Efficient filters + linear algebra review

Logistics

- Office hours question; what does “chance performance” mean?

- Printing PDF

- Next 3 lectures

  - Thurs, this week: image warping (HW2 info)

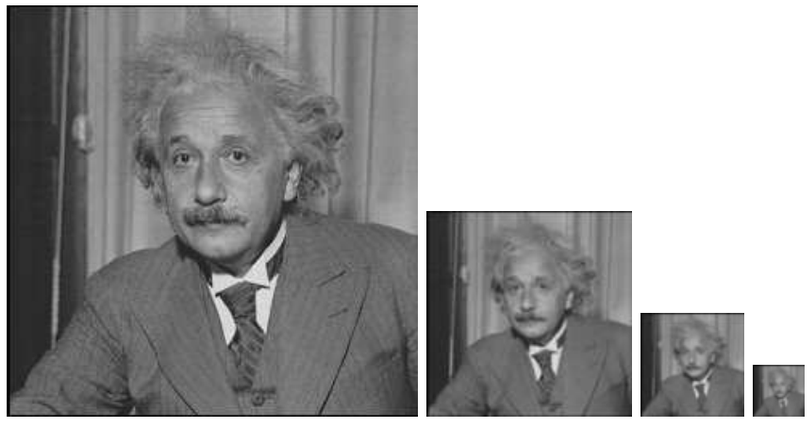
  - Tues, next week: no class

  - Thurs, next week: guest lecture

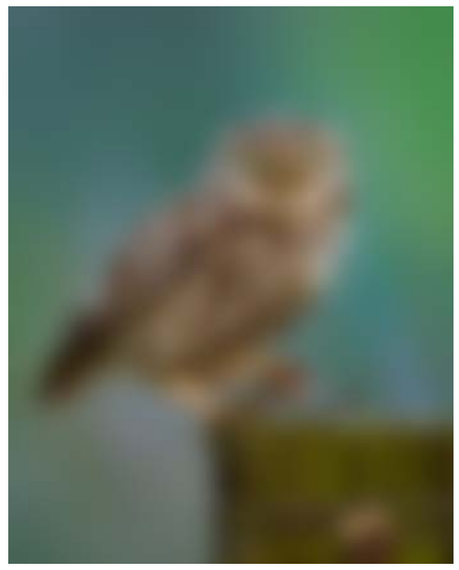
Outline

- Convolution
  - Linear Shift Invariance (LSI)
  - Convolution Properties (commutative, associative, distributive)
  - Normalize Cross-Correlation
- Edges
  - Canny edges (hysterisis)
  - derivative-of-gaussians (DoG)
  - laplacian-of-Gaussians (LoG)
  - filter banks
- **Efficiency**
  - pyramids
  - linear approximiations (SVD, separability)
  - steerability

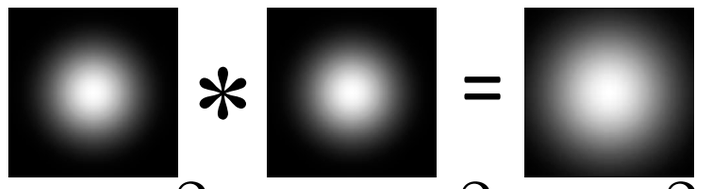
Pyramids



- Big filters (e.g., Gaussians) tend to be smooth, so the output is redundant



- Exploit property that Gaussian*Gaussian = Bigger Gaussian



$$\sigma_{a*b}^2 = \sigma_a^2 + \sigma_b^2$$

Proof: https://en.wikipedia.org/wiki/Sum_of_normally_distributed_random_variables

The Laplacian Pyramid as a Compact Image Code

PETER J. BURT, MEMBER, IEEE, AND EDWARD H. ADELSON

Abstract—**We describe a technique for image encoding in which local operators of many scales but identical shape serve as the basis functions. The representation differs from established techniques in that the code elements are localized in spatial frequency as well as in space.**

Pixel-to-pixel correlations are first removed by subtracting a low-pass filtered copy of the image from the image itself. The result is a net data compression since the difference, or error, image has low variance and entropy, and the low-pass filtered image may represented at reduced sample density. Further data compression is achieved by quantizing the difference image. These steps are then repeated to compress the low-pass image. Iteration of the process at appropriately expanded scales generates a pyramid data structure.

The encoding process is equivalent to sampling the image with Laplacian operators of many scales. Thus, the code tends to enhance salient image features. A further advantage of the present code is that it is well suited for many image analysis tasks as well as for image compression. Fast algorithms are described for coding and decoding.

does not permit simple sequential coding. Noncausal approaches to image coding typically involve image transforms, or the solution to large sets of simultaneous equations. Rather than encoding pixels sequentially, such techniques encode them all at once, or by blocks.

Both predictive and transform techniques have advantages. The former is relatively simple to implement and is readily adapted to local image characteristics. The latter generally provides greater data compression, but at the expense of considerably greater computation.

Here we shall describe a new technique for removing image correlation which combines features of predictive and transform methods. The technique is noncausal, yet computations are relatively simple and local.

The predicted value for each pixel is computed as a local

Peter J. Burt (M'80) received the B.A degeree i physics from Harvard University, Cambridge, MA in 1968, and the M.S. and Ph.D. degrees i computer science from the University of Massa chusetts, Amherst, in 1974 and 1976, respec-tively

From 1968 to 1972 he conducted research i sonar, particularly in acoustic imaging devices a the USN Underwater Sound Laboratory, Nev London, CT and in London, England. As Postdoctoral Fellow, he has studied both natural vision and computer imag understanding at New York University, New York, NY (1976-1978), Be Laboratories (1978-1979), and the University of Maryland, College Par (1979-1989). He has been a member of the faculty at Rensselae Polytechnic Institute, Troy, NY, since 1980.

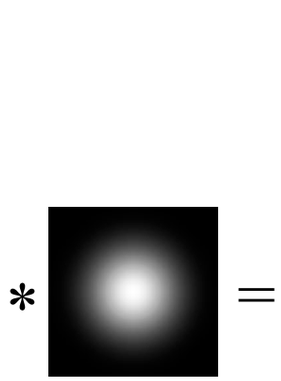
Edward H. Adelson received B.A. degree in physics and philosophy from Yale University, New Haven, CT, in 1974, and the Ph.D degree in experimental psychology from the University of Michigan, Ann Arbor, in 1979.

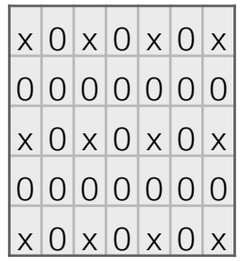
From 1979 to 1981 he was Postdoctoral Fellow at New York University, New York, NY. Since 1981, he has been at RCA David Sarnoff Research Center, Princeton, NJ, as a member of the Technical Staff in the Image Quality and Human Perception Research Group. His research interests center on visual processes in both human and machine visual systems, and include psychophysics, image processing, and artificial intelligence.

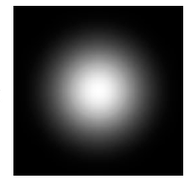
Dr. Adelson is a member of the Optical Society of America, the Association for Research in Vision and Opthalmology, and Phi Beta Kappa.

Key idea: rather than convolving with big gaussian

Convolve with small gaussian, subsample, convolve with small gaussian, subample,….



Can we make even more efficient (since we know we will be subsampling every-other-row-and-column the output)?

yes - perform *strided* convolution that shifts filter by 2 pixels instead of 1

Burt & Adelson, 83

Implementation in 1D

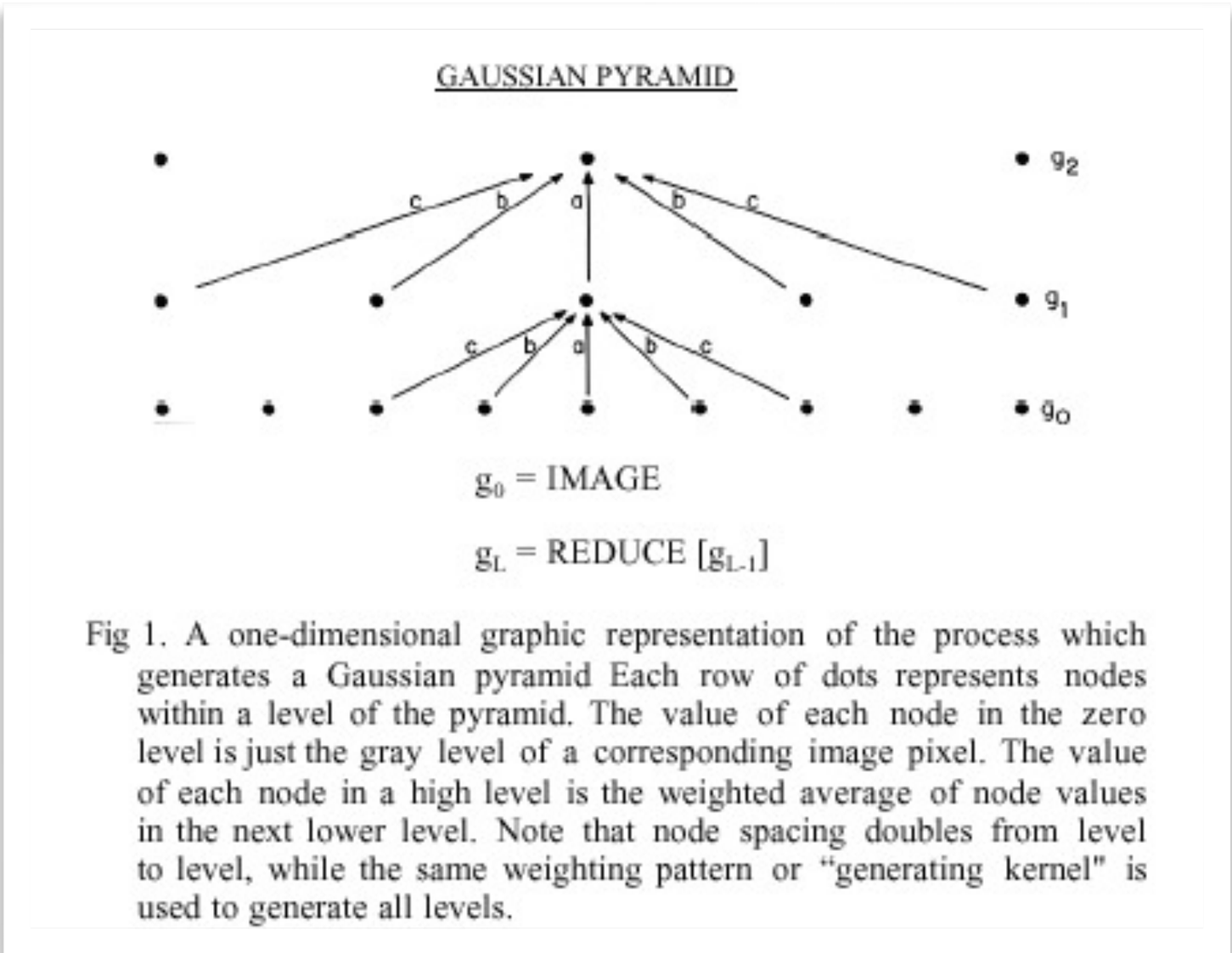
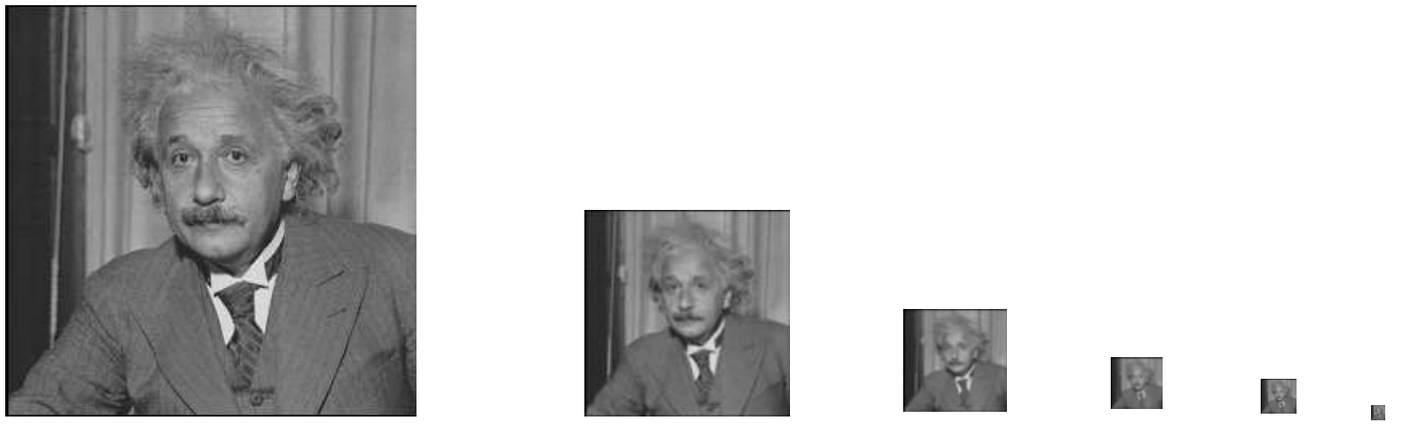


Fig 1. A one-dimensional graphic representation of the process which generates a Gaussian pyramid Each row of dots represents nodes within a level of the pyramid. The value of each node in the zero level is just the gray level of a corresponding image pixel. The value of each node in a high level is the weighted average of node values in the next lower level. Note that node spacing doubles from level to level, while the same weighting pattern or "generating kernel" is used to generate all levels.

$$H[i] = \frac{1}{16}\begin{bmatrix}1 & 4 & 6 & 4 & 1\end{bmatrix}$$

5-tap 1D Gaussian filter



First 6 levels of a Gaussian Pyramid. Original image is 257x257; level 5 is 9x9

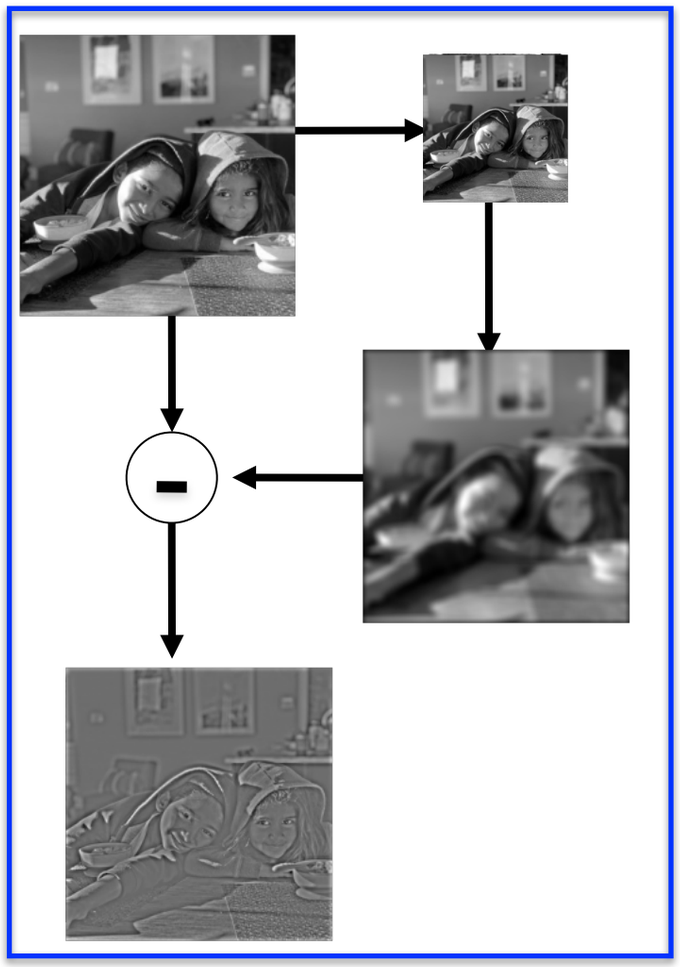
$$REDUCE(F)[i] = \sum_{u=-2}^{2} H[u]F[2i+u]$$

$$H[i] = \frac{1}{16}\begin{bmatrix}1 & 4 & 6 & 4 & 1\end{bmatrix}$$

```
def reduce(image):
# Reduce size of image by 2X by blurring and subsampling
  k = np.array([1,4,6,4,1])/16
  kernal = np.outer(k,k)
  res = np.zeros((image.shape[0]//2,image.shape[1]//2,image.shape[2]))
  for i in range(image.shape[2]):
    res[:,:,i] = signal.convolve2d(image[:,:,i], kernal, mode='same')[::2,::2]
  return res
```

Laplacian pyramid

Store *difference* between upsampled image and original image

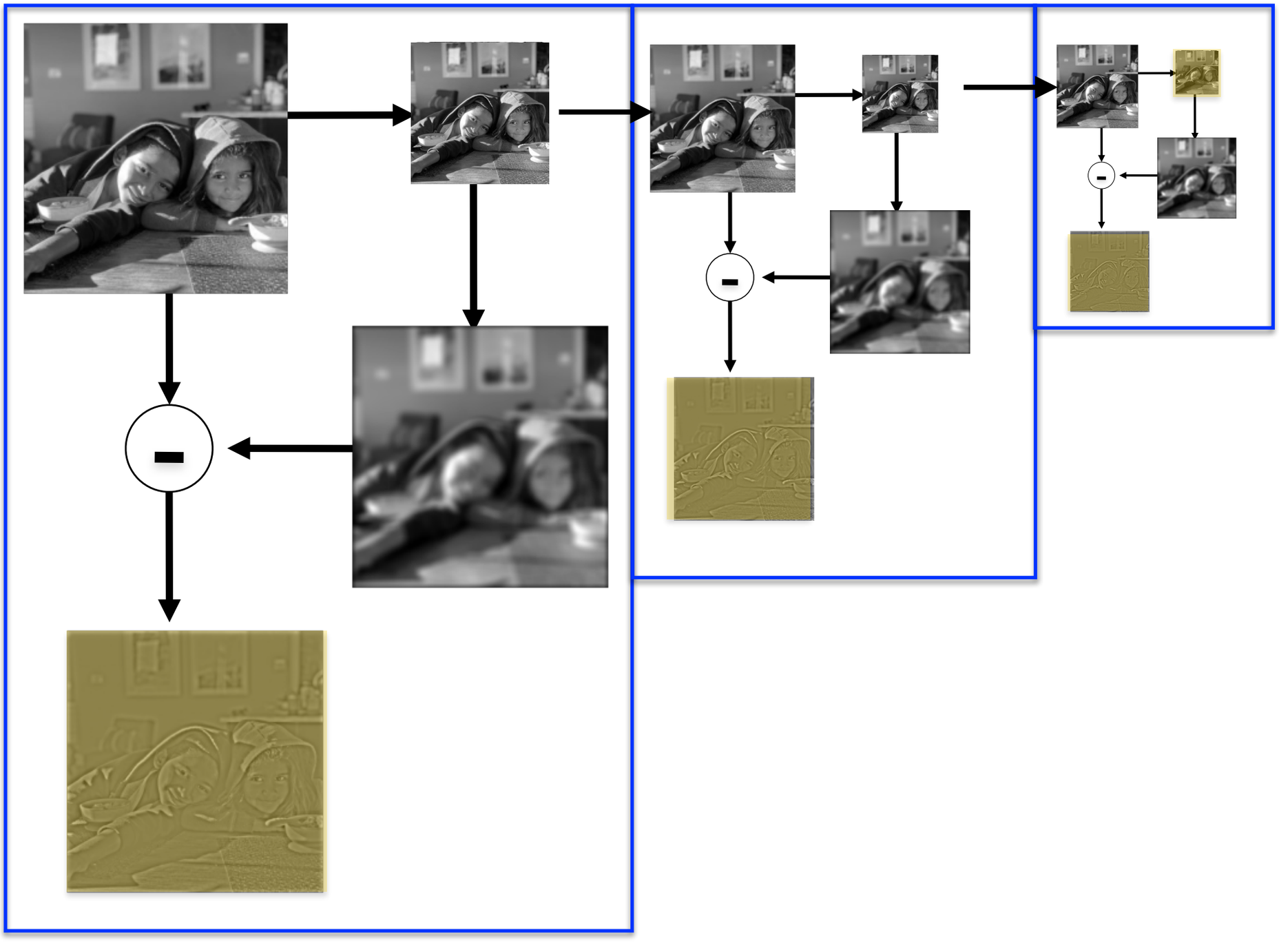


Upsample by inserting zeros between pixels and apply renormalized Gaussian filter:

$$EXPAND(F)[i] = 2 \sum_{u=-2}^{2} H[u]F[(i+u)/2]$$

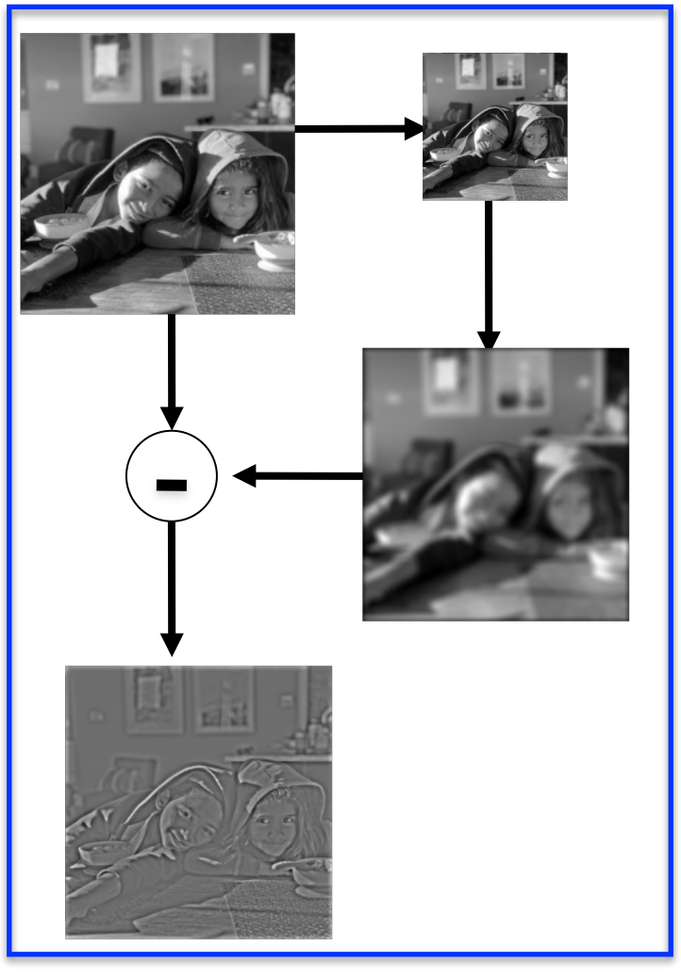

```
def expand(image):
# Expand image size by 2X by zero-interlacing and weighted-averaging
  k = np.array([1,4,6,4,1])/16
  kernal = np.outer(k,k)
  res = np.zeros((2*image.shape[0],2*image.shape[1],image.shape[2]))
  res[::2,::2,:] = image
  for i in range(image.shape[2]):
    res[:,:,i] = signal.convolve2d(res[:,:,i],4*kernal, mode='same')
  return res
```

Laplacian pyramid



Only need to store tiny-res image + sparse delta-images

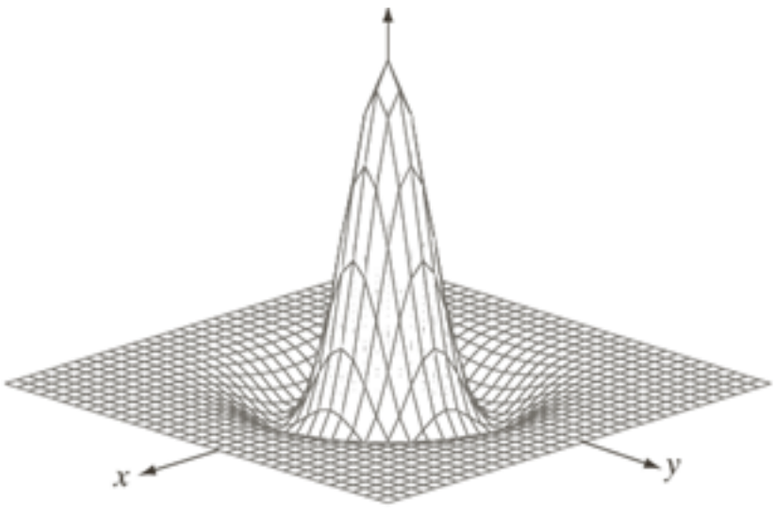
Laplacian pyramid

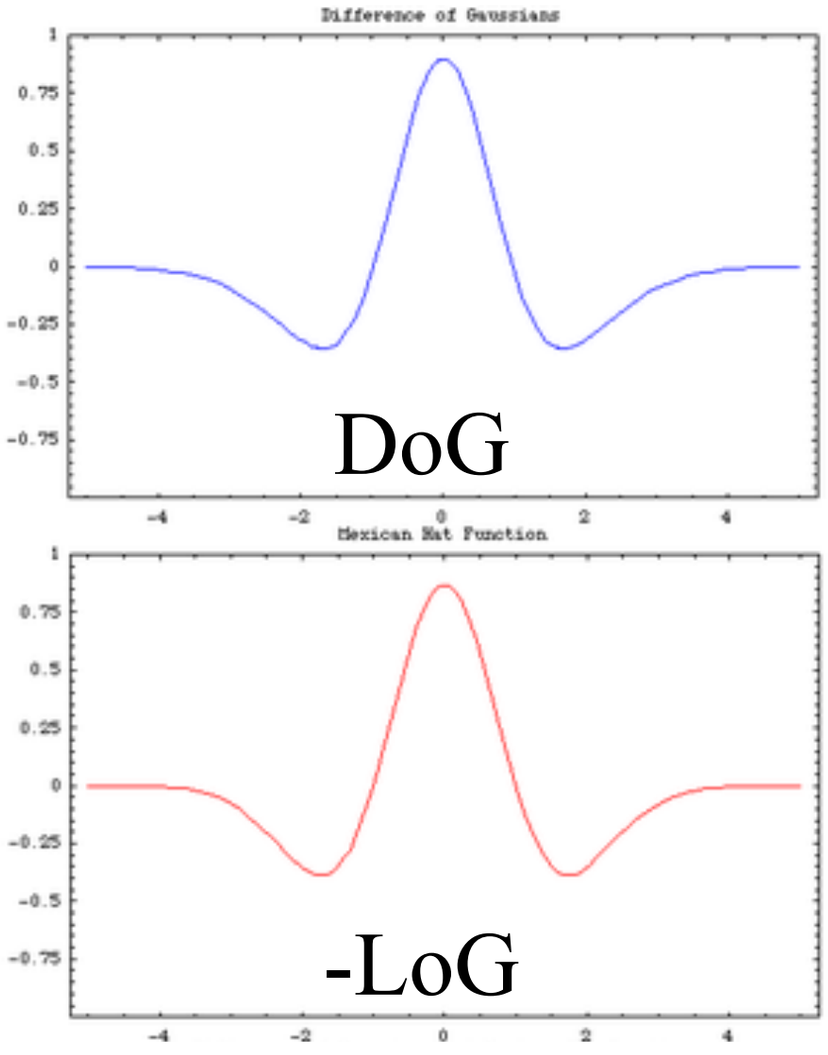


Can we directly produce the difference image with a linear filter?

Difference of Gaussian (DoG) filter

Looks very similar to a laplacian of Gaussian filter, but a different derivation!

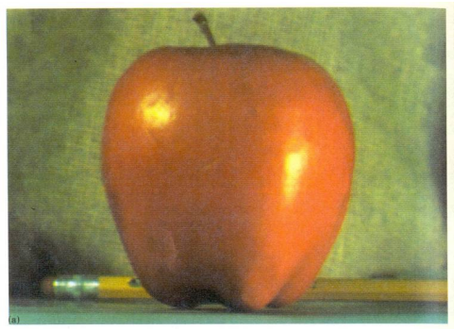




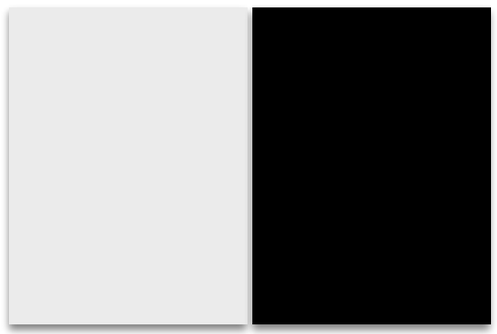
Multiresolution blending

demo_blending.ipynb

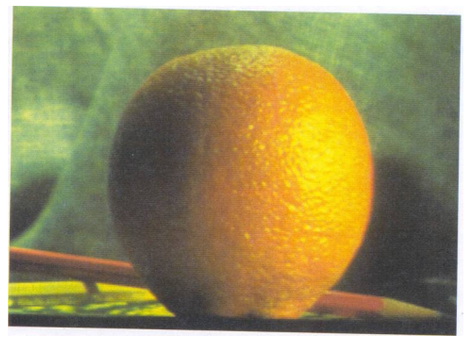
Left image

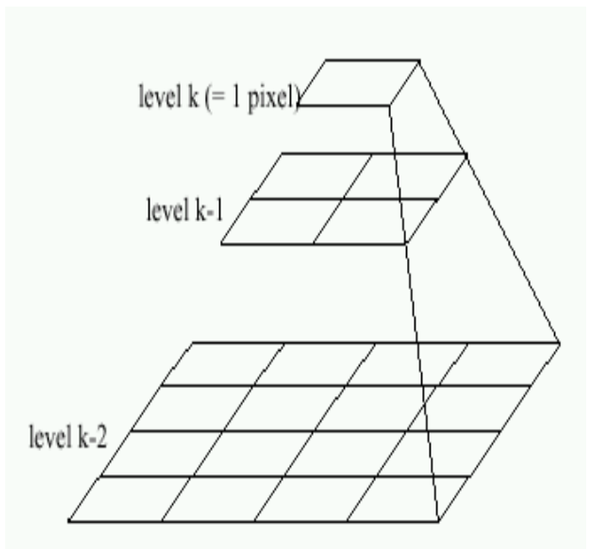


Blend mask

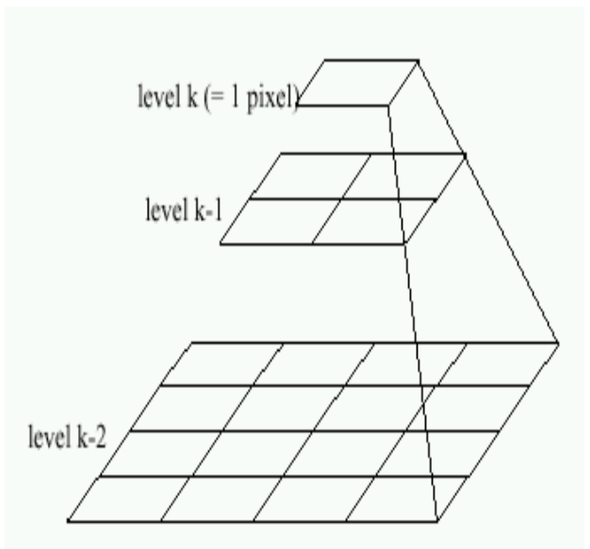


Right image

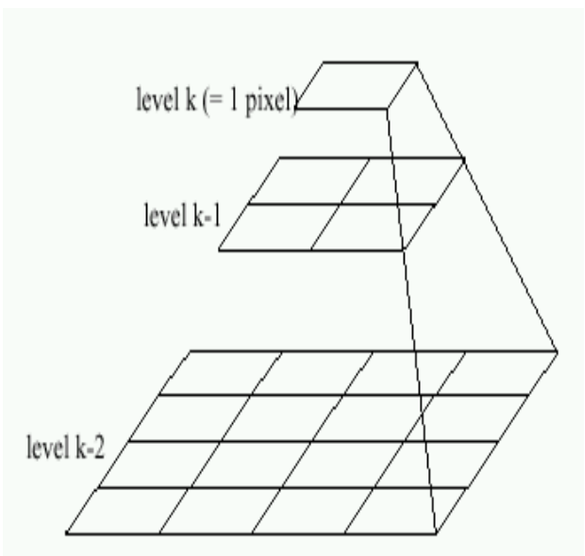




Laplacian pyramid



Gaussian pyramid

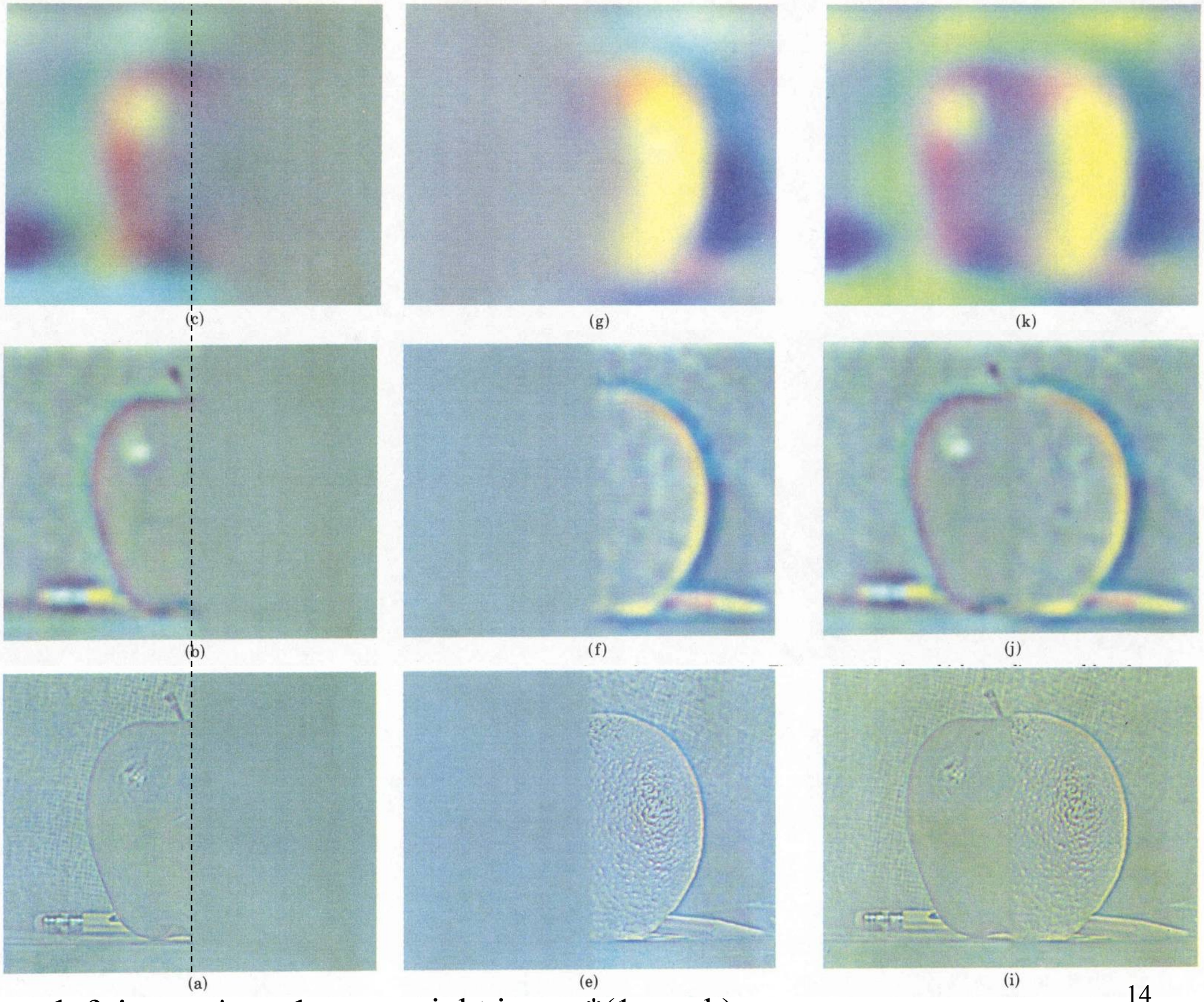


Laplacian pyramid

laplacian level 4

laplacian level 2

laplacian level 0

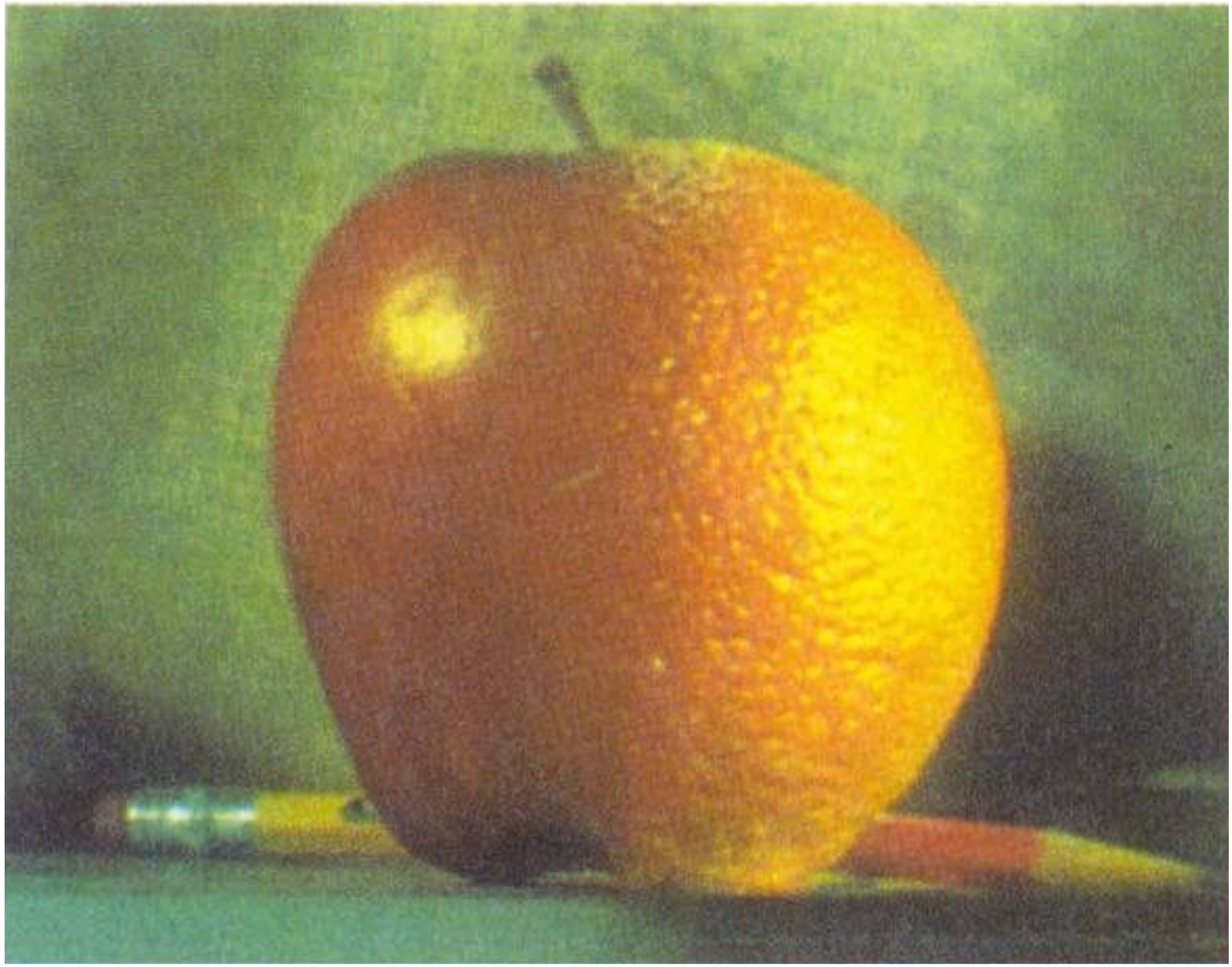


left image*mask

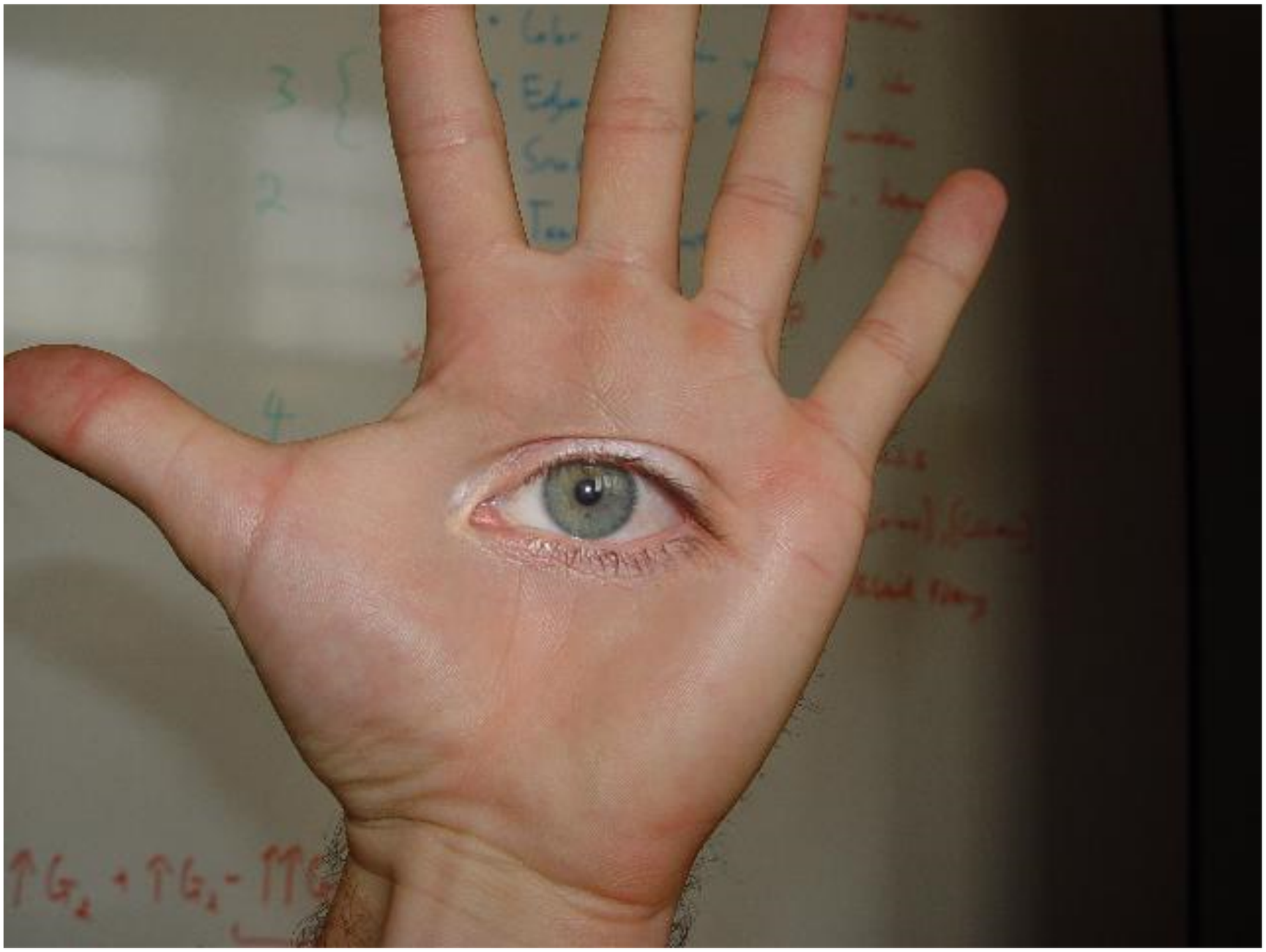
right image*(1-mask)

sum

Pyramid Blending



Fun example!

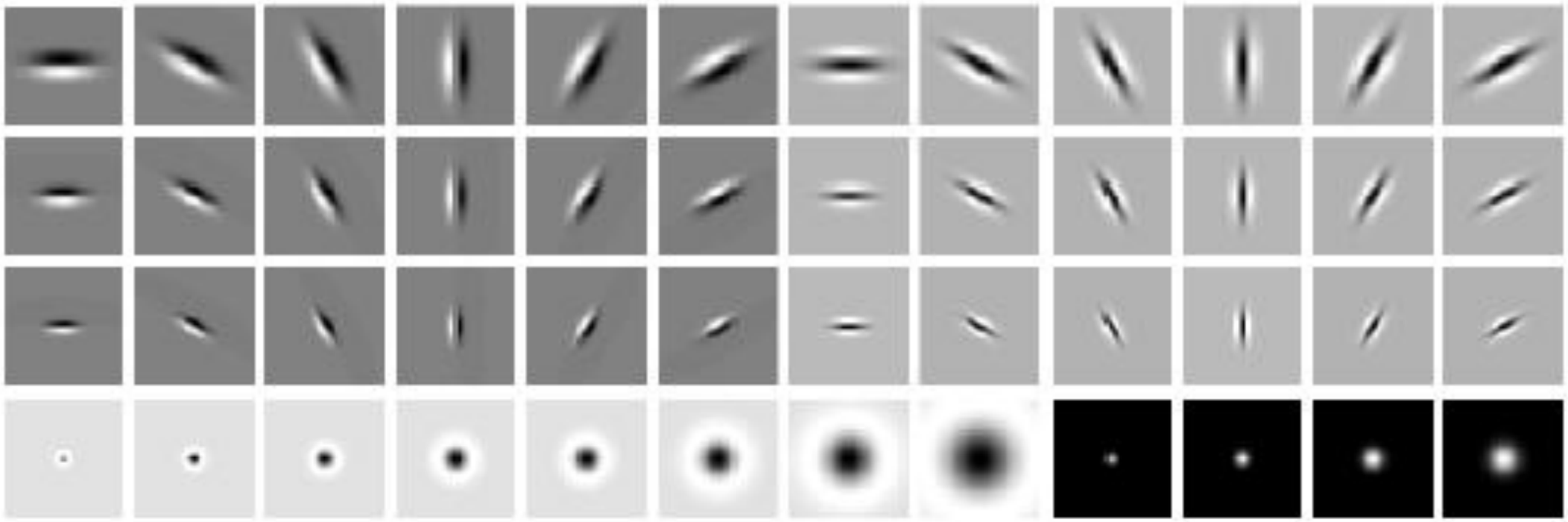


© david dmartin (Boston College)

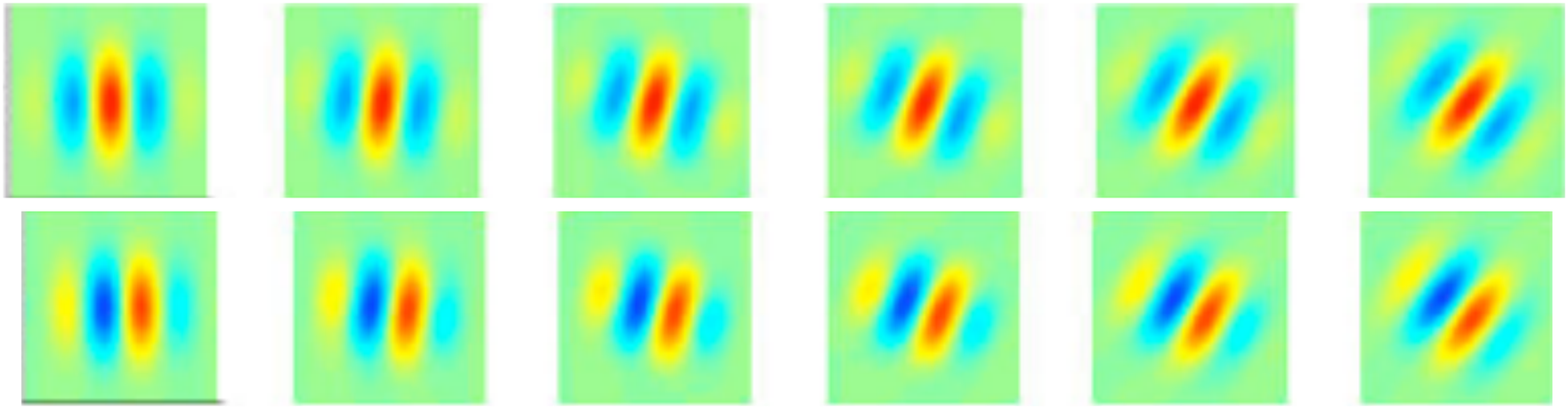
Outline

- Convolution
  - Linear Shift Invariance (LSI)
  - Convolution Properties (commutative, associative, distributive)
  - Normalize Cross-Correlation
- Edges
  - Canny edges (hysterisis)
  - derivative-of-gaussians (DoG)
  - laplacian-of-Gaussians (LoG)
  - filter banks
- Efficiency
  - pyramids
  - **steerability** (skip for now)
  - linear approximiations (SVD, separability)

Recall:
oriented filter banks



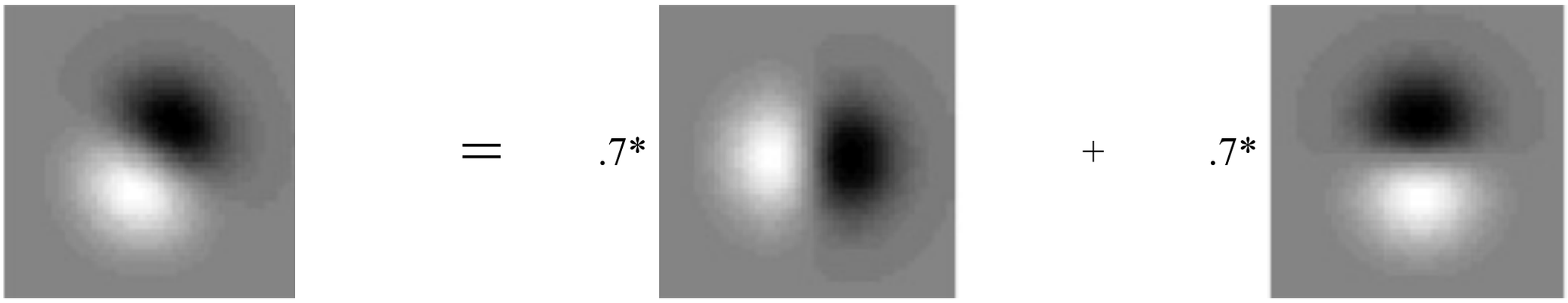
Leung Malik



Gabor filter bank

Remarkable fact

Some oriented filters can be written as linear combinations of a few “basis” filters



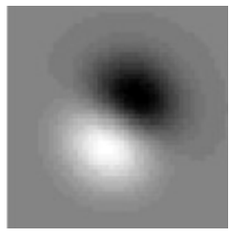
We’ll call such filters *steerable*

IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 13, NO. 9, SEPTEMBER 1991 891

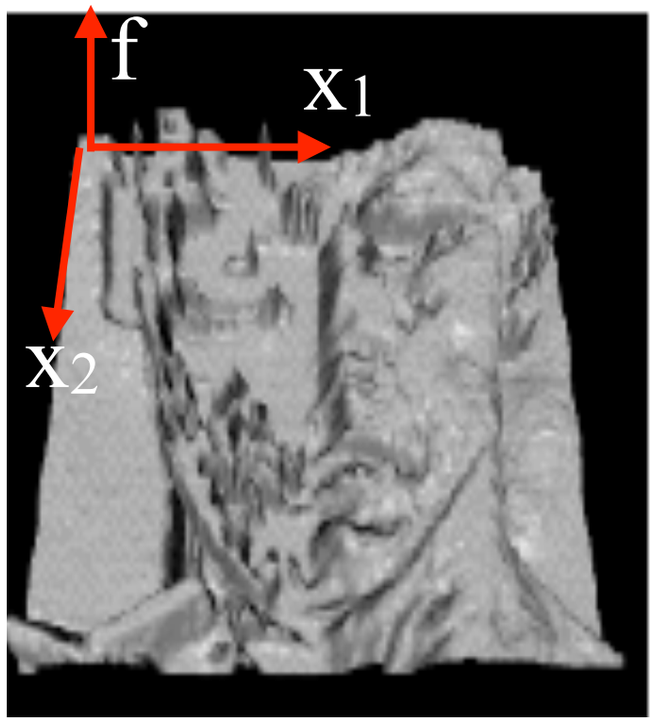
The Design and Use of Steerable Filters

William T. Freeman and Edward H. Adelson

Deriving steerability for derivative-of-guassian filters with a *directional* derivative



https://en.wikipedia.org/wiki/Directional_derivative

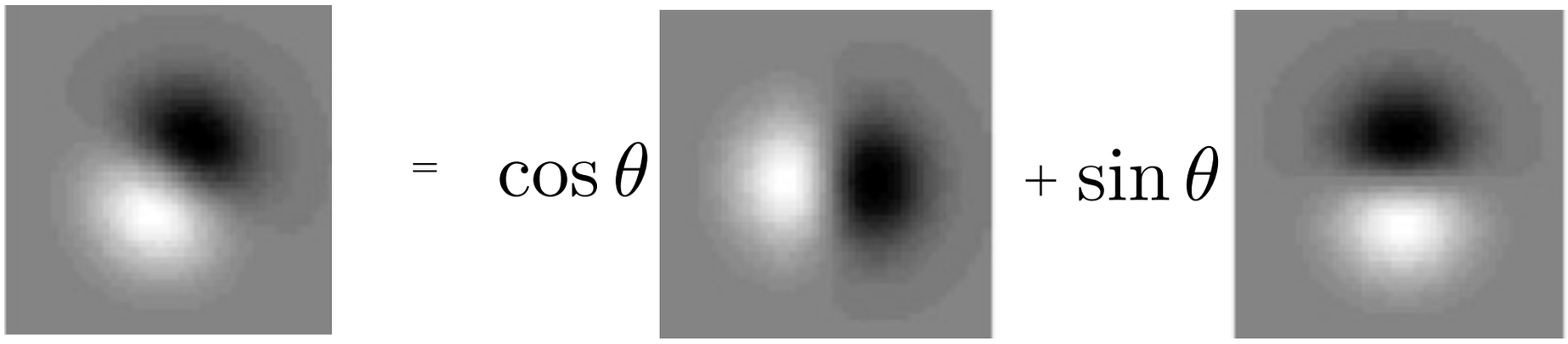


$$f(\mathbf{x}) \quad \text{where} \quad \mathbf{x} = (x_1, x_2), \mathbf{v} = (v_1, v_2)$$

$$\nabla_v f(\mathbf{x}) = \lim_{a \to 0} \frac{f(\mathbf{x} + a\mathbf{v}) - f(\mathbf{x})}{a} = \nabla f(\mathbf{x}) \cdot \mathbf{v}$$

Steering derivative-of-Gaussians

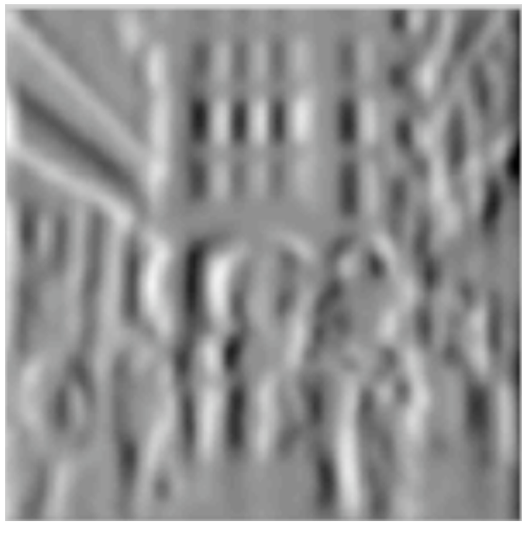
$$\nabla_\theta G_\sigma(x,y) = \cos\theta \frac{\partial G_\sigma}{\partial x} + \sin\theta \frac{\partial G_\sigma}{\partial y}$$



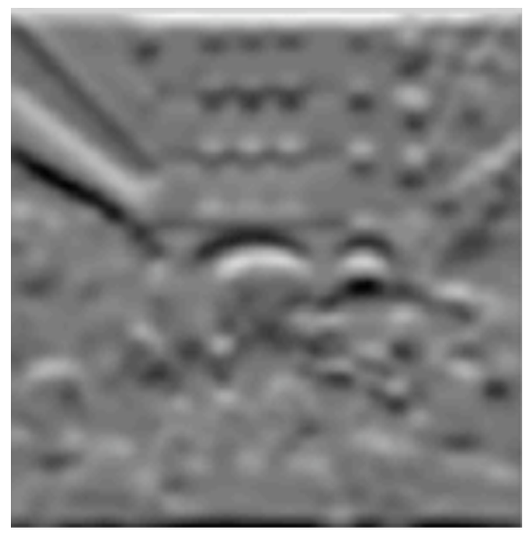
$= \cos\theta$

$+ \sin\theta$

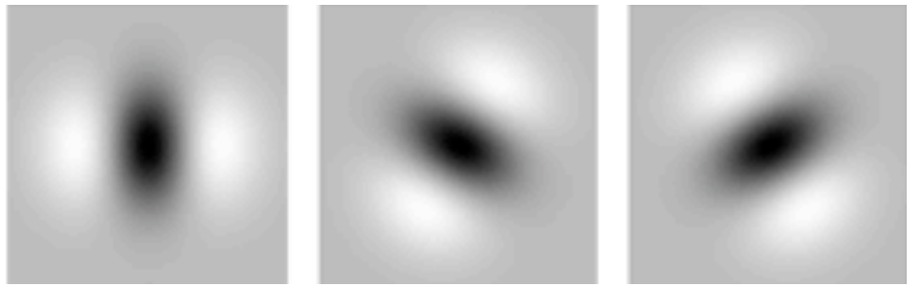
$= \cos\theta$



$+ \sin\theta$



Second case: Second-derivatives of Gaussians



$$G_\theta = k_a(\theta)G_a + k_b(\theta)G_b + k_c(\theta)G_c$$

G_{2a}	=	$0.9213(2x^2 - 1)e^{-(x^2+y^2)}$	$k_a(\theta)$	=	$\cos^2(\theta)$
G_{2b}	=	$1.843xye^{-(x^2+y^2)}$	$k_b(\theta)$	=	$-2\cos(\theta)\sin(\theta)$
G_{2c}	=	$0.9213(2y^2 - 1)e^{-(x^2+y^2)}$	$k_c(\theta)$	=	$\sin^2(\theta)$

When is this possible? Filters must be “smooth” in orientation space

Separability

Image of size N^2
Filter of size M^2

Complexity of filtering? (let's stick with correlation for now)

$O(N^2M^2)$

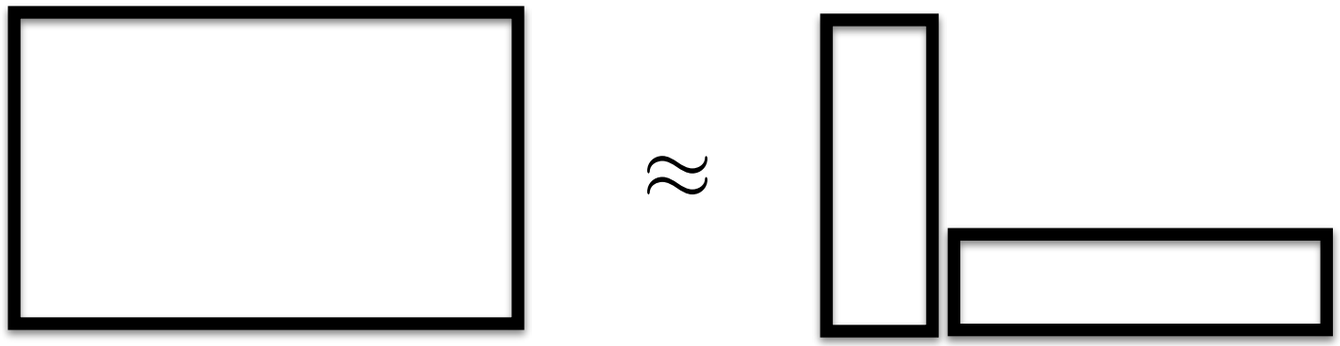
$$H[u,v] = H_x[u]H_y[v]$$

$$G[i,j] = \sum_u \sum_v H[u,v]F[i+u, j+v]$$

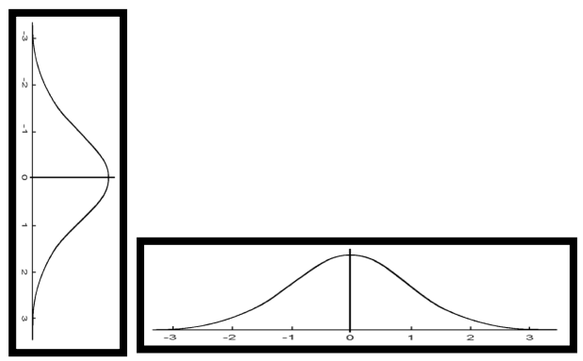
Separability

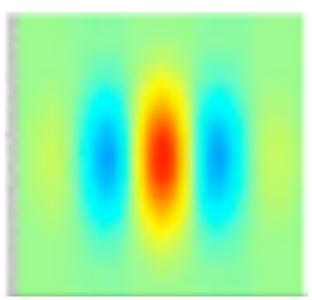
Given a filter, how can we come up with a good separable approximation?

$$H[u, v] \approx H_x[u] H_y[v]$$

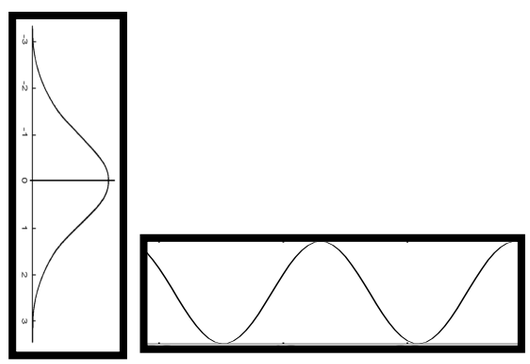


=





=



Linear algebra digression

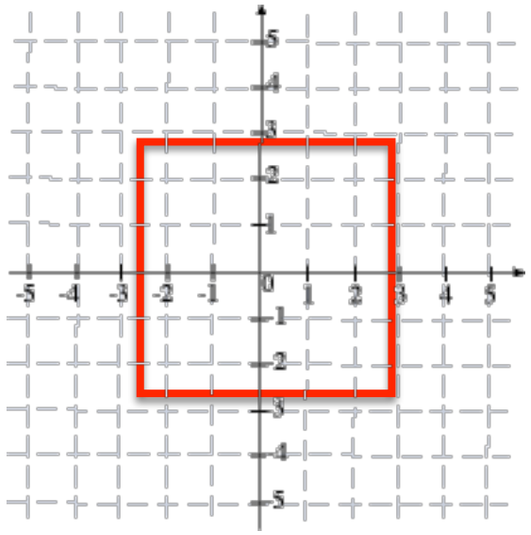
https://www.cs.cmu.edu/~zkolter/course/linalg/linalg_notes.pdf

My own writeup (in lec gdrive folder):

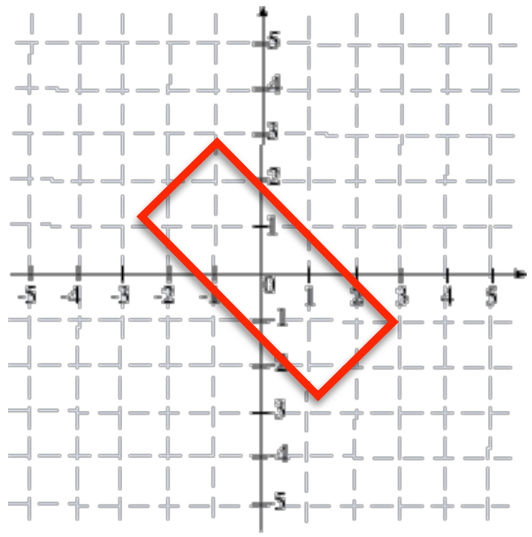
https://drive.google.com/file/d/1c-kjMqIQEKJCHyfMgLzdxRJNReOu6ssk/view?usp=drive_link

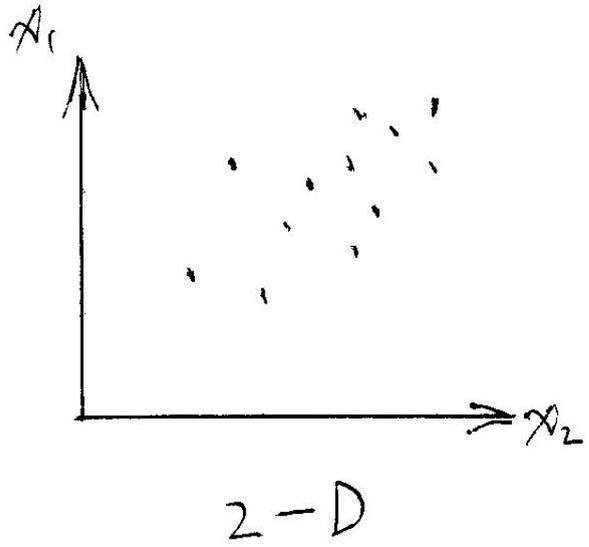
Linear algebra digression

Any matrix can be thought of as a *transformation*

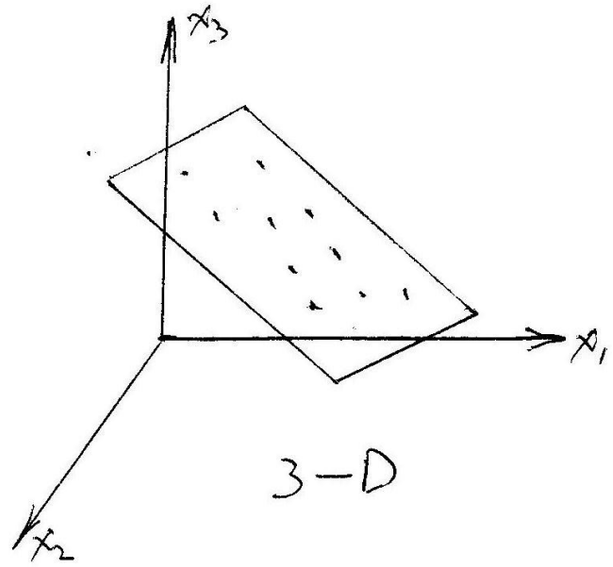


$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$





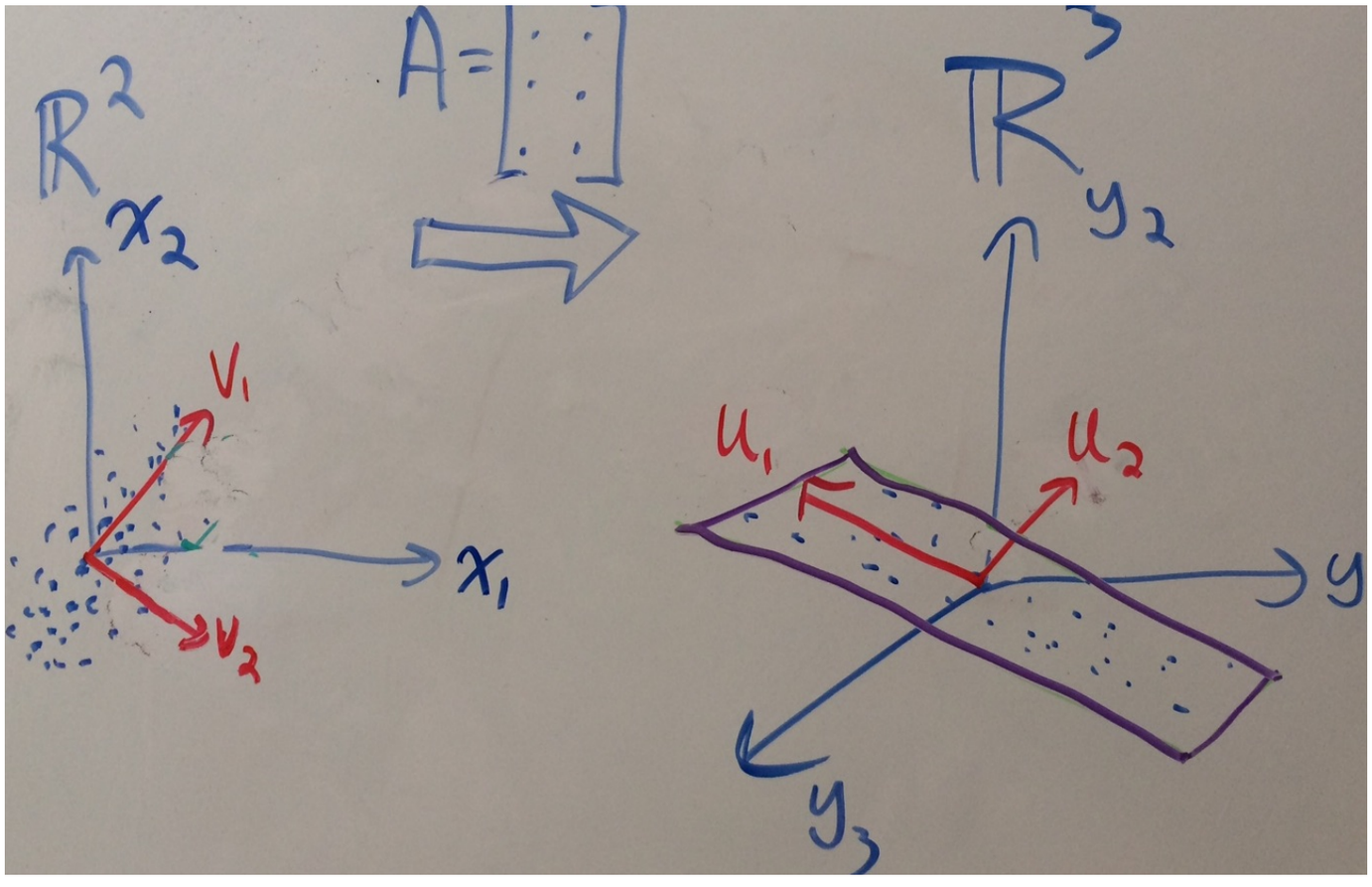
$$A = \begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix}$$



Change of basis

See handout

Recall: SVD

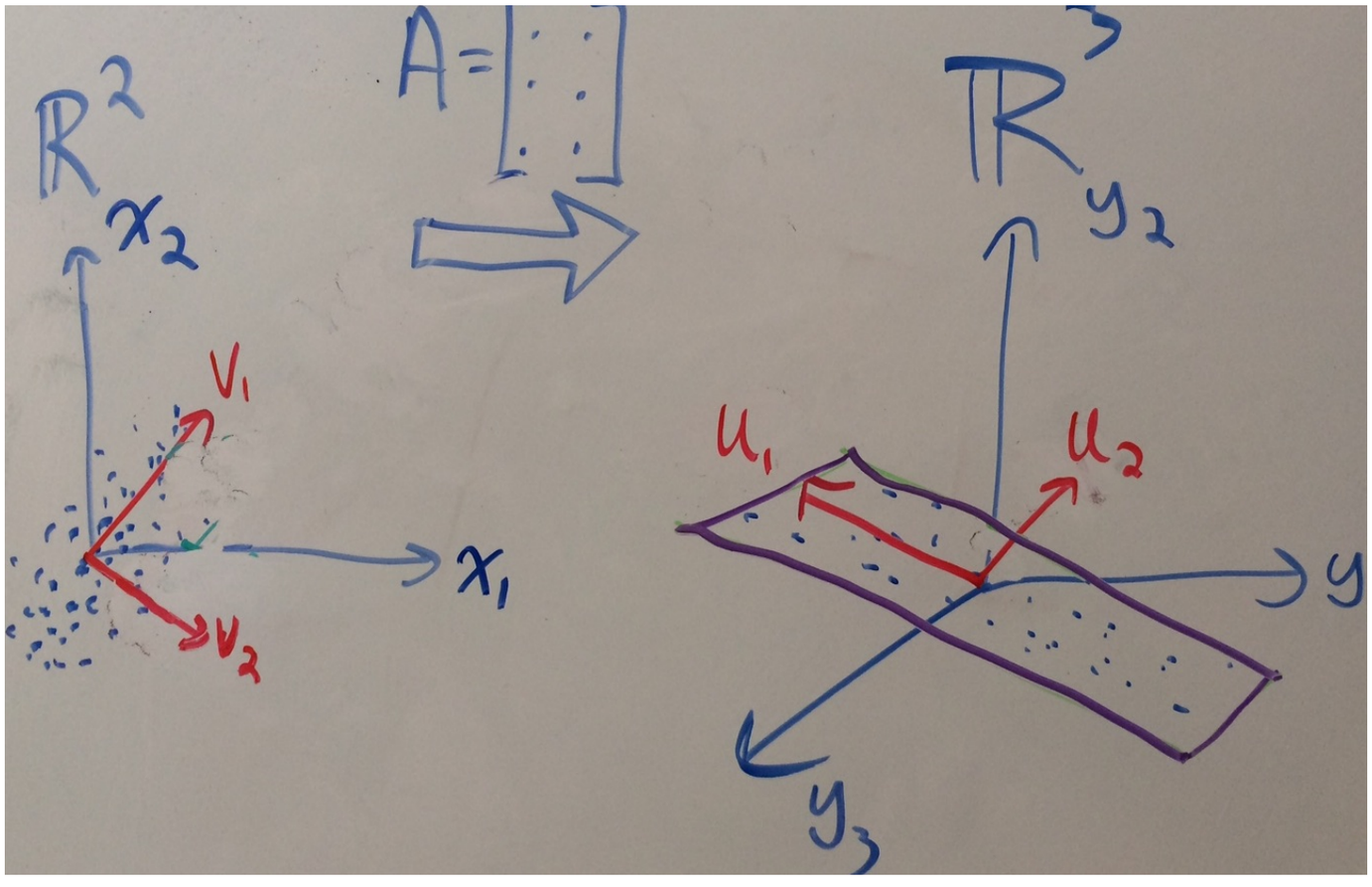


$$y = Ax$$

$$y = U\Sigma V^T x$$

Recall: SVD

$$y = U\Sigma V^T x$$



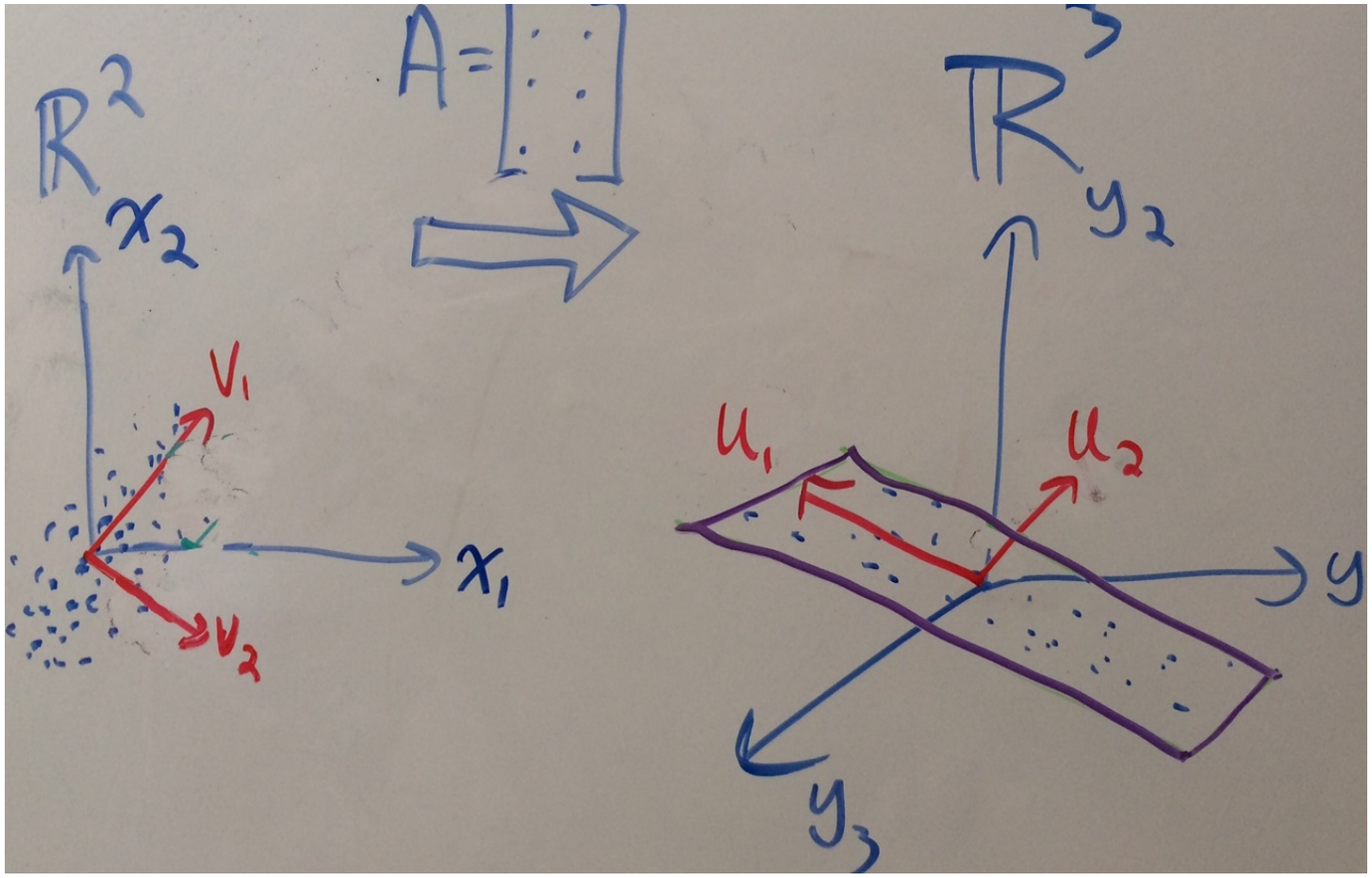
$$\Sigma = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \\ 0 & 0 \end{bmatrix}$$

$$U = \begin{bmatrix} \mathbf{u_1} & \mathbf{u_2} & \mathbf{u_3} \end{bmatrix} \quad U^T U = I$$

$$V = \begin{bmatrix} \mathbf{v_1} & \mathbf{v_2} \end{bmatrix} \quad V^T V = I$$

Recall: SVD

$$y = U\Sigma V^T x$$



Notation: $\mathbf{u}_i$ = left singular vector, σ_i = singular values, $\mathbf{v}_i$ = right singular vectors

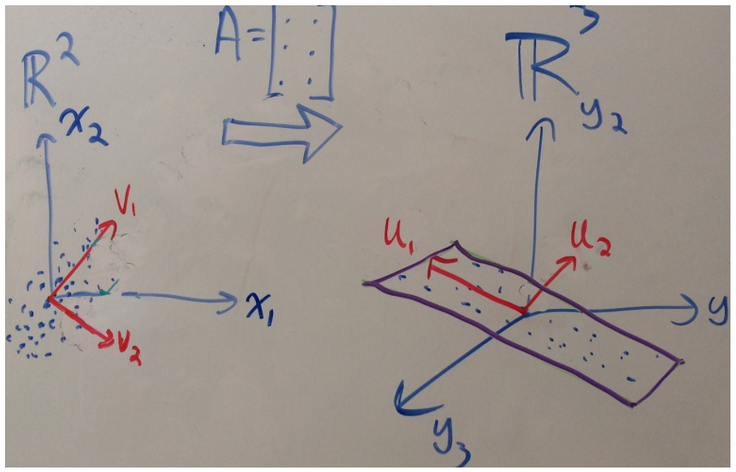
Any linear operator can be thought of as mapping from R^m to R^n

1. projection (with right singular vectors)
2. scaling (with singular values)
3. reconstruction (with left singular vectors)

```
# Compute "narrow" SVD with min(m,n) singular values
U, s, Vh = np.linalg.svd(X, full_matrices=False)
print(U.shape, s.shape, Vh.shape)
```

Recall: SVD

Immediate consequences (by appealing to geometric intuition)



$$||A||_F = A(:)^T A(:)$$

Low rank:
$$\min_{A':rank(A')<=k} ||A - A'||_F = U(:, 1:k)\Sigma(1:k, 1:k)V(:, 1:k)^T$$

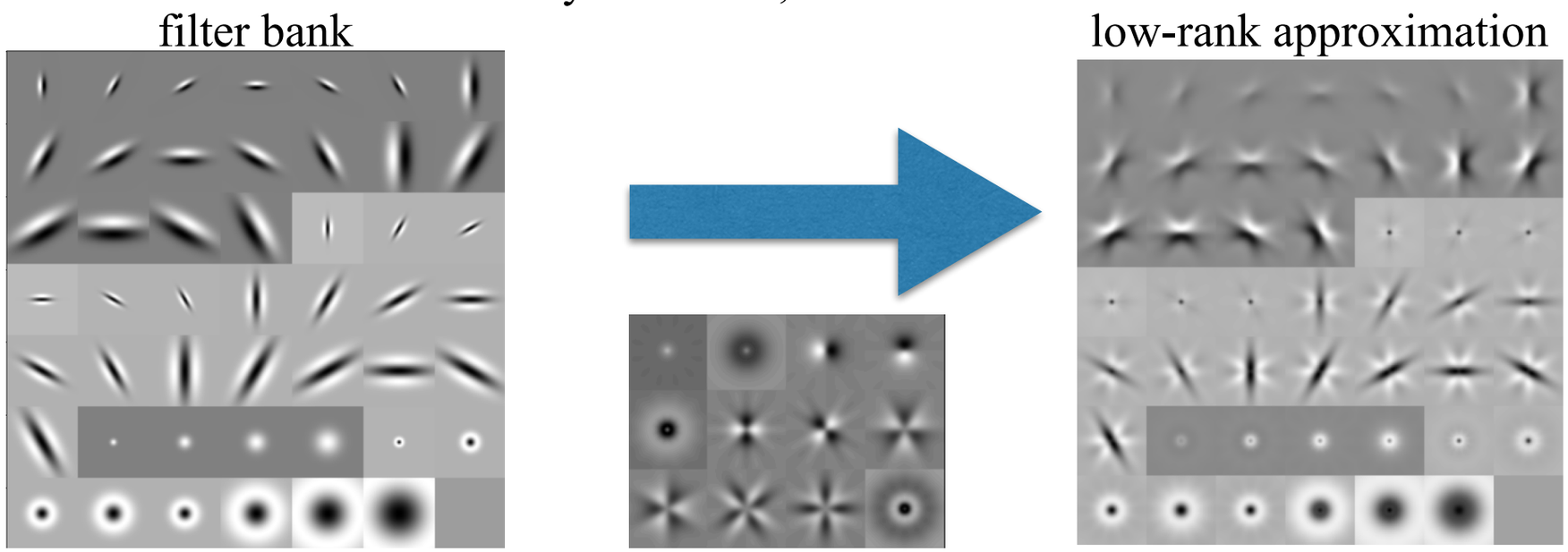
Homogenous least-squares:
$$\min_{h:h^T h=1} ||Ah||^2 = V(:, end)$$

Pseudoinverse:
$$A^+ = \underset{A^+}{\operatorname{argmin}} ||A^+ A - I||_F = V \begin{bmatrix} \frac{1}{\sigma_1} & 0 & 0 \dots \\ 0 & \frac{1}{\sigma_2} & 0 \dots \\ 0 & 0 & \frac{1}{\sigma_3} \dots \\ \vdots & \vdots & \vdots \end{bmatrix}^T U^T$$

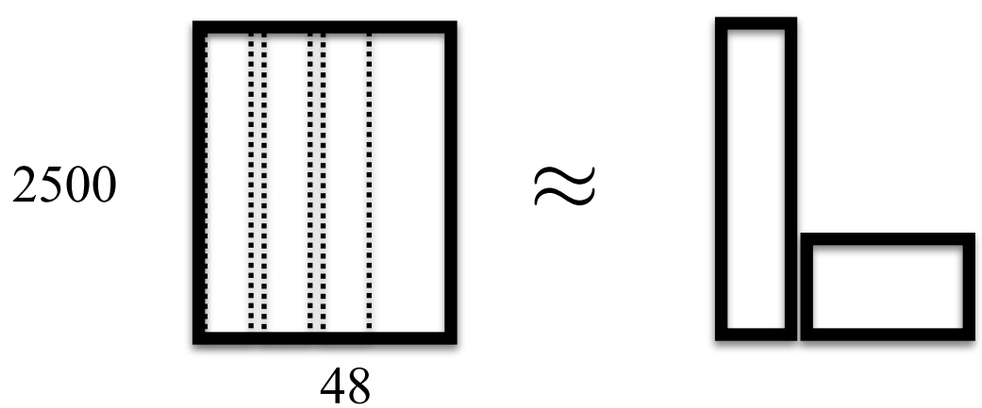
Least-squares method of steerability

demo_filter_bank.ipynb

Shy & Perona, CVPR94



- Column-vectorize each of 48 50x50 filters and stack'em into a 2500x48 matrix
- Apply SVD to generate low-rank approximation of 48 filters as linear combination of 12 *basis* filters



Least-squares method of steerability

Shy & Perona, CVPR94

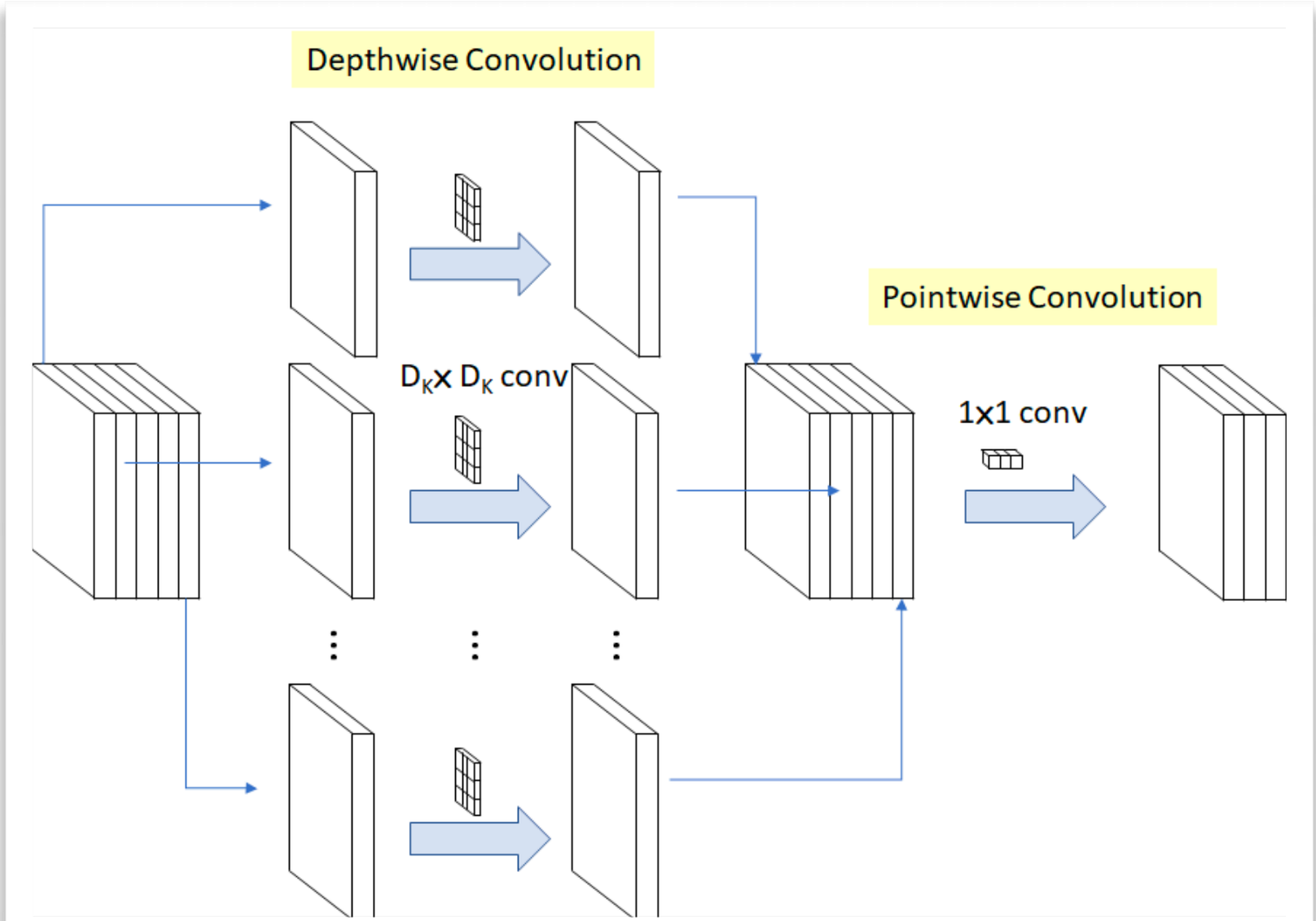
Rank 1 approximation

$$H[u, v, k] = H_s[u, v]c[k]$$

$$G[i, j, k] = \sum_u \sum_v H[u, v, k]F[i + u, j + v]$$

Low-rank separability is ubiquitous in neural networks

Depth-separable convolution in Google's MobileNet



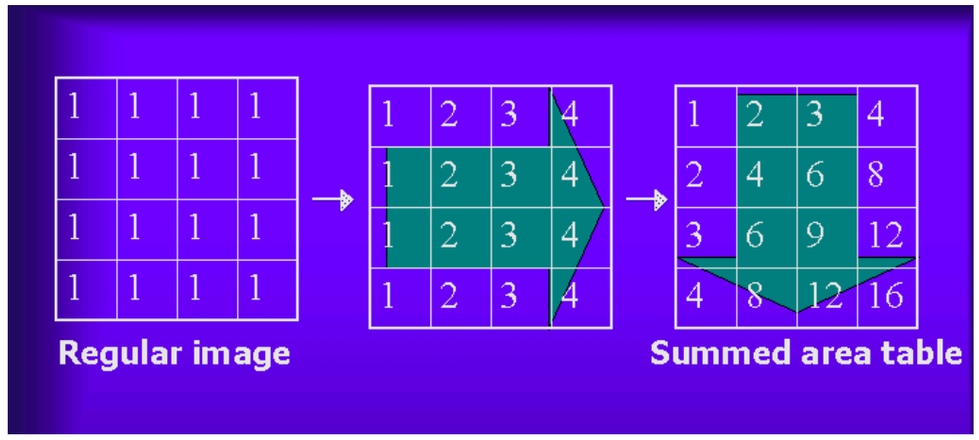
Final trick for efficient *box* filters

Consider a filter of all “1”’s (which sums up values in a neighborhood)

Turns out there is a way to implement filter that is *independant* of filter size

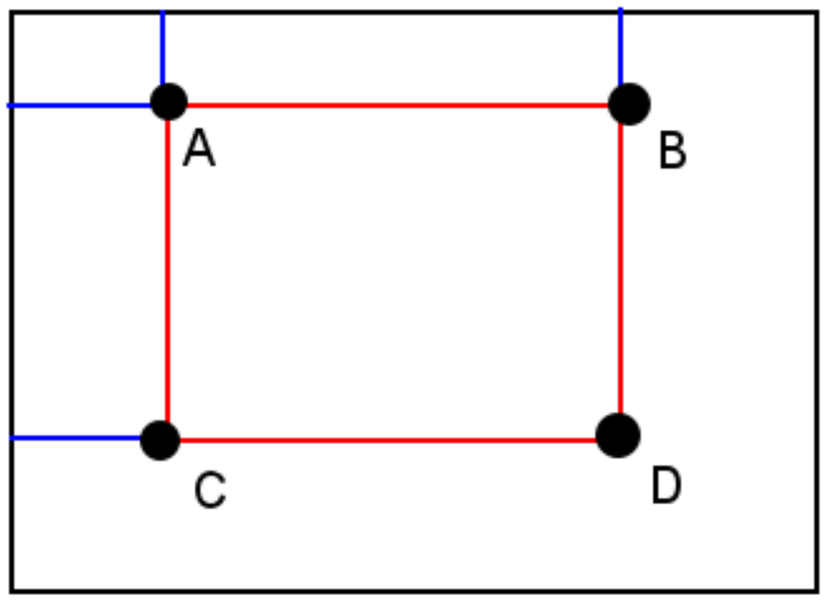
Box filtering with integral images (or summed area tables)

http://en.wikipedia.org/wiki/Summed_area_table



$$I(x,y) = \sum_{\substack{x' \le x \\ y' \le y}} i(x',y')$$

$$I(x,y) = i(x,y) + I(x-1,y) + I(x,y-1) - I(x-1,y-1)$$



Sum = D - B - C + A

Compute filter response for any size window with 4 table lookups (in integral image)!

Reduces $O(N^2M^2)$ to $O(N^2)$

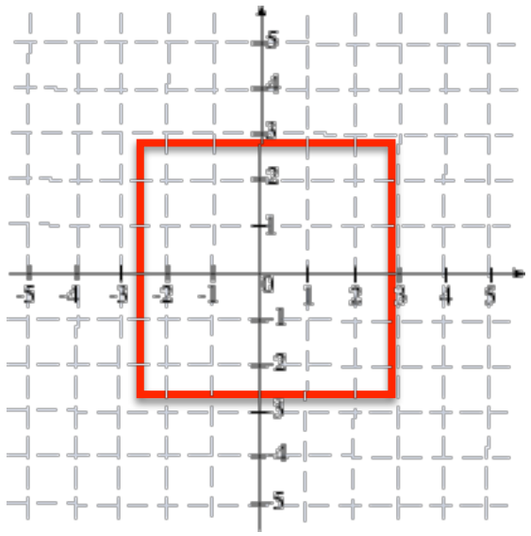
Efficient detection

P. Viola and M. Jones. *Rapid object detection using a boosted cascade of simple features*. CVPR 2001.

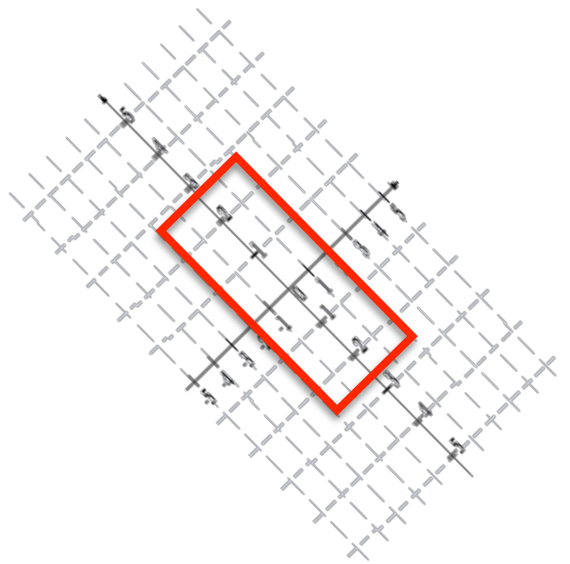
A look back

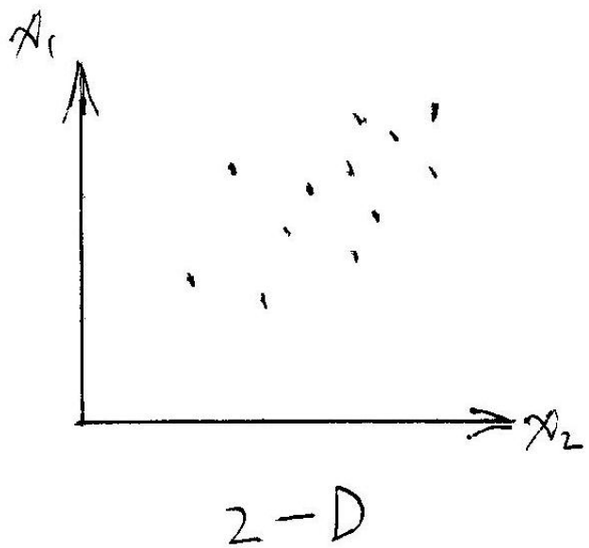
- Convolution (linear-shift-invariant)
- Edges (Canny, hysteresis, LoG)
- Efficiency (pyramids, separability, steerability)
- Next class: image warping

Any matrix can/should be thought of as a *transformation*

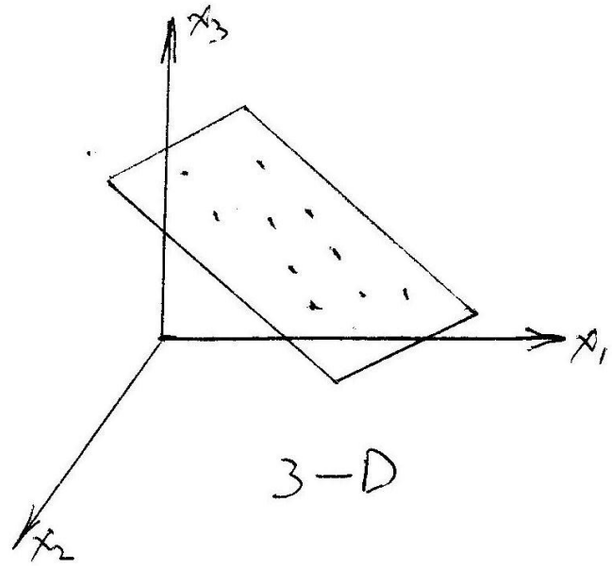


$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

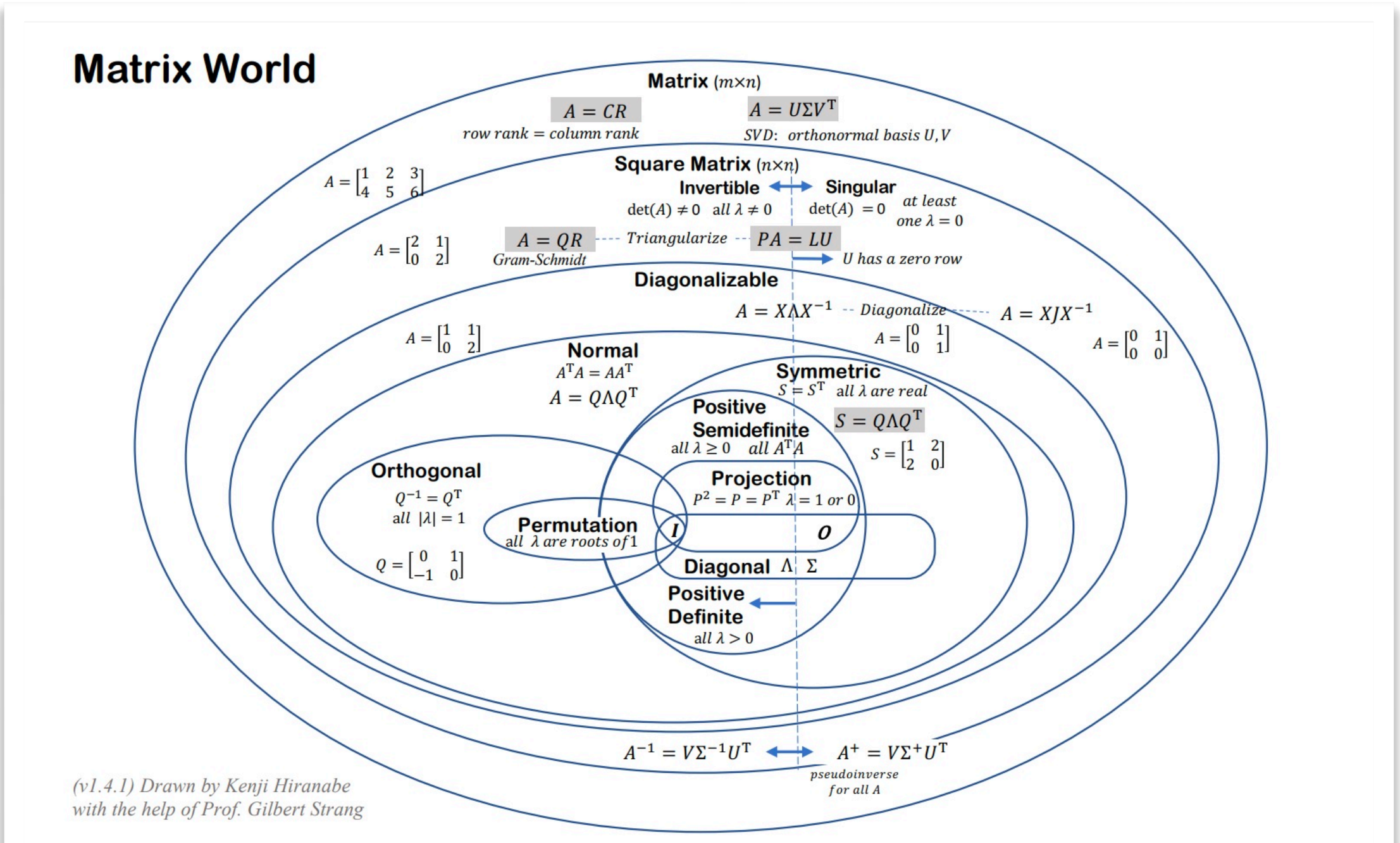




$$A = \begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix}$$

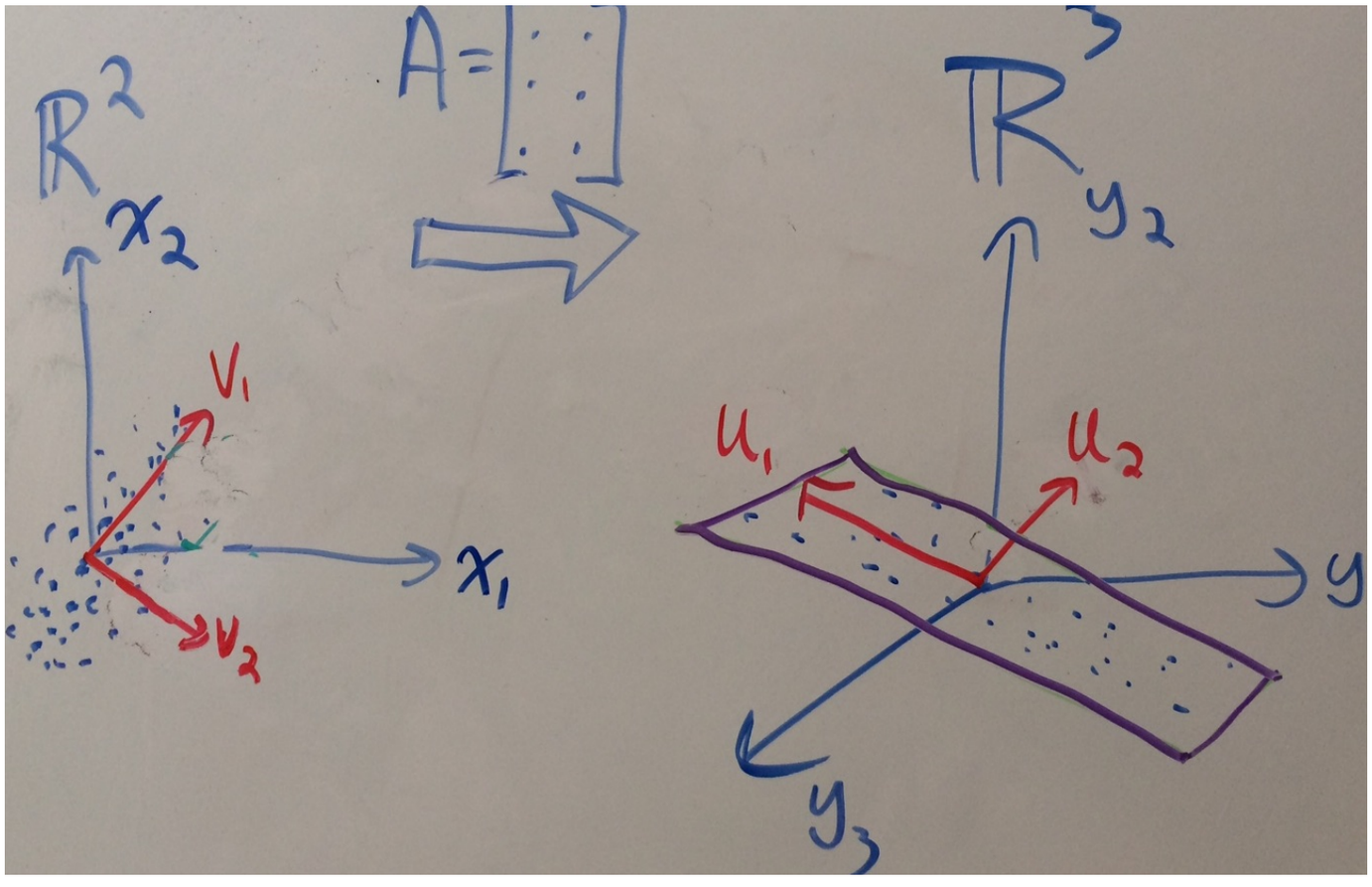


“All” of linear algebra



Recall: SVD

See svd.pdf writeup in lecture directory



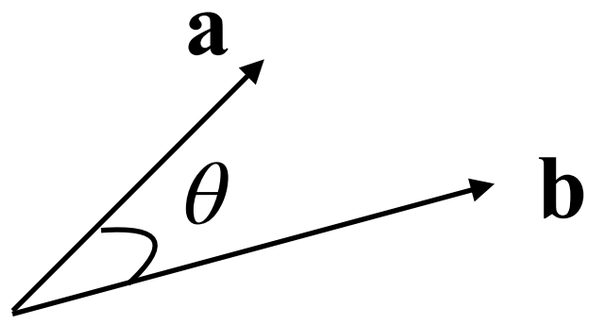
$$y = Ax$$

$$y = U\Sigma V^T x$$

Background

https://en.wikipedia.org/wiki/Dot_product
https://en.wikipedia.org/wiki/Orthonormal_basis

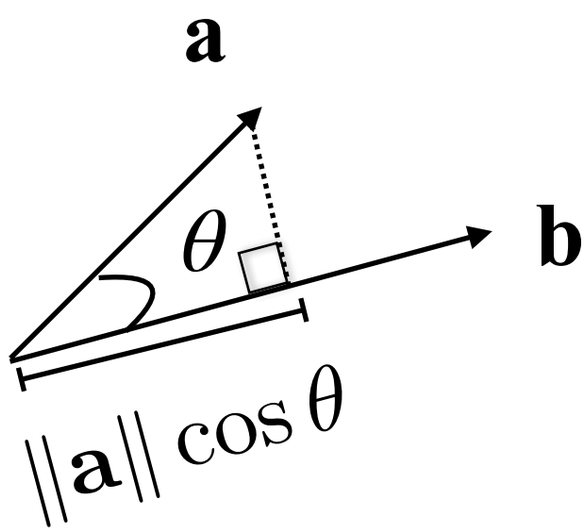
Dot product:



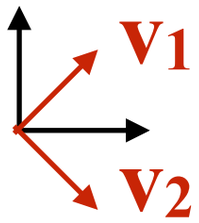
$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \|\mathbf{b}\| \cos\theta$$

$$||\mathbf{a}|| = \sqrt{\mathbf{a}^T \mathbf{a}}$$

Scalar projection:

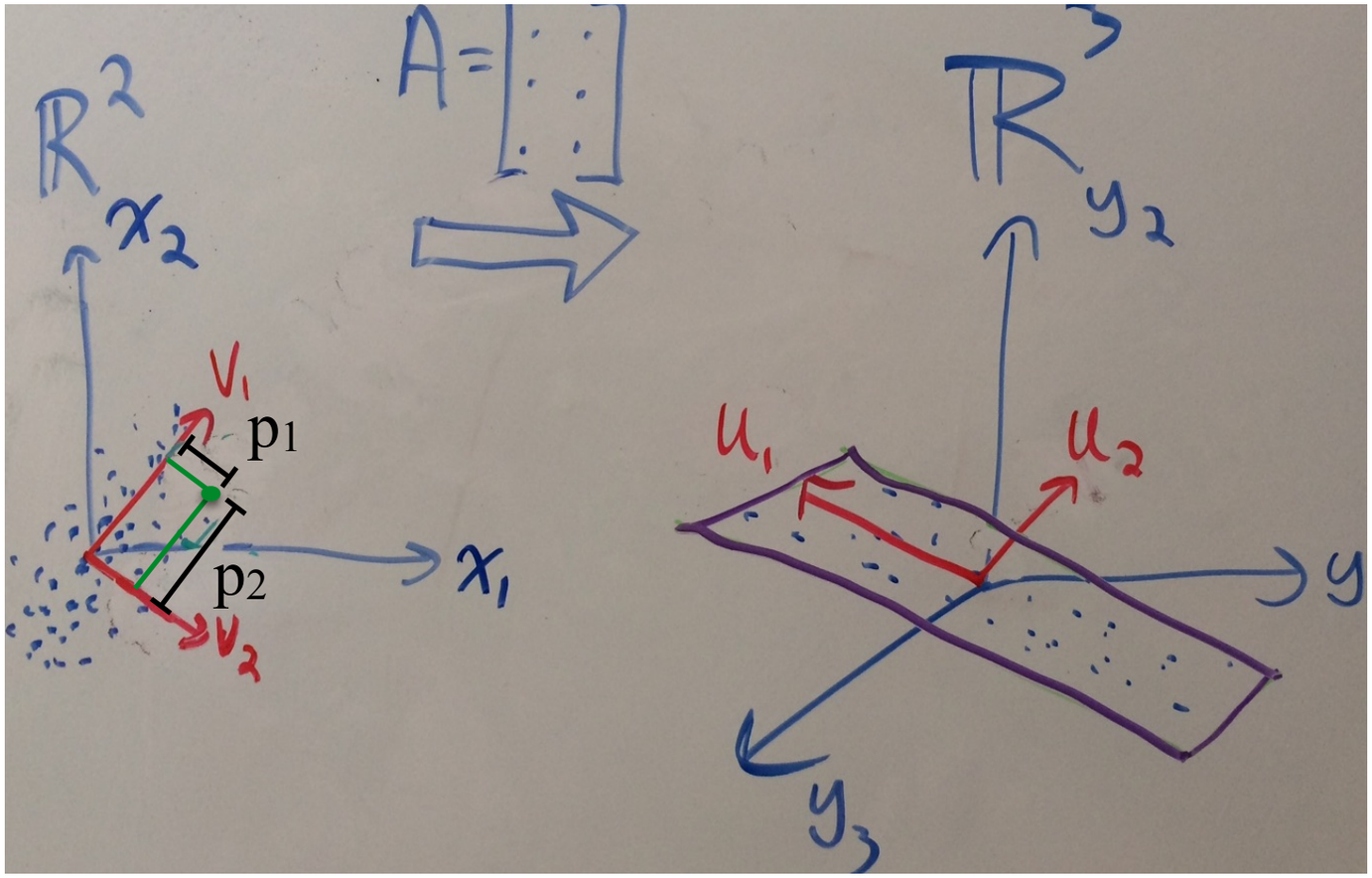


Orthonormal Basis:



$$V = \begin{bmatrix} \mathbf{v_1} & \mathbf{v_2}, \ldots \mathbf{v_n} \end{bmatrix} \qquad \mathbf{v}_i \in R^n$$

$$V^T V = I_{n \times n}$$



$$\mathbf{y} = A\mathbf{x}, \quad \mathbf{y} = U\Sigma V^T\mathbf{x}$$

Let $\mathbf{v_1}, \mathbf{v_2}, \ldots \mathbf{v_n}$ be an orthonormal basis for input space:

$$V = \begin{bmatrix} \mathbf{v_1} & \mathbf{v_2}, \ldots \mathbf{v_n} \end{bmatrix}. \qquad V^T V =$$

Write the *coordinates* of the *projection* of x into this basis:

$$\mathbf{p} =$$

Scale each coordinate:

$$c_i = \sigma_i p_i \qquad \Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 \ldots \\ 0 & \sigma_2 & 0 \ldots \\ 0 & 0 & \sigma_3 \ldots \\ \vdots & \vdots & \vdots \end{bmatrix} \qquad \mathbf{c} =$$

Let $\mathbf{u_1}, \mathbf{u_2}, \ldots \mathbf{u_m}$ be an orthonormal basis for output space:

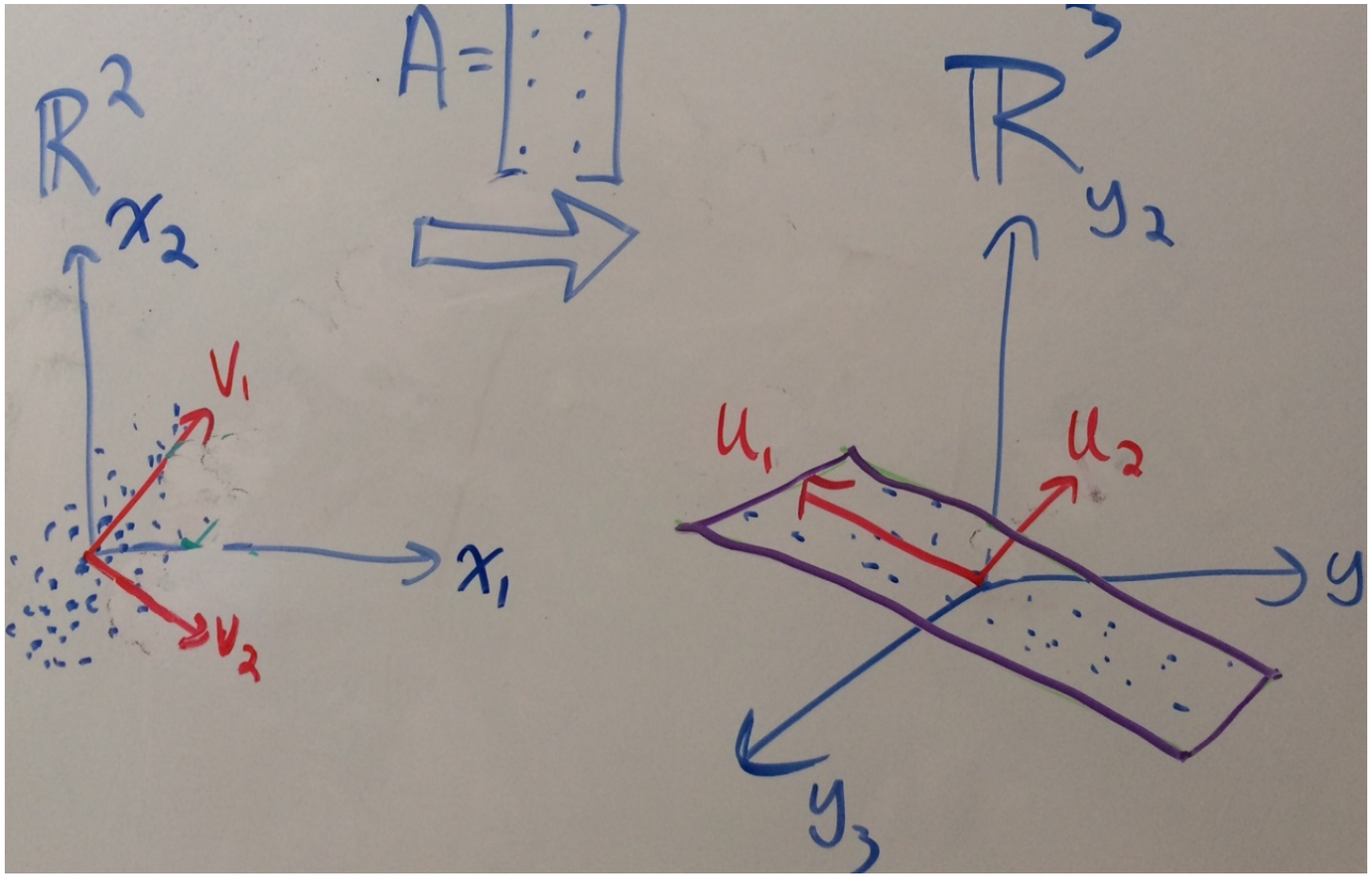
$$U = \begin{bmatrix} \mathbf{u_1} & \mathbf{u_2} \ldots \mathbf{u_m} \end{bmatrix}$$

Reconstruct point in output space with scaled orthonormal basis:

$$\mathbf{y} = c_1\mathbf{u_1} + c_2\mathbf{u_2} \ldots =$$

Recall: SVD

$$y = U\Sigma V^T x$$



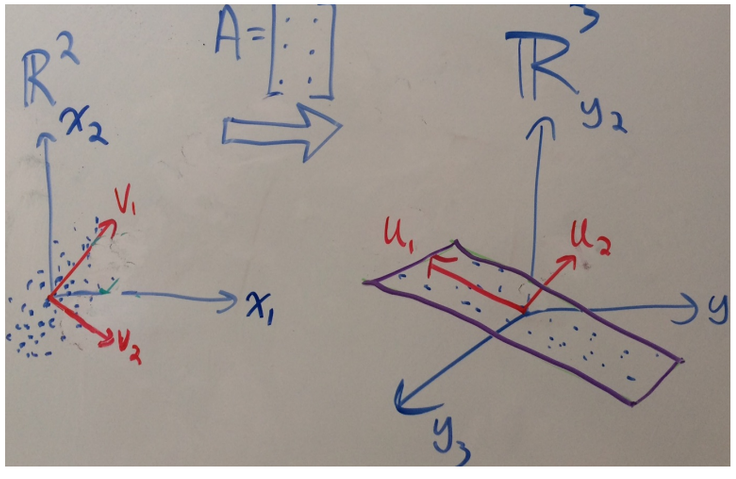
Notation: $\mathbf{u}_i$ = left singular vector, σ_i = singular values, $\mathbf{v}_i$ = right singular vectors

Any linear operator can be thought of as mapping from R^m to R^n

1. projection (with right singular vectors)
2. scaling (with singular values)
3. reconstruction (with left singular vectors)

Recall: SVD

Immediate consequences (by appealing to geometric intuition)



$$||A||_F = \sum A[i,j]^2$$

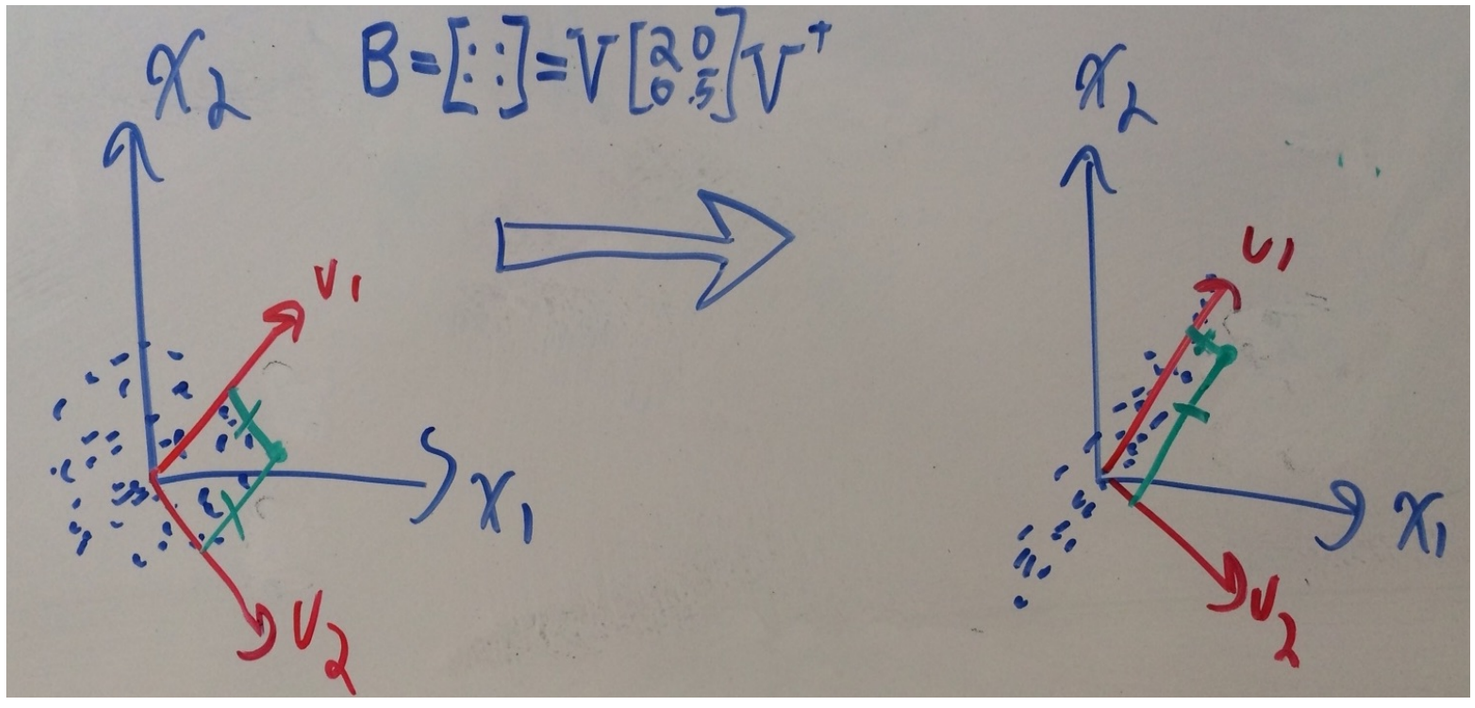
Low rank: $$\min_{A':rank(A')<=k} ||A - A'||_F =$$

Homogenous least-squares: $$\min_{\mathbf{h}:\mathbf{h^T h}=1} ||A\mathbf{h}||^2 =$$

Pseudoinverse: $$A^+ = \operatorname{argmin}_{A^+} ||A^+ A - I||_F =$$

Extension: positive-semidefinite (PSD) matrices

Let's construct a square matrix $B = A^T A$



Any PSD matrix can be thought of as mapping from R^n to R^n

1. projection (with eigenvectors)
2. scaling (with eigenvalues),
3. reconstruction (with *same* eigenvectors)

Special case: V = identity => B = scaling

For general V, B = scaling in rotated coordinate system

Question: how does the SVD of $A = U\Sigma V^T$ relate to the eigenvectors of B?

$$B = V\Sigma^2 V^T$$

Jupyter Notebook Demos

Application efficient filters

Application 1: separable filters

Image of size N^2
Filter of size M^2

Complexity of filtering (let's stick with correlation for now)?

$O(N^2M^2)$

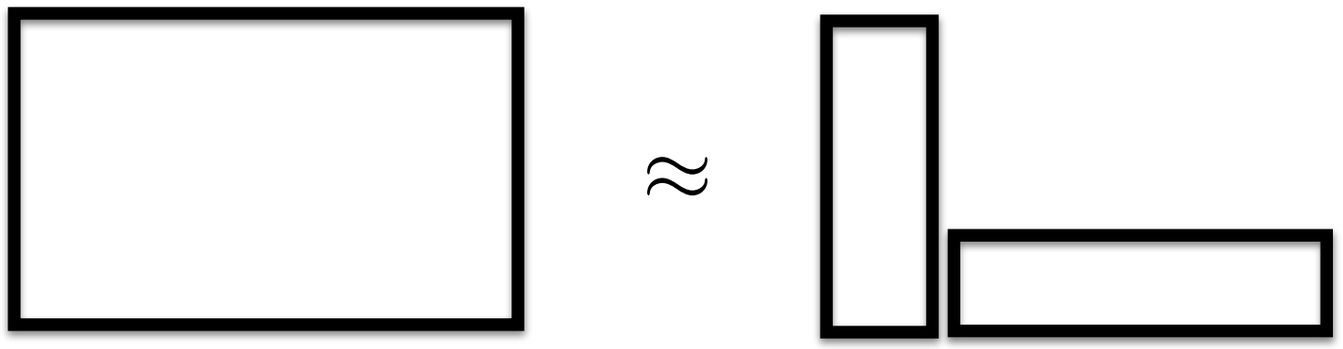
$$H[u,v] = H_x[u]H_y[v]$$

$$G[i,j] = \sum_u \sum_v H[u,v]F[i+u, j+v]$$

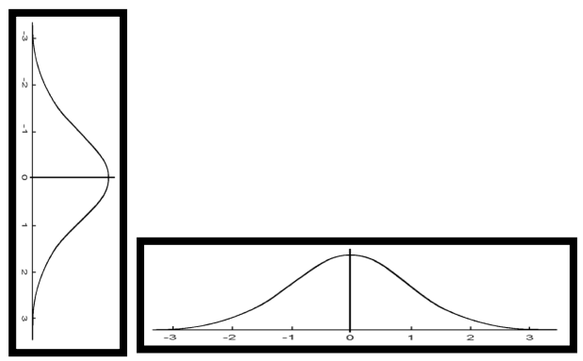
Separability

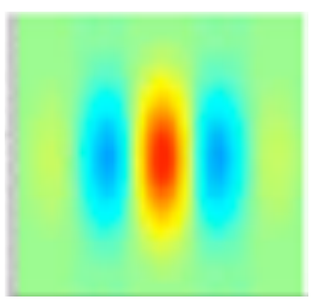
Given a filter, how can we come up with a good separable approximation?

$$H[u, v] \approx H_x[u] H_y[v]$$

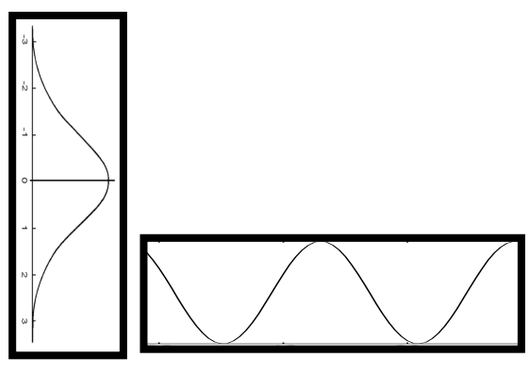


=



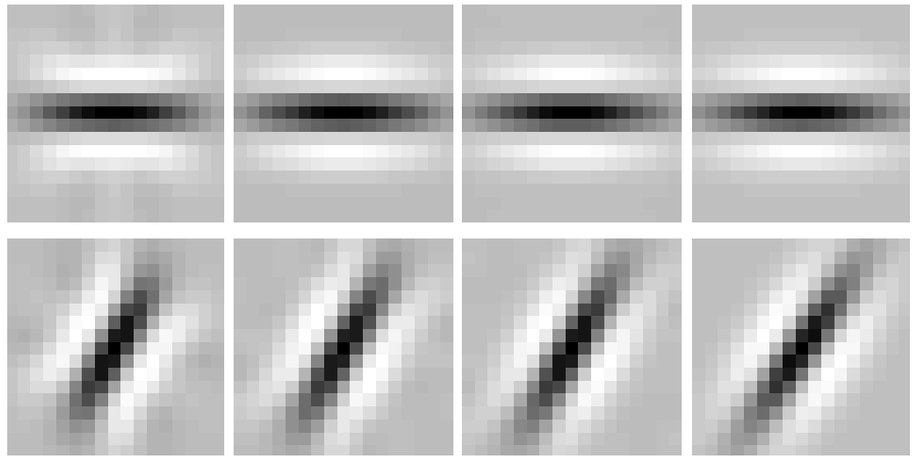


=

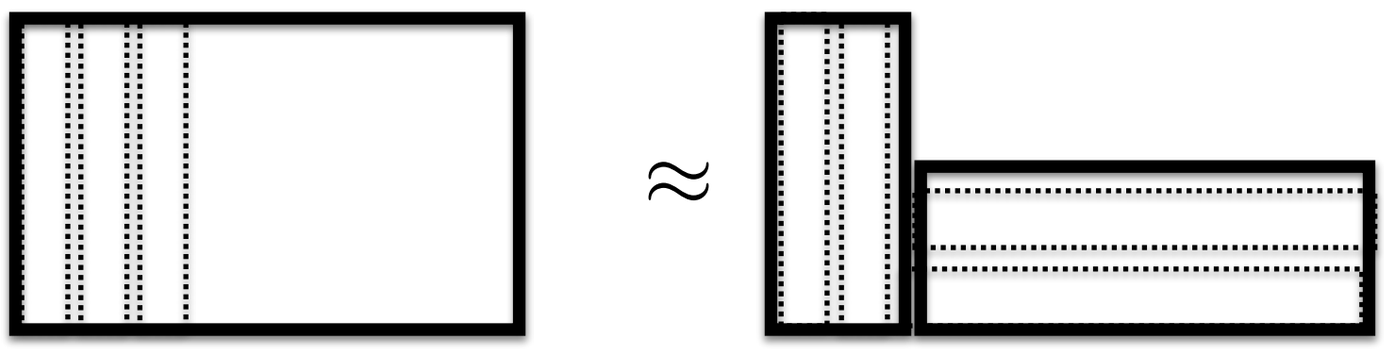


Application 2: eigenfilters

Shy & Perona, CVPR94



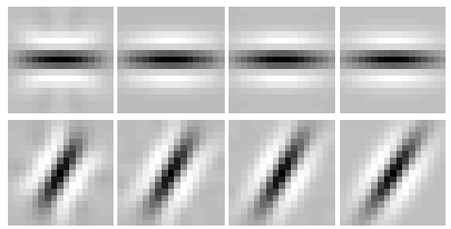
Vectorize each filter and stack each column into a matrix
Apply SVD to generate low-rank approximation



Least-squares method of steerability

Shy & Perona, CVPR94

Rank 1 approximation

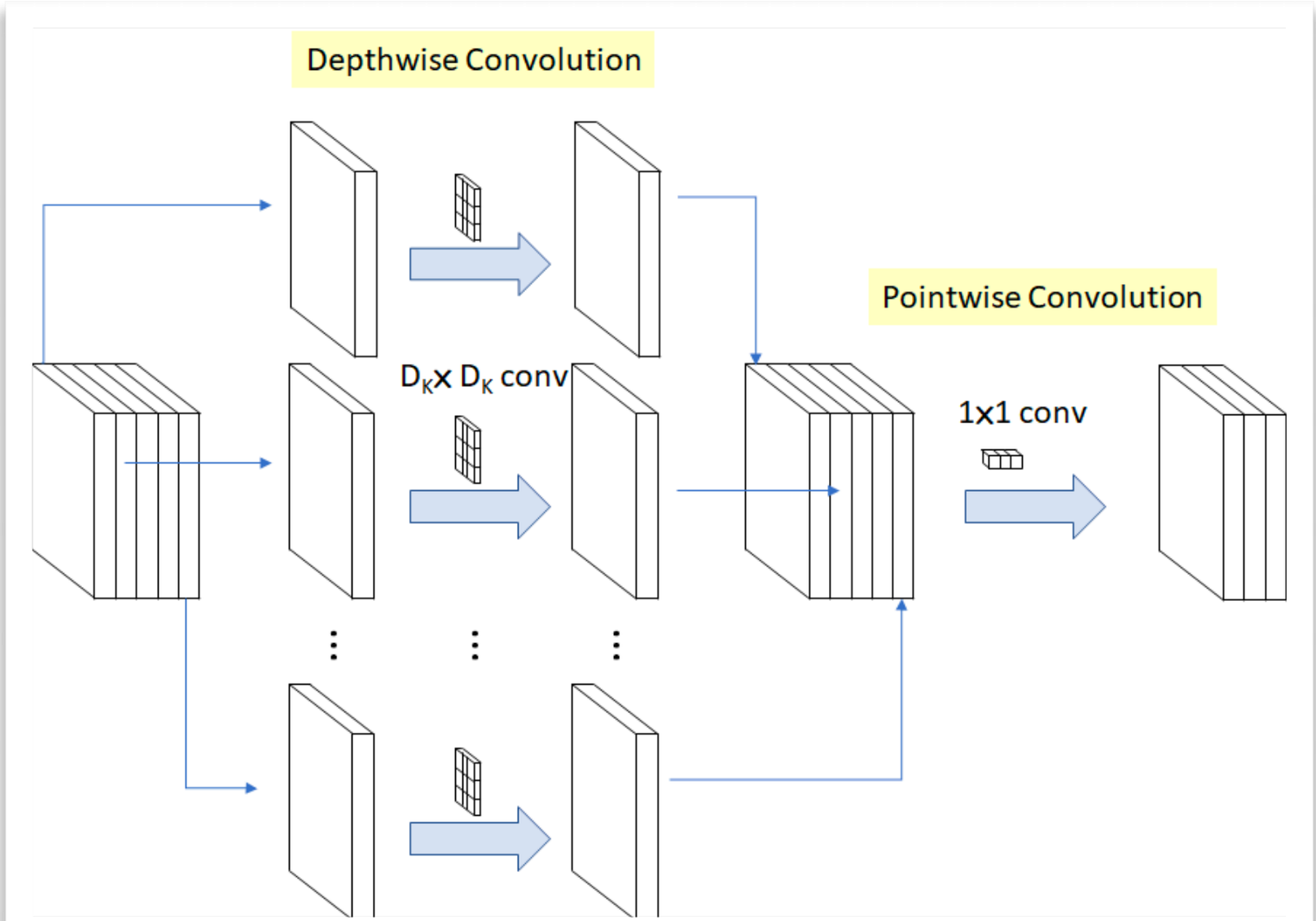


$$H[u,v,k] = H_s[u,v]c[k]$$

$$G[i,j,k] = \sum_u \sum_v H[u,v,k]F[i+u,j+v]$$

Low-rank separability is ubiquitous in neural networks

Depth-separable convolution in Google's MobileNet



Jupyter Notebook Demo (2)