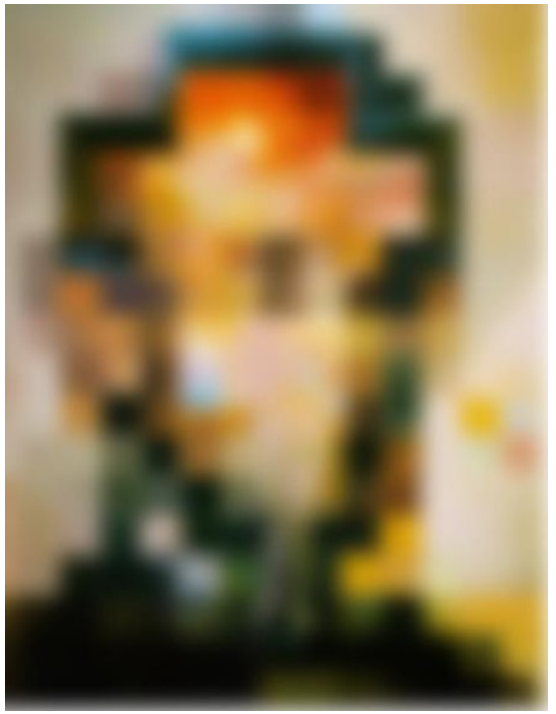
Computer Vision : Insight of object detection algorithms :

Lesson1 Image Preprocessing



By Daniel
Research Scientist
danielshaving@gmail.com

课程大纲：

阶段1 图像预处理

第一课：图像处理基础

知识点：图像处理，灰度值提取，Histogram处理，视觉和图像变换（Warp），图像卷积

第二课：图像处理进阶：图像滤波，特征提取及匹配

知识点：Harris角点检测，Sift，Surf，Orb，图像特征点匹配，边缘检测算法等(含所有带工具箱代码 和 纯numpy代码)

第三课：实践：利用KNN算法和OpenCV进行手写字符识别

阶段2: 创建自己的图像识别神经网络

第四课：深入理解神经网络的前向传递和反向传播及其物理意义

知识点：Loss function，交叉熵代价函数，梯度下降法求导

第五课：训练你自己的网络，重点为调参和工作中用到的一些技巧

知识点：Loss function，交叉熵代价函数，梯度下降法

第六课：卷积神经网络（RNN）在图像分类识别中的应用（附python编程和算法解析）

知识点：数据输入层，卷积计算层，激励层（Sigmoid，Tanh，ReLu，ELU），池化层，全联接层，Batch Normalization，学习率

第七课：实践，不使用任何工具包，训练一个属于你自己的神经网络进行手写字符识别

系列3. 深度卷积神经网络进阶

第八课：不同的神经网络类别和应用

知识点：调参基本技巧，向量点积

第九课：深度卷积神经网络原理及实践

知识点，神经网络的迁移学习技巧

第十课：搭建图片搜索系统，深入理解Triplet Loss 及其训练技巧

第十一课：实践：使用Tensorflow/Keras搭建神经网络，进行图像分类

阶段4: 目标检测和LSTM标注法

第十二课： 目标检测算法

知识点：Fast RCNN , Faster RCNN, Yolo, SSD

第十三课：LSTM 标注学习

第十四课：实践：使用Tensorflow/Keras在数据集上进行目标检测

附加：Kaggle竞赛习题讲解

Python is a must prerequest

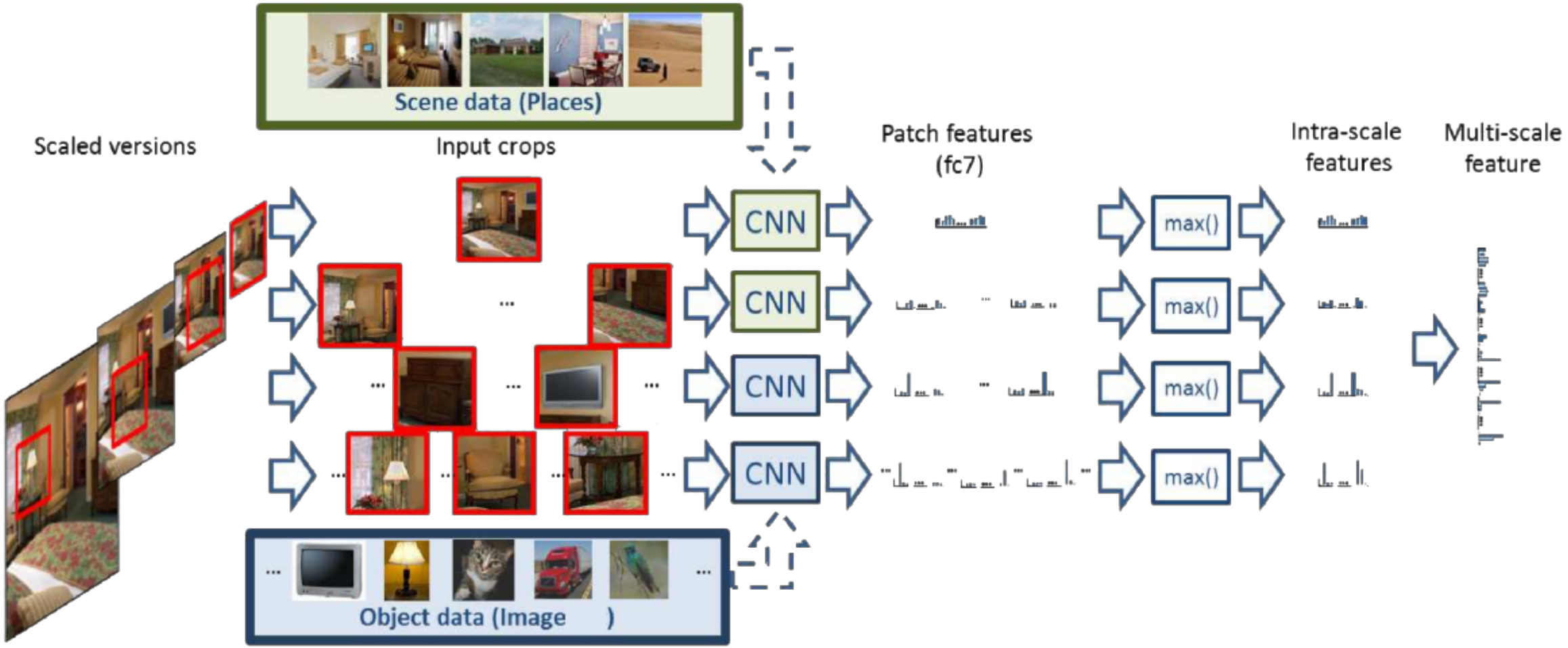
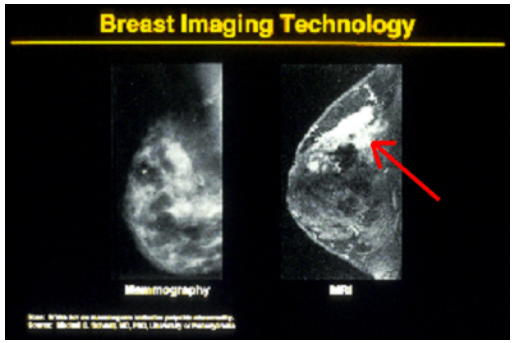

Scene data (Places)
Scaled versions
Input crops
CNN
CNN
CNN
CNN
Patch features
(fc7)
max()
max()
max()
max()
Intra-scale
features
Multi-scale
feature
Object data (Image)

Table of contents

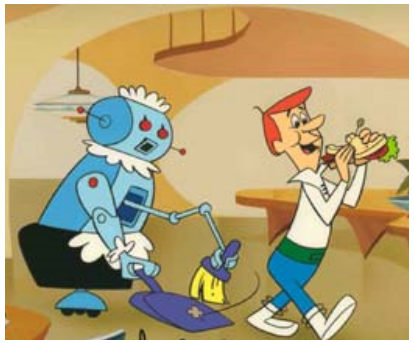
What Computer Vision matters

Safety



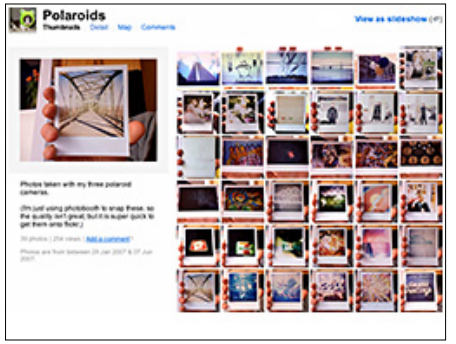
Health

Security



Comfort

Fun

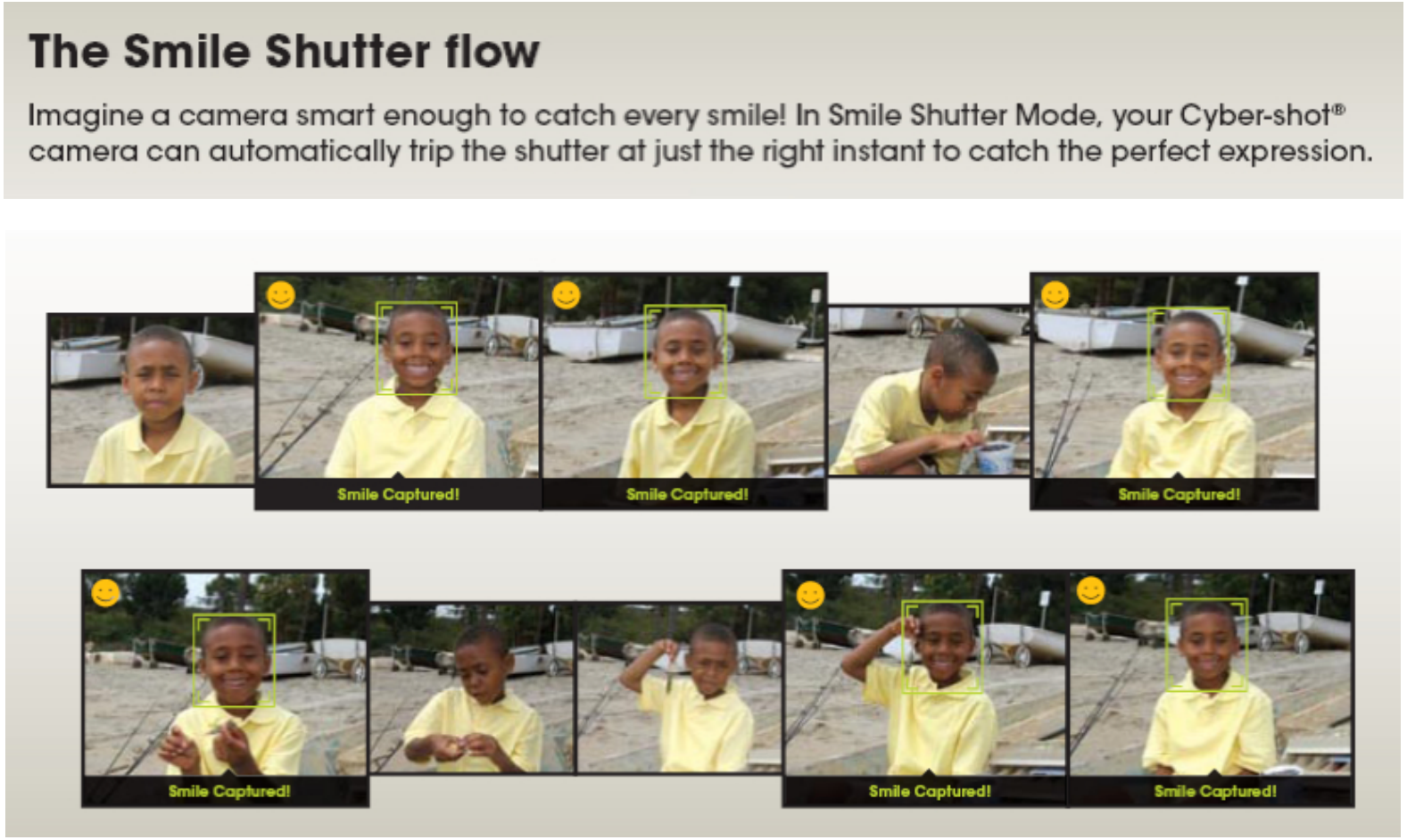


Access

CV applications

Detection	Recognition	Video Feature Processing	3D Modelling	Robotics	Auto-coding
Smile detection	Object Recognition	Target Recognition	Shape Capture	Medical	GAN
Face detection	Face Verification	Human Feature	Motion Capture	Space	...
	Biometrics	Tracing	Automobile	Sport	
	Picture search	Industry motion	AR/VR		
	OCR	SLAM	3D Reconstruction		

Smile detection （ FER ）



Algorithm Example:

ASM Feature
SVM or Bayesian Classification
…

Face detection

Algorithm Example:

Haar Feature
Cascade CNN
DMP Model

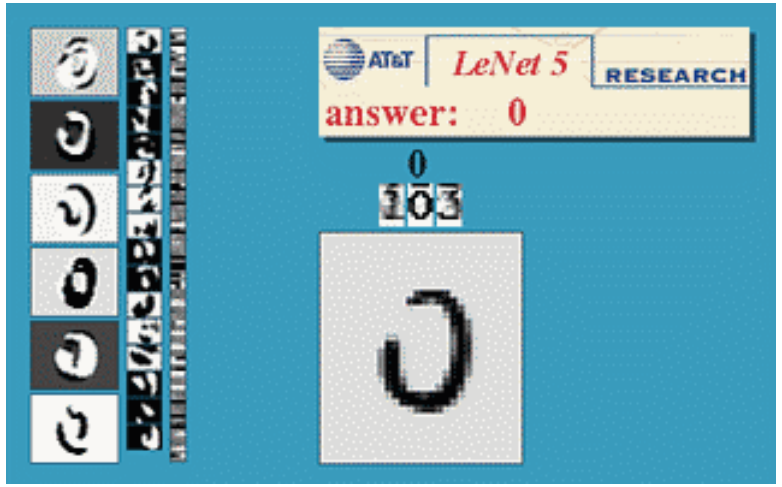
...

- Many new digital cameras and apps now detect faces

OCR

Technology to convert scanned docs to text

- If you have a scanner, it probably came with OCR software



Digit recognition, AT&T labs
http://www.research.att.com/~yann/

License plate readers
http://en.wikipedia.org/wiki/Automatic_number_plate_recognition

Algorithm Example:

Denoise
Fast RCNN / Yolo

…

Object recognition (in supermarkets)

Algorithm Example:

RCNN/Yolo/SSD

…

LaneHawk by EvolutionRobotics

“A smart camera is flush-mounted in the checkout lane, continuously watching for items. When an item is detected and recognized, the cashier verifies the quantity of items that were found under the basket, and continues to close the transaction. The item can remain under the basket, and with LaneHawk,you are assured to get paid for it… “

Object recognition (in Smartphone)

Point & Find, Nokia

Google Goggles

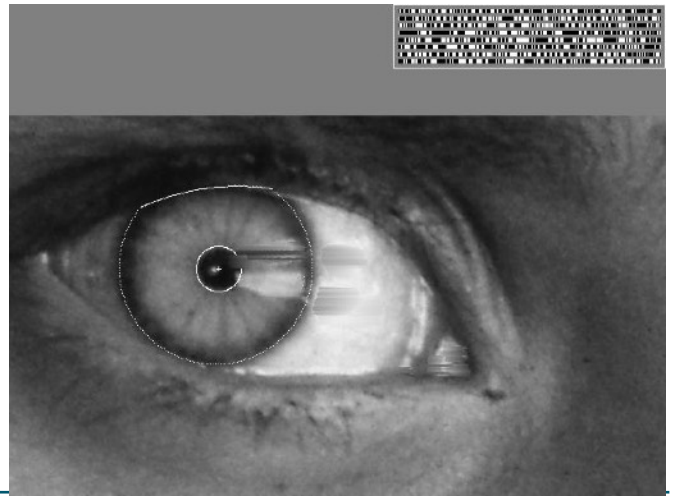
Algorithm Example:

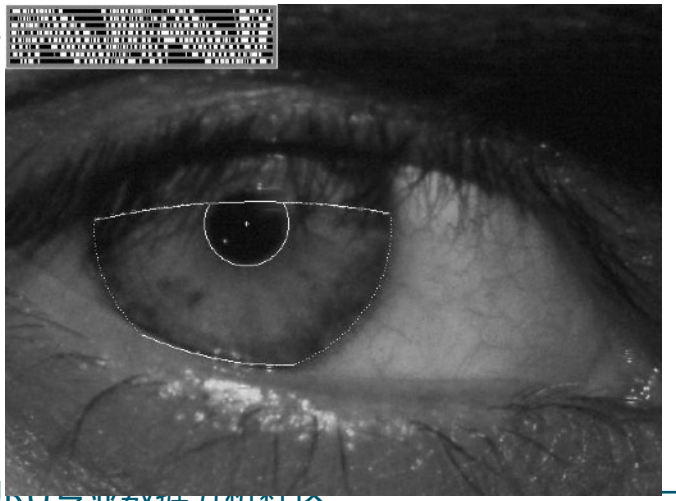
RCNN/Yolo/SSD

...

Iris Recognition

"*How the Afghan Girl was Identified by Her Iris Patterns*" Read the story wikipedia





Algorithm Example:

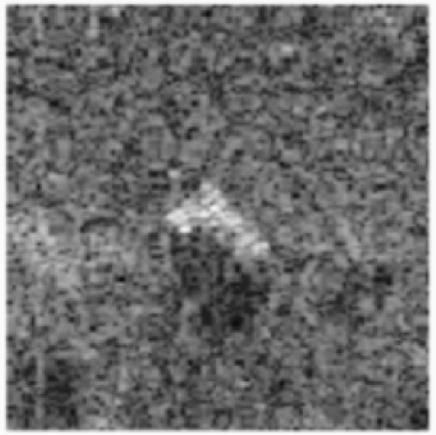
Image Unification

...

Taget recognition, Tracing

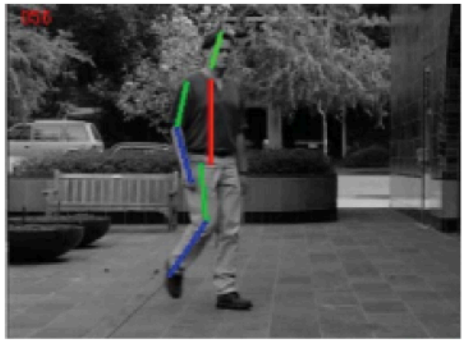
- Target recognition
  - Find enemy vehicles (which are trying not to be found!)

FLIR image from sdvision.kaist.ac.kr/



SAR image from web.mit.edu/6.003 /courseware/

- Human Interfaces
  - Detect faces and identify people
  - Recognize gestures, activities



Algorithm Example:

Lucas-Kanade algorithm
Skeleton
…

Face Verification

Fingerprint scanners on many new laptops, other devices

Face recognition systems now beginning to appear more widely
http://www.sensiblevision.com/

Algorithm Example:

Eigenface Feature
HMM/SVD classification

…

Special effects: shape capture

Algorithm Example:

Deepface algorithm

…

The Matrix movies, ESC Entertainment, XYZRGB, NRC

Special effects: motion capture

Pirates of the Carribean, Industrial Light and Magic

Algorithm Example:

Deepface algorithm

…

Sports

Sportvision first down line
Nice explanation on www.howstuffworks.com

http://www.sportvision.com/video.html

Intelligence car (unmanned vehicle)

- Mobileye
  - Vision systems currently in high-end BMW, GM, Volvo models

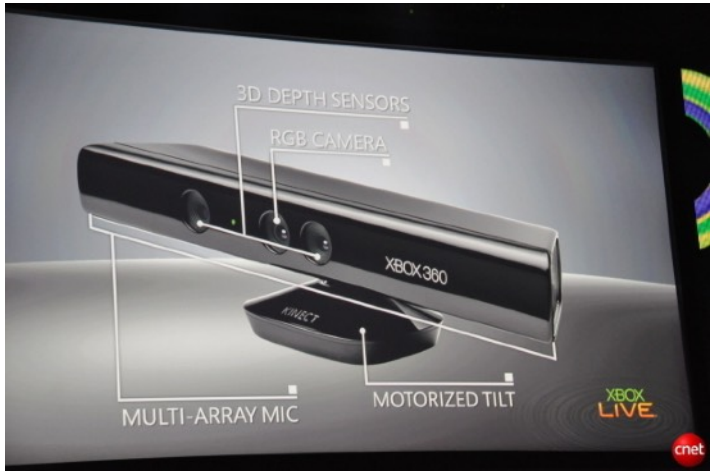
Algorithm Example:

CNN+ SVM decision
SLAM
…

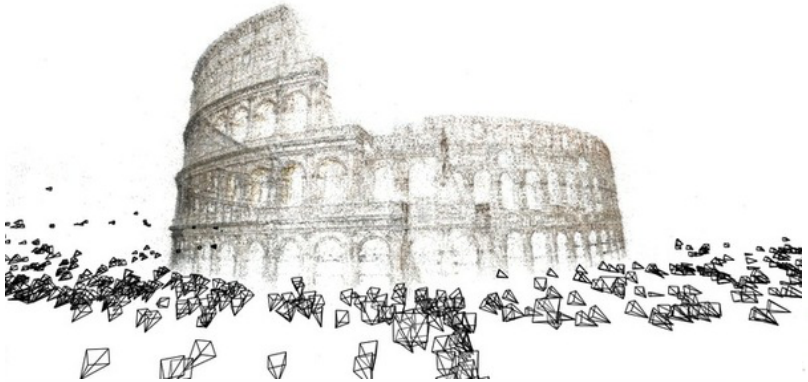
Intelligence car (unmanned vehicle)

AR/VR

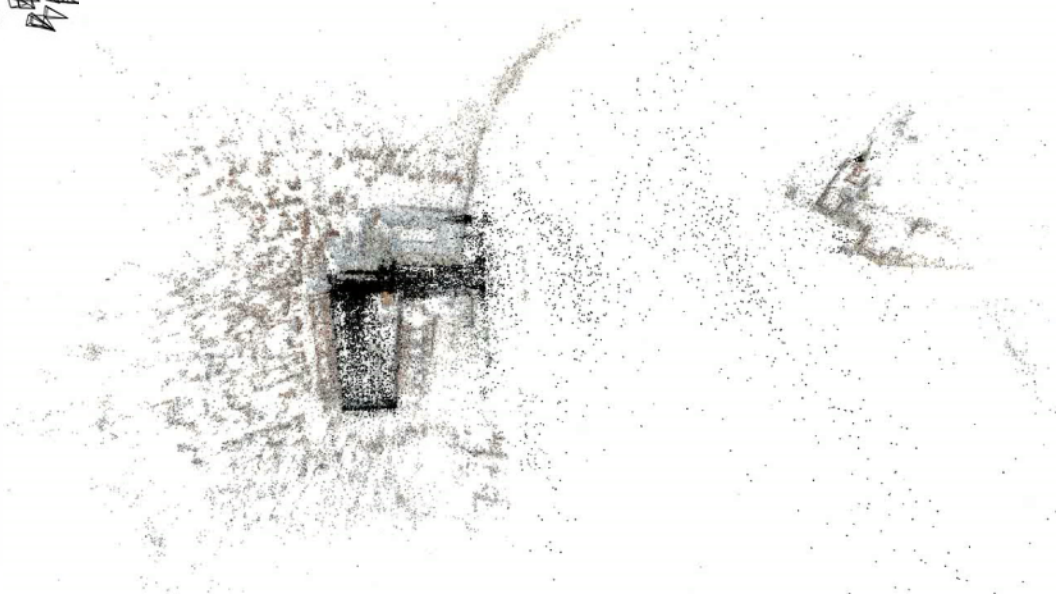
- Object Recognition: http://www.youtube.com/watch?feature=iv&v=fQ59dXOo63o
- Mario: http://www.youtube.com/watch?v=8CTJL5lUjHg
- 3D: http://www.youtube.com/watch?v=7QrnwoO1-8A
- Robot: http://www.youtube.com/watch?v=w8BmgtMKFbY



3D from thousands of images



Building Rome in a Day: Agarwal et al. 2009



Vision in space

NASA'S Mars Exploration Rover Spirit captured this westward view from atop a low plateau where Spirit spent the closing months of 2007.

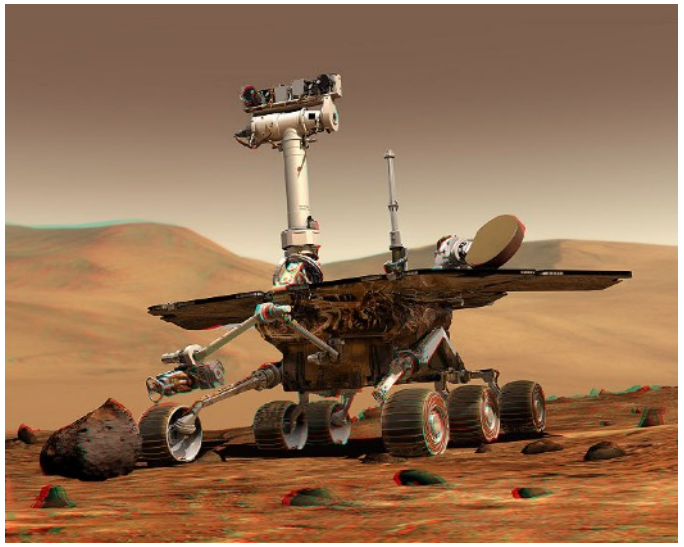
Vision systems (JPL) used for several tasks

- Panorama stitching
- 3D terrain modeling
- Obstacle detection, position tracking
- For more, read “Computer Vision on Mars” by Matthies et al.

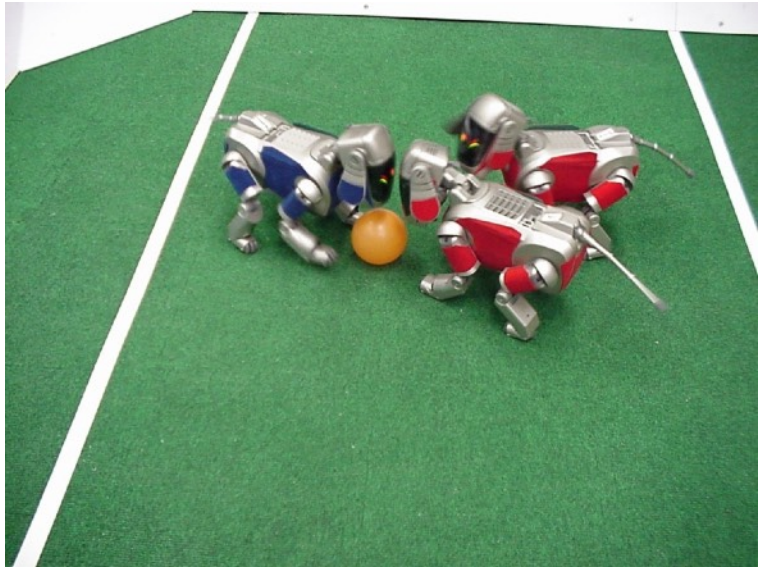
Industrial robots

Vision-guided robots position nut runners on wheels

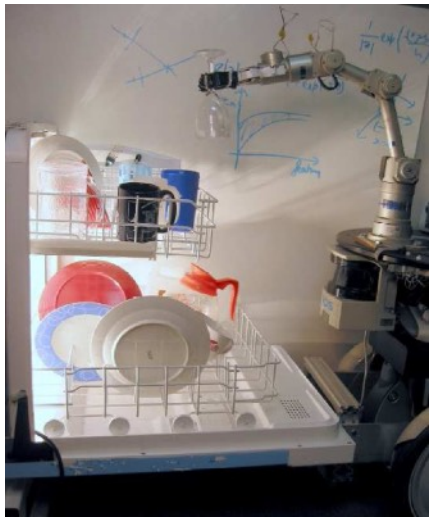
Mobile robots



NASA's Mars Spirit Rover
http://en.wikipedia.org/wiki/Spirit_rover

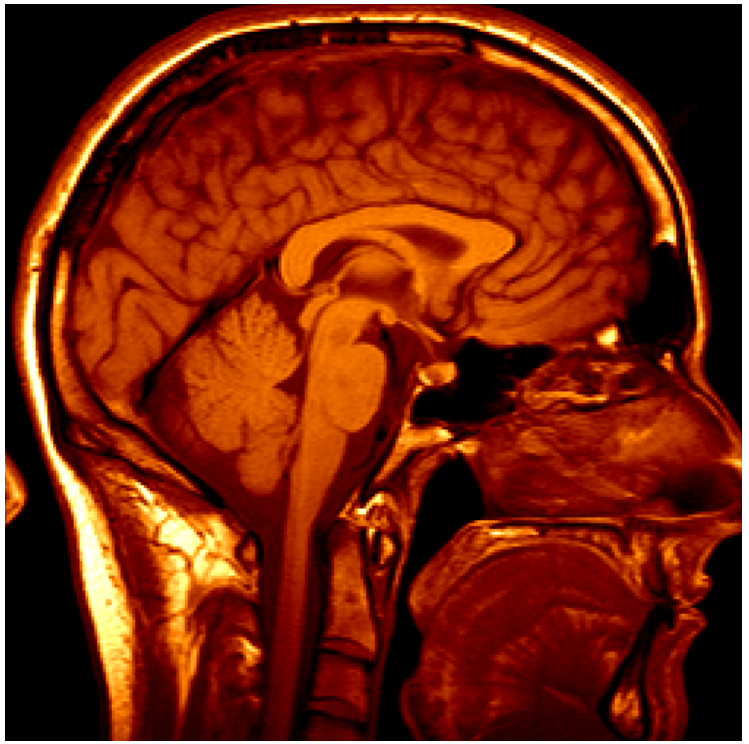


http://www.robocup.org/



Saxena et al. 2008
STAIR at Stanford

Medical Image



3D imaging
MRI, CT

Image guided surgery
Grimson et al., MIT

Table of contents

Unsupervised learning

Algorithm Example:

GAN

...

Translate image/video to computer understood semantic

Make computers understand what they saw

What kind of scene?

Where are the cars?

How far is the building?

...

A computer vision system

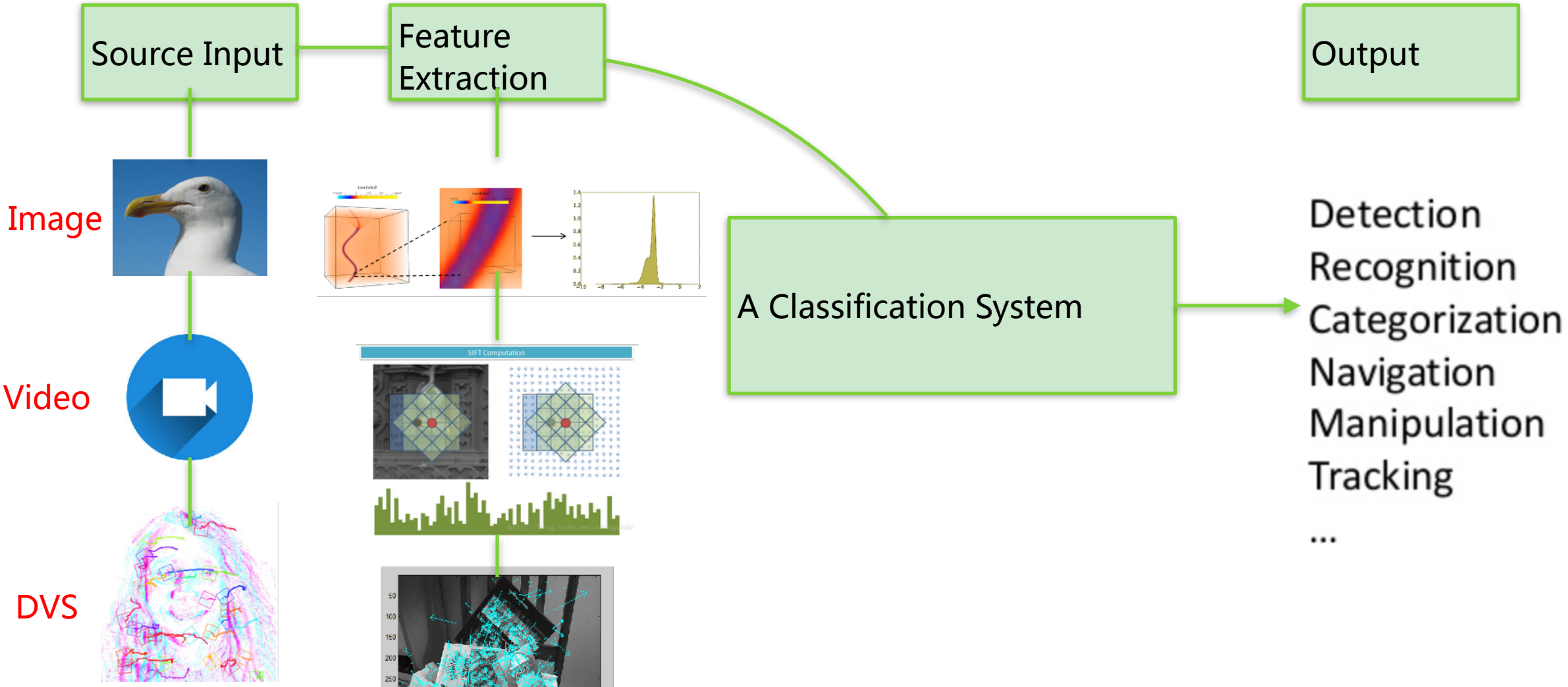


Table of contents

How to learn CV (AFPW method)

Wheel

$$\rightarrow \frac{det\ (M)}{trace\ (M)} = \frac{I_x^2 I_y^2 - (I_x I_y)^2}{I_x^2 + I_y^2}$$

Algorithms

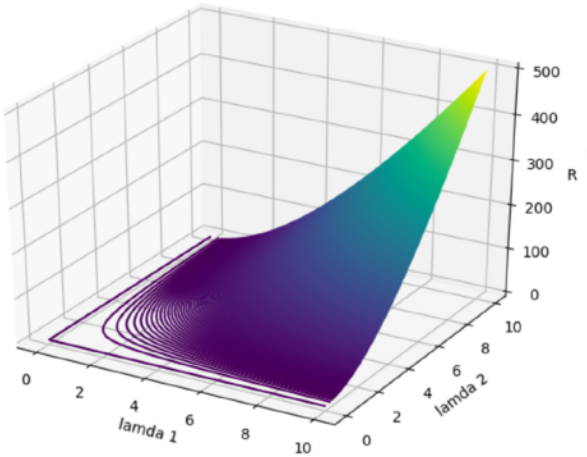
```
def DoG_Extrema(Osctaves, oc, current):
    top = current - 1
    down = current + 1
    current_state = Osctaves[oc][1:-2, 1:-2, current]

    for x in range(0, 3):
        for y in range(0, 3):
            if not (x == 1 and y == 1):
                local_maxima_to_current_neighbors = (current_state > Osctaves[oc][x:x - 3, y:y - 3, current])
            local_maxima_to_top_neighbors = (current_state > Osctaves[oc][x:x - 3, y:y - 3, top])
            local_maxima_to_down_neighbors = (current_state > Osctaves[oc][x:x - 3, y:y - 3, down])

            try:
                local_maxima = local_maxima & local_maxima_to_current_neighbors & local_maxima_to_top_neighbors & local_maxima_to_down_neighbors
            except:
                local_maxima = local_maxima_to_current_neighbors

    for x in range(0, 3):
        for y in range(0, 3):
            if not (x == 1 and y == 1):
                local_minima_to_current_neighbors = (current_state < Osctaves[oc][x:x - 3, y:y - 3, current])
            local_minima_to_top_neighbors = (current_state < Osctaves[oc][x:x - 3, y:y - 3, top])
            local_minima_to_down_neighbors = (current_state < Osctaves[oc][x:x - 3, y:y - 3, down])
            try:
                local_minima = local_minima & local_minima_to_current_neighbors & local_minima_to_top_neighbors & local_minima_to_down_neighbors
            except:
                local_minima = local_minima_to_current_neighbors
```

Perception




```
im = cv2.imread('Images/jobs.jpg')
SIFT = cv2.xfeatures2d.SIFT_create()
SURF = cv2.xfeatures2d.SURF_create()
ORB = cv2.ORB_create()

kps_SIFT,descfs_SIFT = SIFT.detectAndCompute(im, None)
kps_SURF,descfs_SURF = SURF.detectAndCompute(im, None)
kps_ORB, descfs_ORB = ORB.detectAndCompute(im, None)


imgSIFT = cv2.drawKeypoints(im, kps_SIFT, None, color=(255,0,0))
imgSURF = cv2.drawKeypoints(imgSIFT, kps_SURF, None, color=(0,255,0))
imgORB = cv2.drawKeypoints(imgSURF, kps_ORB, None, color=(0,0,255))
```

Functions

AFPW Method Example:

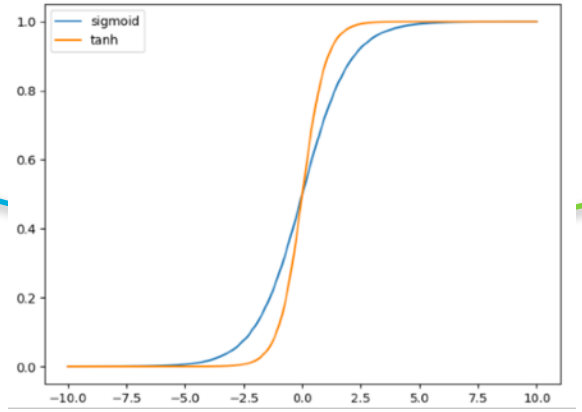
Wheel

Algorithms

$$sigmoid = \frac{1}{1 + e^{-\lambda z}}$$

```
def sigmoid(x):
    sigmoid = 1 / (1 + np.exp(-x))
    return sigmoid
```

tensorflow.sigmoid



Perception

Functions

Table of contents

Overview

Pixel Operation

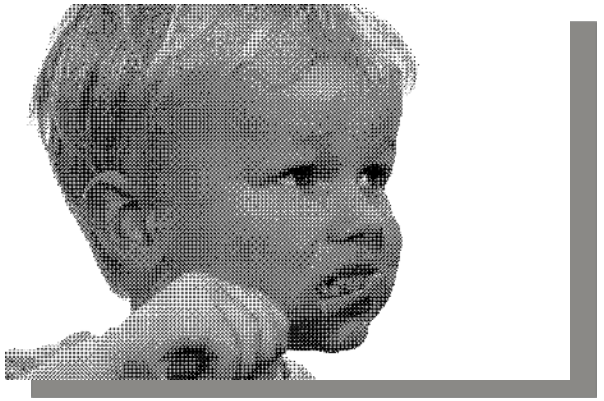
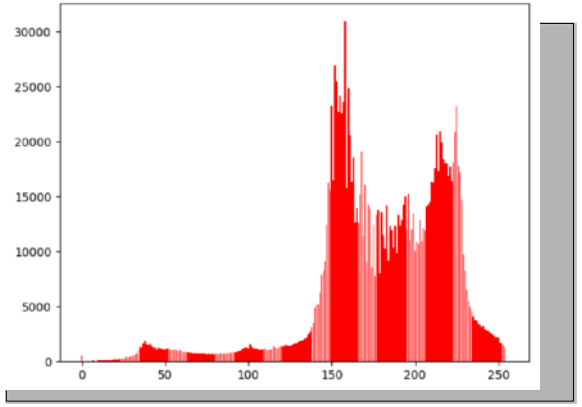


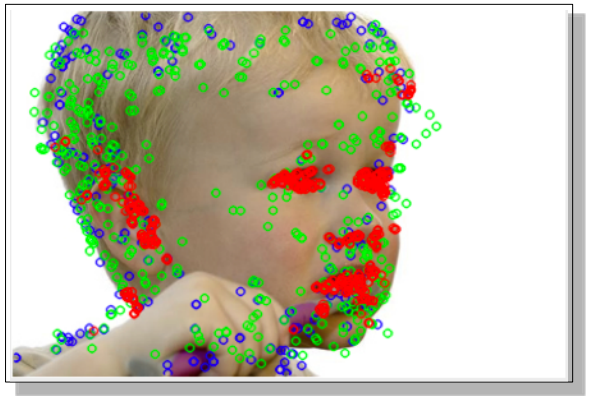
Image Quantilization

Wrap/Affine/combining operations

Image Convolution



Histogram Operation



Interesting point Extraction

Image Preprocessing for machine learning

- Pixel operations
  - Add random noise
  - Add luminance
  - Add contrast
  - Add saturation
- Quantization
  - Uniform quantization
  - Random dither
  - Ordered dither
  - Floyd-Steinberg dither

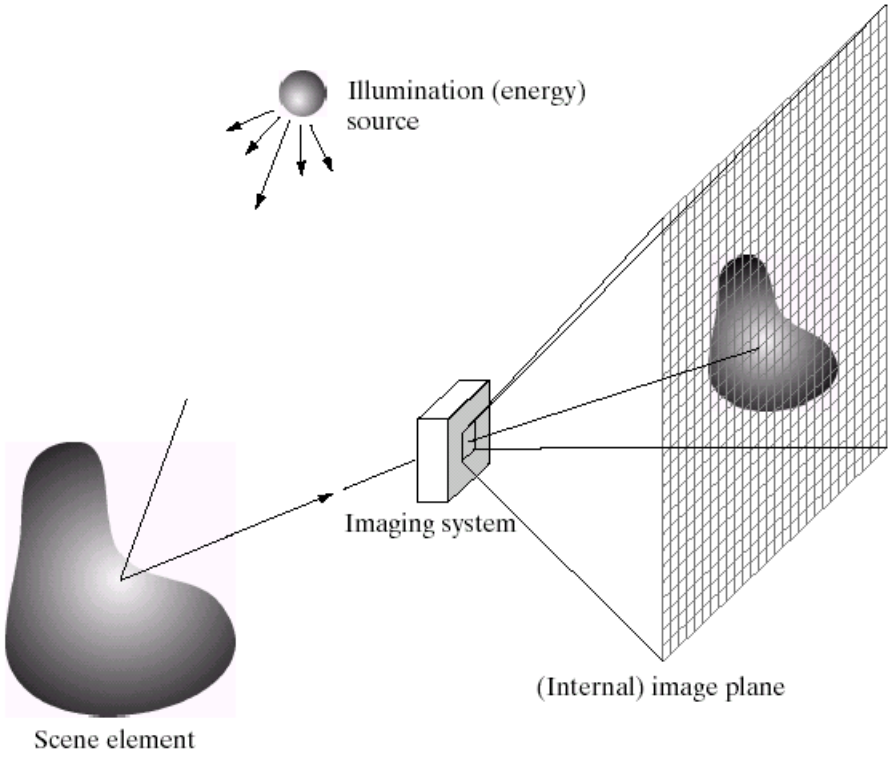
- Warping/Combining/Warping
  - Scale
  - Rotate
  - Warps

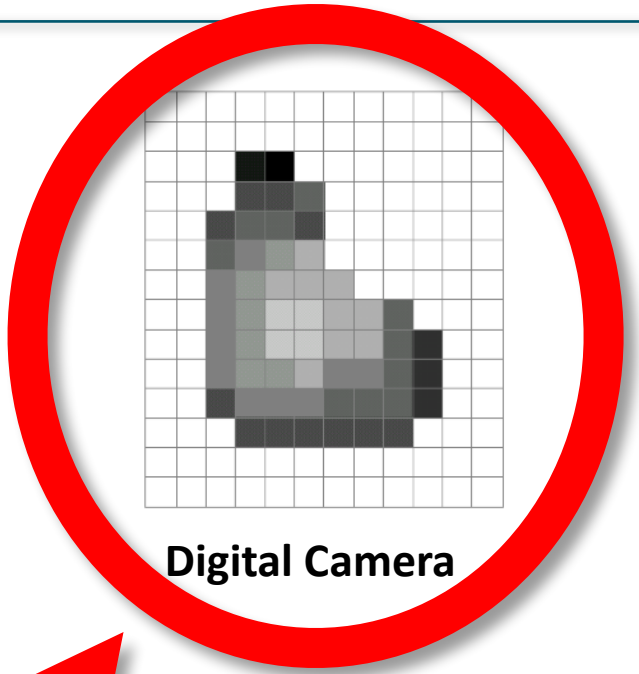
- Convolution
  - Blur
  - Detect edge
  - Haar Feature

- Histogram
  - Histogram equalisation
  - Hog Feature

- Intersting points
  - Haris corner point
  - SIFT/ORB/SURF
  - Discriptor match

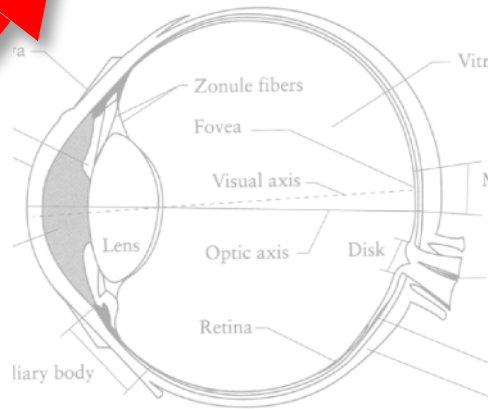
What is an image?





We'll focus on these in this class

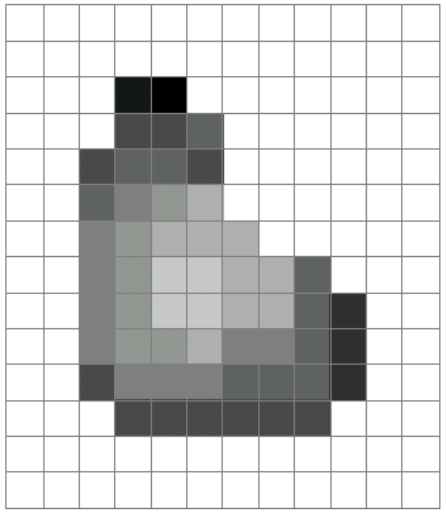
(More on this process later)



The Eye

Source: A. Efros

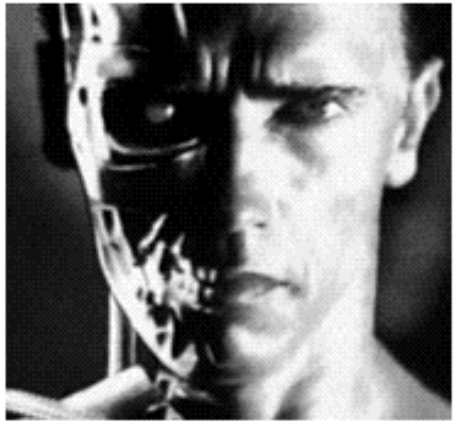
What is an image?

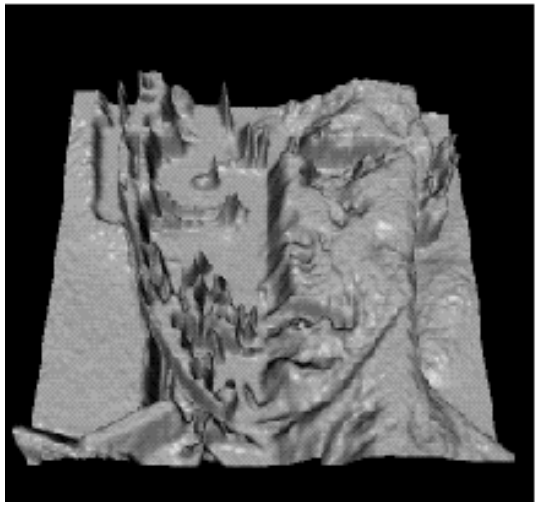


=

255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	20	0	255	255	255	255	255	255	255
255	255	255	75	75	75	255	255	255	255	255	255
255	255	75	95	95	75	255	255	255	255	255	255
255	255	96	127	145	175	255	255	255	255	255	255
255	255	127	145	175	175	175	255	255	255	255	255
255	255	127	145	200	200	175	175	95	255	255	255
255	255	127	145	200	200	175	175	95	47	255	255
255	255	127	145	145	175	127	127	95	47	255	255
255	255	74	127	127	127	95	95	95	47	255	255
255	255	255	74	74	74	74	74	74	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255

Image as a Function





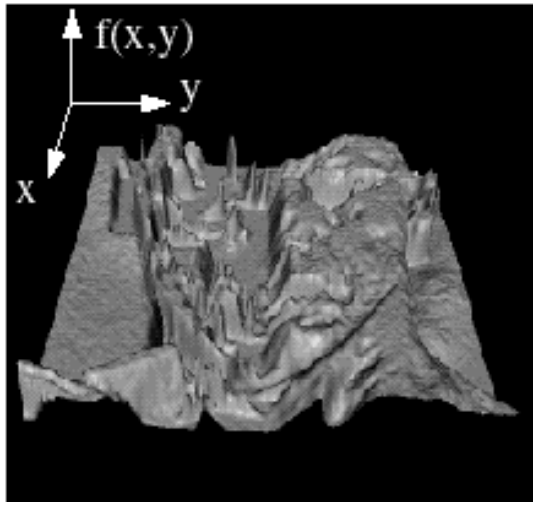
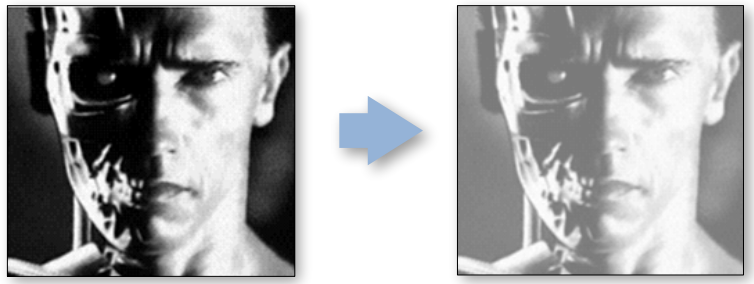
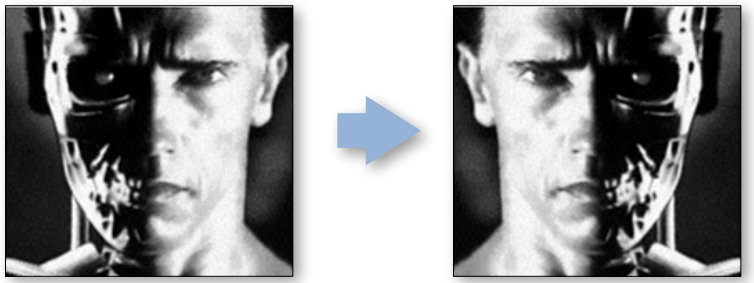


Image as a Function

- As with any function, we can apply operators to an image



$$g(x,y) = f(x,y) + 20$$



$$g(x,y) = f(-x,y)$$

- We'll talk about a special kind of operator, *convolution* (linear filtering)

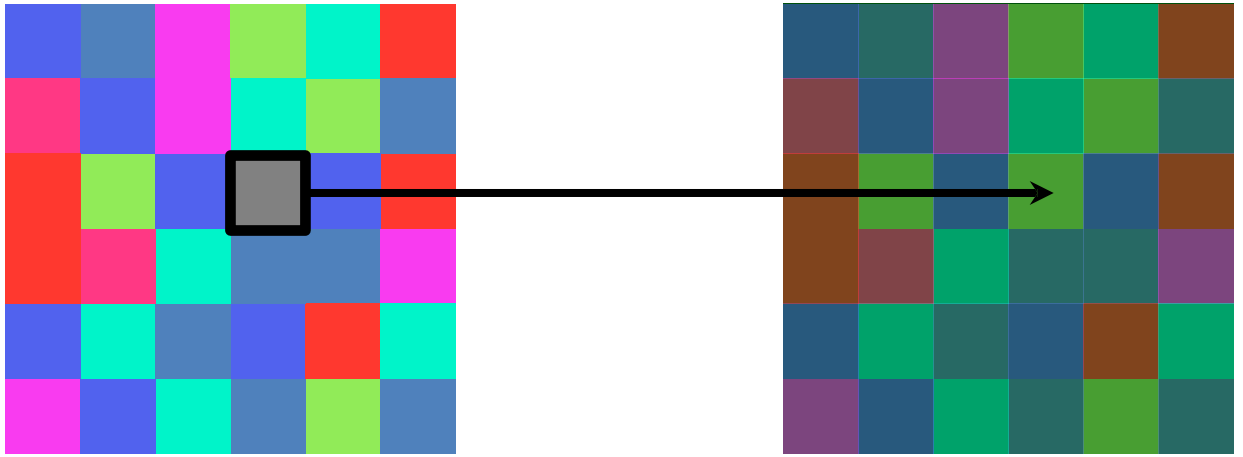
Image as a Function

- We can think of an **image** as a function, *f*,
- f: $R^2 \rightarrow R$
  - $f(x, y)$ gives the **intensity** at position (x, y)
  - Realistically, we expect the image only to be defined over a rectangle, with a finite range:
    - f: $[a,b]$x$[c,d] \rightarrow [0,1]$

- A color image is just three functions pasted together. We can write this as a "vector-valued" function:

$$f(x,y) = \begin{bmatrix} r(x,y) \\ g(x,y) \\ b(x,y) \end{bmatrix}$$

Point Operations



Point Processing

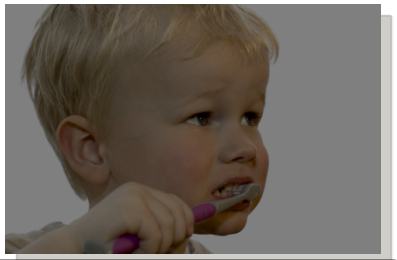
Original

`x`

Darken

`x - 128`

Lower Contrast

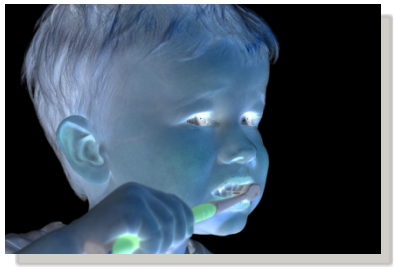


`x / 2`

Nonlinear Lower Contrast

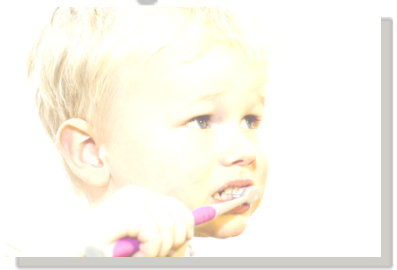
`((x / 255.0) ^ 0.33) * 255.0`

Invert



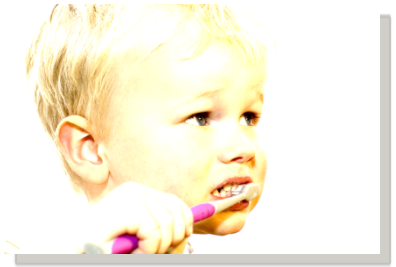
`255 - x`

Lighten



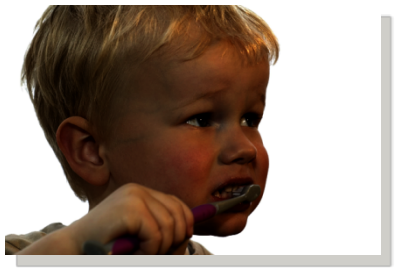
`x + 128`

Raise Contrast



`x * 2`

Nonlinear Raise Contrast



`((x / 255.0) ^2) * 255.0`

- Some operations preserve the range but change the domain of *f* :

$$g(x, y) = f(t_x(x, y), t_y(x, y))$$

What kinds of operations can this perform?

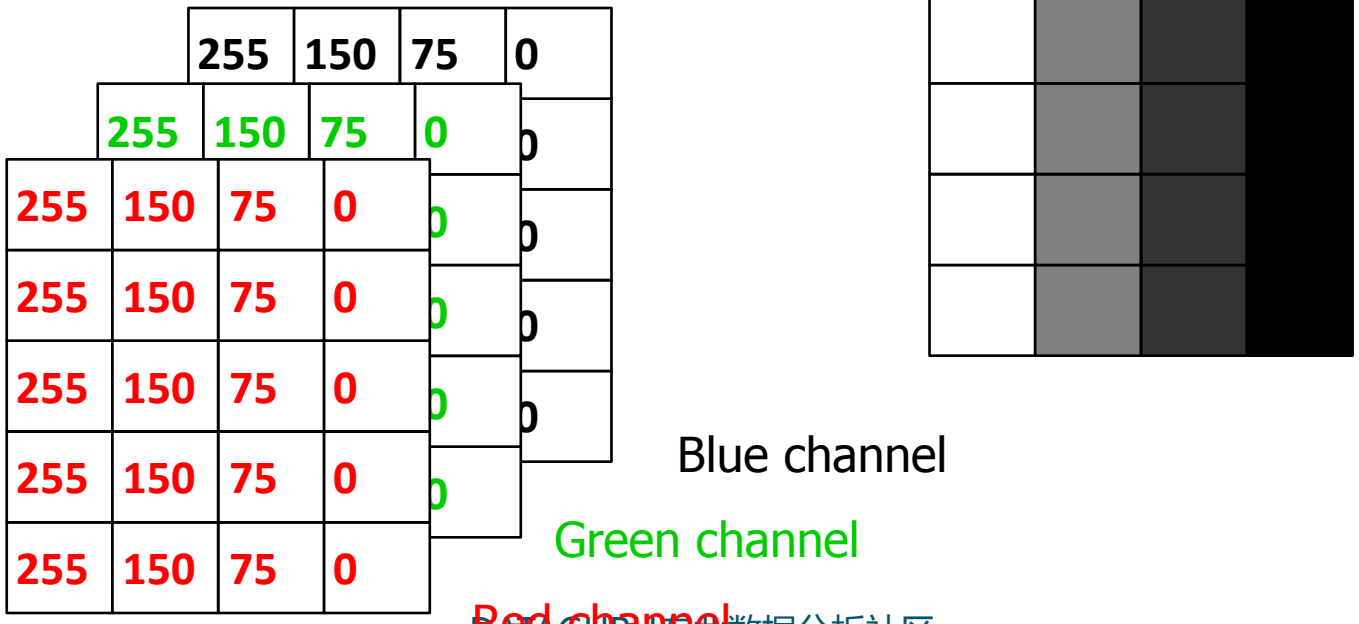
- Still other operations operate on both the domain and the range of f .

Table of contents

Quantization

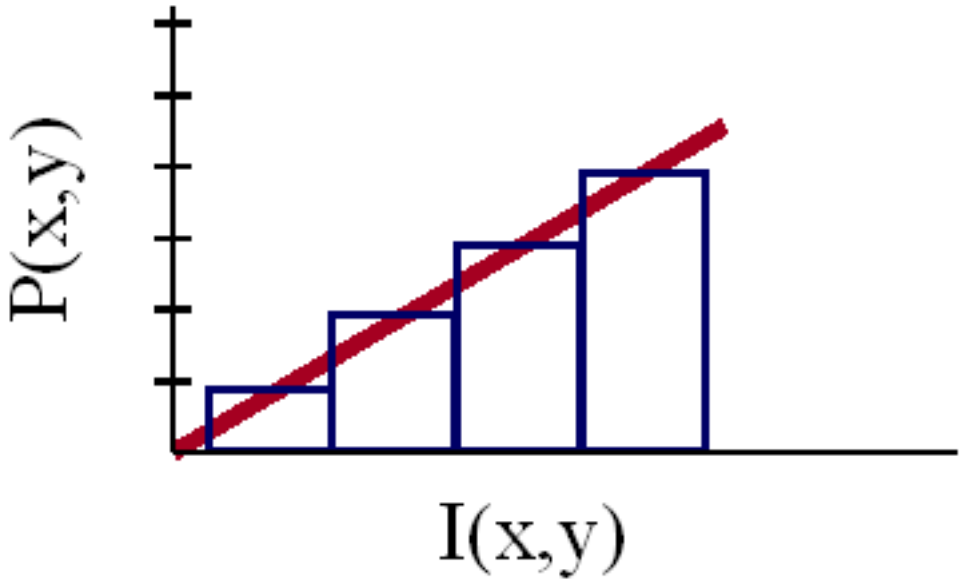
Artifact due to limited intensity resolution

- Frame buffers have limited number of bits per pixel
- Physical devices have limited dynamic range

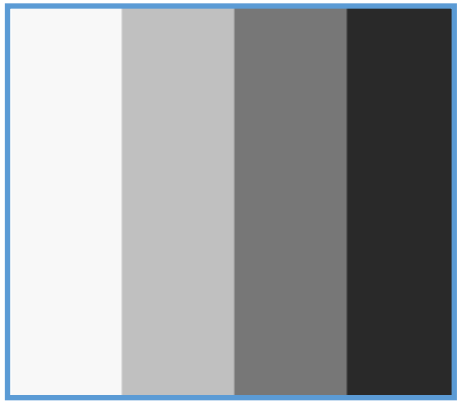


Uniform Quantization

$$P(x,y) = \text{trunc}(I(x,y) + 0.5)$$



I(x, y)



P(x, y)
2 bits per pixel

Uniform Quantization

Image with decreasing bits per pixel:

8 bits

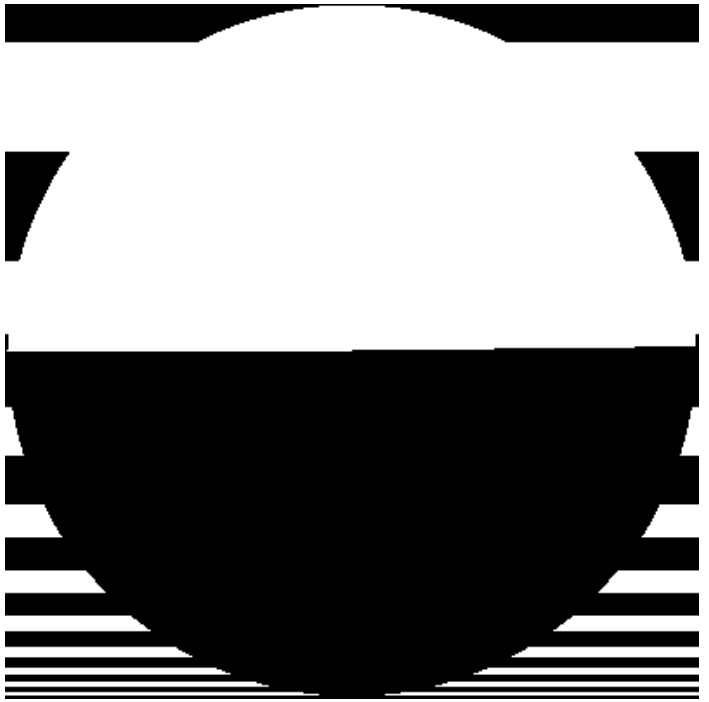
4 bits

2 bits

1 bit

Notice contouring

Naïve Threshold Algorithm



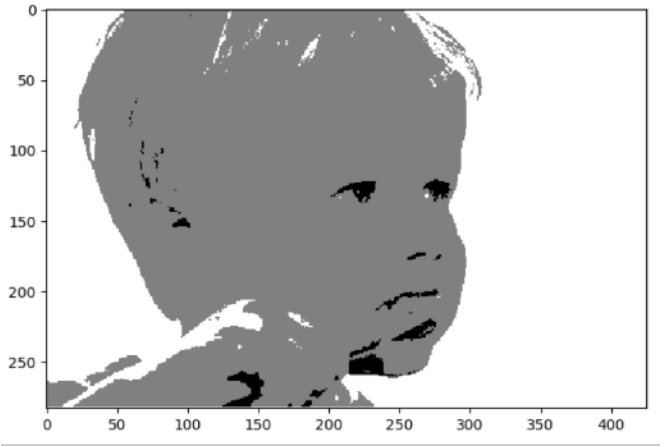
Coding Example

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

N = 2
gray = cv2.imread('images/ChildBrushingTeeth.jpg',cv2.IMREAD_GRAYSCALE)
Uniform_quant = np.round(gray*(N/255))*np.round(255/N)
print(Uniform_quant)

plt.imshow(Uniform_quant,cmap='gray')
plt.show()
```

N=2



N=16

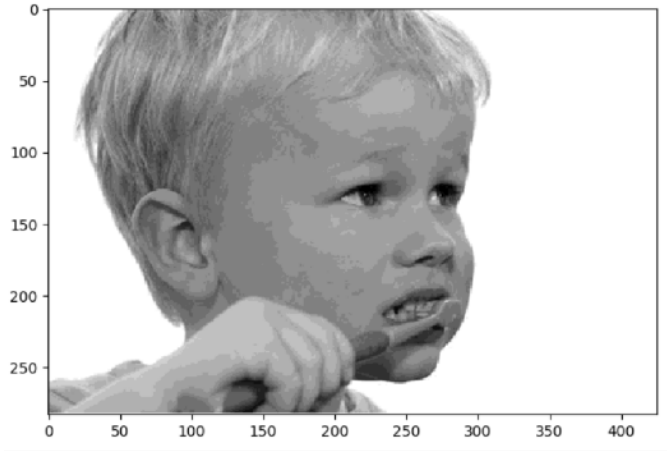


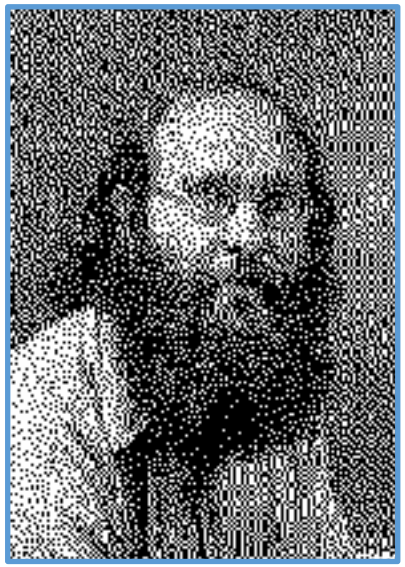
Table of contents

Dithering

- Distribute errors among pixels
  - Exploit spatial integration in our eye
  - Display greater range of perceptible intensities

Original
(8 bits)

Uniform
Quantization
(1 bit)

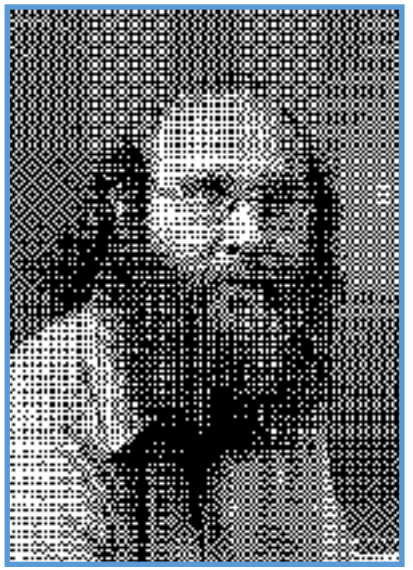


Floyd-Steinberg
Dither
(1 bit)

Ordered Dither

Original
(8 bits)

Uniform
Quantization
(1 bit)



Ordered
Dither
(1 bit)

Ordered Dither

ordered dither matrices

$$D_n = \begin{bmatrix} 4D_{n/2} + D_2(1,1)U_{n/2} & 4D_{n/2} + D_2(1,2)U_{n/2} \\ 4D_{n/2} + D_2(2,1)U_{n/2} & 4D_{n/2} + D_2(2,2)U_{n/2} \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} \qquad D_4 = \begin{bmatrix} 15 & 7 & 13 & 5 \\ 3 & 11 & 1 & 9 \\ 12 & 4 & 14 & 6 \\ 0 & 8 & 2 & 10 \end{bmatrix}$$

Ordered Dithering

- Break the image into small blocks
- Define a *threshold matrix*
  - Use a different threshold for each pixel of the block
  - Compare each pixel to its own threshold
- The thresholds can be clustered, which looks like newsprint
- The thresholds can be "random" which looks better

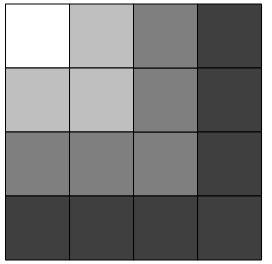
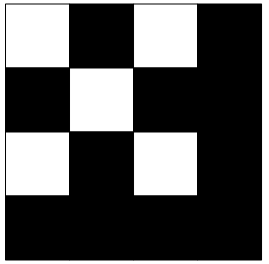


Image block

$$\begin{bmatrix} 1 & 0.75 & 0.5 & 0.25 \\ 0.75 & 0.75 & 0.5 & 0.25 \\ 0.5 & 0.5 & 0.5 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{bmatrix}$$

Threshold matrix

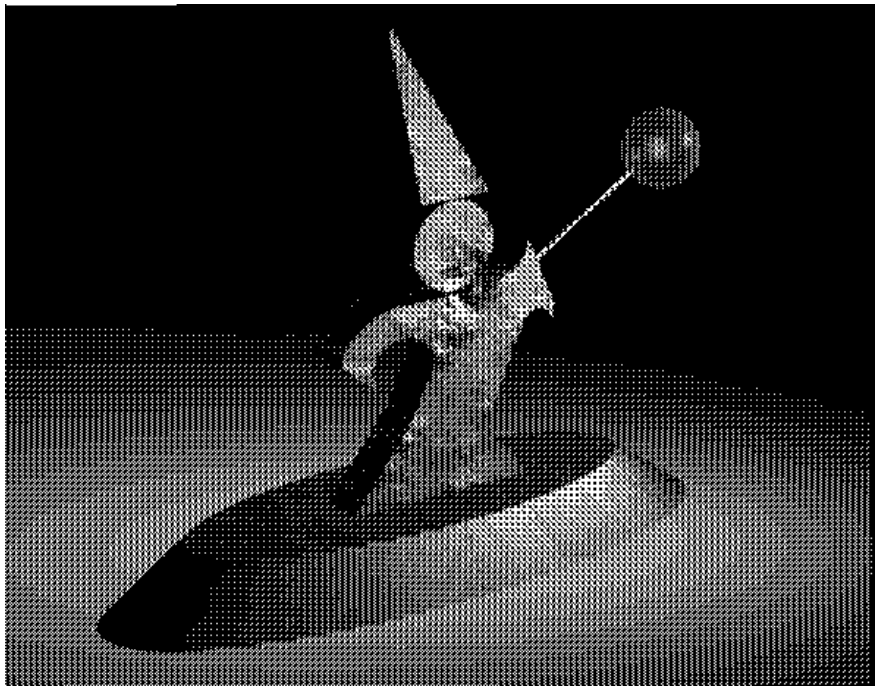
$$\frac{1}{16}\begin{bmatrix} 2 & 16 & 3 & 13 \\ 10 & 6 & 11 & 7 \\ 4 & 14 & 1 & 15 \\ 12 & 8 & 9 & 5 \end{bmatrix}$$

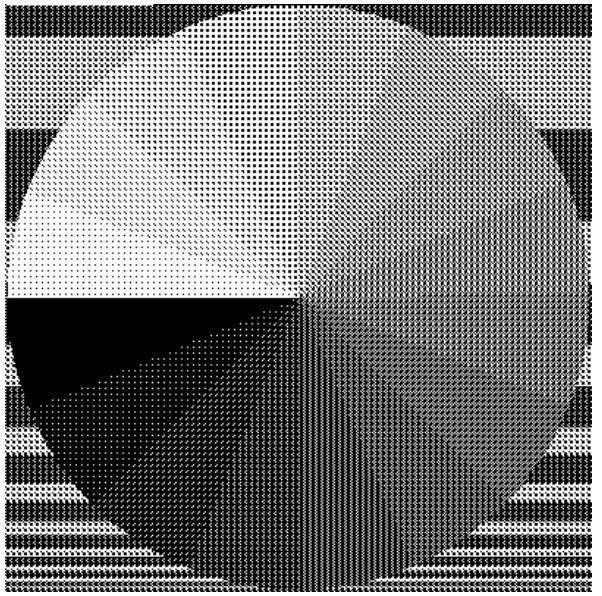


Result

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Ordered Dither





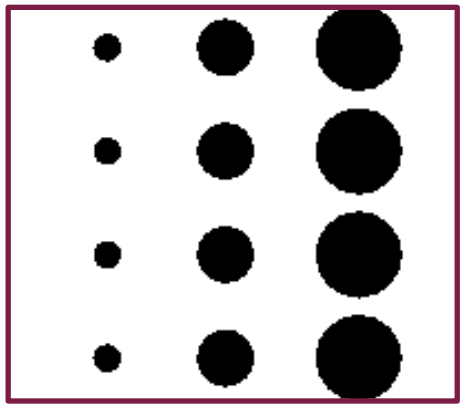
$$\begin{bmatrix} .75 & .375 & .625 & .25 \\ .0625 & 1 & .875 & .4375 \\ .5 & .8125 & .9375 & .125 \\ .1875 & .5625 & .3125 & .6875 \end{bmatrix}$$

Table of contents

Classical Hafltoning

- Use dots of varying size to representation intensities
  - Area of dots proportional to intensity in image

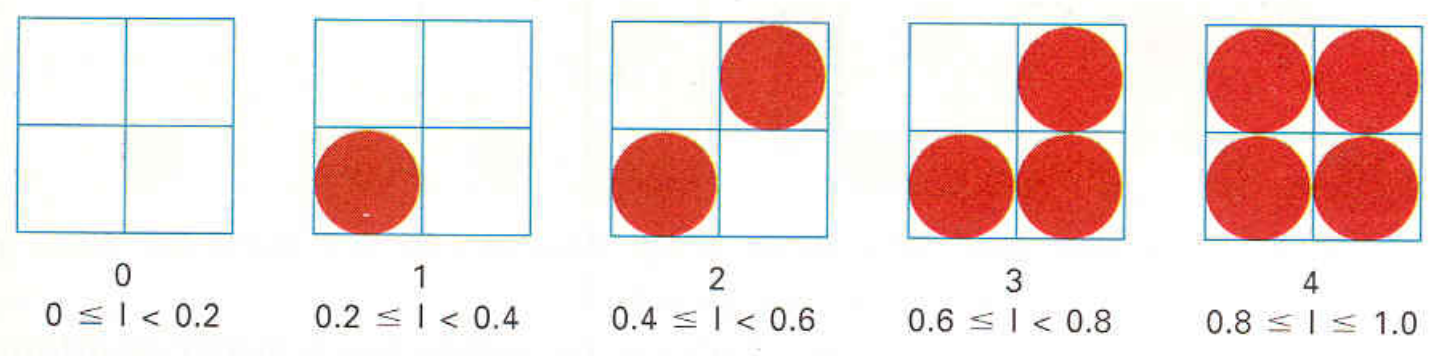
I(x, y)



P(x, y)

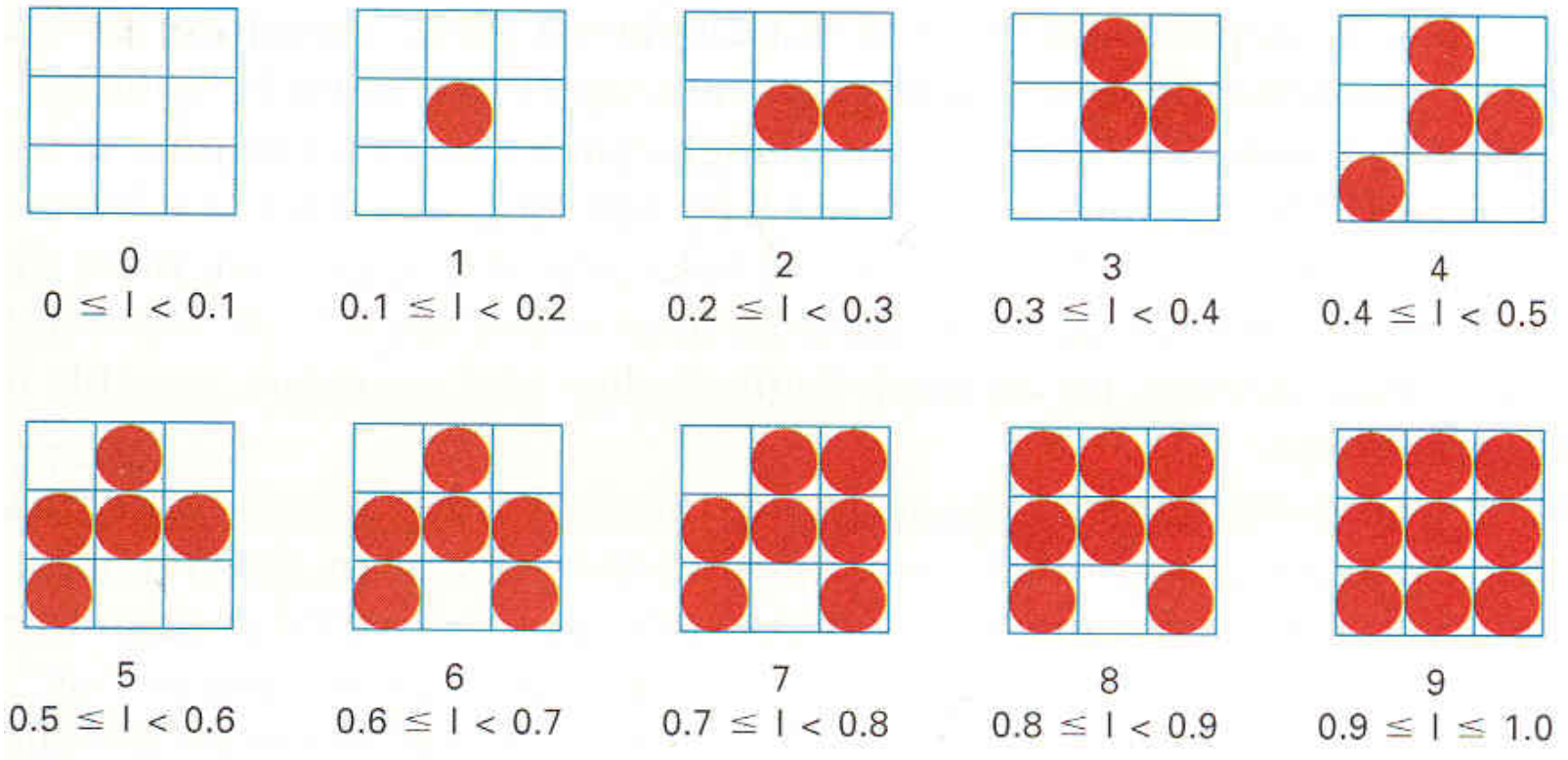
Classical Hafltoning

- Use cluster of pixels to represent intensity
  - Trade spatial resolution for intensity resolution



Classical Hafltoning

- How many intensities in a n x n cluster?



Classical Hafltoning

Newspaper image

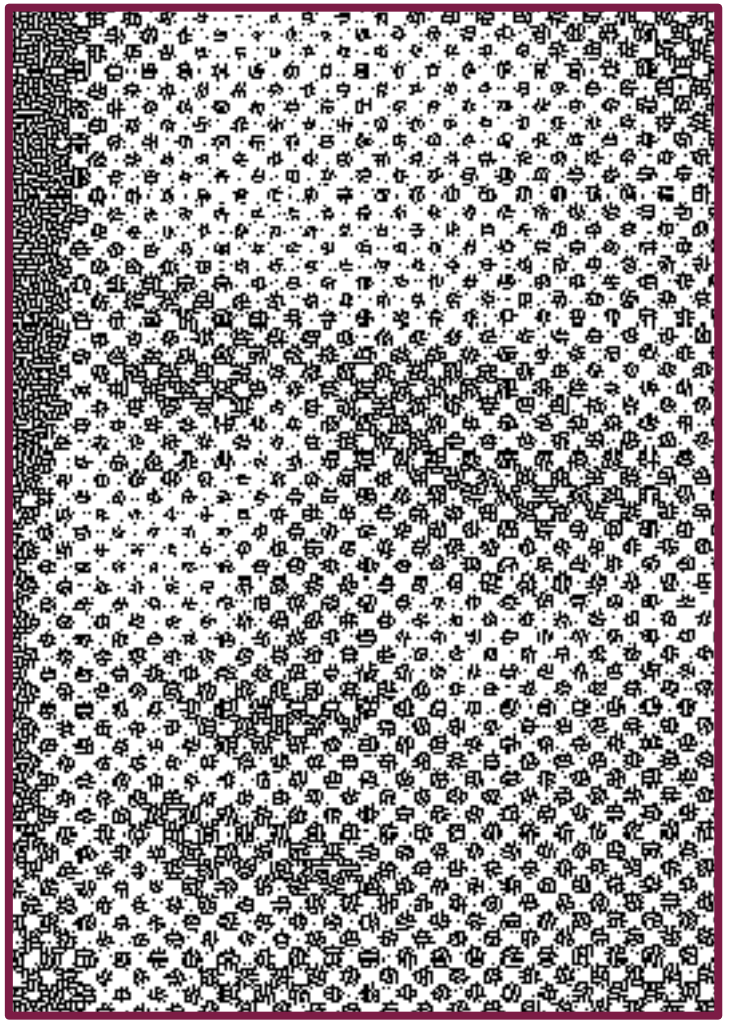


Table of contents

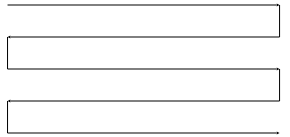
Floyd-Steinberg Dithering

- Any “rounding” errors are distributed to other pixels
  - Specifically to the pixels below and to the right
    - 7/16 of the error to the pixel to the right
    - 3/16 of the error to the pixel to the lower left
    - 5/16 of the error to the pixel below
    - 1/16 of the error to the pixel to the lower right
- Assume the 1 in the middle gets “rounded” to 0

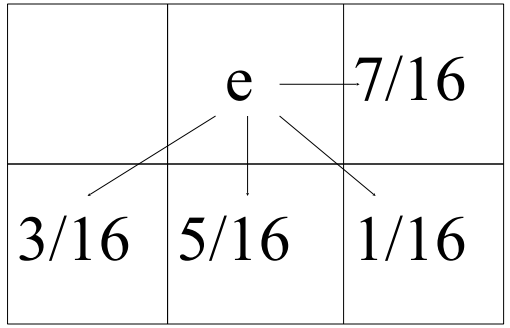
$$\begin{bmatrix} 0.00 & 0.00 & 0.00 \\ 0.00 & 1.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{bmatrix} \longrightarrow \begin{bmatrix} 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.44 \\ 0.19 & 0.31 & 0.06 \end{bmatrix}$$

Floyd-Steinberg Dithering

- Start at one corner and work through image pixel by pixel
  - Usually scan top to bottom in a zig-zag

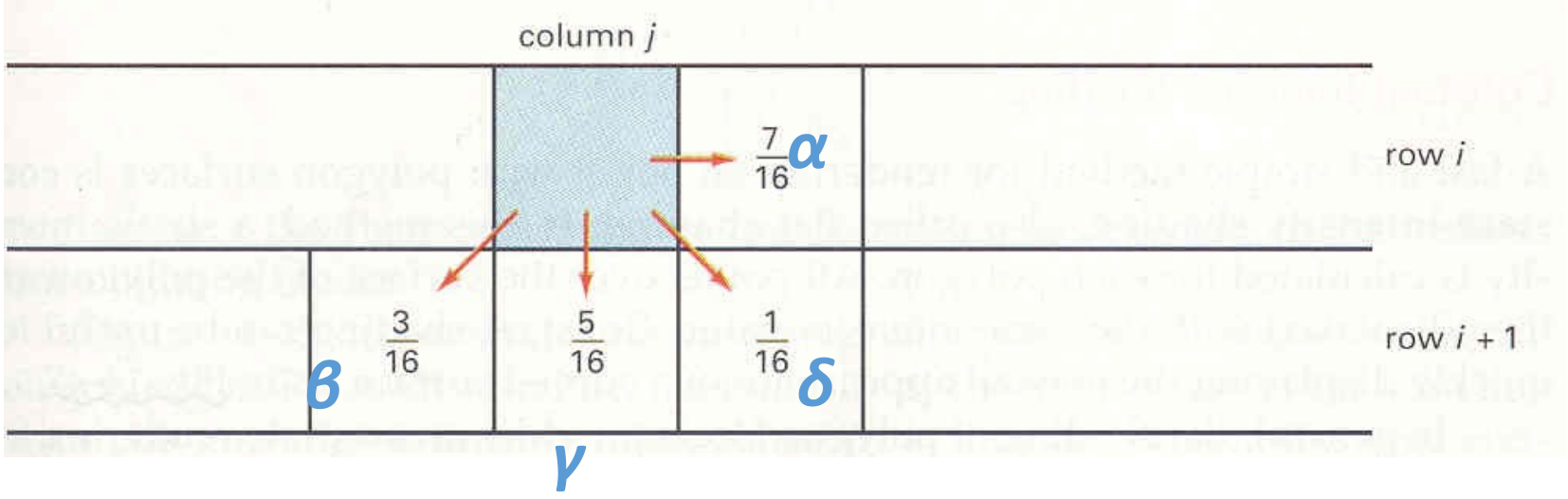


- Threshold each pixel
- Compute the error at that pixel: The difference between what should be there and what you did put there
  - If you made the pixel 0, *e* = original; if you made it 1, *e* = original-1
- Propagate error to neighbors by adding some proportion of the error to each unprocessed neighbor
  - A *mask* tells you how to distribute the error
- Easiest to work with floating point image
  - Convert all pixels to 0-1 floating point



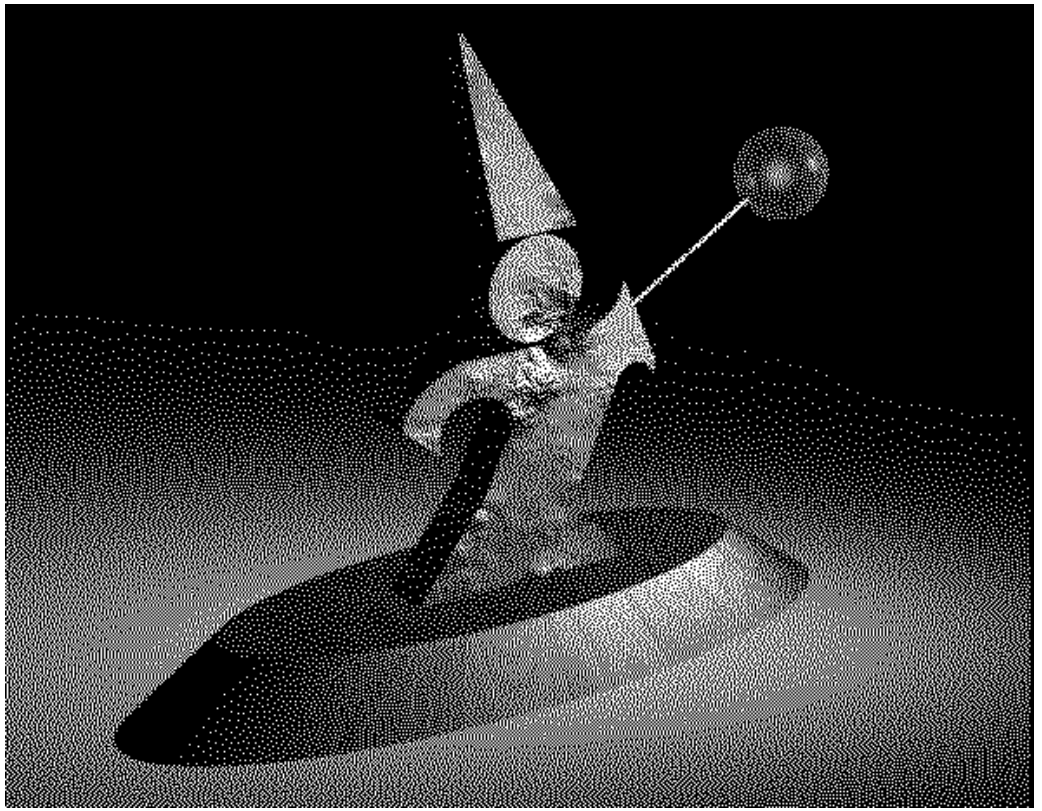
Error Diffusion Dither

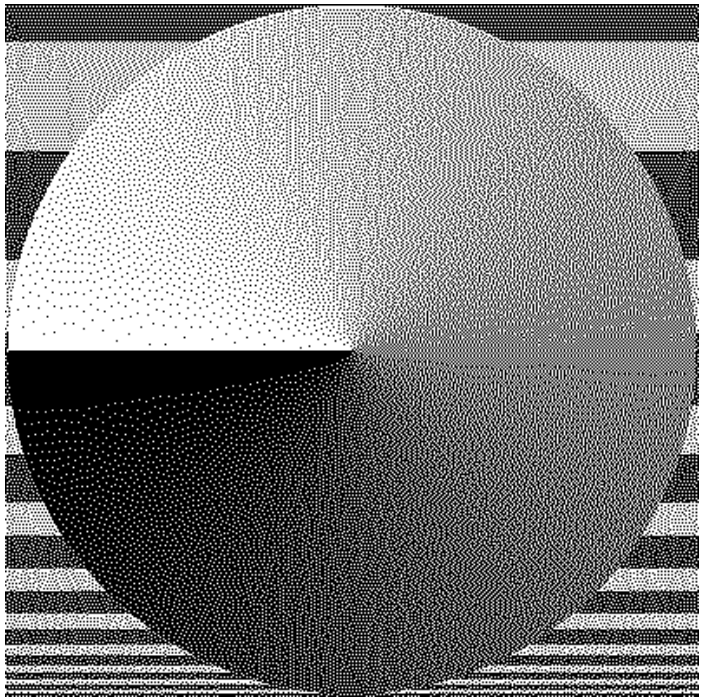
- Spread quantization error over neighbor pixels
  - Error dispersed to pixels right and below



$$\alpha + \beta + \gamma + \delta = 1.0$$

Floyd-Steinberg Dithering

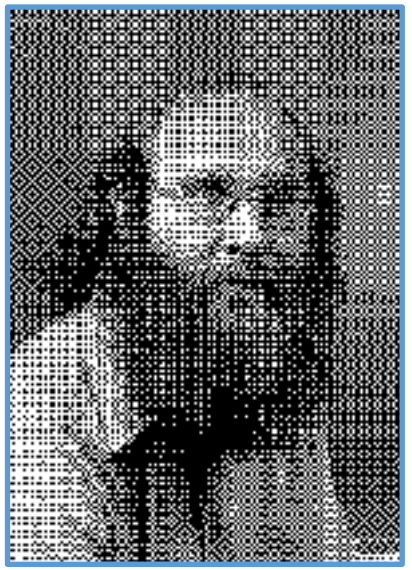




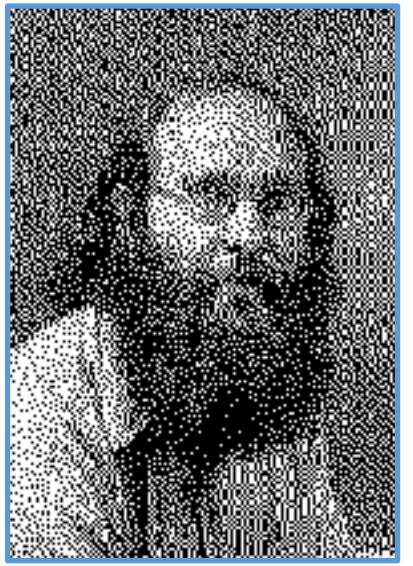
Error Diffusion Dither

Original
(8 bits)

Random
Dither
(1 bit)



Ordered
Dither
(1 bit)



Floyd-Steinberg
Dither
(1 bit)

Dithering Code

```python
import numpy as np
import PIL.Image as Image
import matplotlib.pyplot as plt

def floyd_steinberg_dither(image_file):
    new_img = Image.open(image_file)
    new_img = new_img.convert('RGB')
    pixel = new_img.load()
    x_lim, y_lim = new_img.size
    for y in range(1, y_lim):
        for x in range(1, x_lim):
            red_oldpixel, green_oldpixel, blue_oldpixel = pixel[x, y]
            red_newpixel = np.int(255 * np.floor(red_oldpixel / 128))
            green_newpixel = np.int(255 * np.floor(green_oldpixel / 128))
            blue_newpixel = np.int(255 * np.floor(blue_oldpixel / 128))

            pixel[x, y] = red_newpixel, green_newpixel, blue_newpixel

            red_error = red_oldpixel - red_newpixel
            blue_error = blue_oldpixel - blue_newpixel
            green_error = green_oldpixel - green_newpixel
            print(red_error)
            if x < x_lim - 1:
                red = pixel[x + 1, y][0] + round(red_error * 7 / 16)
                green = pixel[x + 1, y][1] + round(green_error * 7 / 16)
                blue = pixel[x + 1, y][2] + round(blue_error * 7 / 16)

                pixel[x + 1, y] = (red, green, blue)

            if x > 1 and y < y_lim - 1:
                red = pixel[x - 1, y + 1][0] + round(red_error * 3 / 16)
                green = pixel[x - 1, y + 1][1] + round(green_error * 3 / 16)
                blue = pixel[x - 1, y + 1][2] + round(blue_error * 3 / 16)

                pixel[x - 1, y + 1] = (red, green, blue)

            if y < y_lim - 1:
                red = pixel[x, y + 1][0] + round(red_error * 5 / 16)
                green = pixel[x, y + 1][1] + round(green_error * 5 / 16)
                blue = pixel[x, y + 1][2] + round(blue_error * 5 / 16)

                pixel[x, y + 1] = (red, green, blue)

            if x < x_lim - 1 and y < y_lim - 1:
                red = pixel[x + 1, y + 1][0] + round(red_error * 1 / 16)
                green = pixel[x + 1, y + 1][1] + round(green_error * 1 / 16)
                blue = pixel[x + 1, y + 1][2] + round(blue_error * 1 / 16)

                pixel[x + 1, y + 1] = (red, green, blue)

    plt.imshow(new_img)
    plt.show()
```

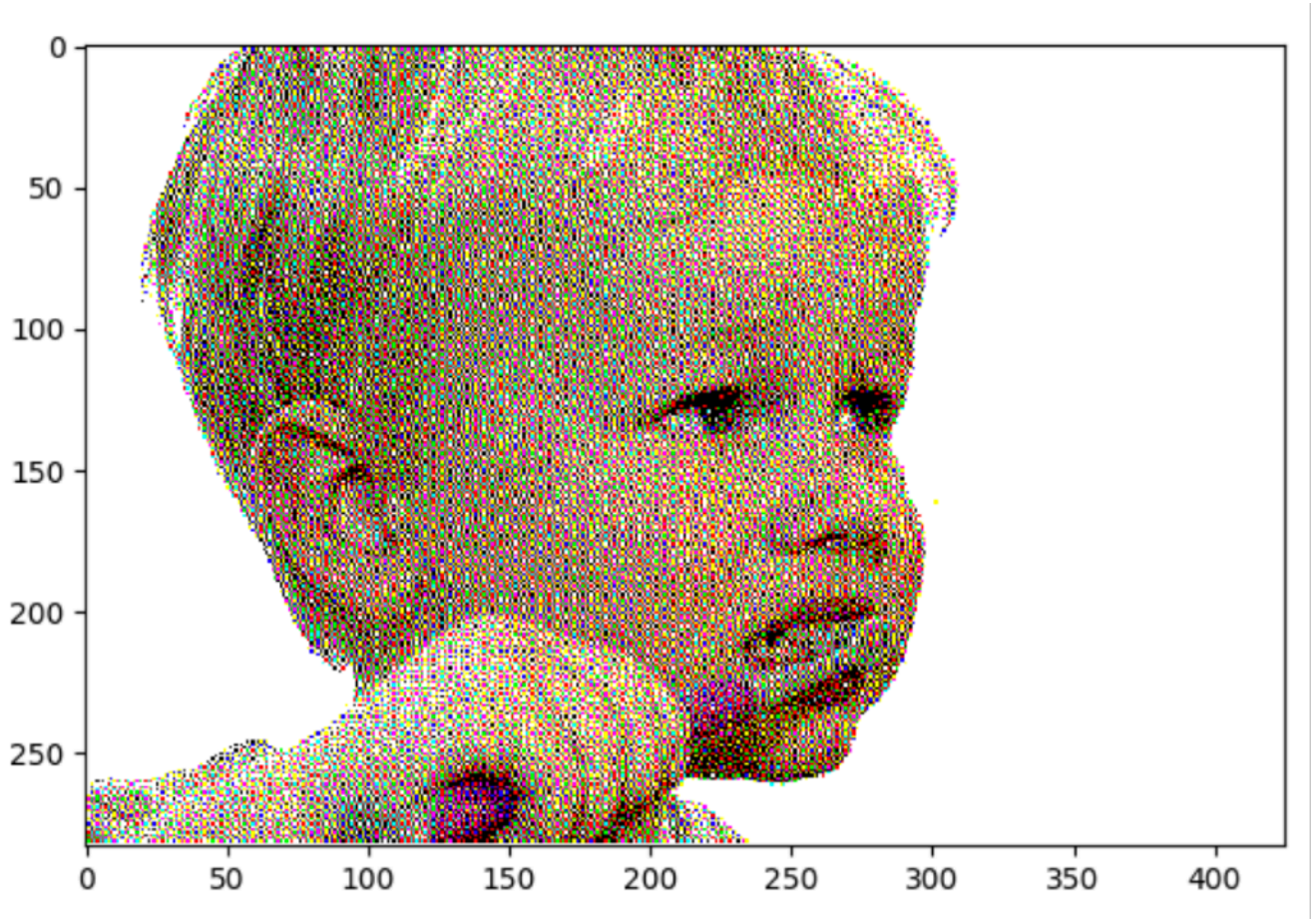
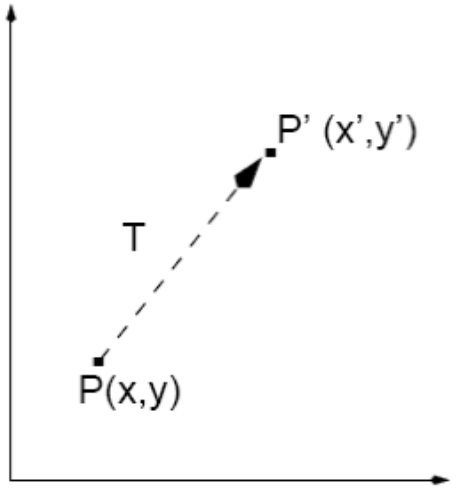


Table of contents

Geometric operations - Translation

- Moves a point to a new location by adding translation amounts to the coordinates of the point.



$$x' = x + dx, \quad y' = y + dy$$

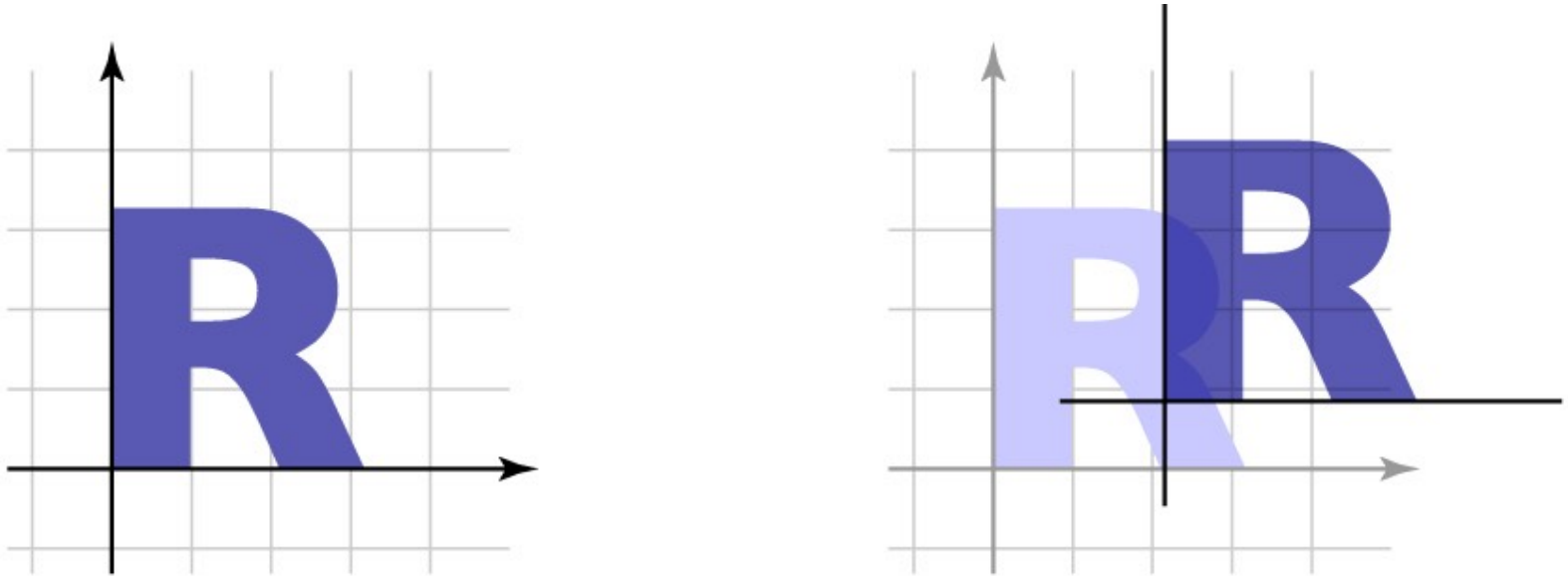
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} dx \\ dy \end{bmatrix}$$

$$P' = P + T$$

Affine Transformation - Translation

- Translation

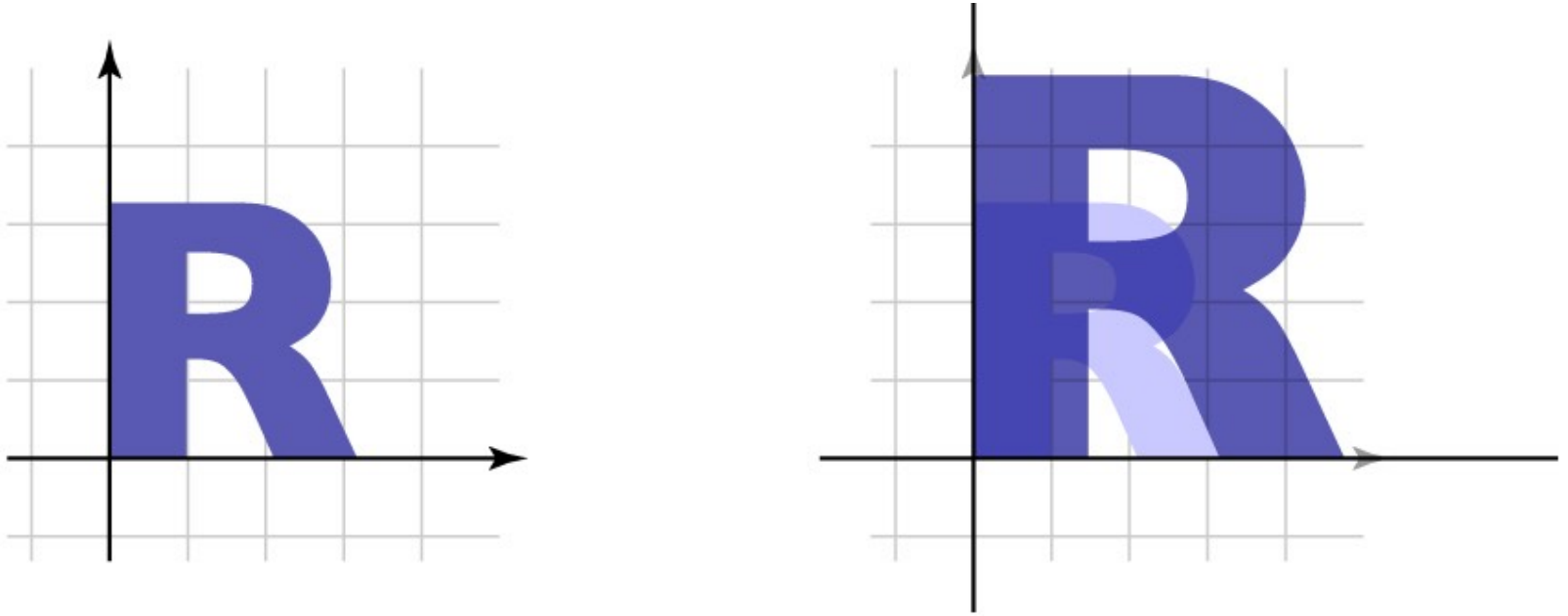
$$\begin{bmatrix} 1 & 0 & a_x \\ 0 & 1 & a_y \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 2.15 \\ 0 & 1 & 0.85 \end{bmatrix}$$



Geometric operations - Uniform Scale

- Uniform scale $\begin{bmatrix} s & 0 & a_x \\ 0 & s & a_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} sx + a_x \\ sx + a_y \end{bmatrix}$

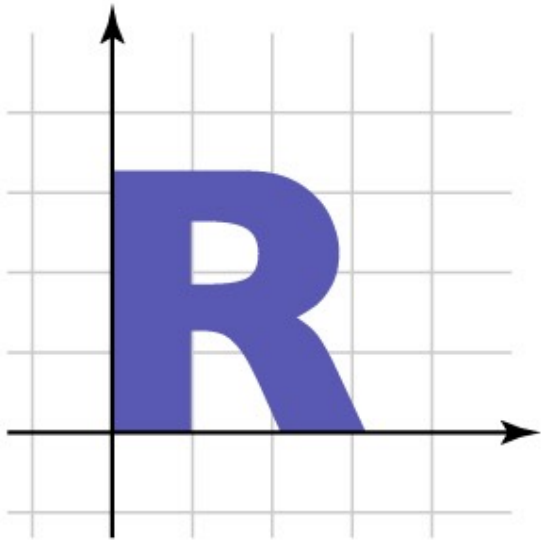
$$\begin{bmatrix} 1.5 & 0 & 0 \\ 0 & 1.5 & 0 \end{bmatrix}$$

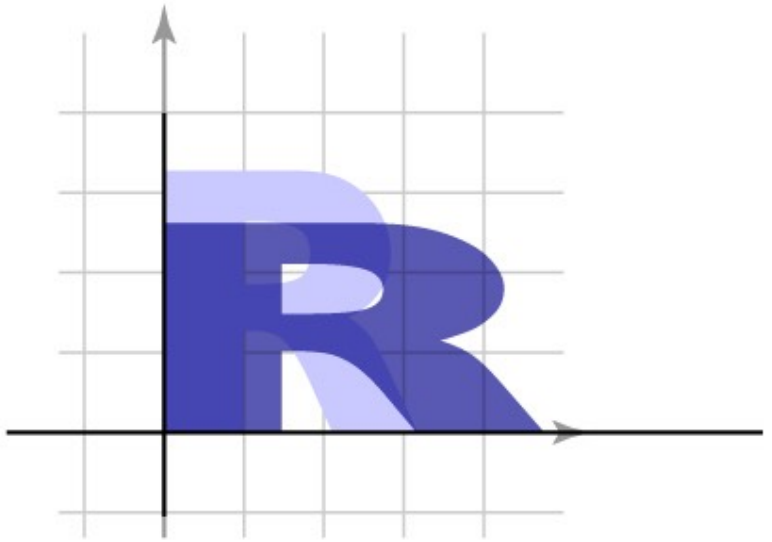


Geometric operations - Linear transformation

- Nonuniform scale $\begin{bmatrix} s_x & 0 & a_x \\ 0 & s_y & a_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x x + a_x \\ s_y x + a_y \end{bmatrix}$

$$\begin{bmatrix} 1.5 & 0 & 0 \\ 0 & 0.8 & 0 \end{bmatrix}$$



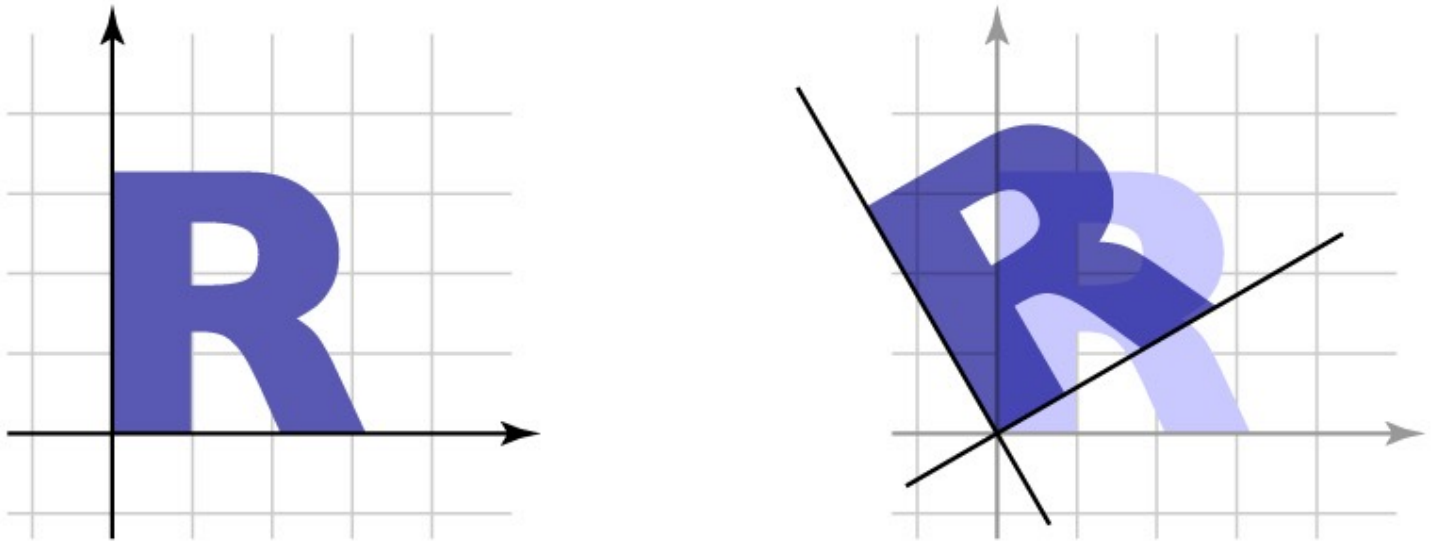


Geometric operations - Rotation

- Rotation

$$\begin{bmatrix} \cos\theta & -\sin\theta & a_x \\ \sin\theta & \cos\theta & a_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta + a_x \\ x\sin\theta + y\cos\theta + a_y \end{bmatrix}$$

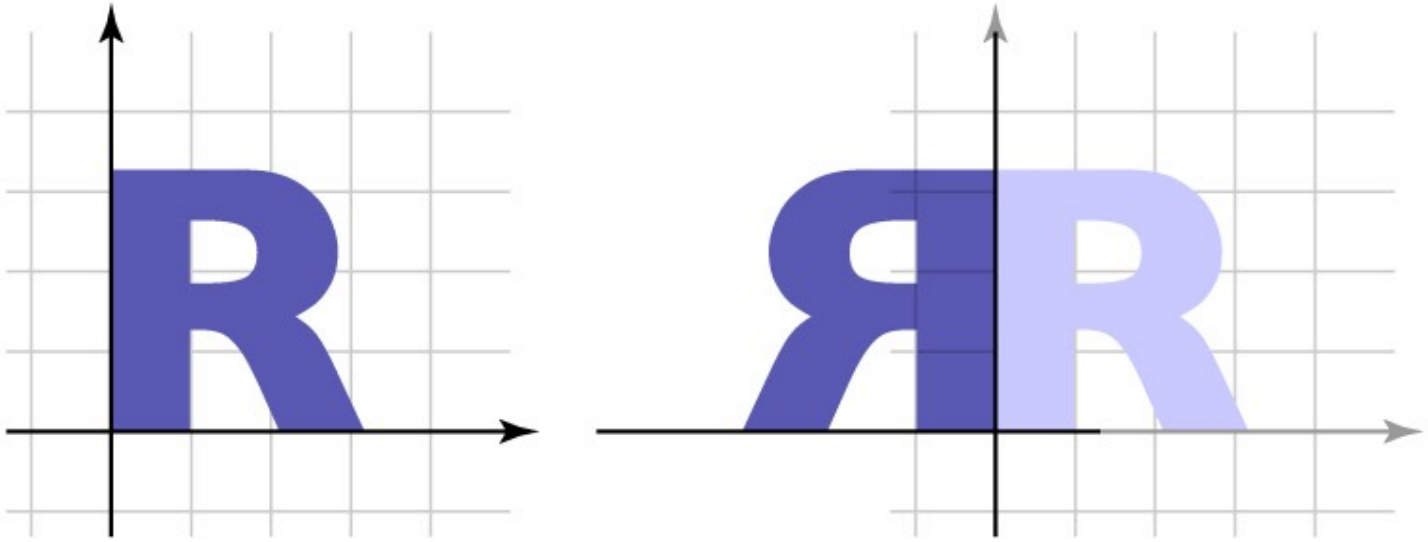
$$\begin{bmatrix} 0.86 & -0.5 & 0 \\ 0.5 & 0.866 & 0 \end{bmatrix}$$



Geometric operations - Reflection

- Reflection
  - can consider it a special case of nonuniform scale

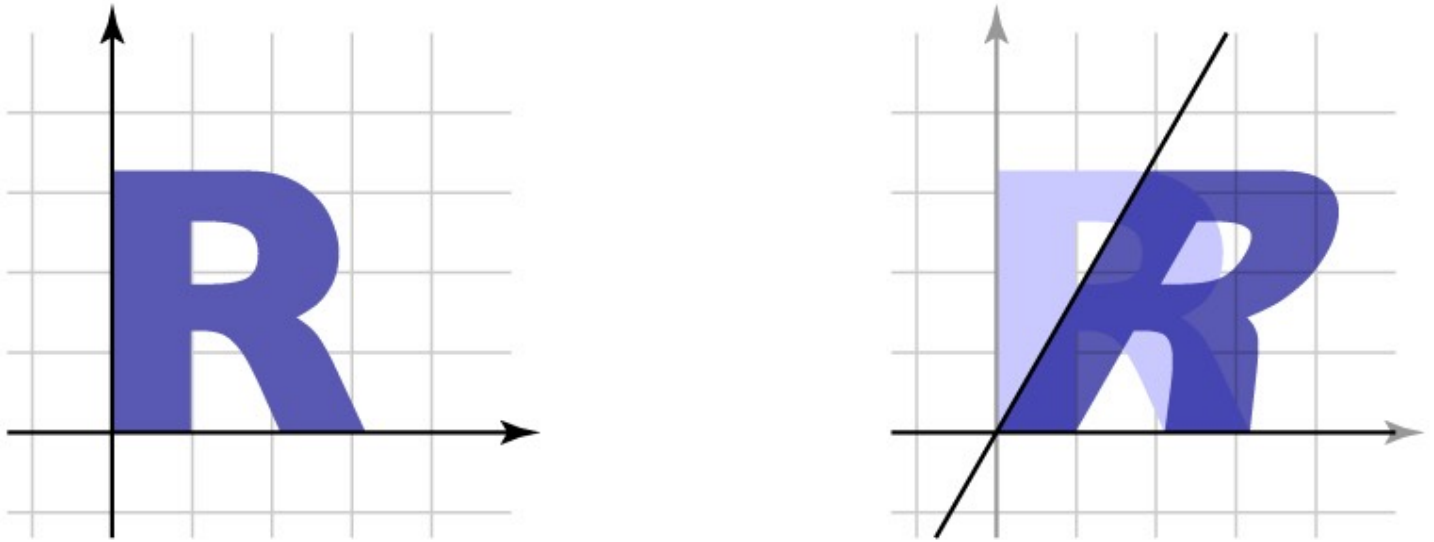
$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$



Geometric operations - Shear

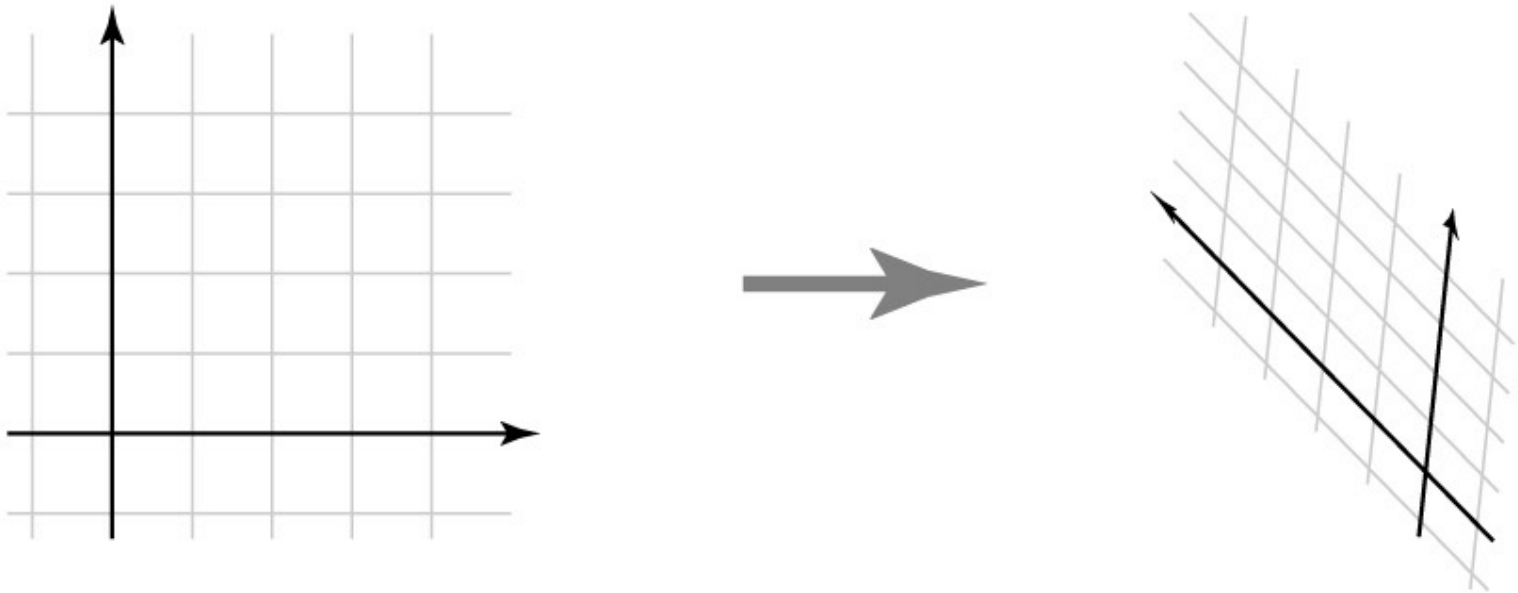
- Shear

$$\begin{bmatrix} 1 & a & a_x \\ 0 & 1 & a_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + ay + a_x \\ y + a_y \end{bmatrix}$$



Affine transformations

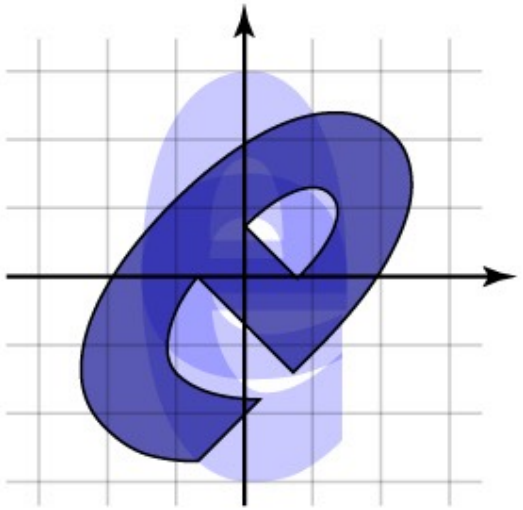
- The set of transformations we have been looking at is known as the "affine" transformations
  - straight lines preserved; parallel lines preserved
  - ratios of lengths along lines preserved (midpoints preserved)



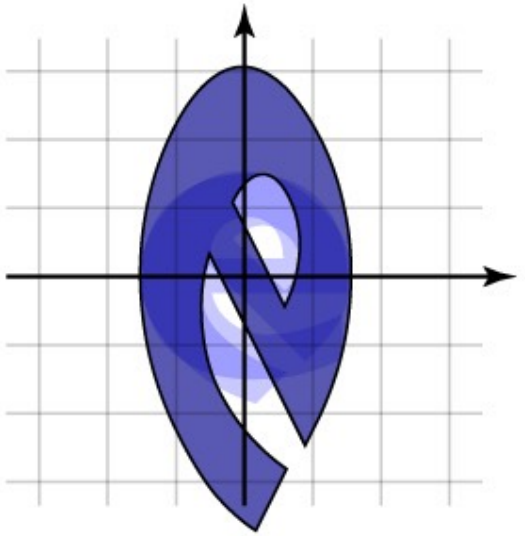
Composite affine transformations

- Another example

$$\begin{bmatrix} s_{11}\cos\theta & -s_{12}\sin\theta & a_x \\ s_{21}\sin\theta & s_{22}\cos\theta & a_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_{11}x\cos\theta - s_{12}y\sin\theta + a_x \\ s_{21}x\sin\theta + s_{22}y\cos\theta + a_y \end{bmatrix}$$



scale, then rotate

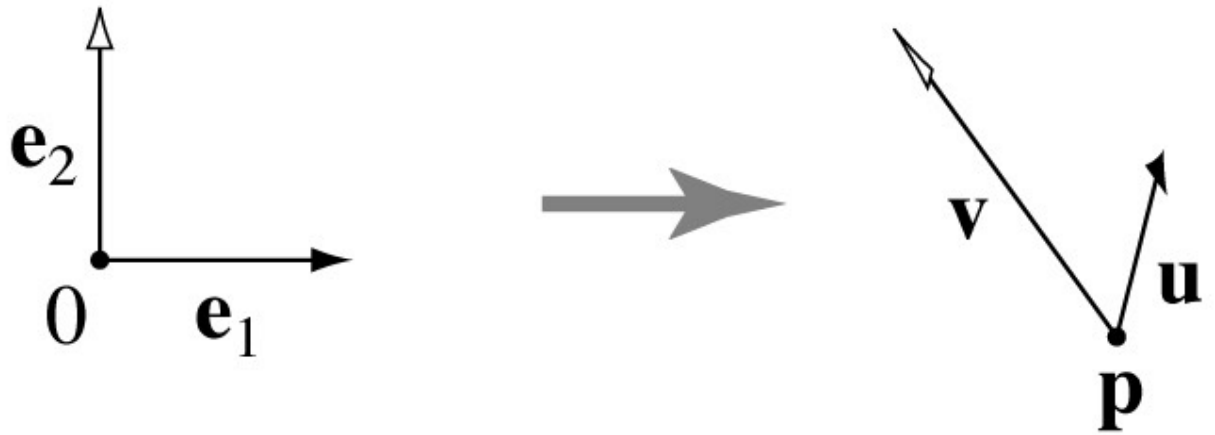


rotate, then scale

Affine change of coordinates

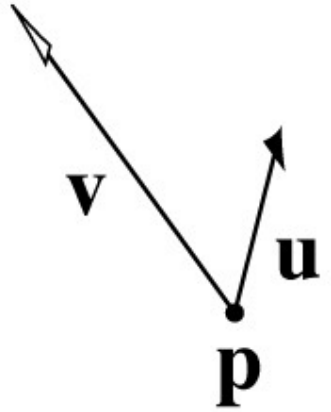
- Six degrees of freedom

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \end{bmatrix} \quad \text{or} \quad [u \quad v \quad p]$$



Affine change of coordinates

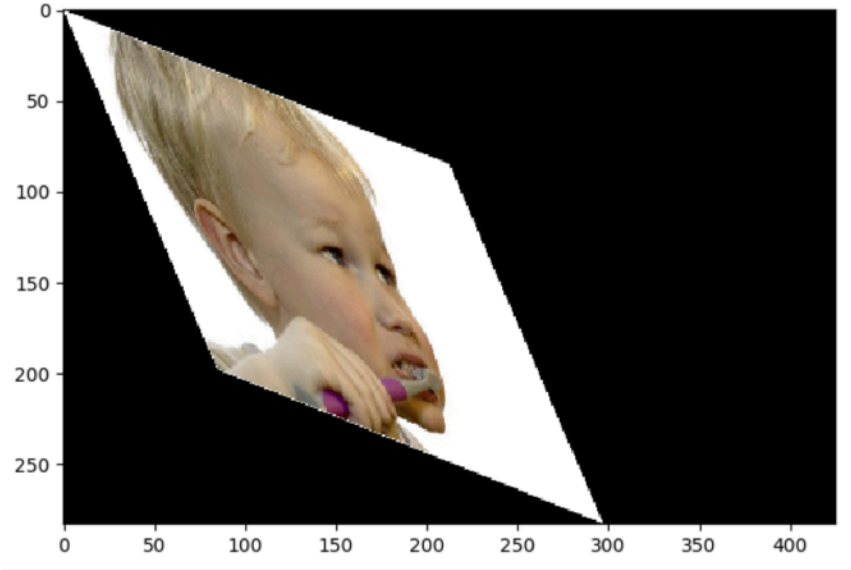
- Coordinate frame: point plus basis
- Interpretation: transformation changes representation of point from one basis to another
- "Frame to canonical" matrix has frame in columns
  - takes points represented in frame
  - represents them in canonical basis
  - e.g. [0 0], [1 0], [0 1]
- Seems backward but bears thinking about



$$[u \quad v \quad p]$$

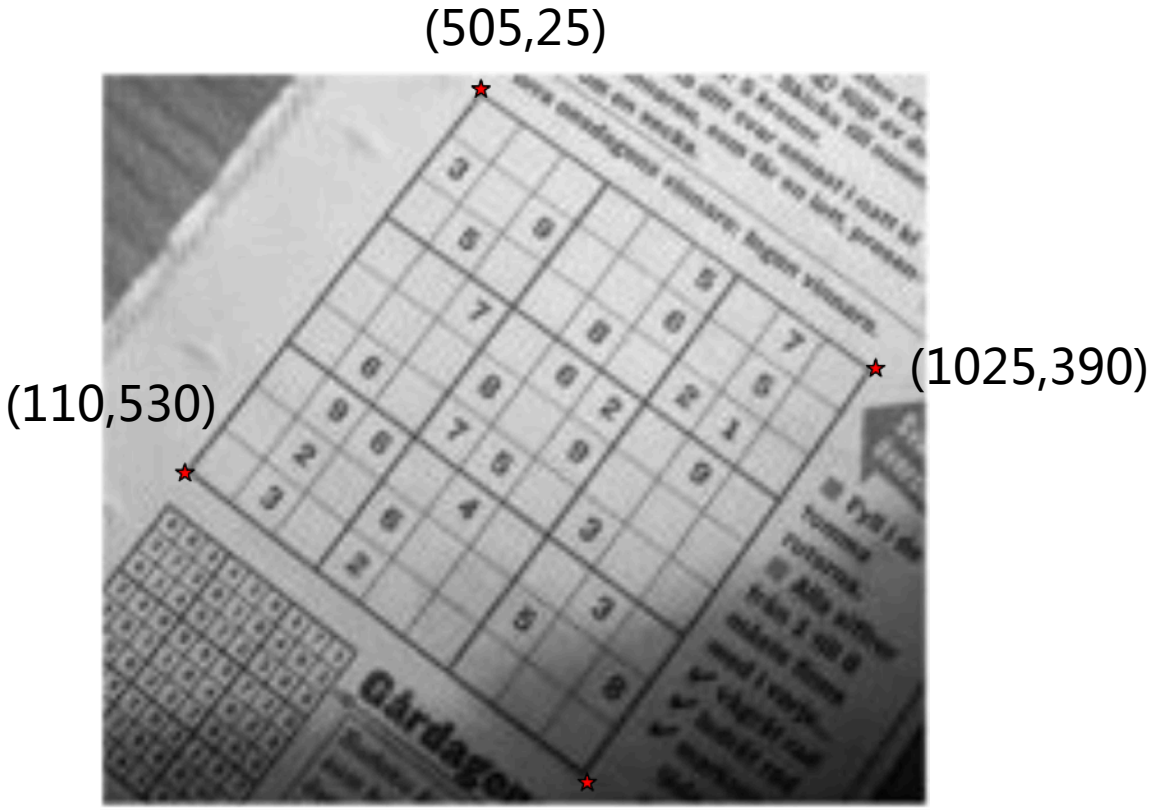
Affine change of coordinates

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('ChildBrushingTeeth.jpg',cv2.IMREAD_COLOR)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
(height,width,deep) = img.shape
matRotation = np.array(
 [[0.5,0.3,0],
  [0.2, 0.7, 0]])
dest_img= cv2.warpAffine(img,matRotation,(0,0))
plt.imshow(dest_img)
plt.figure(1)
plt.show()
```



Affine change of coordinates

```python
import cv2
import numpy as np
img = cv2.imread('sudoku.png',cv2.IMREAD_COLOR)
(height,width,deep) = img.shape
# lefttop, letbottom,rightbuttom
matSrc = np.float32([[505,25],[110,530],[1025,390]])
matDest = np.float32([[0,0],[0, 970],[1100,0]])
matAffine = cv2.getAffineTransform(matSrc,matDest)
dest_img = cv2.warpAffine(img,matAffine,(0,0))
cv2.imshow('img',dest_img)
cv2.waitKey(0)
```



Affine change of coordinates

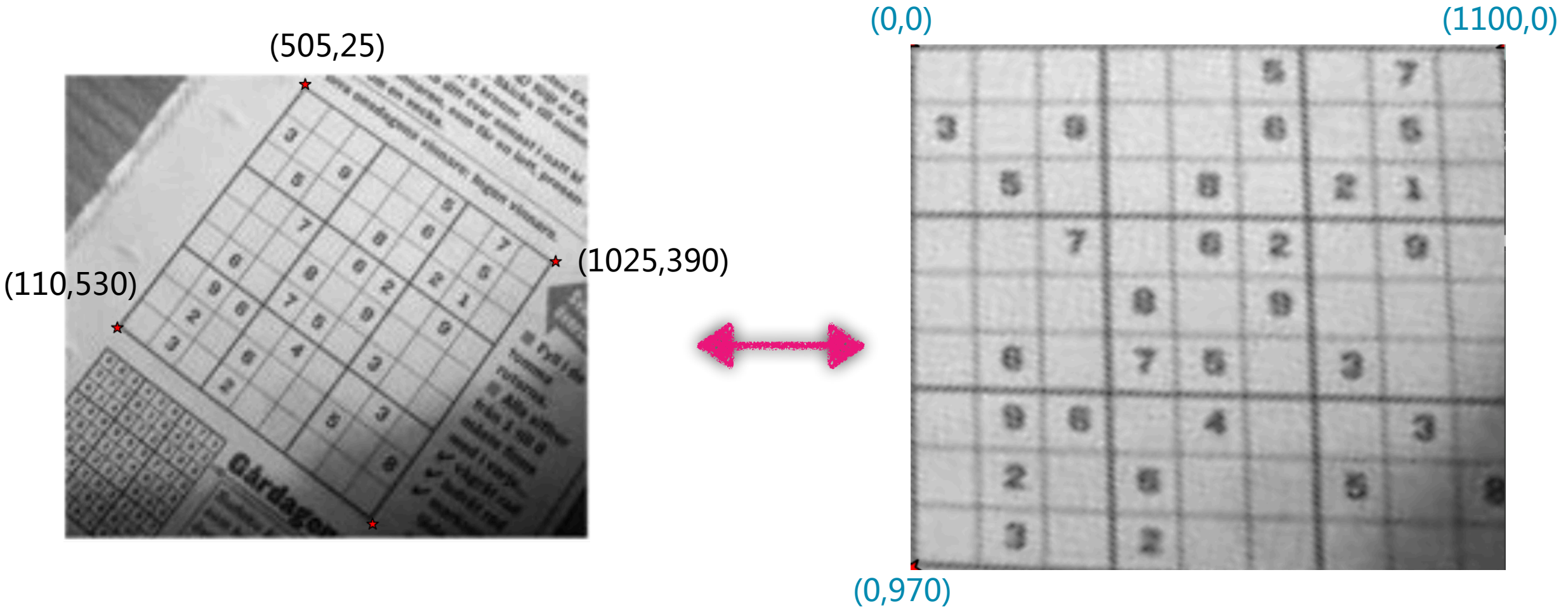
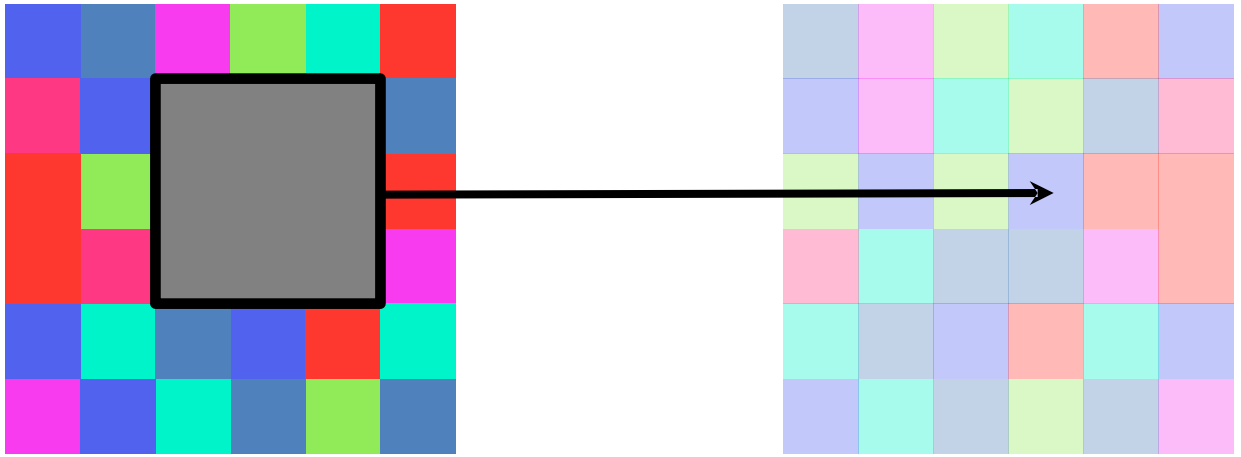
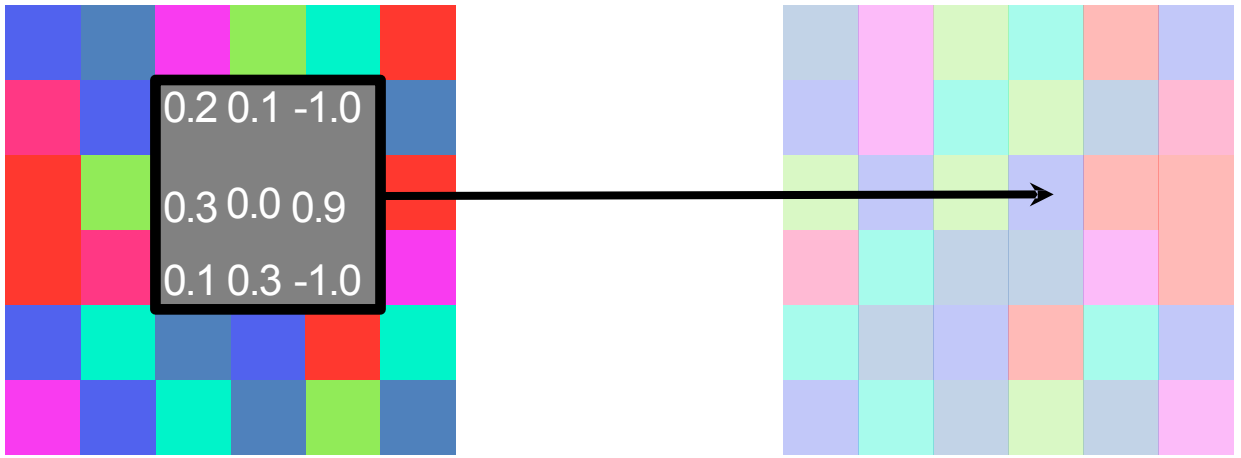


Table of contents

Convolution: Neighborhood Operations



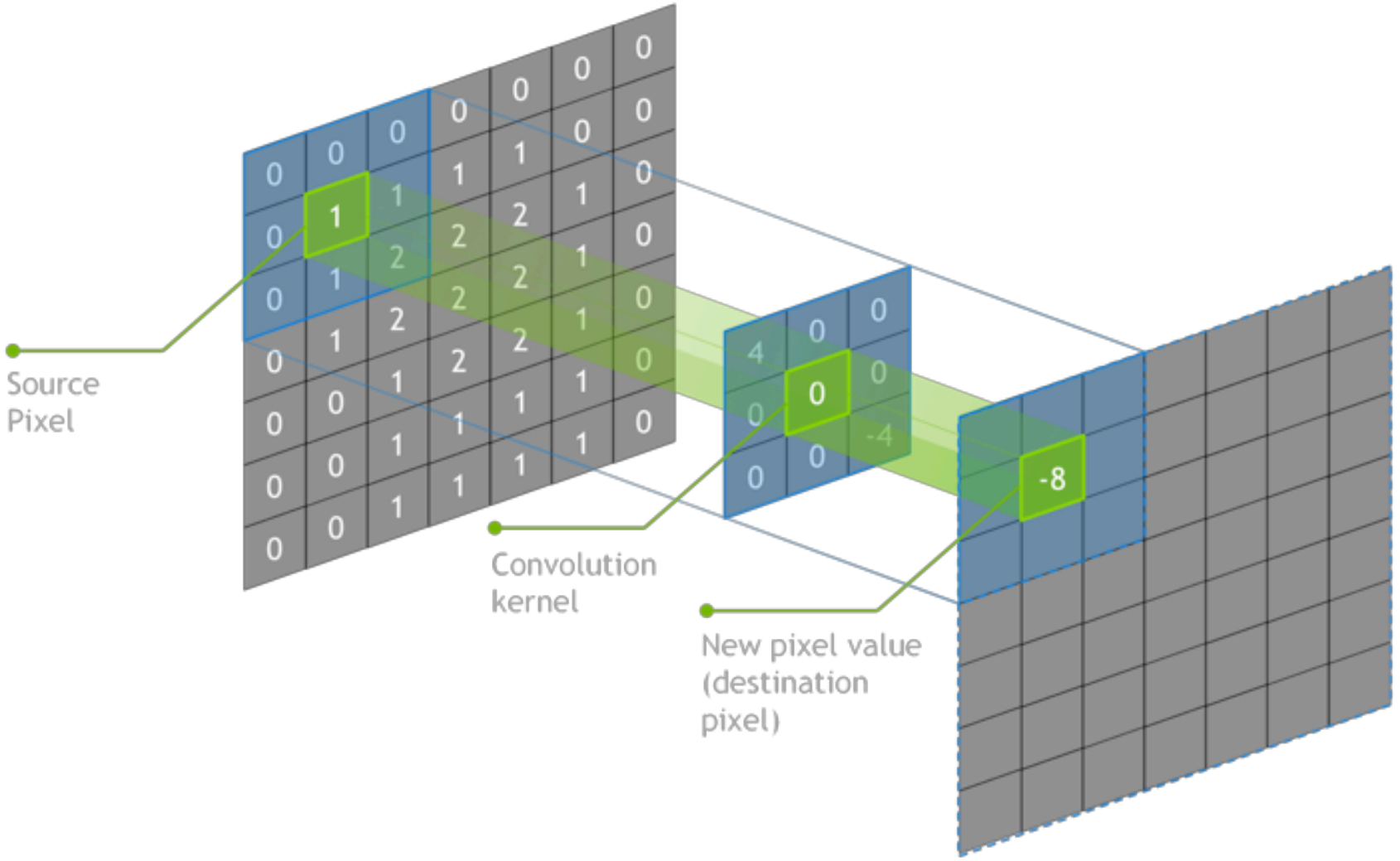
Convolution



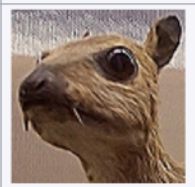
$$F = \begin{bmatrix} 0.2 & 0.1 & -1.0 \\ 0.3 & 0.0 & 0.9 \\ 0.1 & 0.3 & -1.0 \end{bmatrix}$$

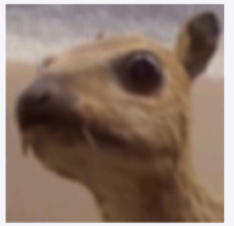
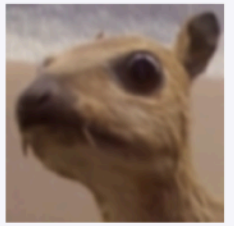
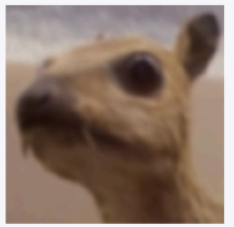
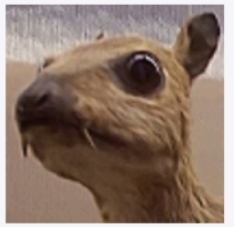
$$I' = F * I$$

Convolution images



Convolution Kernels

Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5 × 5 (approximation)	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
Unsharp masking 5 × 5 Based on Gaussian blur with amount as 1 and threshold as 0 (with no image mask)	$\frac{-1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

Thanks

FAQ时间