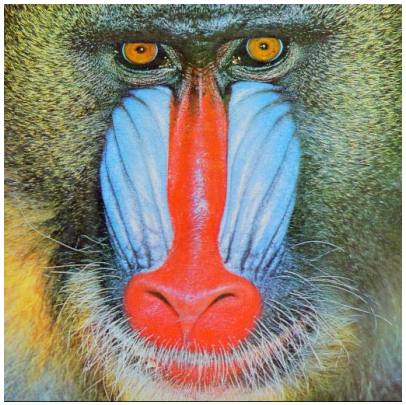
Fast and Somewhat Accurate Algorithms (Team 4)

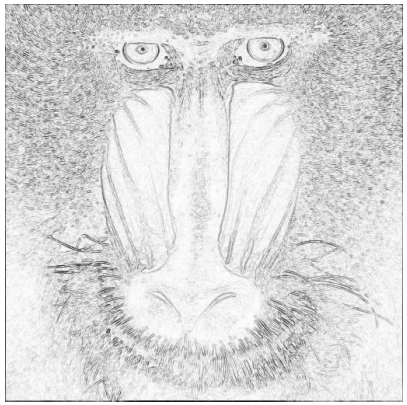
Mario Barela, Su Chen, Emily Gunawan, Morgan Schreffler, Minho Song, Doyeob Yeo, with mentor: Chai Wah Wu (IBM)

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.

Reduce processing time using look-up tables (LUT)

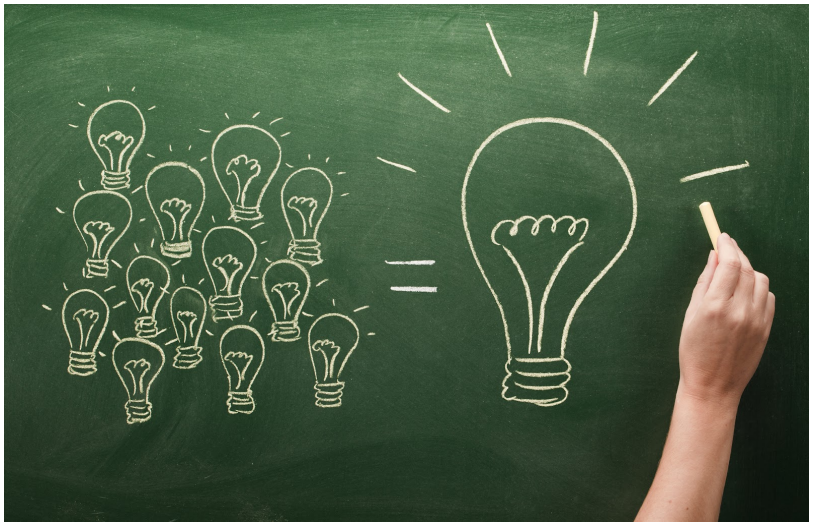
IDEA

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

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

- 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).

Some Strategies to reduce LUTs' size



1. LUT and Truncation
2. Matrix Decomposition and Approximate decomposition

Truncation to reduce LUT's size

Example: truncating by 3 digits.
Replace

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

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

Decomposing rank 1 matrix

IDEA

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

$$K = cb^T$$

where $c, b \in \mathbb{R}^n$. For example, the low pass filter kernel.

Emily writes: the person doing this slide can also mention quickly the existence of other commonly-used rank-1 filters. The gradient filters such as Sobel gradient filter (an edge-detection filter) is also of rank 1 and can be decomposed similarly.

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

Applying a filter K to an Image then becomes

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

Decomposing rank 1 matrix

An example: blurring filter with kernel

$$K = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} = c \cdot c^T$$

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

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:

$$K = A * B$$

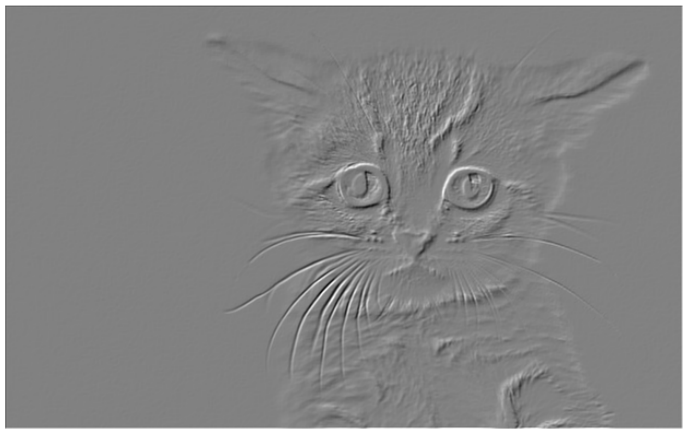
where

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

- 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



$$K = \begin{bmatrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}, A = \begin{bmatrix} -0.000 & -0.000 & 0.000 \\ -0.000 & 1.702 & 0 \\ 0.000 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} -0.587 & -0.587 & 0 \\ -0.587 & 0 & 0.587 \\ 0 & 0.587 & 0.587 \end{bmatrix}$$

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

	Emboss	Low-Pass	Averaging
PSNR	42.0500	71.0895	
l_2 - error	0.0079	2.7895e-04	
max - error	0.0144	6.1630e-04	

Table: Comparison of errors

The errors are measured as such

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

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

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

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.

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.

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.

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!

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

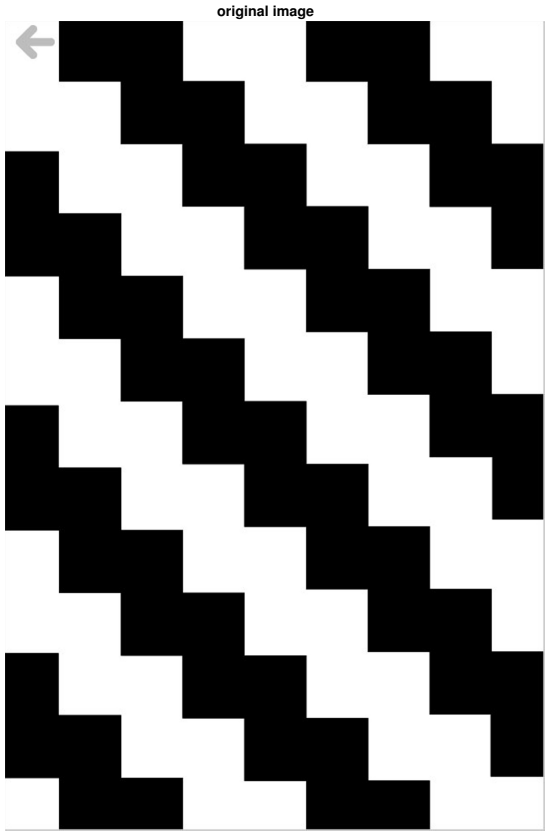


Figure: Original Image

Decomposing a Kernel Matrix by Rows

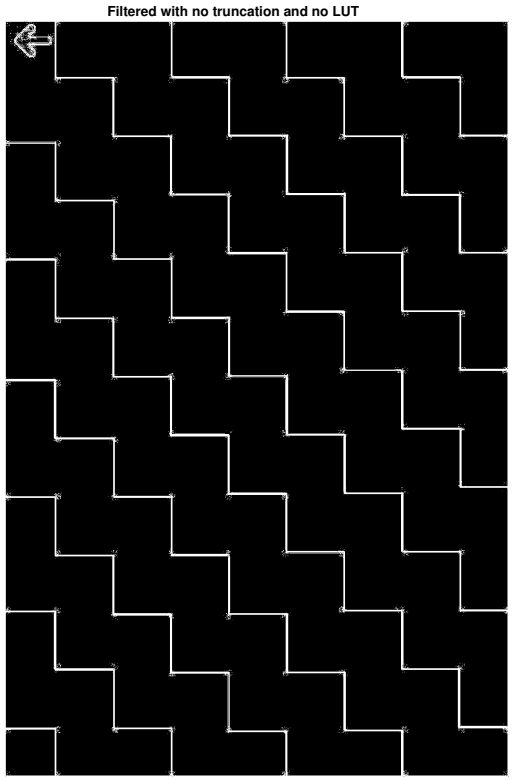


Figure: Image after HP filter

Decomposing a Kernel Matrix by Rows

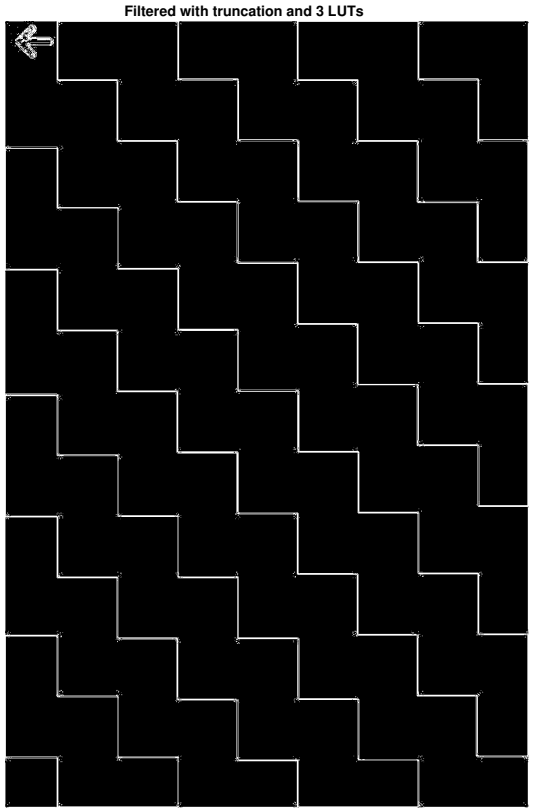


Figure: Image after HP filter with 3 LUT's containing 16, 4096, and 16 entries

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

1. Refine convolution decomposition
2. SVD (singular value decomposition)

Thank you!