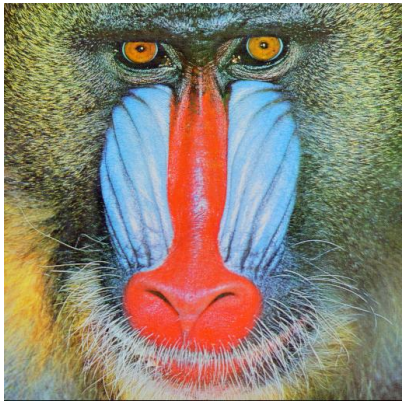
Fast and Somewhat Accurate Algorithms (Team 4)

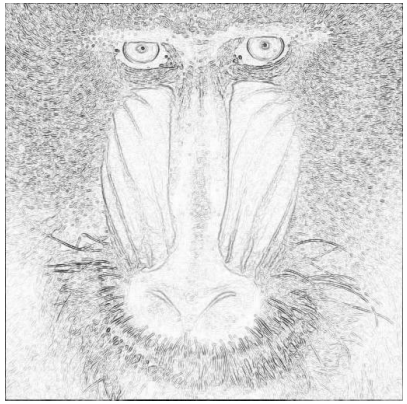
Mario Barela, Su Chen, Emily Gunawan, Morgan Schreffler, Minho Song, Doyeob Yeo, with mentor: Chai Wah Wu (IBM)

IMA: Modeling in Industry Final Presentation

Friday, August 14, 2015

Outline

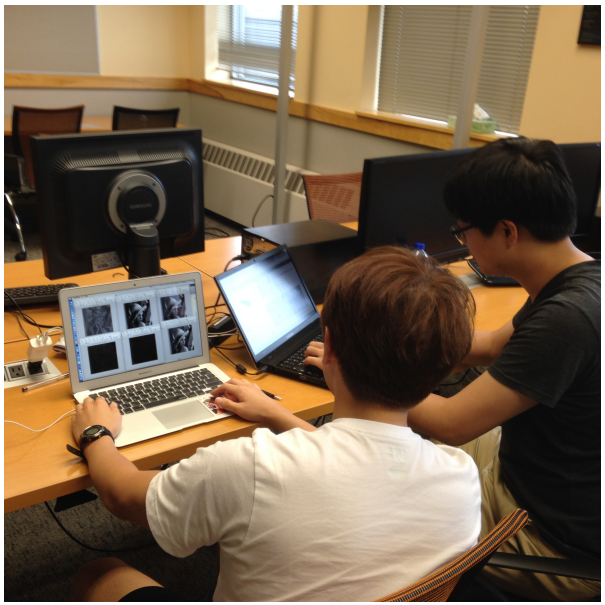




Problem

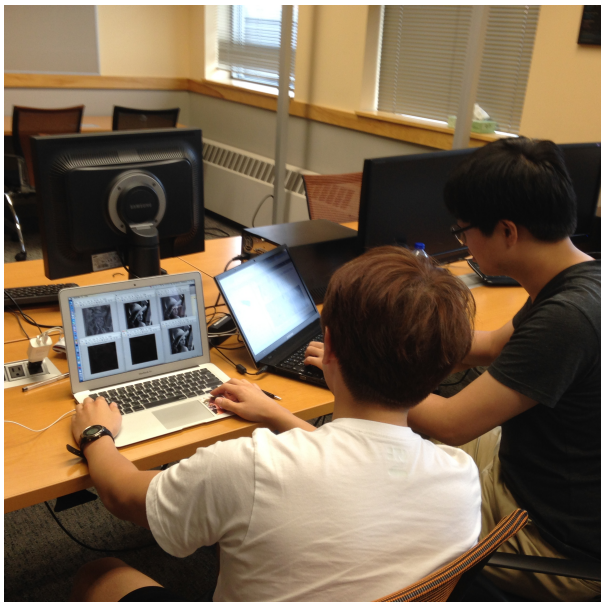
We study the tradeoff between accuracy and system complexity as measured by processing speed and hardware complexity.

Strategies to Simplify Computations in Image Filtering



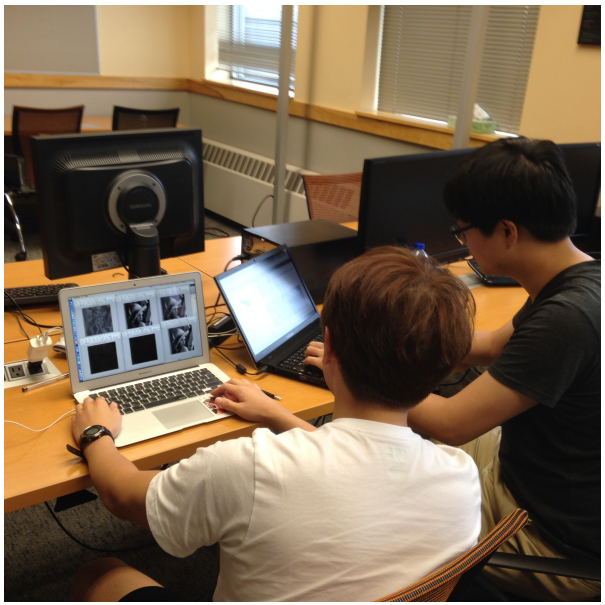
1. Use lookup tables (LUTs) to reduce computation time

Strategies to Simplify Computations in Image Filtering



1. Use lookup tables (LUTs) to reduce computation time
2. Use bit truncation to reduce the size of LUTs

Strategies to Simplify Computations in Image Filtering



1. Use lookup tables (LUTs) to reduce computation time
2. Use bit truncation to reduce the size of LUTs
3. Decompose kernel exactly or approximately to simplify LUTs

Reduce Processing Time Using Look-up Tables (LUTs)

IDEA

Prior to applying filter, pre-compute convolutions for all possible square matrices and store them in a multi-dimensional table.

$$K * \begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix} \rightarrow LUT(x_1, x_2, \ldots, x_9)$$

- Save time by retrieving the values in $LUT(x_1, \ldots, x_9)$ (in place of direct algebraic computations).
- Challenge: a full-size LUT for a 3×3 kernel requires 2^{72} bytes (assuming each pixel is 8 bits).

Truncating to Reduce LUT Size

IDEA

If we reduce the number of possible inputs, we can reduce the number of elements in the LUT!

Truncating to Reduce LUT Size

IDEA

If we reduce the number of possible inputs, we can reduce the number of elements in the LUT!

Solution

Replace the inputs

$$\left.\begin{array}{rl} 40 = & 00101\underline{000} \\ & 00101\underline{001} \\ & 00101\underline{010} \\ & 00101\underline{011} \\ & 00101\underline{100} \\ & 00101\underline{101} \\ & 00101\underline{110} \\ 47 = & 00101\underline{111} \end{array}\right\} \textit{with} \longrightarrow 00101\underline{100}$$

to do the kernel convolution, and pre-store the results in the LUT with index 00101.

Truncation Matrices and Image Norms

If our convolution kernel K is $n \times n$, we can choose a *truncation matrix* $T = [t_{ij}]_{n \times n}$ with integer elements from 0 to 8 to indicate how many bits to truncate from each pixel when convolving with K. An example might be $T = \begin{bmatrix} 5 & 5 & 5 \\ 5 & 2 & 5 \\ 5 & 5 & 5 \end{bmatrix}$

Truncation Matrices and Image Norms

If our convolution kernel K is $n \times n$, we can choose a *truncation matrix* $T = [t_{ij}]_{n \times n}$ with integer elements from 0 to 8 to indicate how many bits to truncate from each pixel when convolving with K. An example might be $T = \begin{bmatrix} 5 & 5 & 5 \\ 5 & 2 & 5 \\ 5 & 5 & 5 \end{bmatrix}$

Definition (Image Norms)

l_∞-error(A,B)$= \max_{i,j} |a_{ij} - b_{ij}|$

l_2-error(A,B)$= \sqrt{\frac{\sum_{i,j}(a_{ij} - b_{ij})^2}{h \times w}}$

PSNR(A,B)$= 20 \cdot \log_{10}(\frac{\mathrm{MAX}_B}{\sqrt{\mathrm{MSE}}})$

$d^2_{\mathsf{IMED}}(A, B) =$
$\frac{1}{2\pi \cdot h \cdot w} \sum_{i,k=1}^{h} \sum_{j,\ell=1}^{w} \exp\left(\frac{-(k^2+\ell^2)}{2}\right)(a_{ij} - b_{ij})(a_{i+k,j+\ell} - b_{i+k,j+\ell})$

High Pass Filters: Direct Computation vs. Truncation

For the next few slides, we will consider these birds.

Example 1 of a 3×3 High Pass Filter

This is them after the HP kernel $K_1 = \frac{1}{4}\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ was applied.

3x3 HP filtered with direct computation

Example 1 of a 3×3 High Pass Filter

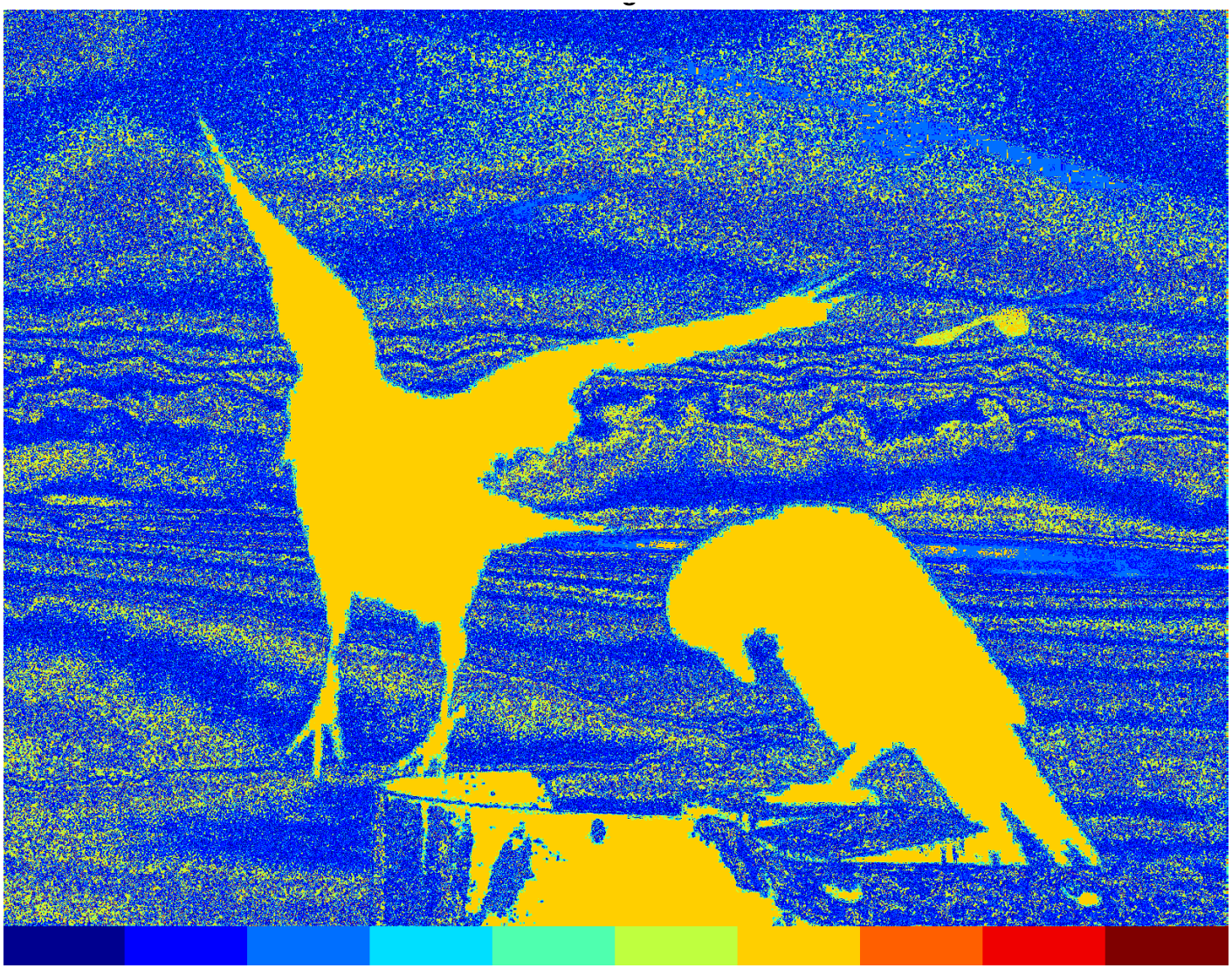
C'mon baby, make it HOT!

Example 1 of a 3×3 High Pass Filter

Here they are with K_1 applied with truncation $T_1 = \begin{bmatrix} 8 & 3 & 8 \\ 3 & 1 & 3 \\ 8 & 3 & 8 \end{bmatrix}$.

Example 1 of a 3×3 High Pass Filter

Here is the difference image, color-coded for convenience.



Example 2 of a 3×3 High Pass Filter

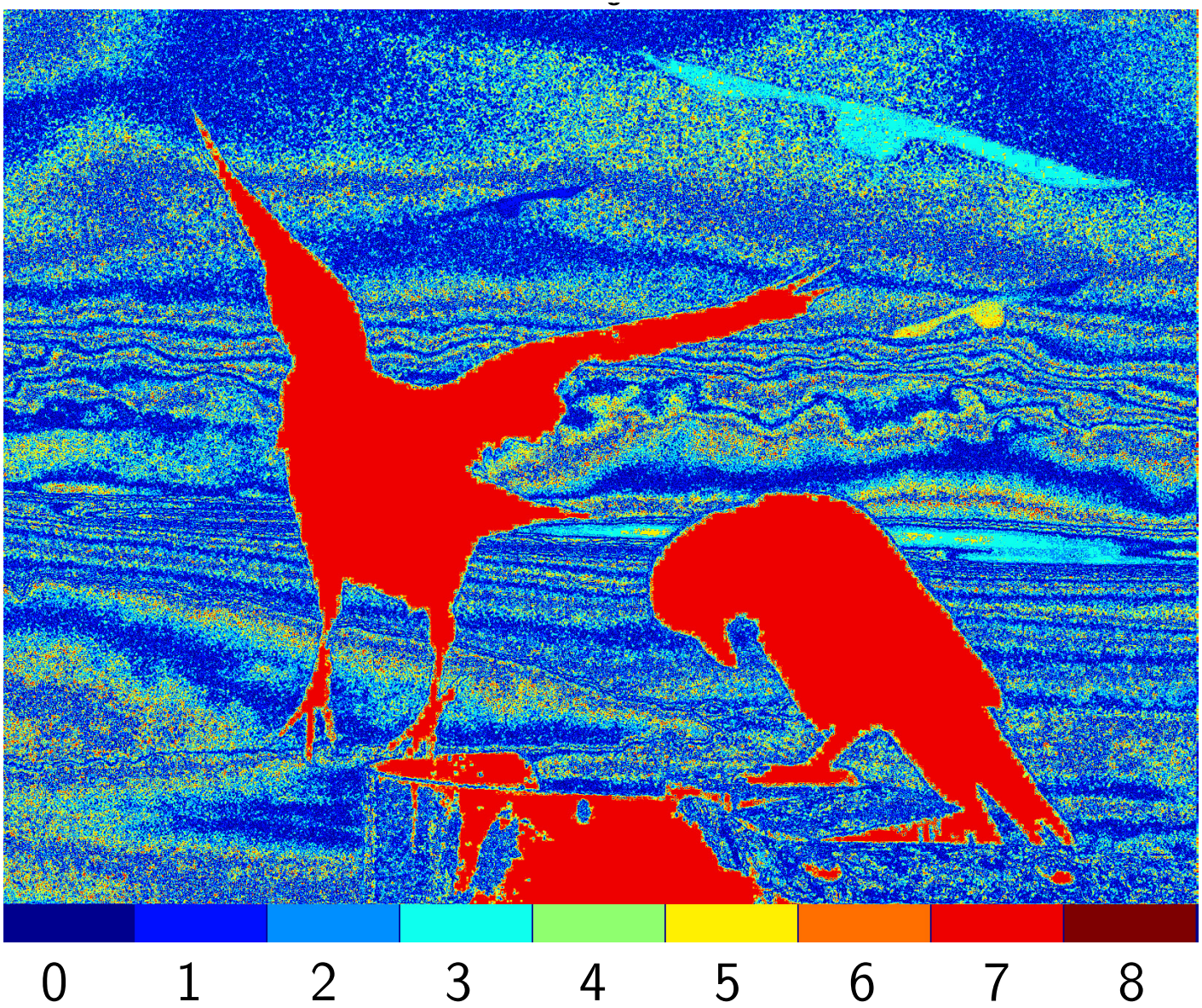
This is the gulls after the HP kernel $K_2 = \frac{1}{8}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$ was applied.

Example 2 of a 3×3 High Pass Filter

Here they are with K_1 applied with truncation $T_2 = \begin{bmatrix} 4 & 4 & 4 \\ 4 & 1 & 4 \\ 4 & 4 & 4 \end{bmatrix}$.

Example 2 of a 3×3 High Pass Filter

Here is the difference image, color-coded for convenience.



Example of a 5×5 High Pass Filter

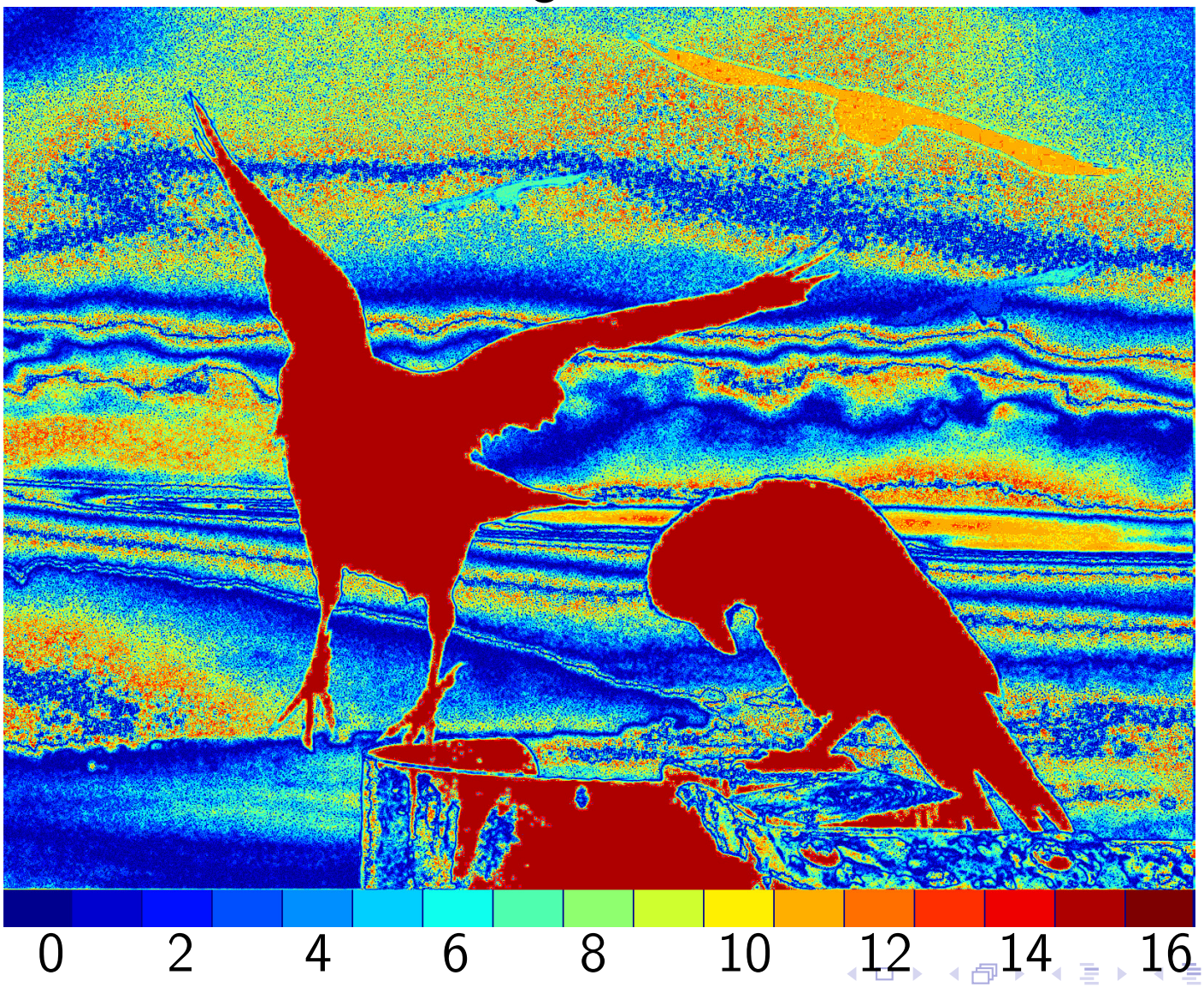
This is the gulls after the HP kernel $K_3 = \frac{1}{24}\begin{bmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & 24 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix}$.

Example of a 5×5 High Pass Filter

Here they are with K_3 applied with truncation $T_3 = \begin{bmatrix} 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 1 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 \end{bmatrix}$.

Example of a 5×5 High Pass Filter

Here is the difference image, color-coded for convenience.



And the Data to Back it Up!

Here are three very different pictures.

(a) A lighthouse

(b) A sunset

(c) A waterfall

And the Data to Back it Up!

Here are three very different pictures.

(a) A lighthouse

(b) A sunset

(c) A waterfall

When converted to grayscale and compared, we have the following chart:

	ℓ^2	ℓ^∞	PSNR	d_{IMED}
(a) vs. (b)	64.8743	251	9.3835	64.043
(a) vs. (c)	59.2807	247	7.7803	58.6937
(b) vs. (c)	36.9946	229	10.3046	36.4582

Table: Comparison of image distances

And the Data to Back it Up!

	ℓ^2	ℓ^∞	PSNR	d_{IMED}
(a) vs. (b)	64.8743	251	9.3835	64.043
(a) vs. (c)	59.2807	247	7.7803	58.6937
(b) vs. (c)	36.9946	229	10.3046	36.4582

Keep these numbers in mind as we compare our filtered images.

And the Data to Back it Up!

	ℓ^2	ℓ^∞	PSNR	d_{IMED}
(a) vs. (b)	64.8743	251	9.3835	64.043
(a) vs. (c)	59.2807	247	7.7803	58.6937
(b) vs. (c)	36.9946	229	10.3046	36.4582

Keep these numbers in mind as we compare our filtered images.

	ℓ^2	ℓ^∞	PSNR	d_{IMED}
K_1 DC vs. K_1 Trunc.	3.4992	9	37.2515	3.1401
K_2 DC vs. K_2 Trunc.	3.6366	8	36.917	3.3553
K_3 DC vs. K_3 Trunc.	8.5556	16	29.4858	8.2883

Table: Comparison of filters and their truncated forms

And the Data to Back it Up!

	ℓ^2	ℓ^∞	PSNR	d_{IMED}
(a) vs. (b)	64.8743	251	9.3835	64.043
(a) vs. (c)	59.2807	247	7.7803	58.6937
(b) vs. (c)	36.9946	229	10.3046	36.4582

Keep these numbers in mind as we compare our filtered images.

	ℓ^2	ℓ^∞	PSNR	d_{IMED}
K_1 DC vs. K_1 Trunc.	3.4992	9	37.2515	3.1401
K_2 DC vs. K_2 Trunc.	3.6366	8	36.917	3.3553
K_3 DC vs. K_3 Trunc.	8.5556	16	29.4858	8.2883

Table: Comparison of filters and their truncated forms

The PSNR values are particularly indicative of acceptable error, since lossy compression typically has a PSNR of about 30-50.

Decomposing rank 1 matrix

IDEA

If the kernel $(n \times n)$ matrix K has rank 1, then

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

where $c, b \in \mathbb{R}^{n \times 1}$. For example,

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

Applying a filter K to an Image then becomes

$$\text{IMAGE} * K = \text{IMAGE} * (c * b^T) = (\text{IMAGE} * c) * b^T$$

so that we need two LUTs (with b and c) for one convolution.

Decomposing rank 1 matrix

An example: low pass filter with kernel

$$K = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \frac{1}{4}\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \triangleq c * c^T$$

Truncation matrix:

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

By symmetry, only one look up table is needed.

1. Step 1: Generate look up table for $c = [1 \;\; 2 \;\; 1]^T$.
2. Step 2: Convolve the image with c.
3. Step 3: Convolve transpose of the image with c.

(a) Original image

(b) Low pass, direct

(c) Low pass, LUT with T_1

(d) Low pass, LUT with T_2

Figure: Comparison of filtered images.

Scheme	LUT Size	PSNR (dB)
LUT with T_1	512KB	41.8459
LUT with T_2	16KB	30.4442

Table: Error and size of LUT for LUTs with different truncation schemes.

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: Best truncation scheme for fixed-size LUT.

Singular value decomposition

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):

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

Here u_i,$v_i \in \mathbb{R}^{n\times 1}$ are the left and right singular vectors respectively and σ_i's 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.

Singular value decomposition

Example:

$$K = \frac{1}{8}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} = U\Sigma V^T$$

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

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

T_1, T_2 as above.

(a) Original image

(b) High pass, direct

(c) High pass, LUT with T_1

(d) High pass, LUT with T_2

Figure: Comparison of filtered images.

Figure: Difference between the filtered image with direct computation and with LUT, T_2. Maximum pixel-wise difference: 18.

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

Table: Error and size of LUT for LUTs with different truncation schemes.

An Approximate Decomposition

IDEA

The more zeros in our filter the smaller the LUT. Can we decompose the (3× 3) filter K in a better way? Consider:

$$K = A * B$$

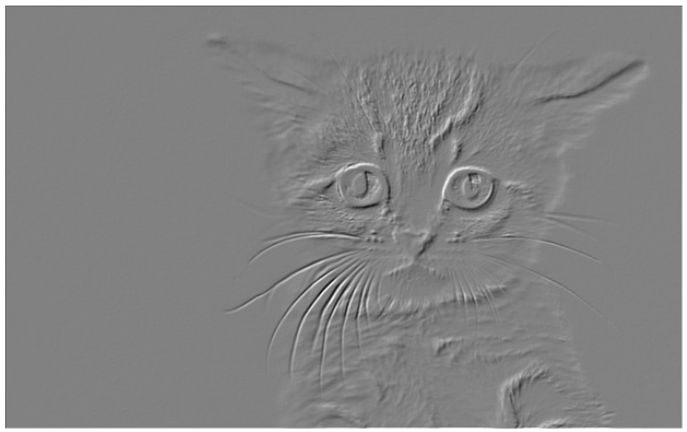
where

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

- Size of the LUT reduces from 2^{72} bytes to 2^{49} bytes before any truncation.
- We can solve the minimization problem

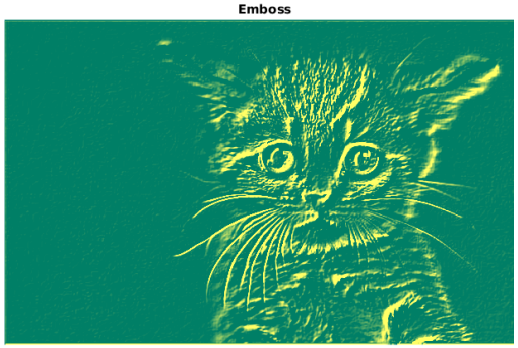
$$\min(\|K - A * B\|_F + TotalEdgeSum)$$

Approximate Decomposition Example



$$K = \begin{bmatrix} .1 & .1 & .1 \\ .1 & .2 & .1 \\ .1 & .1 & .1 \end{bmatrix}, A = \begin{bmatrix} 0 & 0.2922 & 0 \\ -0.0000 & 0.4132 & 0 \\ 0.0000 & 0.2922 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 & 0 \\ 0.2922 & 0.4132 & 0.2922 \\ -0.0000 & -0.0000 & 0 \end{bmatrix}$$

- $\|K - A_T * B_T\|_F < 3.04 * 10^{-5}$



(a) Emboss

(b) High-Pass

(c) Averaging

	Emboss	Low-Pass	Averaging	High-Pass
PSNR	109.7337	71.0895	51.2598	25.6056
l_2 - error	8.2883e-04	2.7895e-04	0.7163	13.3739
l_∞ - error	0.0019	6.1630e-04	4.2445	24.4963

Table: Comparison of errors

Decomposing a Kernel Matrix by Rows

- A kernel matrix $K_{n\times n}$ of *any* rank can always be decomposed as

$$K = \begin{bmatrix}1\\0\\\vdots\\0\end{bmatrix} * [r_1] + \begin{bmatrix}0\\1\\\vdots\\0\end{bmatrix} * [r_2] + \cdots + \begin{bmatrix}0\\0\\\vdots\\1\end{bmatrix} * [r_n],$$

  where r_i is the ith row of K.
- Essentially, this decomposition allows us to convolve our image by one row of K at a time, and simply add the results at the end.
- If using LUTs, computation is reduced to n look-ups and $n-1$ additions per pixel, with each LUT having 2^{8n} entries.
- Compare this to a single LUT, which requires only one look-up operation, but the table contains 2^{8n^2} entries!
- Even when $n = 3$, this is the difference between one 4-zebibyte table (≈1% of one mole of bytes) and three 16MB tables!

Decomposing a Kernel Matrix by Rows

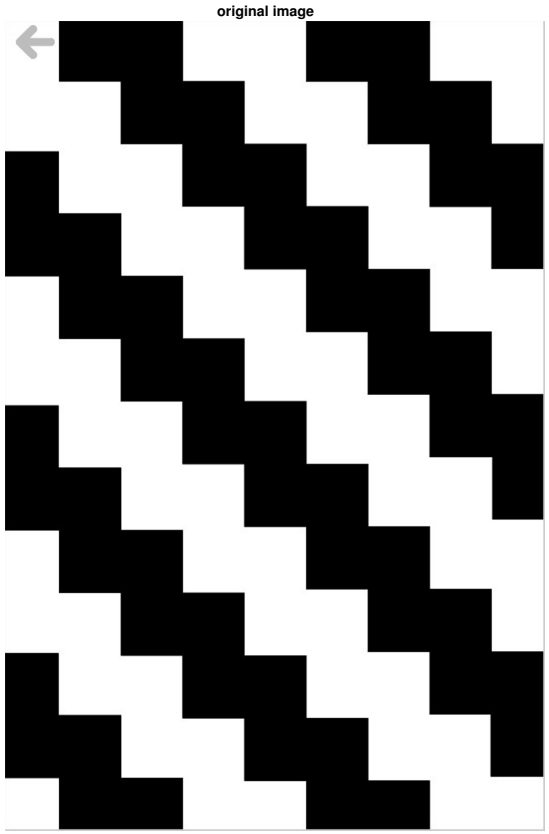


Figure: Original Image

Decomposing a Kernel Matrix by Rows

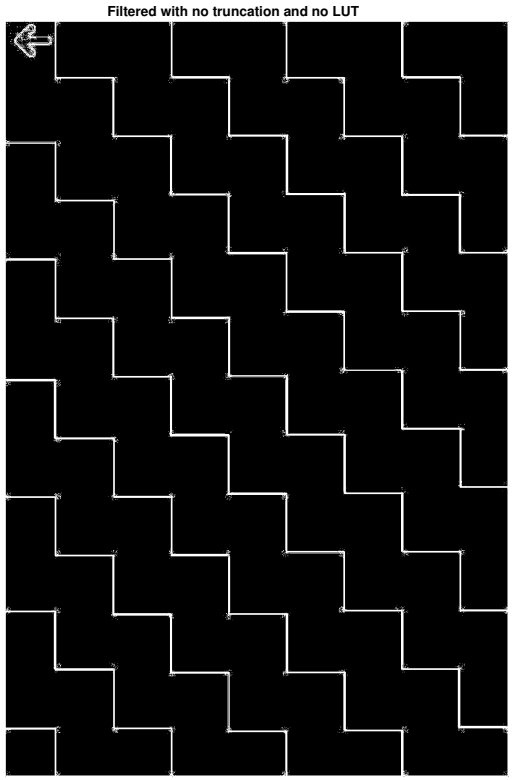


Figure: Image after HP filter

Decomposing a Kernel Matrix by Rows

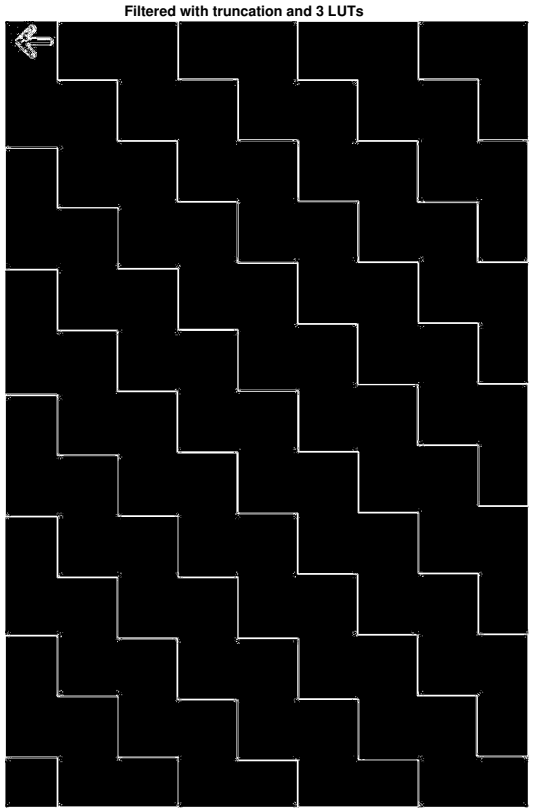


Figure: Image after HP filter with 3 LUT's containing 16, 4096, and 16 entries

Truncation and Blurring Filter (using Edge-detection filter)

Observation

1. The truncation method can be used to reduce running time.
2. Truncation makes some errors such as staircase effects for blurring, but is decent for edge-detecting.

Truncation and Blurring Filter (using Edge-detection filter)

Observation

1. The truncation method can be used to reduce running time.
2. Truncation makes some errors such as staircase effects for blurring, but is decent for edge-detecting.

$$\Downarrow$$

Truncation and Blurring Filter (using Edge-detection filter)

Observation

1. The truncation method can be used to reduce running time.
2. Truncation makes some errors such as staircase effects for blurring, but is decent for edge-detecting.

$$\Downarrow$$

Goal

Take advantage of the truncation method avoiding staircase effects!

Truncation and Blurring Filter (using Edge-detection filter)

IDEA

$I - L$ can be regarded as an edge-detection filter even though it is not a high-pass filter in general.

Truncation and Blurring Filter (using Edge-detection filter)

IDEA

$I - L$ can be regarded as an edge-detection filter even though it is not a high-pass filter in general.

Example of $I - L$.

Strategy

1. Let L be given low-pass filter.
2. Set $H := I - L$.
3. Find $I - H(T)$ for a certain truncation T.

Truncation and Blurring Filter (using Edge-detection filter)

IDEA

$I - L$ can be regarded as an edge-detection filter even though it is not a high-pass filter in general.

Example of $I - L$.

Strategy

1. Let L be given low-pass filter.
2. Set $H := I - L$.
3. Find $I - H(T)$ for a certain truncation T.

$\Downarrow$

Truncation and Blurring Filter (using Edge-detection filter)

IDEA

$I - L$ can be regarded as an edge-detection filter even though it is not a high-pass filter in general.

Example of $I - L$.

Strategy

1. Let L be given low-pass filter.
2. Set $H := I - L$.
3. Find $I - H(T)$ for a certain truncation T.

$$\Downarrow$$

$$L \approx I - H(T)$$

Truncation and Blurring Filter (using Edge-detection filter)

Let

$$L = \frac{1}{8}\begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \text{ and } T = \begin{bmatrix} 8 & 6 & 8 \\ 6 & 4 & 6 \\ 8 & 6 & 8 \end{bmatrix}.$$

(a) Low-pass filter, L (b) Truncation, $L(T)$ (c) New Idea, $I - H(T)$

Figure: Comparison of filtered images

The table 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.

	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: Comparison of errors

Further Directions

Figure: Eat more Punch pizza!

Thank you!