IMAGE} = \sum_{k=1}^{n} r_k * (e_k * \text{IMAGE})$. This is just another way of writing that summation is distributive, i.e.,

$$\sum_{q=(1-n)/2}^{(n-1)/2} \sum_{r=(1-n)/2}^{(n-1)/2} k_{i+q,j+r} p_{i-q,j-r} = \sum_{q=(1-n)/2}^{(n-1)/2} \left(\sum_{r=(1-n)/2}^{(n-1)/2} k_{i+q,j+r} p_{i-q,j-r} \right).$$

If using lookup tables to compute a matrix convolution, this observation greatly reduces the amount of memory required. Indeed, rather than having a single LUT which takes n^2 inputs and contains 2^{8n^2} entries, we have n LUTs (one for each row of K), each of which takes n inputs and contains only 2^{8n} entries. Even when $n = 3$ this is a tremendous saving of memory (with 8-bit entries, one 4-zebibyte (2^{72}) LUT versus three 16MB (2^{24}) LUTs).

Further, the multiple-LUT approach can be easily implemented, and at a lower cost than direct computation. Before filtering any pixels, rotate K by 180^o about its center. Then, take an $n \times n$ submatrix IMAGE_B of IMAGE centered about the pixel to which you wish to apply the kernel. Now simply enter the ith row of IMAGE_B into the LUT associated with r_i for $i = 1, 2, \ldots, n$, and add the results. Thus, if we make the naive assumption that lookups and additions are computationally equivalent, only $2n - 1$ operations (n lookups and $n - 1$ additions) are required per pixel. Compare this to $n^2 + 2n - 2$ operations (n^2 multiplications and $2(n - 1)$ additions) per pixel using direct computation.

Finally, when combined with bit truncation, each LUT can be made even smaller at the cost of an extra n^2 truncation operations per pixel. Unfortunately, if truncation is assumed equivalent to lookups and additions, this is not much more efficient than direct computation, since it saves only one operation per pixel. However, if truncation is cheaper than lookups and/or addition (which is suspected to be the case), combining bit truncation with this particular LUT scheme will drastically reduce computation time when compared with direct computation.

As an example, choose an arbitrary 3×3 kernel K and a truncation matrix $T = \begin{bmatrix} 5 & 5 & 5 \\ 5 & 2 & 5 \\ 5 & 5 & 5 \end{bmatrix}$, using the ith rows of K and T to produce the ith LUT. The associated LUTs will have 512, 4096, and 512 entries. Even if each entry uses double precision, only 20kB of memory is required to store the LUTs, which is certainly small enough to fit into cache memory. However, 14 operations (5 to use the LUTs as described previously, and 9 truncations) are required per pixel, versus 13 for direct computation.

3.8. **Using the truncating technique for a blurring filter: Taking advantage of the fact that truncating works better for edge-detecting than for blurring.** We notice that the bit truncation method (see Section 3.2) introduces too much error in low-pass filters but produces acceptable errors for high-pass filters. We would like to take advantage of this fact to reduce running time for running a low-pass filter.

Let I, L, and H denote the identity, a low-pass filter operation, and a high-pass filter operation on an image. Let $L(T)$, and $H(T)$ denote L and H with a truncation using the truncation matrix T. To informally say that applying a truncation technique to a high-pass filter gives close enough result, we can write $H \approx H(T)$.

Since $I = L + H$, we have $L = I - H \approx I - H(T)$.

Example 3.2. Consider a high-pass filter $H = \frac{1}{8}\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$. Let

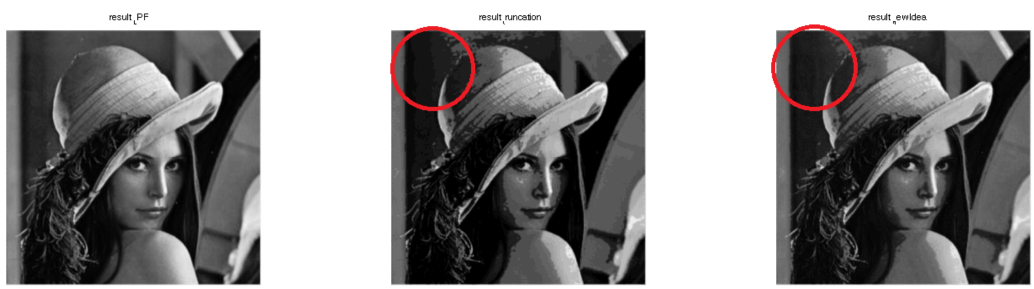
$$T_1 = \begin{bmatrix} 8 & 4 & 8 \\ 4 & 2 & 4 \\ 8 & 4 & 8 \end{bmatrix}, T_2 = \begin{bmatrix} 8 & 5 & 8 \\ 5 & 3 & 5 \\ 8 & 5 & 8 \end{bmatrix} \text{ and } T_3 = \begin{bmatrix} 8 & 6 & 8 \\ 6 & 4 & 6 \\ 8 & 6 & 8 \end{bmatrix}$$

be truncation matrices. The table 5 shows us how $L(T_i)$ and $I - H(T_i)$ are different from the low-pass filter $L = I - H$ for each i. $L(T_i)$ is Old and $I - H(T_i)$ is New in the table 5.

	T_1		T_2		T_3	
	Old	New	Old	New	Old	New
PSNR	38.12	39.58	30.80	32.84	22.59	25.59
l_2 - error	3.06	2.80	7.40	5.87	18.87	13.52
l_∞ - error	8.42	7.40	19.64	15.81	43.35	35.70

Table 5. Comparison of errors

As you see, our new approach gives us the better results no matter which truncation matrix you use. See also Figures 4 made via the truncation matrix T_3.



(A) LPF (B) LPF with Truncation (C) Result of the New Idea

Figure 4. Comparison of filtered images

As you see, you can easily check that our new idea $\tilde{L}$ reduces staircase effects against the usual truncation method.

Likewise, we can apply our method to any other low-pass filter as follows:

1. Let L be the our target low-pass filter.
2. Set $\tilde{H} := I - L(T)$ for a certain truncation T.
3. Find $\tilde{L} := I - \tilde{H}$ as an approximation of L.

References

[WSLQE06] Wu, C. W., Stanich, M., Li, H., Qiao, Y., Ernst, L., "Fast Error Diffusion and Digital Halftoning Algorithms Using Look-up Tables," Proceedings of NIP22: International Conference on Digital Printing Technologies, Denver, Colorado, pp. 240-243, September 2006.

[SF68] Sobel, Irwin and Feldman, Gary, A 3x3 isotropic gradient operator for image processing, 1968.

[HG08] Huynh-Thu, Quan and Ghanbari, Mohammed, Scope of validity of PSNR in image/video quality assessment, Electronics letters, vol. 44, no. 13, pp. 800-801, 2008.

[FWZ05] Jufu Feng, Liwei Wang, Yan Zhang, "On the Euclidean Distance of Images," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, no. 8, pp. 1334-1339, 2005.

Mathematical Modeling in Industry XIX (August 5-14, 2015)