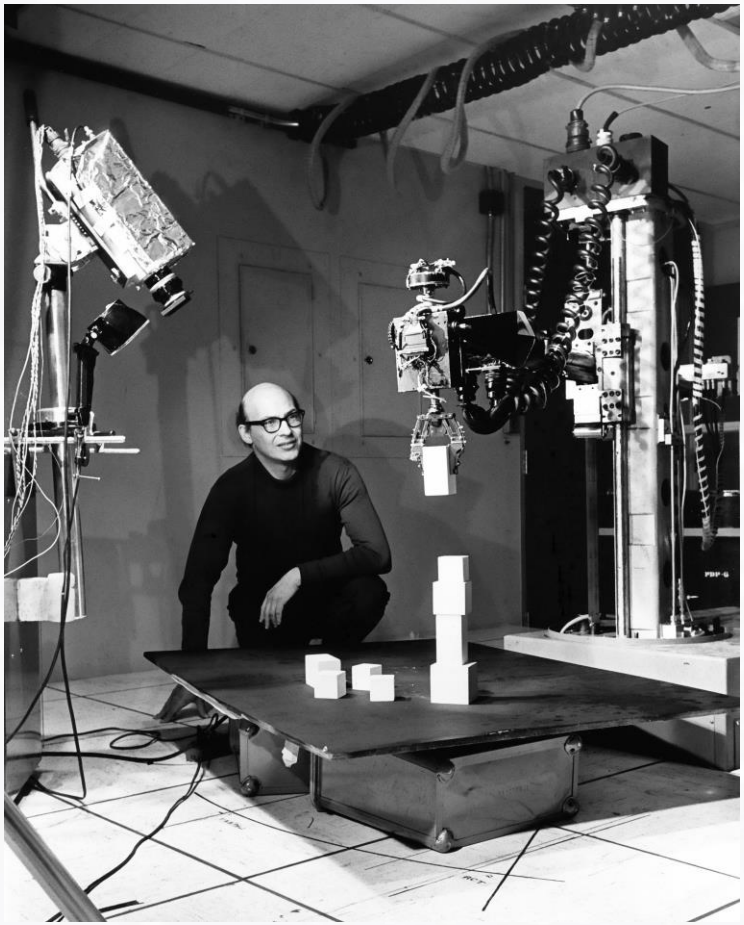
Elements of Classical Computer Vision

CS 410/510: CV & DL

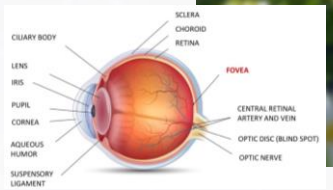
Outline

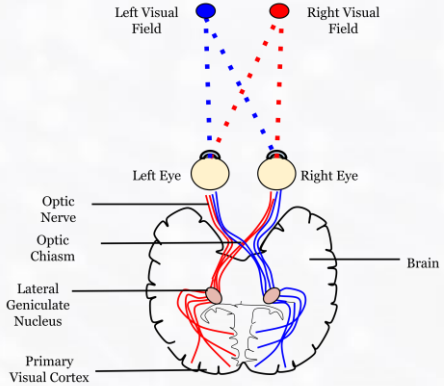
- Insights from the Visual System
- Brief History of CV
- Image Pre-Processing
- Convolution & Feature Extraction
- Edge and Keypoint Detection
- Descriptors
- Video Tracking with Correlation Filters
- Segmentation: *K-means*, *GMM*, *Graph Cut*
- Motion Estimation: *Optical Flow*
- Facial Recognition: *EigenFace*
- Real-Time Face Detection: *Viola-Jones*

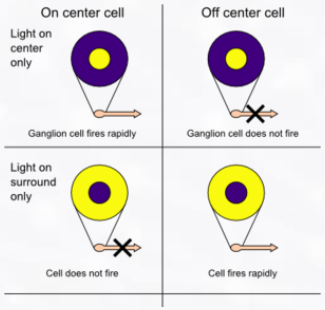


Insights from the Visual System

• Vision is our most acute and also our most studied sense. From the inception of Computer Vision (CV), the organization of the visual cortex has served as the inspiration for the most successfully deep learning networks.





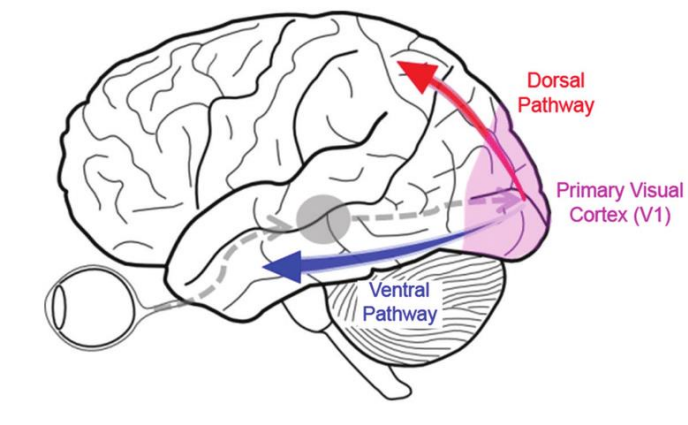


• What we see through our eyes is only a very small part of the world around us. At any given time, our eyes are sampling only a fraction of the surrounding light field. Even within this fraction, most of the resolution is dedicated to the center of gaze which has the highest concentration of *ganglion cells*.

• In the **eye**, a tiny pit located in the macula of the retina that provides the clearest vision of all. Only in the **fovea** are the layers of the retina spread aside to let light fall directly on the cones, the cells that give the sharpest image.

• Information processing in the visual system starts in the retina, where photo receptors convert light into electrical signals. There are generally two types of ganglion cells in the retina (see image), **on-center** and **off-center**.

Insights from the Visual System



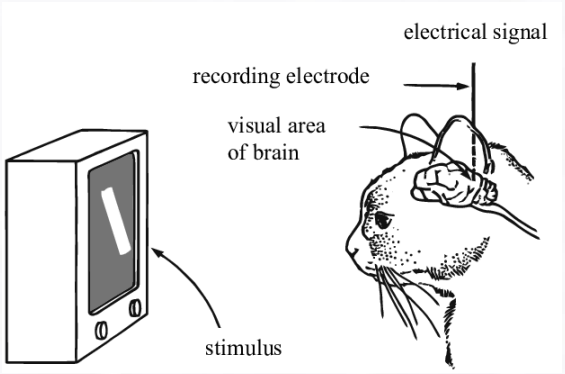
• From the retina, this sensory information travels the lateral geniculate nucleus (LGN) to the **visual cortex** (V1). V1 is known to process simple visual forms, such as edges and corners (see next slides).

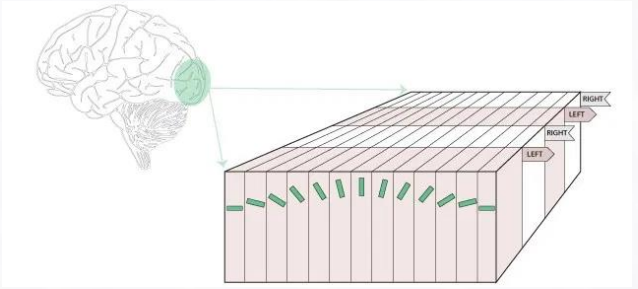
• V1 transmits information to two primary neural pathways, called the **ventral stream** and **the dorsal stream**.

• The ventral stream begins with V1, goes through visual area V2, then through visual area **V4** (processes intermediate visual forms, feature groups, etc.) and to the **inferior temporal cortex** (high-level object descriptions). The ventral stream, sometimes called the "**What Pathway**", is associated with form recognition and object representation. It is also associated with storage of long-term memory.

• The dorsal stream begins with V1, goes through Visual area V2, then to the dorsomedial area (DM/V6) and medial temporal area (MT/V5) and to the posterior parietal cortex. The dorsal stream, sometimes called the "**Where Pathway**" or "How Pathway", is associated with motion, representation of object locations, and control of the eyes and arms.

Insights from the Visual System





• In perhaps the most influential set of experiments in the history of CV, **David Hubel** and **Torsten Wiesel** (Nobel prize recipients in 1981) laid the groundwork for understanding the **hierarchical nature of the mammalian visual system** by demonstrating how complex representations of visual information are built from simple cells in the primary visual cortex.

• In one experiment (1959) they inserted a microelectrode into the primary visual cortex of an anesthetized cat. They then projected patterns of light and dark on a screen in front of the cat. They found that some neurons fired rapidly when presented with lines at one angle, while others responded best to another angle. Some of these neurons responded to light patterns and dark patterns differently. Hubel and Wiesel called these neurons "simple cells." Still other neurons, which they termed **complex cells**, detected edges regardless of where they were placed in the receptive field of the neuron.

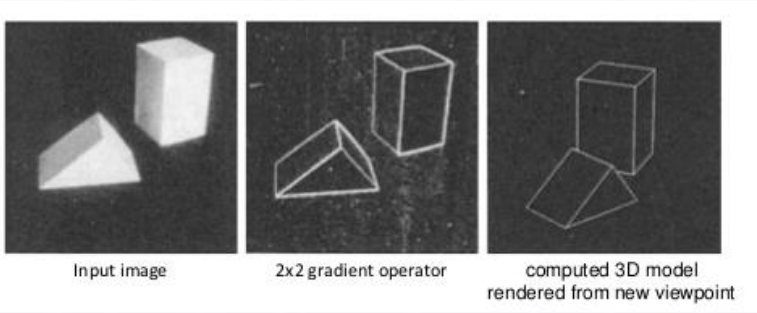
• The visual information relayed to V1 is not coded in terms of spatial (or optical) imagery but rather are better described as **edge detection**. In this way, each cortical neuron in the visual cortex can be thought of as a visual feature detector, which only becomes active when it receives inputs above a certain threshold for its preferred feature in a particular patch of the visual field.

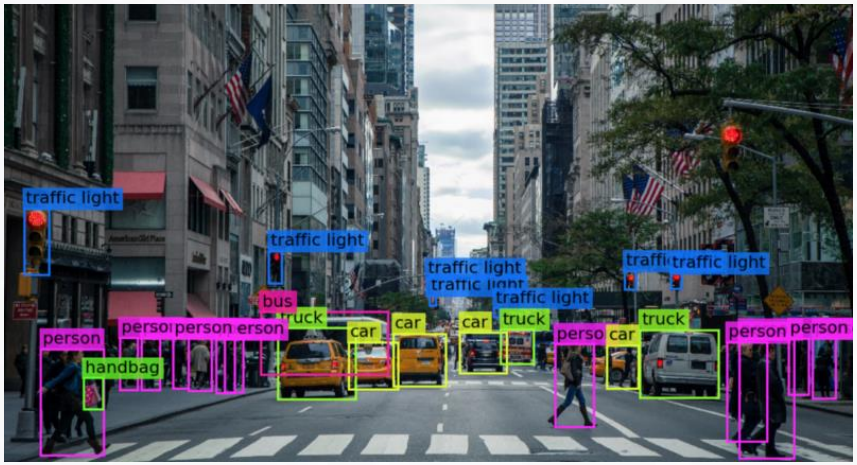
Brief History of CV

• When CV began in the early 1970s, it was initially viewed as the visual perception component of an ambitious agenda to mimic human intelligence and to endow robots with intelligent behavior.

• At the time, it was believed by some of the early pioneers in AI and robotics than solving the "visual input" problem would be an easy step along the path to solving AGI.

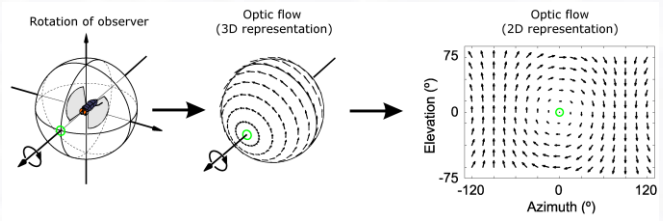
• Famously, Minsky at MIT asked his undergraduate student to "spend the summer linking a camera to a computer and getting the compute to describe what it saw" – five decades later, we are still working on this problem.

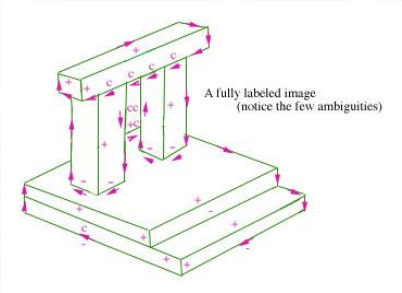




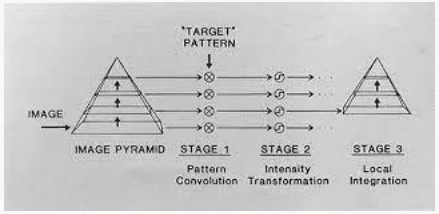
Brief History of CV

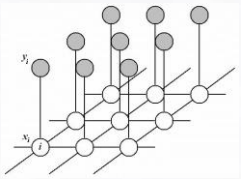
• 1970s: Inception of CV; high-level attempt to recover 3D structure of the world from images as steppingstone toward systems of visual understanding; early codification of **optical flow, edge extraction, motion estimation, polyhedral modeling.**





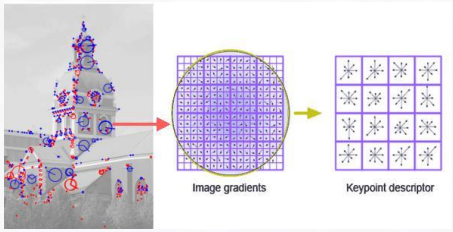
• 1980s: More attention on mathematical rigor and quantitative image and scene analysis. Development of **image pyramids**; stereo image analysis; **Canny edge detection**; snakes; incorporation of **Markov Random Fields**; **Kalman filters**.

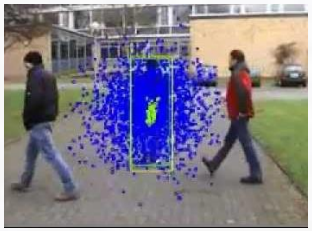


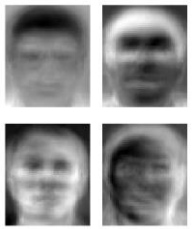


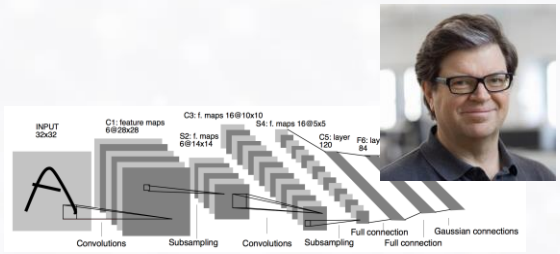
Brief History of CV

• 1990s: Physics-based vision, optical flow methods; multi-view stereo algorithms, including stereo correspondence; tracking algorithms (particle filters); image segmentation (normalized cuts); facial recognition, statistical learning (Eigenface); feature extraction, invariance (SIFT); invention of CNNs.

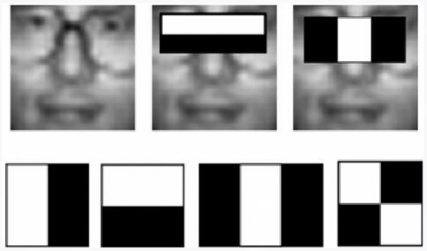


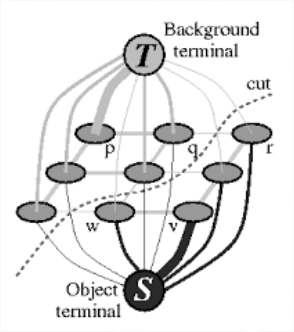


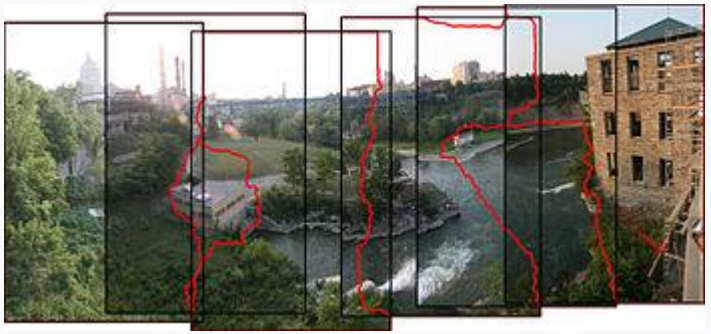




• 2000s: Increased interplay between CV and graphics; image stitching; computational photography algorithms (HDR image capture); texture synthesis; BoW in CV (representation of visual features as words); real-time face detection (Viola-Jones); interactive segmentation (graph cuts).

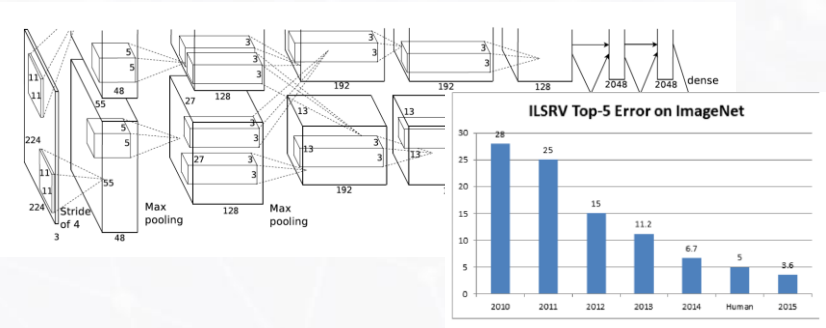


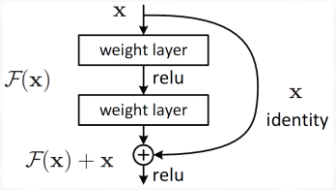


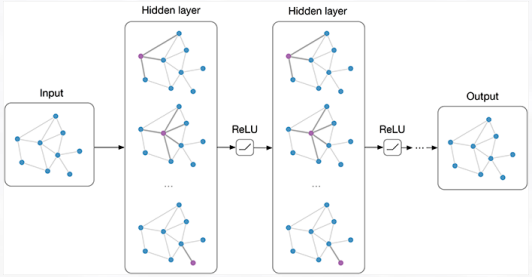


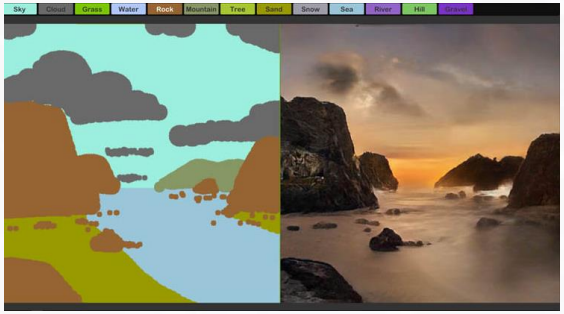
Brief History of CV

• 2010s: Dominance of DL in CV (**AlexNet**); efficient training of very deep DL models (**ResNet**) introduction of large-scale image datasets (**ImageNet**); incremental development of CNN architectures (**VGG**, **Inception**); generative models (**GAN**s); real-time localization (**YOLO**); object localization (**R-CNN**); pixel-level segmentation (**Mask R-CNN**); CNN architecture optimization (**NAS**); emergence of compact DL models (**SqueezeNet**); computational creativity (GauGAN); refined hierarchical models (**CapsNet**); graph convolution nets (**GCN**s).









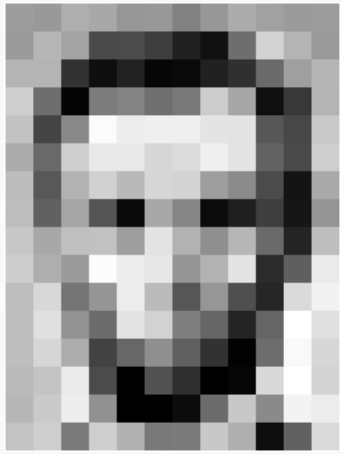
http://nvidia-research-mingyuliu.com/gaugan

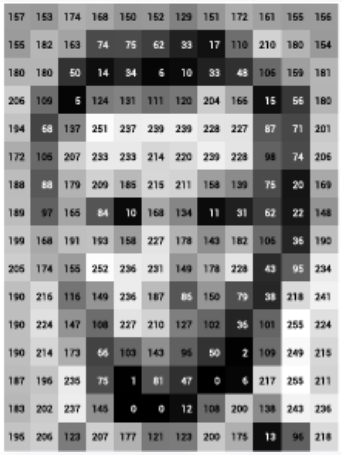
Pre-Processing

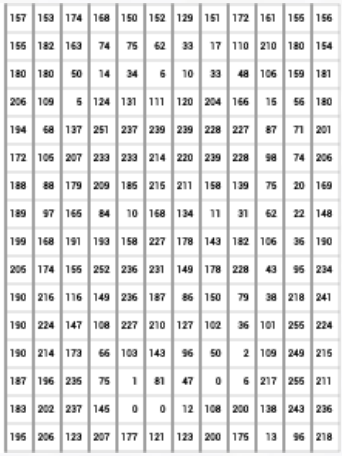
• Image data are almost always **preprocessed** prior to ingestion into a model; such preprocessing steps can benefit feature extraction, model performance/convergence, etc.

• The general goal of preprocessing is to remove as much unwanted variation in the data as possible while retaining the aspects of the image that are critical to the task at hand. Preprocessing must be applied with care.

• Note that the **choice of preprocessing technique(s) can have a large influence** on the performance of a CV algorithm. Many CV algorithms are sensitive to the application of preprocessing; oftentimes preprocessing improves performance and/or the stability of various CV algorithms.







Pre-Processing

Image normalization

There are many related techniques for image normalization. The (2) most common being:

$$x_{ij} \leftarrow \frac{x_{ij}}{I_{\max}} \quad \text{and} \quad x_{ij} \leftarrow \left(x_{ij} - I_{\min}\right)\frac{newMax - newMin}{I_{\max} - I_{\min}} + newMin$$

Where $I_{max}(I_{min})$ denote the maximum(minimum) pixel intensity in the image; (newMin, newMax) denotes the pixel intensity range of the transformed image.

Image standardization (also: whitening)

Image standardization is applied by subtracting the mean intensity and dividing by the standard deviation of pixel intensities (wrt each channel for an RGB image):

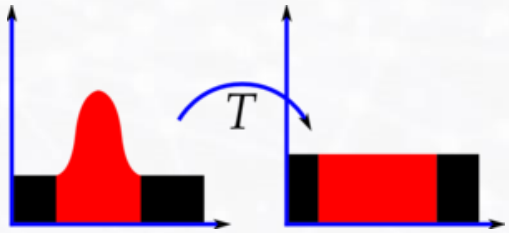
$$x_{ij} \leftarrow \frac{x_{ij} - I_{\mu}}{I_{\sigma}}$$

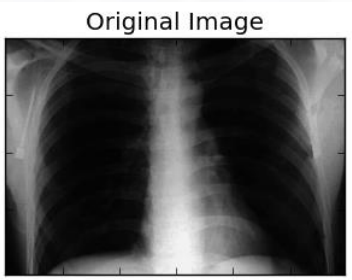
- Standardization helps ensure that each pixel has a similar data distribution.

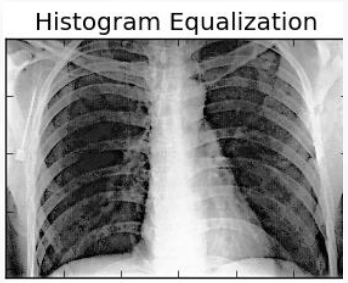
Pre-Processing

Histogram Equalization

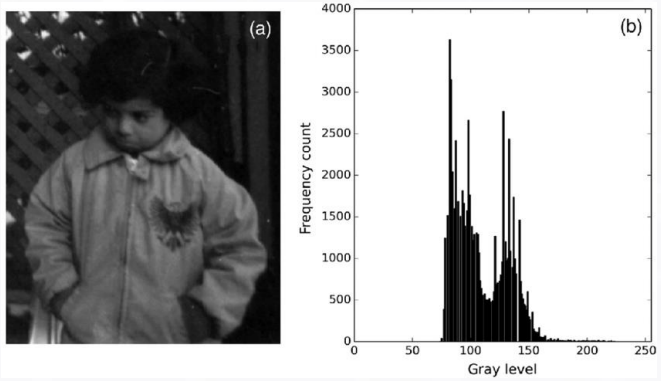
Histogram equalization (HA) is used to modify the statistics of the intensity values so that all of their moments take predefined values. HA forces the distribution of pixel intensities to be flat.

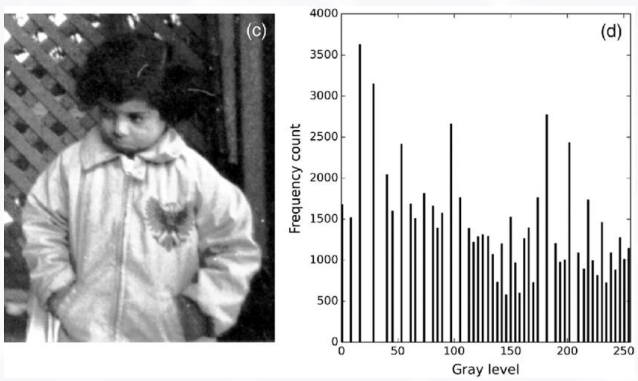






• HA is useful in images with backgrounds and foregrounds that are both bright or dark. In medical imaging, this can lead to better views of, say, bone structure in x-rays, and to better detail in over and under-exposed images. (+) HA is computationally cheap; (-) HA is indiscriminate, may increase signal/amplify noise.





Pre-Processing

Histogram Equalization

(1) Compute histogram of the original intensities **h** (for 8-bit image **k** here ranges over {0,1,2,…, $K = 255$}):

$$h_k = \sum_{i=1}^{I} \sum_{j=1}^{J} \delta\left[x_{ij} - k \right]$$

where $\delta[\cdot]$ denotes the Dirac delta function (i.e., when argument is zero, $\delta = 1$; otherwise $\delta = 0$); I and J are the image dimensions (check that you understand this is simply a mathematical formulation of the histogram of a greyscale image).

Pre-Processing

Histogram Equalization

(1) Compute histogram of the original intensities **h** (for 8-bit image **k** here ranges over {0,1,2,…, $K = 255$}):

$$h_k = \sum_{i=1}^{I} \sum_{j=1}^{J} \delta\left[x_{ij} - k \right]$$

where $\delta[\cdot]$ denotes the Dirac delta function (i.e., when argument is zero, $\delta = 1$; otherwise $\delta = 0$); I and J are the image dimensions (check that you understand this is simply a mathematical formulation of the histogram of a greyscale image).

(2) Determine the cumulative proportion c of pixels that are less than or each to each intensity level:

$$c_k = \frac{\sum_{l=1}^{k} h_l}{IJ}$$

Pre-Processing

Histogram Equalization

(1) Compute histogram of the original intensities **h** (for 8-bit image **k** here ranges over {0,1,2,…, $K = 255$}):

$$h_k = \sum_{i=1}^{I}\sum_{j=1}^{J}\delta\left[x_{ij} - k\right]$$

where $\delta[\cdot]$ denotes the Dirac delta function (i.e., when argument is zero, $\delta = 1$; otherwise $\delta = 0$); I and J are the image dimensions (check that you understand this is simply a mathematical formulation of the histogram of a greyscale image).

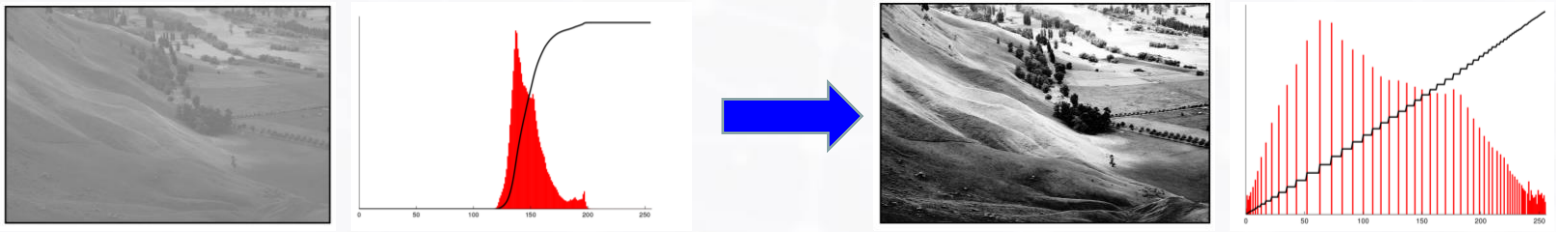
(2) Determine the cumulative proportion c of pixels that are less than or each to each intensity level:

$$c_k = \frac{\sum_{l=1}^{k} h_l}{IJ}$$

(3) Finally, use the cumulative histogram as a look up table to compute the transformed value so that:

$$x_{ij} \leftarrow K \cdot c_{x_{ij}} \quad \text{(where } K \text{ is the max intensity, e.g., } K = 255\text{)}$$

• For instance, if $x_{ij} = 90$ (pixel has intensity 90), and suppose $c_{90} = 0.29$, then the transformed pixel value would be: $K \cdot c_{x_{ij}} = 255 \cdot 0.29 = 74$.



Convolution and Feature Extraction

Convolution

• A **convolution** is a mathematical operation of two functions (e.g., f and g) that produces A third function: $(f * g)$ that expresses how the shape of one is modified by the other.

• More formally, the convolution (in continuous domains) is defined as the integral of the product of the two functions; one can conceptualize f as a **signal** and g as a "windowed" sample, i.e., **filter** (*cf.* signal processing):

$$\boxed{(f * g)(t) := \int f(\tau) g(t - \tau) d\tau}$$

Notice that if $f(t)$ is a unit impulse $\delta(t)$, we get: $(\delta * g)(t) := \int \delta(\tau) g(t - \tau) d\tau = g(t)$. The inverse of the convolution operation is known as **deconvolution**.

Convolution and Feature Extraction

Convolution

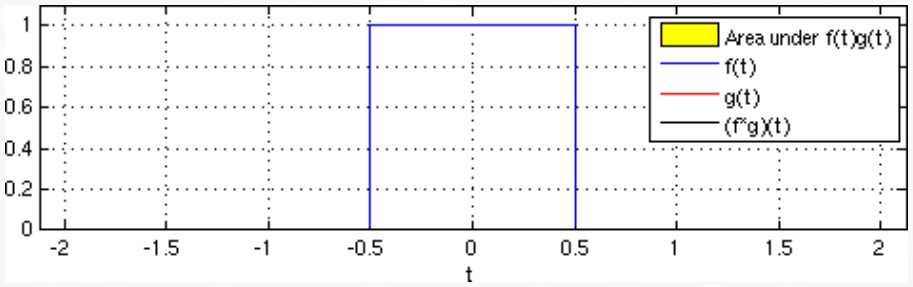
• A **convolution** is a mathematical operation of two functions (e.g., f and g) that produces A third function: $(f * g)$ that expresses how the shape of one is modified by the other.

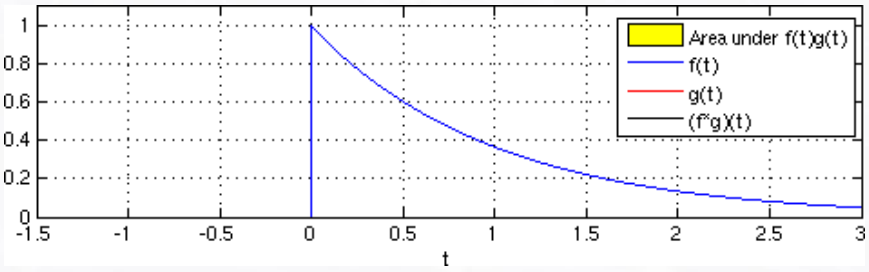
• More formally, the convolution (in continuous domains) is defined as the integral of the product of the two functions; one can conceptualize f as a **signal** and g as a "windowed" sample, i.e., **filter** (cf. signal processing):

$$(f * g)(t) := \int f(\tau) g(t - \tau) d\tau$$

Notice that if $f(t)$ is a unit impulse $\delta(t)$, we get: $(\delta * g)(t) := \int \delta(\tau) g(t - \tau) d\tau = g(t)$. The inverse of the convolution operation is known as **deconvolution**.

• At a high-level, one can think of the resultant convolution waveform $(f * g)$ as the response signal when we sample f using the filter g.

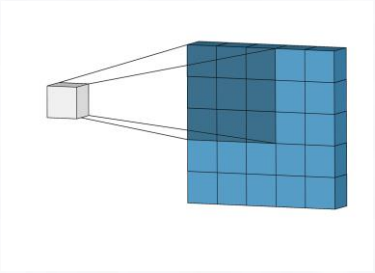




Convolution and Feature Extraction

Convolution

- In CV, we primarily consider convolution operations in discrete domains.



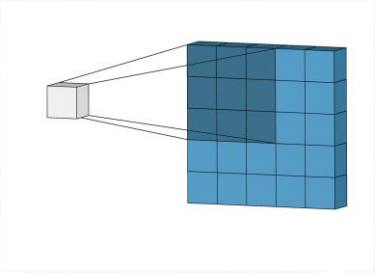
- Given an image $\mathbf{X}$, with individual pixel intensities x_{ij}, the 2D convolution of $\mathbf{X}$ with a filter F with entries f_{mn} where $m \in \{-M, \ldots, M\}$ and $\mathrm{n} \in \{-N, \ldots, N\}$ amounts to computing:

$$x_{ij} \leftarrow \sum_{m=-M}^{M} \sum_{n=-N}^{N} x_{i-m,\, j-n}\, f_{m,n}$$

Convolution and Feature Extraction

Convolution

• In CV, we primarily consider convolution operations in discrete domains.

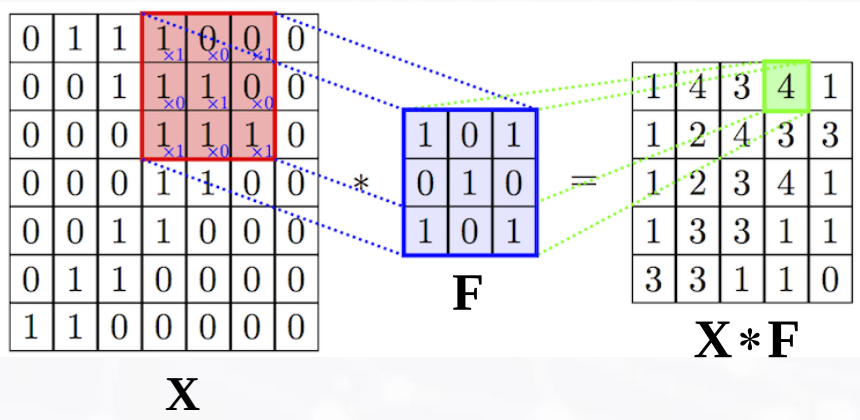


• Given an image **X**, with individual pixel intensities x_{ij}, the 2D convolution of **X** with a filter F with entries f_{mn} where $m \in \{-M, \ldots, M\}$ and $\mathrm{n} \in \{-N, \ldots, N\}$ amounts to computing:

$$x_{ij} \leftarrow \sum_{m=-M}^{M} \sum_{n=-N}^{N} x_{i-m,\, j-n}\, f_{m,n}$$

In the animation above, for instance, $\{-M, \ldots, M\} = \{-1,0,1\}$, and $\{-M, \ldots, M\} = \{-1,0,1\}$. (We deal with issues of **padding**, **stride**, etc., later).

• Here is a numerical example of 2D convolution:



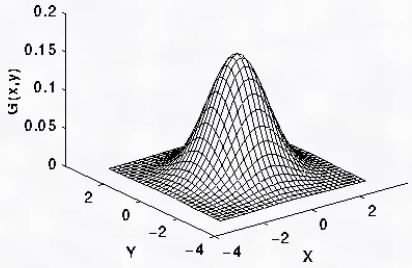
Convolution and Feature Extraction

Gaussian Filter

- One can introduce common filter types with respect to the discrete convolution operation.

Define the **2D Gaussian filter**:

$$g(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



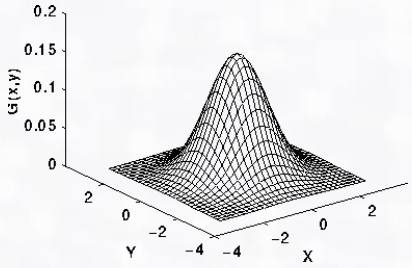
Convolution and Feature Extraction

Gaussian Filter

- One can introduce common filter types with respect to the discrete convolution operation.

Define the **2D Gaussian filter**:

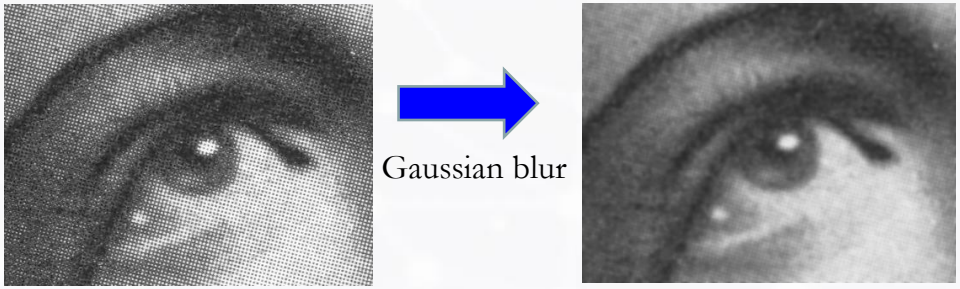
$$g(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



- Here is an example of discretized 5×5 Gaussian filter ($\sigma = 1$, with binning applied):

$$\underbrace{\frac{1}{273}}_{\text{normalization constant}} \cdot \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

- Applying a Gaussian filter to an image has the effect of reducing noise; oftentimes Gaussian filters are used in CV as a pre-processing step to enhance image structure at varying scales. Notice that because the Gaussian filter is **isotropic** it is <u>not orientation-selective</u>.

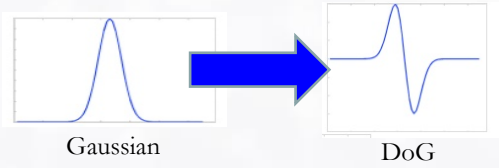


Convolution and Feature Extraction

Derivative of Gaussian Filter

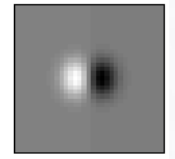
• Just as one can use a Gaussian filter to blur an image and hence remove pixilation artifacts, the **derivative of a Gaussian** (DoG) filter can be used for basic edge detection.

Consider the partial derivatives of the Gaussian filter:



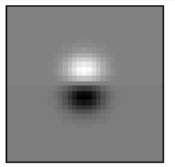
$$g_x = \frac{\partial g}{\partial x} g(x,y) = \frac{-x}{N 2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} = \frac{-x \cdot g(x,y)}{N'\sigma^2}$$

$$\mathbf{g}_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$



$$g_y = \frac{\partial g}{\partial y} g(x,y) = \frac{-y}{N 2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} = \frac{-y \cdot g(x,y)}{N'\sigma^2}$$

$$\mathbf{g}_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

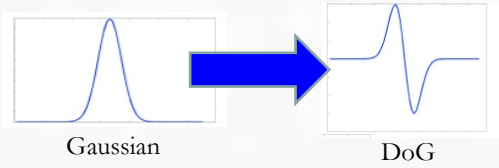


Convolution and Feature Extraction

Derivative of Gaussian Filter

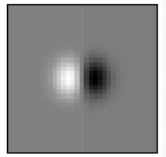
• Just as one can use a Gaussian filter to blur an image and hence remove pixilation artifacts, the **derivative of a Gaussian** (DoG) filter can be used for basic edge detection.

Consider the partial derivatives of the Gaussian filter:



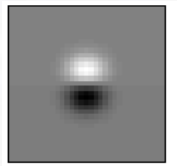
$$g_x = \frac{\partial g}{\partial x} g(x,y) = \frac{-x}{N 2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} = \frac{-x \cdot g(x,y)}{N'\sigma^2}$$

$$\mathbf{g}_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

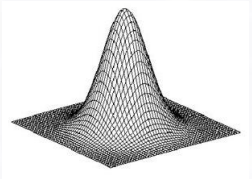


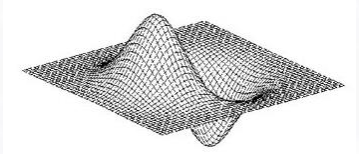
$$g_y = \frac{\partial g}{\partial y} g(x,y) = \frac{-y}{N 2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} = \frac{-y \cdot g(x,y)}{N'\sigma^2}$$

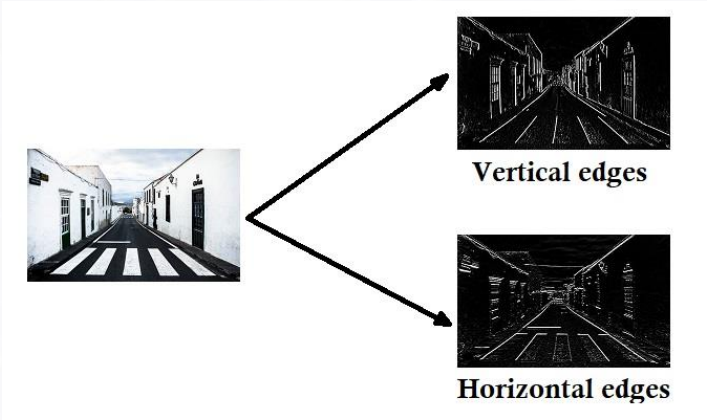
$$\mathbf{g}_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$



• On their own, the DoG filters: $\boldsymbol{g_x}$ and $\boldsymbol{g_y}$ provide vertical and horizontal edge detectors, respectively.







Convolution and Feature Extraction

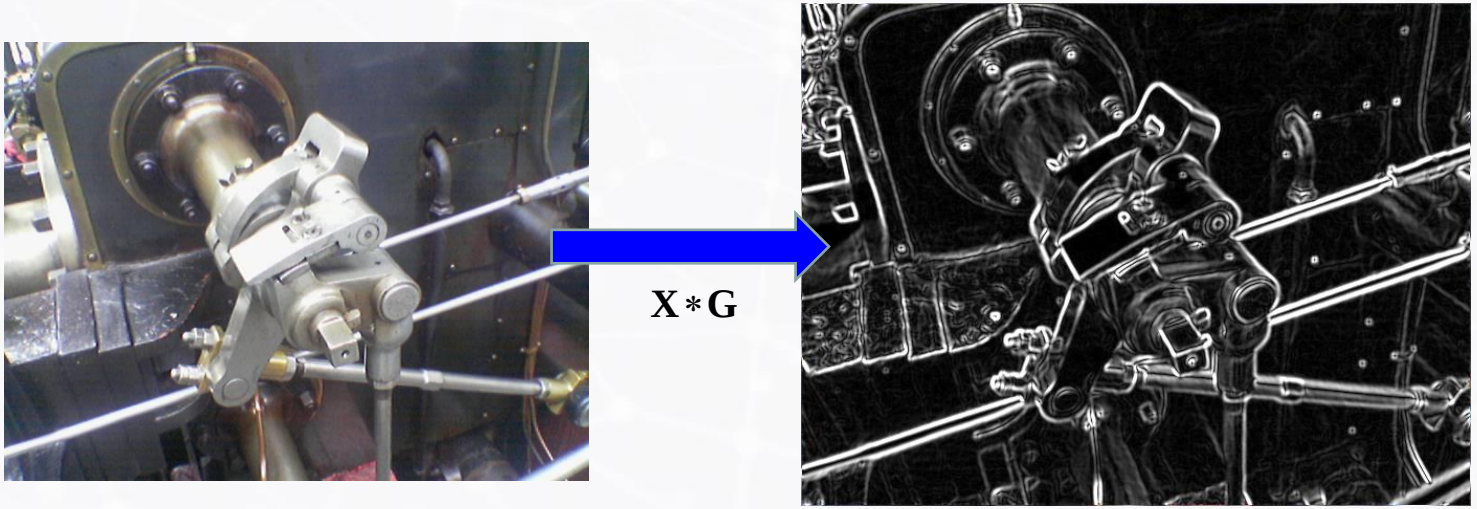
Derivative of Gaussian Filter

- On their own, the DoG filters: $\boldsymbol{g_x}$ and $\boldsymbol{g_y}$ provide vertical and horizontal edge detectors, respectively.

$$\mathbf{g}_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad \mathbf{g}_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

- In combination, one can create a **Sobel filter** (edge detector), by defining the filter as:

$$\mathbf{X} * \mathbf{G} = \sqrt{\left(\mathbf{X} * \mathbf{g}_x\right)^2 + \left(\mathbf{X} * \mathbf{g}_y\right)^2}$$



Convolution and Feature Extraction

Laplacian Filter

• The second derivative (i.e., the Laplacian operator) of the Gaussian filter gives rise to the **Laplacian filter**, defined:

$$\nabla^2 g(x,y) = g_{xx} + g_{yy}$$

Convolution and Feature Extraction

Laplacian Filter

• The second derivative (i.e., the Laplacian operator) of the Gaussian filter gives rise to the **Laplacian filter**, defined:

$$\nabla^2 g(x,y) = g_{xx} + g_{yy}$$

$$g(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Computing each second partial derivative of the Gaussian yields:

$$g_{xx} = \frac{\partial g \partial g}{\partial x^2} g(x,y) = \frac{\partial g}{\partial x} \frac{-x \cdot g(x,y)}{N'\sigma^2} = \frac{-g(x,y)}{N'\sigma^2} + \frac{x^2 \cdot g(x,y)}{N''\sigma^4} = \frac{1}{N'''} \left(\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2} \right) g(x,y)$$

$$g_{yy} = \frac{\partial g \partial g}{\partial y^2} g(x,y) = \frac{\partial g}{\partial y} \frac{-y \cdot g(x,y)}{N'\sigma^2} = \frac{-g(x,y)}{N'\sigma^2} + \frac{y^2 \cdot g(x,y)}{N''\sigma^4} = \frac{1}{N'''} \left(\frac{y^2}{\sigma^4} - \frac{1}{\sigma^2} \right) g(x,y)$$

Thus,

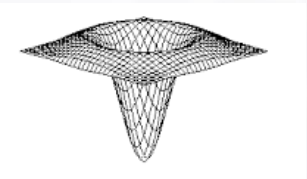
$$\nabla^2 g(x,y) = \frac{1}{N'''\sigma^4} \left(1 - \frac{x^2+y^2}{\sigma^2} \right) g(x,y)$$

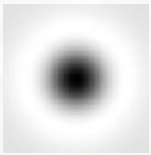
Convolution and Feature Extraction

Laplacian Filter

• The second derivative (i.e., the Laplacian operator) of the Gaussian filter gives rise to the **Laplacian filter**, defined:

$$\nabla^2 g(x,y) = \frac{1}{N'''\sigma^4}\left(1 - \frac{x^2+y^2}{\sigma^2}\right) g(x,y)$$

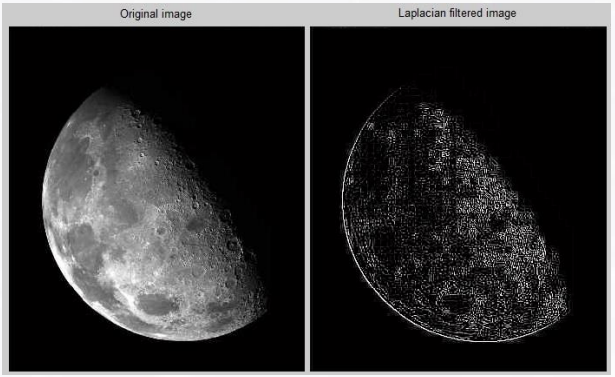




Two commonly used (3×3) discrete variants of the Laplacian filter are:

$$\nabla^2 \mathbf{g} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \qquad \nabla^2 \mathbf{g} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

• The Laplacian filter detects sudden intensity transitions in the image and highlights the edges; the Laplacian is therefore commonly used as an edge/feature detector (sometimes known as a "zero cross" feature detector). Note that the Laplacian filter is sensitive to noise – for this reason one typically applies a Gaussian blur prior to application of the Laplacian operator.



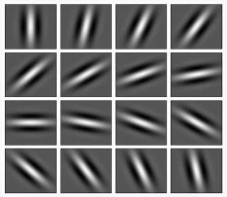
Convolution and Feature Extraction

Gabor Filters

• Another common family of filters, **Gabor filters**, are defined as a product of Gaussian and sinusoid functions. As such, Gabor filters are selective for both scale and orientation.

Gabor filters are parametrized by the standard deviation σ of the Gaussian, and the phase φ, orientation ω, and wavelength λ of the sine wave:

$$f_{mn} = \frac{1}{2\pi\sigma^2} e^{-\frac{m^2+n^2}{2\sigma^2}} \sin\left[\frac{2\pi(\cos[\omega]m + \sin[\omega]n)}{\lambda} + \phi\right]$$



• Notice that Gabor filters closely resemble the features discovered by Hubel and Wiesel to which "simple cells" in the visual cortex were responsive.

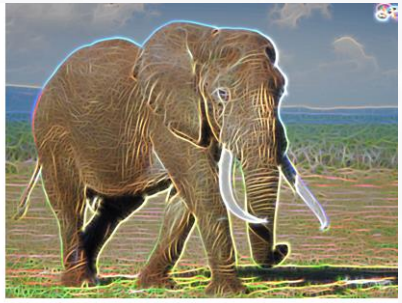
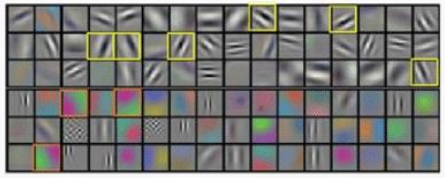
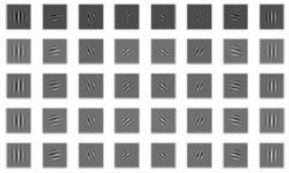


Image with bank of Gabor filter activations shown





(Left) Primitive filters "discovered" by AlexNet CNN architecture; notice the close resemblance with Gabor filters (Right)

Canny Edge Detection

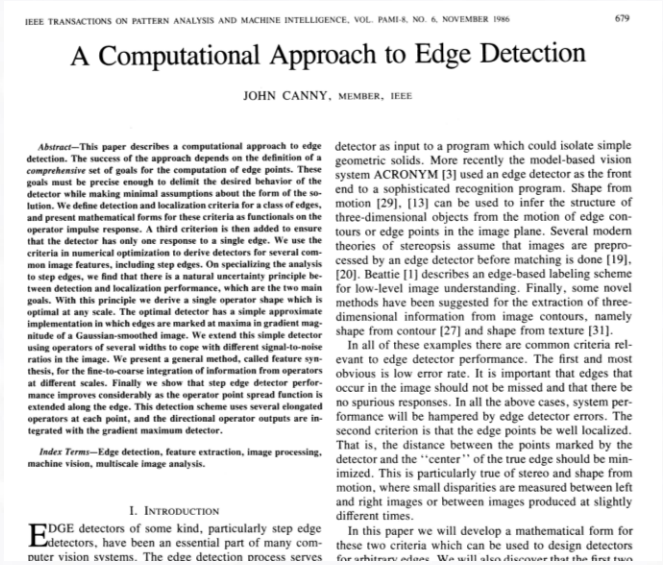
Canny Edge Detection

• Canny edge detection (1986) is a classic edge detection algorithm know by all CV practitioners; it is still in wide use today.

• At its core, Canny edge detection is an intuitive and relatively simple algorithm, following (5) key steps:

(1) We first apply a Gaussian filter to reduce noise in the input image.

Original image (left) — Blurred image with a Gaussian filter (sigma=1.4 and kernel size of 5×5)



IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. PAMI-8, NO. 6, NOVEMBER 1986 679

A Computational Approach to Edge Detection

JOHN CANNY, MEMBER, IEEE

Abstract—This paper describes a computational approach to edge detection. The success of the approach depends on the definition of a *comprehensive* set of goals for the computation of edge points. These goals must be precise enough to delimit the desired behavior of the detector while making minimal assumptions about the form of the solution. We define detection and localization criteria for a class of edges, and present mathematical forms for these criteria as functionals on the operator impulse response. A third criterion is then added to ensure that the detector has only one response to a single edge. We use the criteria in numerical optimization to derive detectors for several common image features, including step edges. On specializing the analysis to step edges, we find that there is a natural uncertainty principle between detection and localization performance, which are the two main goals. With this principle we derive a single operator shape which is optimal at any scale. The optimal detector has a simple approximate implementation in which edges are marked at maxima in gradient magnitude of a Gaussian-smoothed image. We extend this simple detector using operators of several widths to cope with different signal-to-noise ratios in the image. We present a general method, called feature synthesis, for the fine-to-coarse integration of information from operators at different scales. Finally we show that step edge detector performance improves considerably as the operator point spread function is extended along the edge. This detection scheme uses several elongated operators at each point, and the directional operator outputs are integrated with the gradient maximum detector.

Index Terms—Edge detection, feature extraction, image processing, machine vision, multiscale image analysis.

I. Introduction

EDGE detectors of some kind, particularly step edge detectors, have been an essential part of many computer vision systems. The edge detection process serves as input to a program which could isolate simple geometric solids. More recently the model-based vision system ACRONYM [3] used an edge detector as the front end to a sophisticated recognition program. Shape from motion [29], [13] can be used to infer the structure of three-dimensional objects from the motion of edge contours or edge points in the image plane. Several modern theories of stereopsis assume that images are preprocessed by an edge detector before matching is done [19], [20]. Beattie [1] describes an edge-based labeling scheme for low-level image understanding. Finally, some novel methods have been suggested for the extraction of three-dimensional information from image contours, namely shape from contour [27] and shape from texture [31].

In all of these examples there are common criteria relevant to edge detector performance. The first and most obvious is low error rate. It is important that edges that occur in the image should not be missed and that there be no spurious responses. In all the above cases, system performance will be hampered by edge detector errors. The second criterion is that the edge points be well localized. That is, the distance between the points marked by the detector and the "center" of the true edge should be minimized. This is particularly true of stereo and shape from motion, where small disparities are measured between left and right images or between images produced at slightly different times.

In this paper we will develop a mathematical form for these two criteria which can be used to design detectors for arbitrary edges. We will also discover that the first two

https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

https://ieeexplore.ieee.org/document/4767851

Canny Edge Detection

Canny Edge Detection

(2) Using the Sobel kernels, $\boldsymbol{g_x}$ and $\boldsymbol{g_y}$, we next calculate the magnitude and slope of the input image gradient using:

$$\mathbf{X} * \mathbf{G} = \sqrt{\left(\mathbf{X} * \mathbf{g}_x\right)^2 + \left(\mathbf{X} * \mathbf{g}_y\right)^2}$$

$$\theta(x, y) = \arctan\left(\frac{\mathbf{g}_y}{\mathbf{g}_x}\right)$$

This process yields a gradient intensity map.

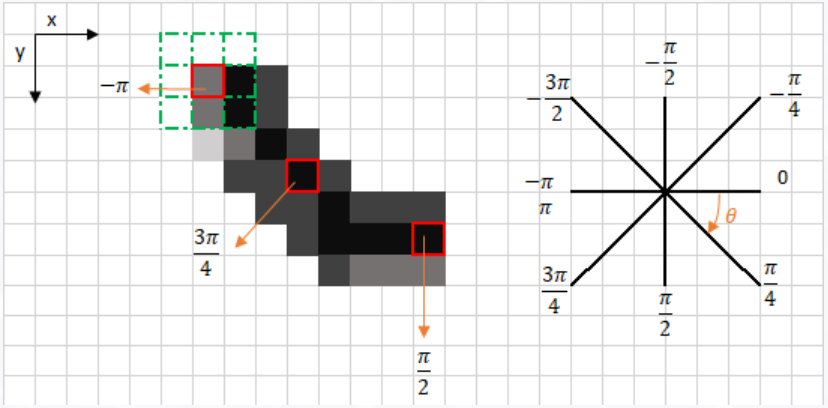
https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

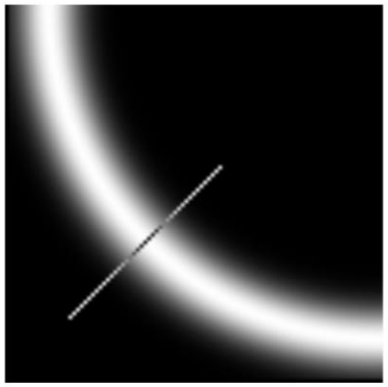
Canny Edge Detection

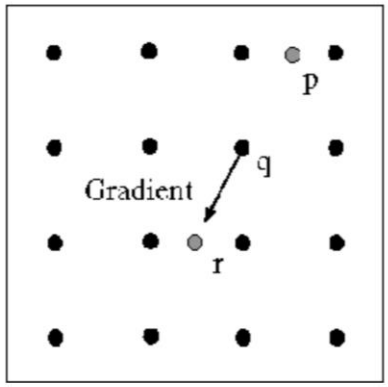
Canny Edge Detection

(3) Step (2) generates a general gradient intensity map – however, many of the rendered contours are thick and often noisy. To help produce more distinct, thin contour lines, we apply a process called **non-maximum suppression**.

• The basic idea is a as follows: we use the gradient intensity map – specifically the angle $\theta(x, y)$ generated from the Sobel kernel (notice that the angle θ yields a vector that "points" in the direction of the highest gradation of low intensity transitioning to high intensity).





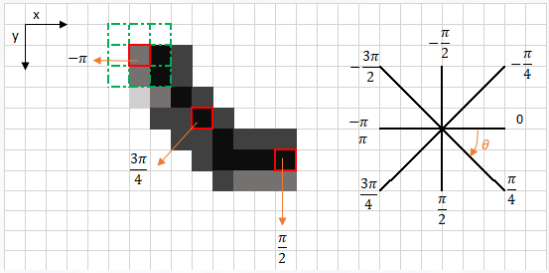


• The <u>orientation of an edge contour is orthogonal (generally) to the gradient angle θ</u>.

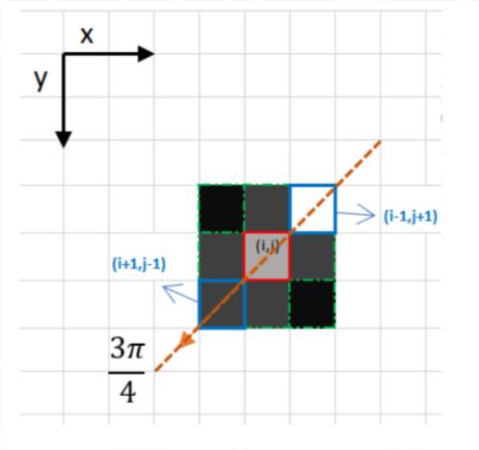
Canny Edge Detection

Canny Edge Detection

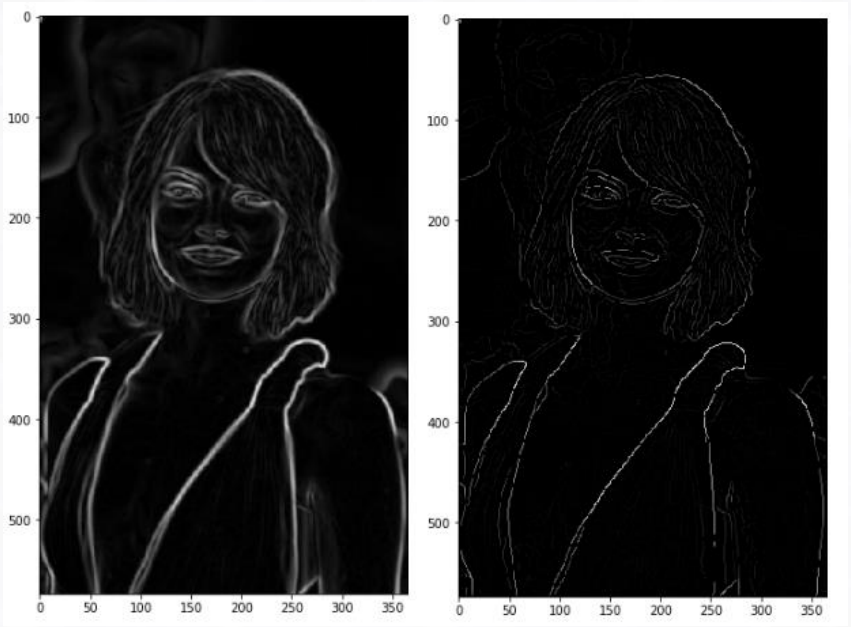
(3) **Non-maximum suppression** (NMS)



• If the current pixel under consideration for non-maximum suppression does not have the maximum gradient intensity compared to the neighboring pixels along the gradient vector induce by θ, **this pixel is suppressed** (i.e., we set the intensity to zero). The effect of non-maximum suppression is to thin contour lines.



Pixel (i,j) is under consideration for NMS; looking at the neighboring pixels along the edge contour, we suppress the intensity of pixel (i,j), because it is not the maximum along the edge contour.



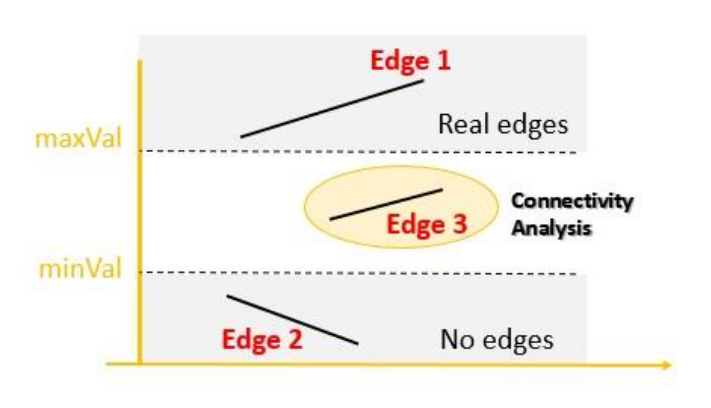
https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

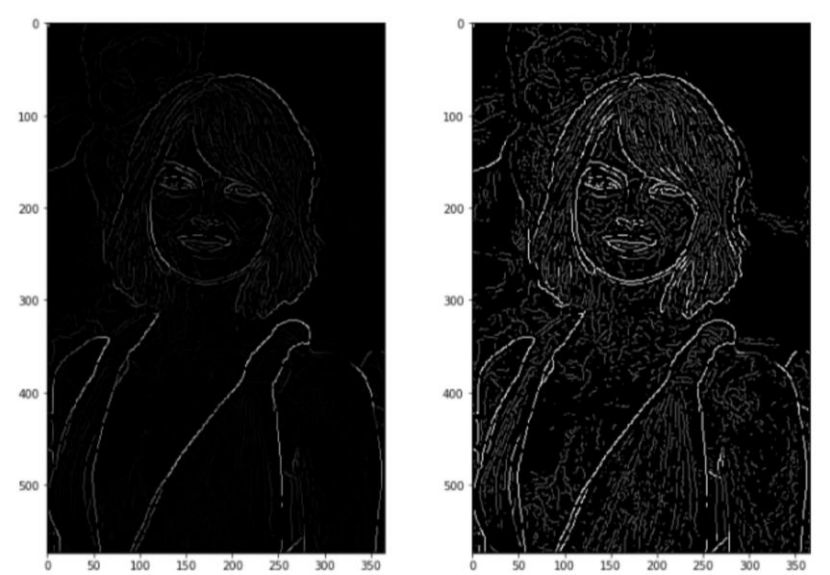
Canny Edge Detection

Canny Edge Detection

(4) **Double Thresholding**

• Following NMS, we apply double thresholding. In this step we are provided two parameters: (minValue, maxValue); using these parameters, we identify pixels as either: **strong**, **weak** or **irrelevant** as edge pixels.





Non-Max Suppression image (left) — Threshold result (right): weak pixels in gray and strong ones in white.

• Any pixel values above maxValue are identified as true edge pixels; intensities below minValue are discarded from consideration as edge pixels. Finally, pixels falling in the range (minValue, maxValue) are considered "weak" and subject to further analysis using hysteresis (step 5).

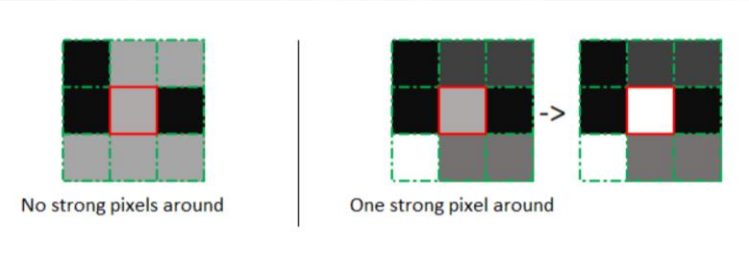
https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

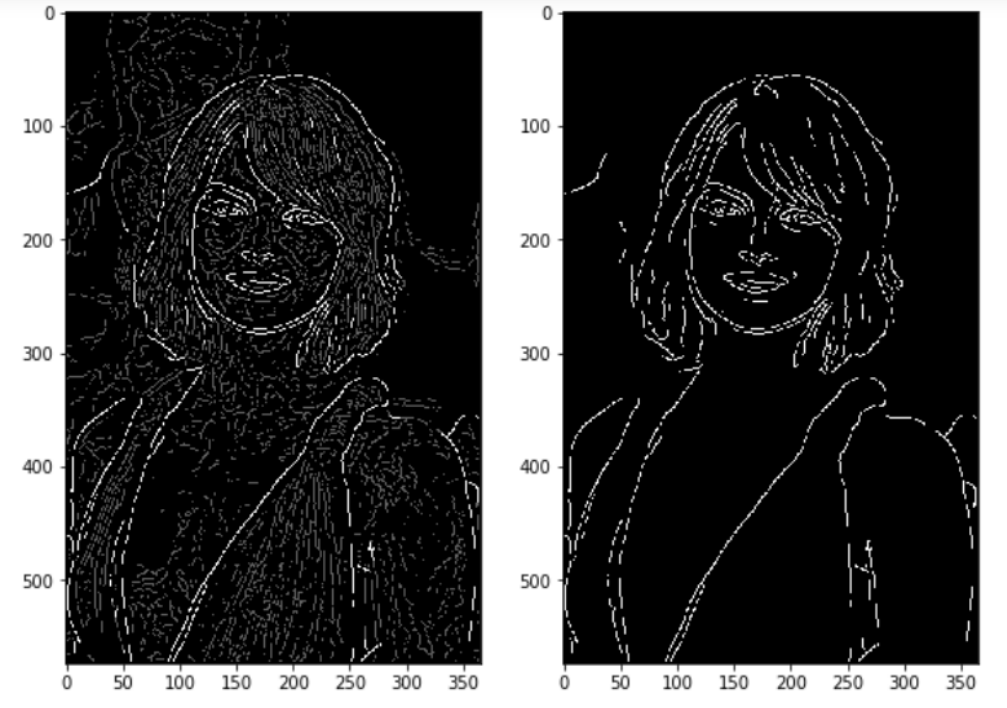
Canny Edge Detection

Canny Edge Detection

(5) **Edge Refinement with Hysteresis**

• Finally, the hysteresis consists of transforming weak pixels into strong ones, if and only if at least one of the pixels around the one being processed is designated strong.

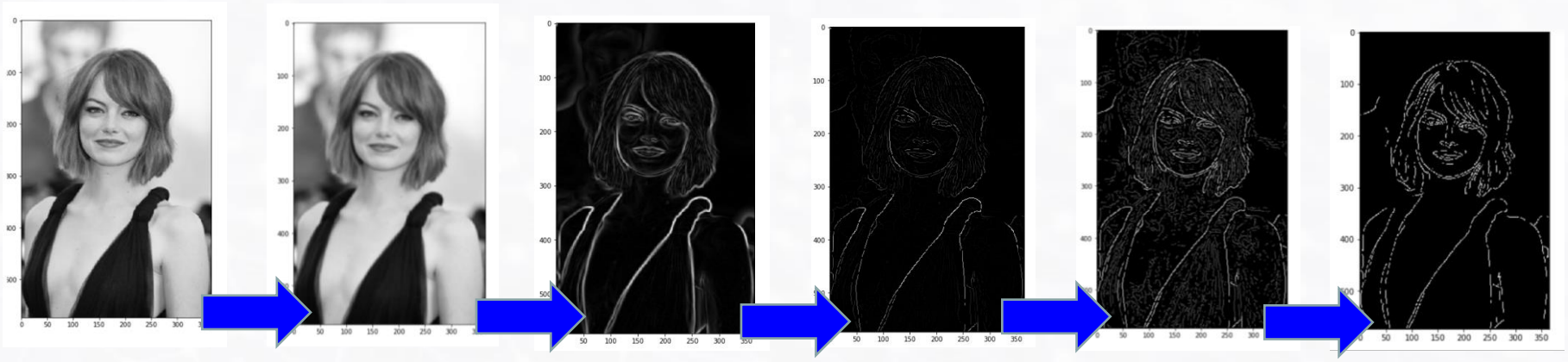




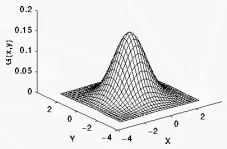
https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

Canny Edge Detection

Canny Edge Detection Summary

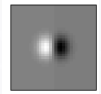


(1) Gaussian blur

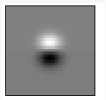


(2) Sobel transform

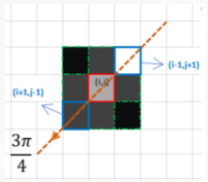
$$\mathbf{g}_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$



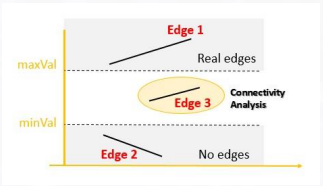
$$\mathbf{g}_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$



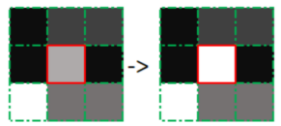
(3) NMS



(4) Double Thresholding



(5) Hysteresis



$$\mathbf{X} * \mathbf{G} = \sqrt{\left(\mathbf{X} * \mathbf{g}_x\right)^2 + \left(\mathbf{X} * \mathbf{g}_y\right)^2}$$

$$\theta(x, y) = \arctan\left(\frac{\mathbf{g}_y}{\mathbf{g}_x}\right)$$

https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

OpenCV

• **OpenCV** (Intel, opencv.org) is a comprehensive, open-source library of CV-related algorithms, available in C++ and Python. Functionality is broad, including image processing, feature extraction, segmentation, edge detection, video tracking, segmentation, image stitching, camera calibration, DNN-based algorithms, etc.

• I highly recommend exploring some of the OpenCV tutorials: https://docs.opencv.org/master/d9/df8/tutorial_root.html

• Every core algorithm mentioned in this lecture series (pre-processing, filter types, Canny, descriptors, segmentation, tracking, etc.) can be executed in OpenCV (often using just a few lines of code!).

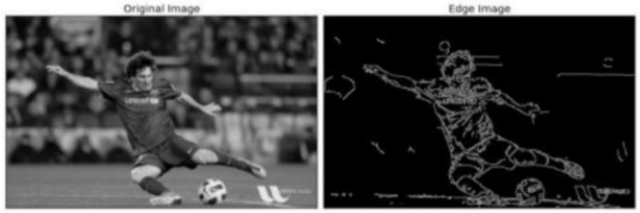
Source code: https://github.com/opencv/opencv

OpenCV

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

img = cv.imread('messi5.jpg',0)
edges = cv.Canny(img,100,200)

plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])

plt.show()
```

See the result below:

Canny edge detection

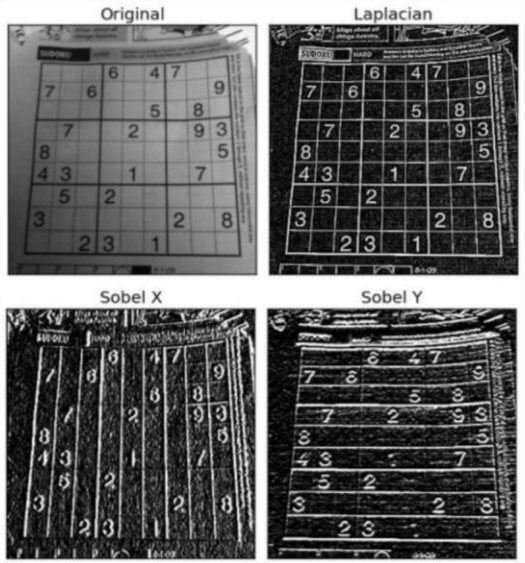


image

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('dave.jpg',0)

laplacian = cv2.Laplacian(img,cv2.CV_64F)
sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)
sobely = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=5)

plt.subplot(2,2,1),plt.imshow(img,cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,2),plt.imshow(laplacian,cmap = 'gray')
plt.title('Laplacian'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,3),plt.imshow(sobelx,cmap = 'gray')
plt.title('Sobel X'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,4),plt.imshow(sobely,cmap = 'gray')
plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])

plt.show()
```

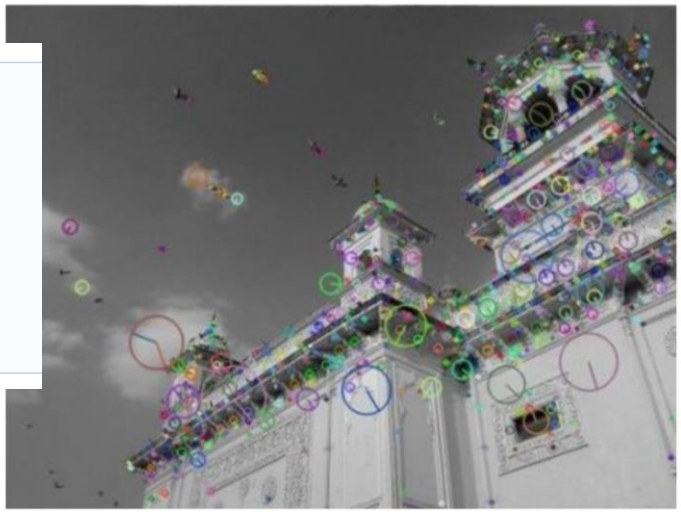
Sobel Transform

Result:




```
import numpy as np
import cv2 as cv

img = cv.imread('home.jpg')
gray= cv.cvtColor(img,cv.COLOR_BGR2GRAY)

sift = cv.SIFT_create()
kp = sift.detect(gray,None)

img=cv.drawKeypoints(gray,kp,img)

cv.imwrite('sift_keypoints.jpg',img)
```

SIFT descriptor



SIFT Descriptor

• Oftentimes, we are interested in identifying "interesting points" (i.e., **keypoints**) in an image; these points can be leveraged in edge detection, keypoint matching, image stitching, pose classification, object tracking, and related CV problems.

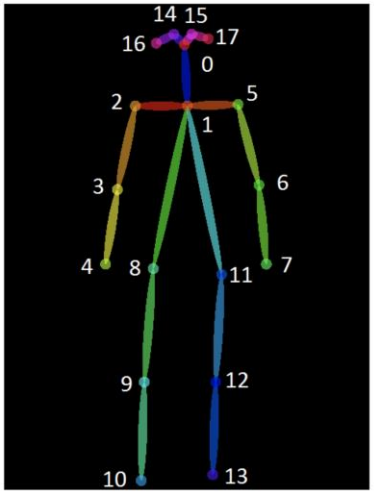
Distinctive Image Features from Scale-Invariant Keypoints

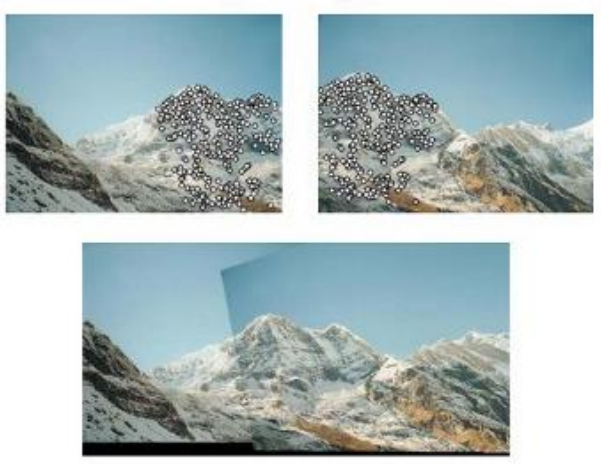
David G. Lowe
Computer Science Department
University of British Columbia
Vancouver, B.C., Canada
lowe@cs.ubc.ca

January 5, 2004

Abstract

This paper presents a method for extracting distinctive invariant features from images that can be used to perform reliable matching between different views of an object or scene. The features are invariant to image scale and rotation, and are shown to provide robust matching across a a substantial range of affine distortion, change in 3D viewpoint, addition of noise, and change in illumination. The features are highly distinctive, in the sense that a single feature can be correctly matched with high probability against a large database of features from many images. This paper also describes an approach to using these features for object recognition. The recognition proceeds by matching individual features to a database of features from known objects using a fast nearest-neighbor algorithm, followed by a Hough transform to identify clusters belonging to a single object, and finally performing verification through least-squares solution for consistent pose parameters. This approach to recognition can robustly identify objects among clutter and occlusion while achieving near real-time performance.





• The *scale invariant feature transform* (**SIFT**, 1999) descriptor is a common, robust method used to detect and describe local features in images. SIFT descriptors are 128-dimensional vectors that summarize unique visual features within a patch centered at a keypoint pixel.

https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf

SIFT Descriptor

(1) **Scale-space extrema detection**: In order for the SIFT detector to be scale-invariant, we first generate a **scale-space of an image**.

• **The goal of the scale-space calculation is to generate a multi-scale Laplacian** (more accurately: an *approximation* of the Laplacian) for the original image.

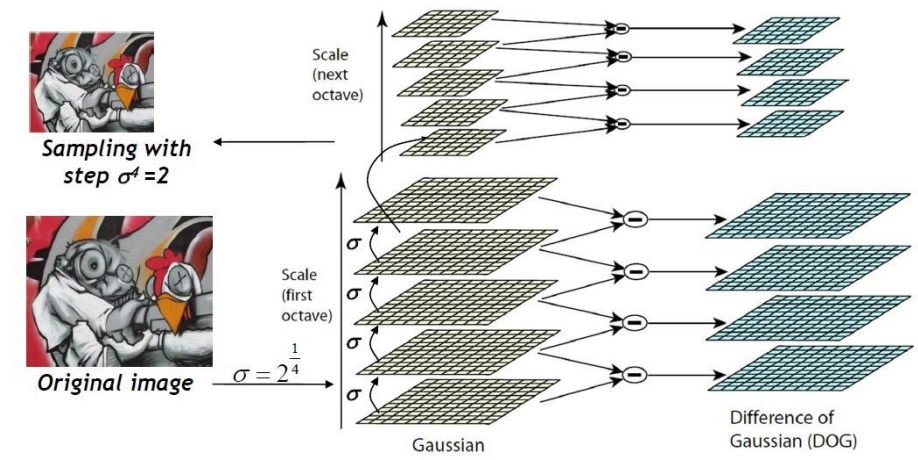
SIFT Descriptor

(1) **Scale-space extrema detection**: In order for the SIFT detector to be scale-invariant, we first generate a **scale-space of an image**.

• **The goal of the scale-space calculation is to generate a multi-scale Laplacian** (more accurately: an approximation of the Laplacian) for the original image.

• The first step toward this end is to convolve a Gaussian kernel at different scales with the input image. Concretely, we produce different "**octaves**"; within each octave we use different σ parameters to generate smoothing at different scales; we generate different octaves by halving the size of the input image for each successive octave.

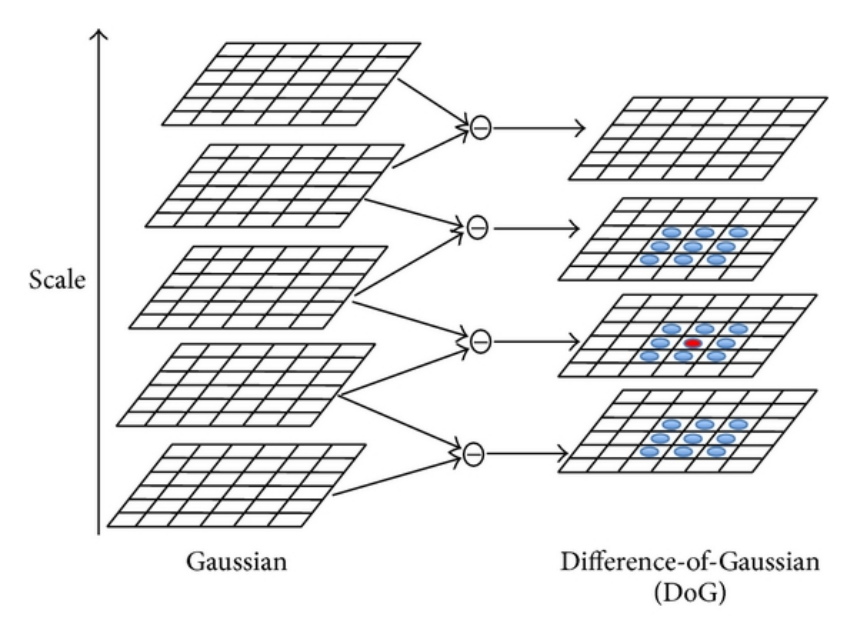
• Next, we compute the **difference of Gaussians** (for pairs in each octave); the difference of Gaussians is a well-known approximation to the Laplacian of an image.



SIFT Descriptor

(1) **Scale-space extrema detection**: In order for the SIFT detector to be scale-invariant, we first generate a **scale-space of an image**.

• Then we determine a set of candidate keypoints. One pixel in each image is compared with its 8 neighbors as well as the 9 pixels in the next scale and the 9 pixels in the previous scale; in this way, a total of 26 pixels are compared. If the pixel under consideration is an extremum in relation to this set, it is designated as a candidate keypoint.



SIFT Descriptor

(2) Keypoint Selection

• At each candidate keypoint, the authors **determine whether the keypoint is a low-contrast point**, in which case it is rejected. To discard the keypoints with low contrast, we compute the second-order Taylor expansion at its local extremum $\hat{x}$; if the intensity of this pixel is less than a threshold value (0.3), it is discarded; for D(x,y,σ), the *difference of Gaussian space*, compute:

$$D(\mathbf{x}) = D + \frac{\partial D}{\partial \mathbf{x}}^T \mathbf{x} + \frac{1}{2}\mathbf{x}^T \frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x}$$

Low-contrast keypoint removal

SIFT Descriptor

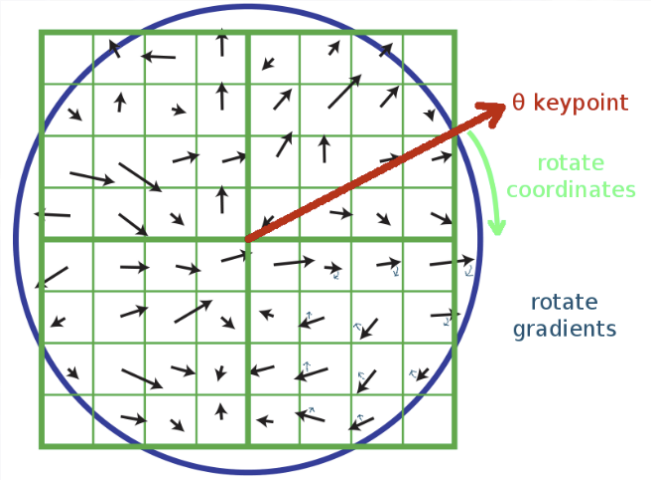
(3) Orientation Assignment: This is the key step in achieving invariance to rotation.

• We compute the gradient magnitude m(x,y) and direction θ(x,y) with respect to the Gaussian-smoothed image L(x,y, σ) for a neighborhood of 36 points (x,y) surrounding the keypoint where σ is the scale identified with the keypoint:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + L(x,y+1) - L(x,y-1))^2}$$

$$\theta(x,y) = \arctan((L(x,y+1) - L(x,y-1)) / (L(x+1,y) - L(x-1)))$$

• From the histogram of the orientations of these 36 surrounding pixels, **we assign an orientation for the keypoint** (aligned with the maximum of the orientation histogram). Next, rotate the gradient directions and locations relative to the keypoint orientation (this will make the SIFT detector invariant to rotations).

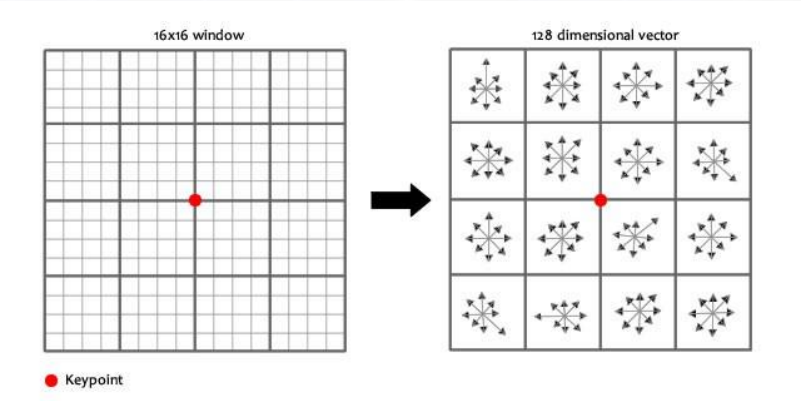


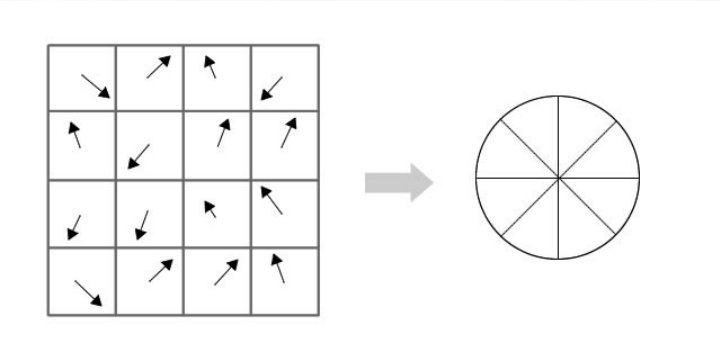
SIFT Descriptor

(4) Keypoint Descriptor: At this juncture we have identified keypoints in the image, along with their relative scale and orientation.

• The final step is to compute the **128-dimensional keypoint descriptor**. When defining this descriptor vector, we want it to be distinctive (i.e., specific to the particular keypoint), and invariant to changes in viewpoint and illumination.

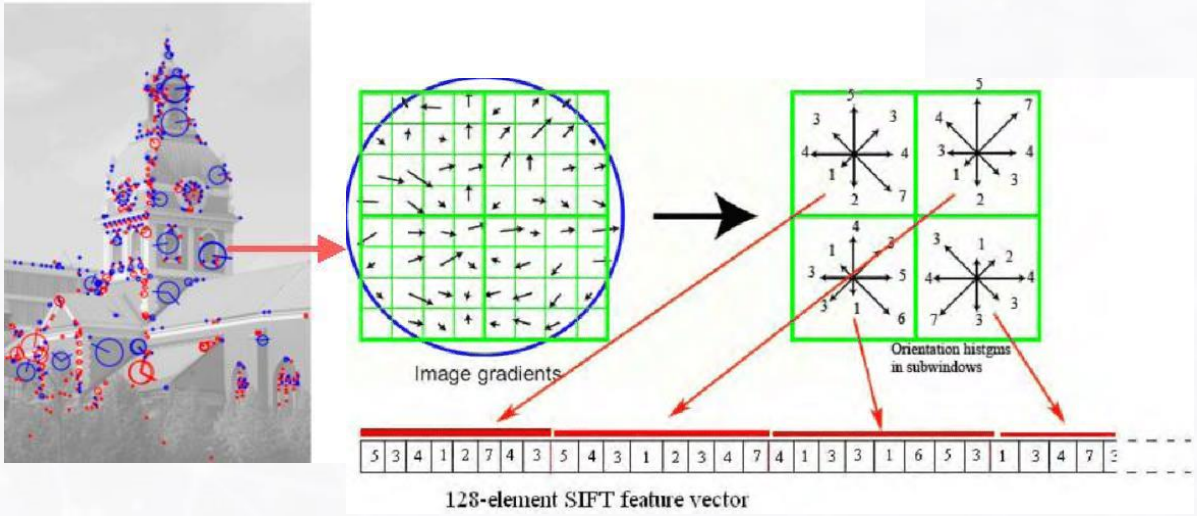
• Using a 16x16 window (divided into 4x4 sub-regions) of rotated gradients (rotated relative to the keypoint orientation) we construct a histogram (using 8 bins, as shown) for each 4x4 subregion. This yields a 4x4x8=128 dimensional SIFT feature vector.





*Note that illumination invariance can be achieved by thresholding/normalizing this descriptor vector.

SIFT Descriptor



SIFT Algorithm summary:

(1) Generate scale-space of image using Gaussian smoothing for different σ and different image sizes; estimate Laplacian from this scale-space using difference of Gaussians; determine keypoint candidate from local neighborhoods.

(2) Remove low-contrast candidates.

(3) Compute local gradients wrt keypoint; determine orientation of keypoint.

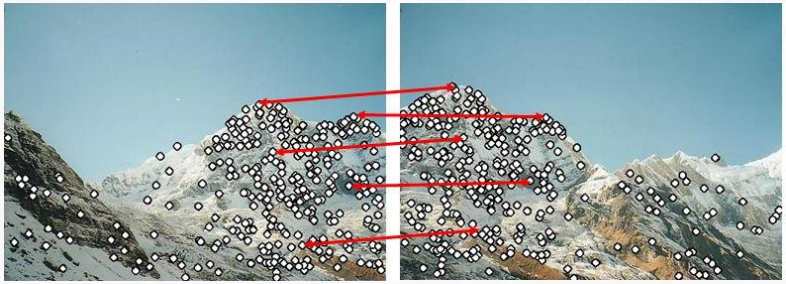
(4) Generate keypoint descriptor: from 16x16 grid of neighboring pixels, generate 8-bin histograms of each 4x4 subregion (of gradients rotated relative to keypoint orientation).

SIFT Descriptor: Image Stitching

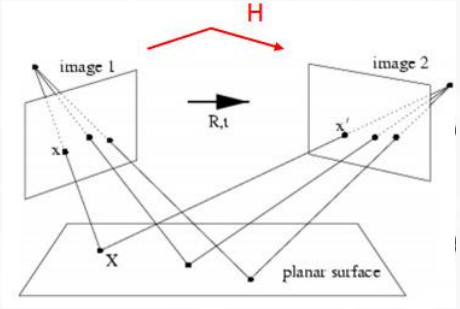
• Image descriptors (e.g., SIFT), are essential to many CV tasks, including **image stitching**, **image retrieval**, **pose estimation**, and general image **feature extraction**.

Image Stitching

• Image stitching is the process of combining multiple photographic images with overlapping fields of view to produce a cohesive panorama image.



• Using local image descriptors such as SIFT, one can perform **keypoint matching** between two images by simply identifying matches based on their nearest neighbors (i.e., L2 distance b/w descriptor vectors). Finally, we learn a linear transformation (called a **Homography matrix** in CV) which relates the mapping between two planes from a single point of reference.



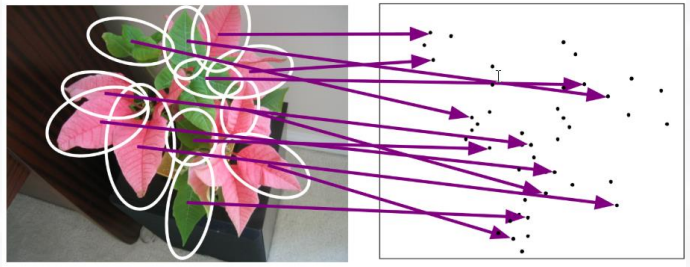
*Note that there are many heuristics to reduce the instance of false matchings in this setting, e.g., checking ratio of closest distance with second closest distance, and rejecting based on a threshold criterion.

SIFT Descriptor: Image Retrieval

Bag of Visual Worlds (BoVW)

• Using a collection of local descriptors (e.g., SIFT descriptors of keypoints) of an image, we can generate a "global" description of an image, called a **BoVW model**.

(1) Given a training set, for each image, we extract the set of SIFT keypoint descriptors (notice that images can yield different numbers of keypoints, but each will have an associated vector of equal dimension).



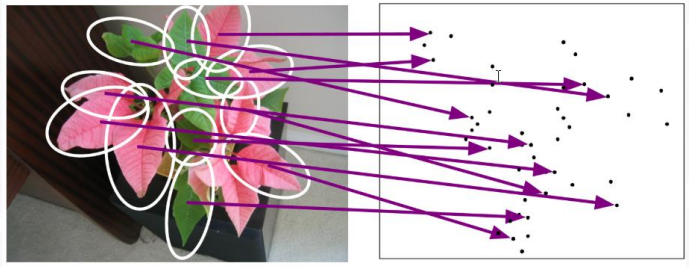
Each SIFT vector is of 128-dimensions

SIFT Descriptor: Image Retrieval

Bag of Visual Worlds (BoVW)

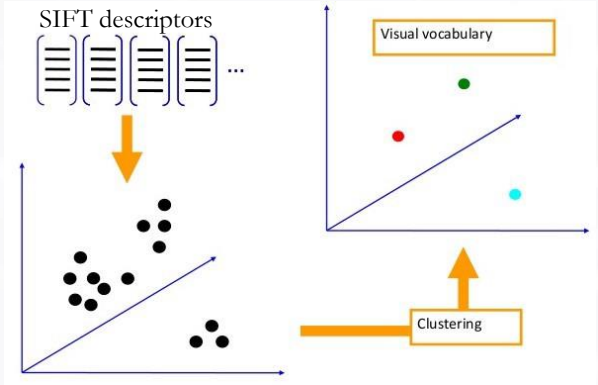
• Using a collection of local descriptors (e.g. SIFT descriptors of keypoints) of an image, we can generate a "global" description of an image, called a **BoVW model**.

(1) Given a training set, for each image, we extract the set of SIFT keypoint descriptors (notice that images can yield different numbers of keypoints, but each will have an associated vector of equal dimension).



Each SIFT vector is of 128-dimensions

(2) Collectively, we take all the SIFT keypoint descriptors and perform k-means clustering. The hyperparameter K (the number of clusters) will represent out visual "vocabulary" size; each centroid corresponds with a visual "word" in the SIFT representation feature space.

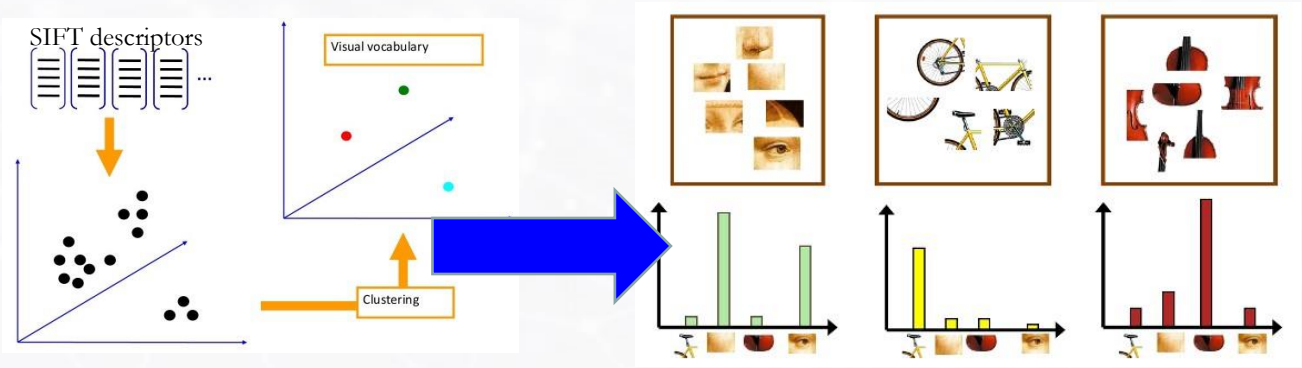


SIFT Descriptor: Image Retrieval

Bag of Visual Worlds (BoVW)

• Using a collection of local descriptors (e.g., SIFT descriptors of keypoints) of an image, we can generate a "global" description of an image, called a **BoVW model**.

(3) For each training image we, **create a histogram based on the visual vocabulary** rendered by k-means in (2). This per image histogram denotes the frequency of each word in the visual vocabulary

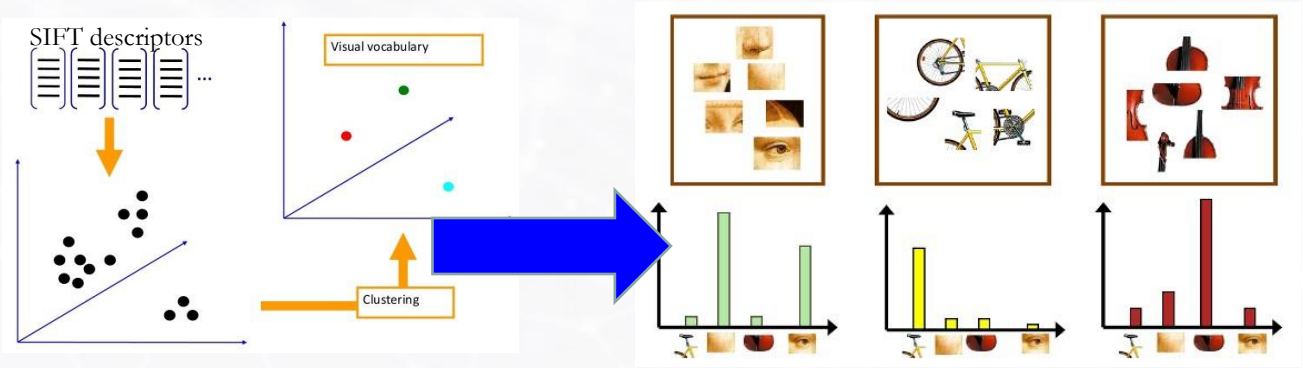


SIFT Descriptor: Image Retrieval

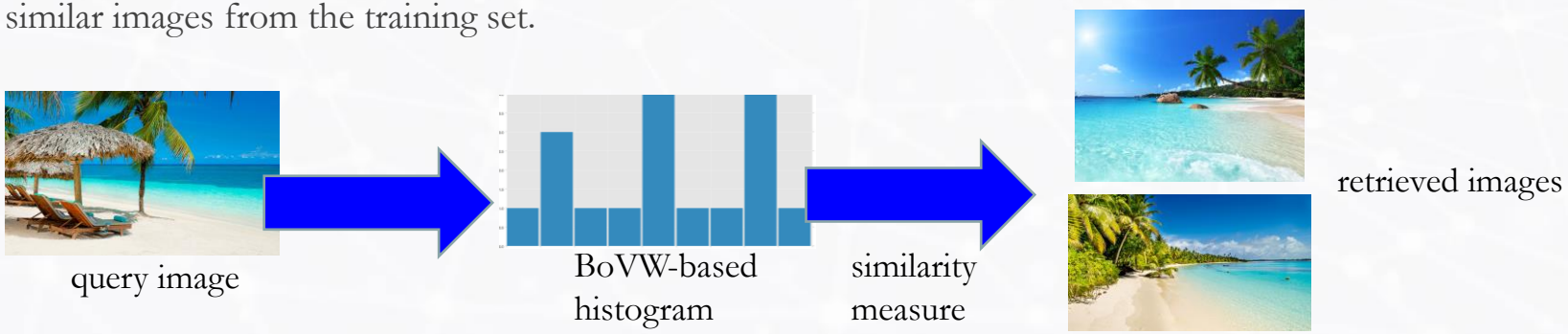
Bag of Visual Worlds (BoVW)

• Using a collection of local descriptors (e.g. SIFT descriptors of keypoints) of an image, we can generate a "global" description of an image, called a **BoVW model**.

(3) For each training image we, **create a histogram based on the visual vocabulary** render by k-means in (2). This per image histogram denotes the frequency of each word in the visual vocabulary



• Now, from this BoVW model, we can perform **image retrieval**. Given a query image, we generate its SIFT features, and then construct the histogram for this test image based on our previously identified visual vocabulary. Using a basic similarity measure with respect to this histogram (i.e., nearest-neighbor, L2-distance, etc.) we can generate similar images from the training set.



*Notice that **general image classification** using hand-crafted features can be executed in a similar manner.

HOG Descriptor

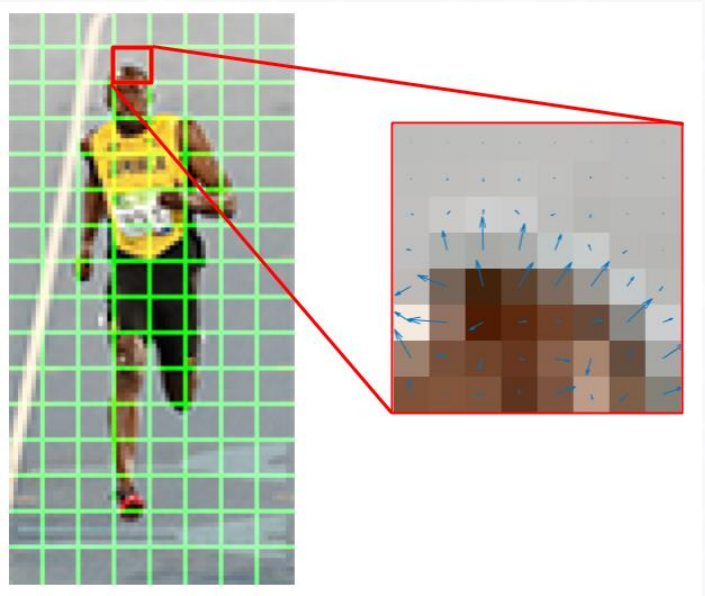
• Like the SIFT descriptor, the **histogram of oriented gradients** (HOG) descriptor attempts to compactly represent salient features in an image. In general, The HOG descriptor gives a more detailed characterization of the spatial structure of an image.

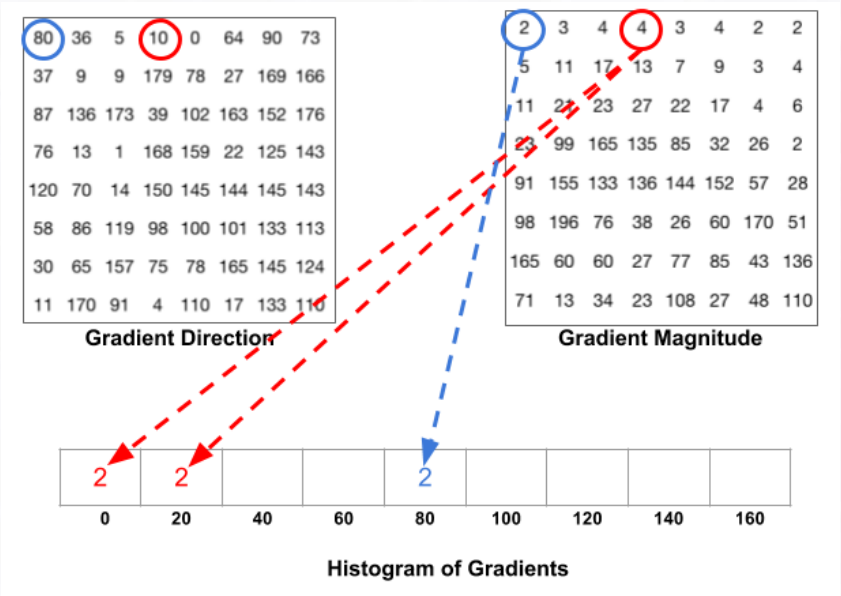
The HOG descriptor is simple to calculate:

(1) First*, we apply the Sobel transformation to the input image.

(2) Next, we divide the image (or image patch) uniformly into small cells (e.g., 8x8, 16x16 cells). Within each of these cells each pixel now has an associated magnitude and direction (from the Sobel transformation).

(3) We then "bin" the direction values of each pixel in the cell using 9 bins (0, 20, 40, …, 160 – direction signs are ignored; this is known as an "unsigned" gradient).





*Generally, the calculation of the HOG descriptor requires no pre-processing (due to block normalization step).

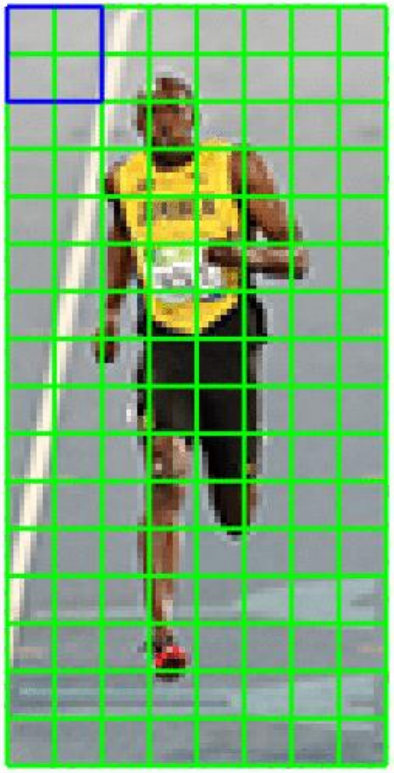
HOG Descriptor

(1) First*, we apply the Sobel transformation to the input image.

(2) Next, we divide the image (or image patch) uniformly into small cells (e.g. 8x8, 16x16 cells). Within each of these cells each pixel now has an associated magnitude and direction (from the Sobel transformation).

(3) We then "bin" the direction values of each pixel in the cell using 9 bins (0, 20, 40, …, 160 – direction signs are ignored; this is known as an "unsigned" gradient).

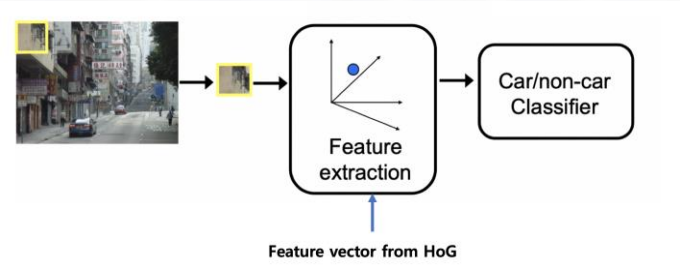
(4) Finally, for robustness to lighting changes, we generate a **normalized block descriptor** by concatenating 2x2 (usually, or 3x3) neighborhoods of cells (then normalizing over this entire neighborhood); this yields the final HOG descriptor.



HOG Descriptor: Object Localization

• We can develop an **object localization algorithm** using HOG descriptors in combination with a classifier model.

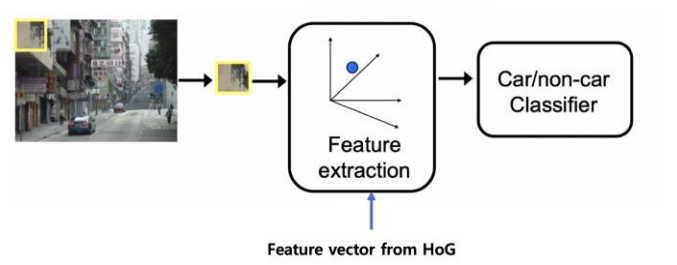
• Suppose that we train a simple SVM (support vector machine) to **classify cars vs non-cars** based on the HOG descriptor of an image patch. Which is to say, we train the SVM on a set of HOG descriptors of image patches from our training set in order to differentiate cars from non-cars.



HOG Descriptor: Object Localization

• We can develop an **object localization algorithm** using HOG descriptors in combination with a classifier model.

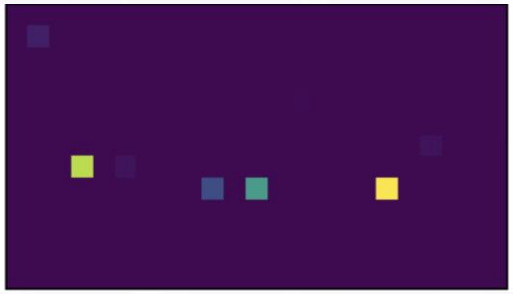
• Suppose that we train a simple SVM (support vector machine) to **classify cars vs non-cars** based on the HOG descriptor of an image patch. Which is to say, we train the SVM on a set of HOG descriptors of image patches from our training set in order to differentiate cars from non-cars.



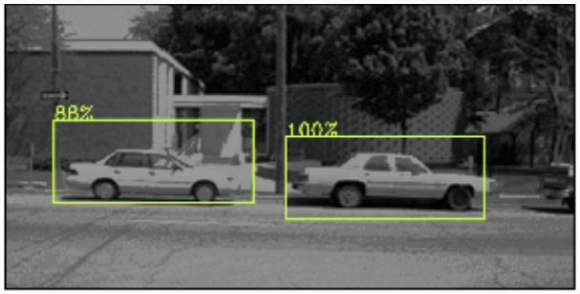
• Using a simple "sliding window" approach -- we extract patches over all regions in an image and compute their corresponding HOG descriptor. Each of these HOG descriptors is fed into our trained SVM, rendering a "score map". The maximum scores (above a threshold) are determined to be locations of a car.

HOG Descriptor

SVM Score Map



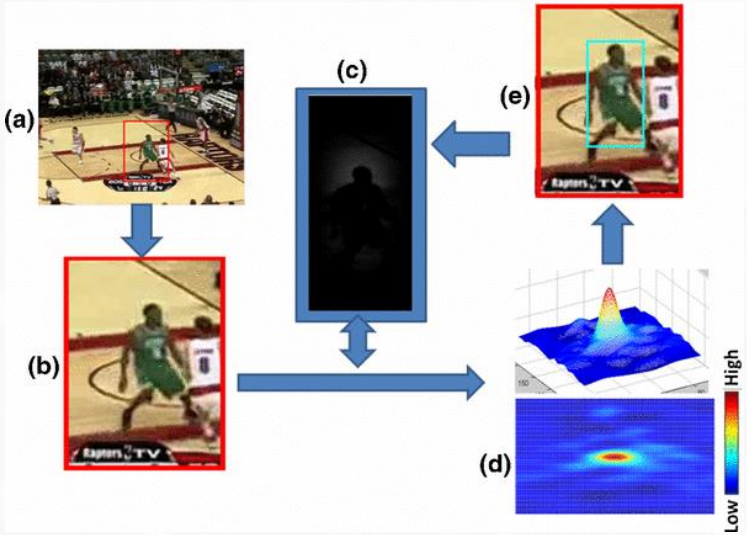
Localization



Video Tracking: Correlation Filter

Visual Object Tracking using Adaptive Correlation Filters

David S. Bolme J. Ross Beveridge Bruce A. Draper Yui Man Lui
Computer Science Department
Colorado State University
Fort Collins, CO 80521, USA
bolme@cs.colostate.edu



• Filter- based trackers model the appearance of objects using filters trained on example images.

• The target is initially selected based on a small tracking window centered on the object in the first frame. The target is then tracked by **correlating the filter over a search window in the next frame**; the location corresponding to the maximum value in the correlation output indicates the new position of the target.

• When executed efficiently, correlation filter tracking can run in real-time, e.g., **Minimum Output Sum of Squared Error** (MOSSE, 2010) tracker, which we review next.

Video Tracking: Correlation Filter

MOSSE Tracker

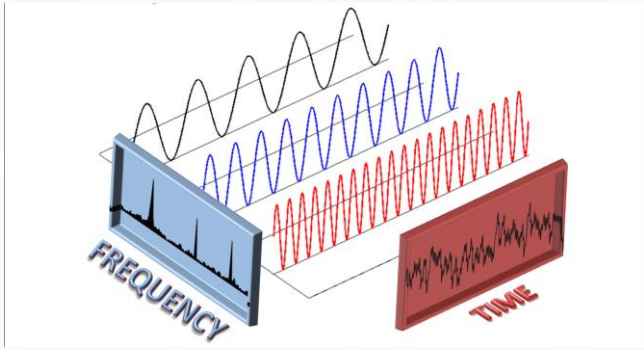
• We wish to develop a computationally efficient method to define a robust correlation filter for object tracking.

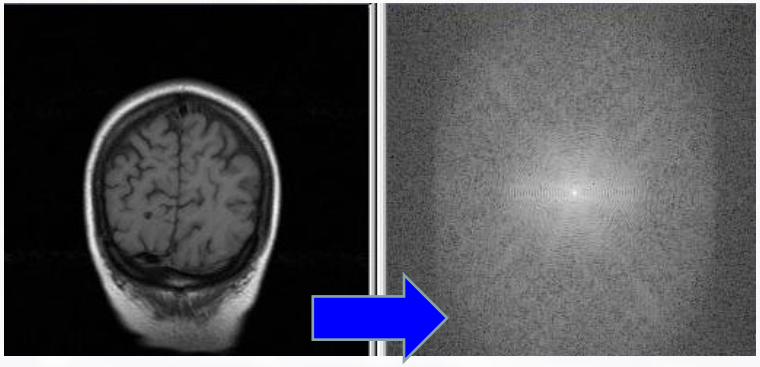
• To this end, we want to define a correlation filter H, satisfying:

$$G = F \odot H^*$$

where G is the **Fast Fourier Transform*** (FFT) of an idealized correlation output (e.g., a Gaussian peak), $F = FFT(f)$ the input image patch and $\mathrm{H} = FFT(h)$ of the learned correlation filter; H^* denotes the complex conjugate of H; $\odot$ denotes elementwise multiplication.

• **The Convolution Theorem**** states that correlation is mathematically equivalent to convolution in the Fourier domain.





FFT

*http://www.dsp-book.narod.ru/DSPMW/07.PDF
**https://www.sciencedirect.com/topics/engineering/convolution-theorem

Video Tracking: Correlation Filter

MOSSE Tracker

• To this end, we want to define a correlation filter H, satisfying:

$$G = F \odot H^*$$

Let $G_i = e^{\frac{(x-x_i)^2+(y-y_i)^2}{\sigma^2}}$, a Gaussian correlation.

Video Tracking: Correlation Filter

MOSSE Tracker

• To this end, we want to define a correlation filter H, satisfying:

$$G = F \odot H^*$$

Let $G_i = e^{\frac{(x-x_i)^2+(y-y_i)^2}{\sigma^2}}$, a Gaussian correlation.

• A reasonable **optimization criterion** is:

$$\min_{H^*} \sum_i | F_i \odot H^* - G_i |^2$$

where we wish to minimize the distance between the idealized correlation output G_i and the predicted correlation output using the learned filter H^*, namely: $F_i \odot H^*$; note that the sum is performed over a training dataset of images/patches. The solution to this optimization problem is the **MOSSE tracker**.

Video Tracking: Correlation Filter

MOSSE Tracker

• To this end, we want to define a correlation filter H, satisfying:

$$G = F \odot H^*$$

Let $G_i = e^{\frac{(x-x_i)^2+(y-y_i)^2}{\sigma^2}}$, a Gaussian correlation.

• A reasonable **optimization criterion** is:

$$\min_{H^*} \sum_i | F_i \odot H^* - G_i |^2$$

where we wish to minimize the distance between the idealized correlation output G_i and the predicted correlation output using the learned filter H^*, namely: $F_i \odot H^*$; note that the sum is performed over a training dataset of images/patches. The solution to this optimization problem is the **MOSSE tracker**.

• The authors show that a closed form solution is given by:

$$H^* = \frac{\sum_i G_i \odot F_i^*}{\sum_i F_i \odot F_i^*}$$

where the numerator represents the correlation between the input and the desired output, and the denominator is the energy spectrum of the input.

Video Tracking: Correlation Filter

MOSSE Tracker

• In practice, one usually extracts several crops of the object of interest from the first several frames of a video clip (or at minimum – from the first frame); this gives us our training set $\{F_i\}$

• Using a Gaussian filter for G_i, we calculate the **MOSSE correlation filter**:

$$H^* = \frac{\sum_i G_i \odot F_i^*}{\sum_i F_i \odot F_i^*}$$

Video Tracking: Correlation Filter

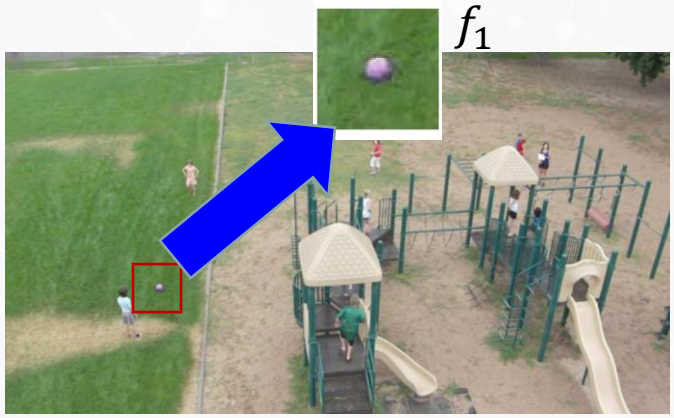
MOSSE Tracker

• In practice, one usually extracts several crops of the object of interest from the first several frames of a video clip (or at minimum – from the first frame); this gives us our training set $\{F_i\}$

• Using a Gaussian filter for G_i, we calculate the MOSSE correlation filter:

$$H^* = \frac{\sum_i G_i \odot F_i^*}{\sum_i F_i \odot F_i^*}$$

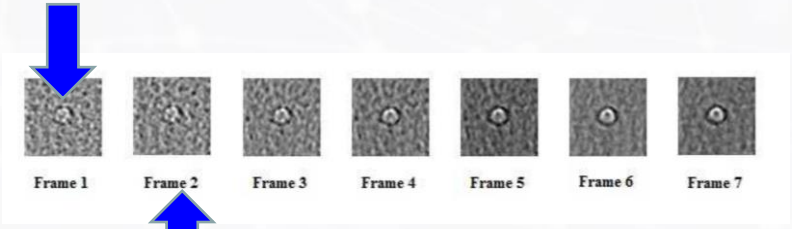
• The target is then tracked by **correlating the filter over a search window in the next frame**; the location corresponding to the maximum value in the correlation output indicates the new position of the target.



Frame 1 of video

g_i

$G_1 \odot F_1^* \rightarrow INVFFT$



$G_2 \odot F_2^* \rightarrow INVFFT$

Video Tracking: Correlation Filter

MOSSE Tracker

• During tracking, a target object can often change appearance by changing rotation, scale, undergoing illumination changes, deforming etc. Therefore, filters to need to quickly adapt in order to follow objects; the authors apply a running average for this purpose; for the *i*th video frame:

$$H_i^* = \frac{A_i}{B_i}$$

$$A_i = \eta G_i \odot F_i^* + (1-\eta) A_{i-1}$$

$$B_i = \eta F_i \odot F_i^* + (1-\eta) B_{i-1}$$

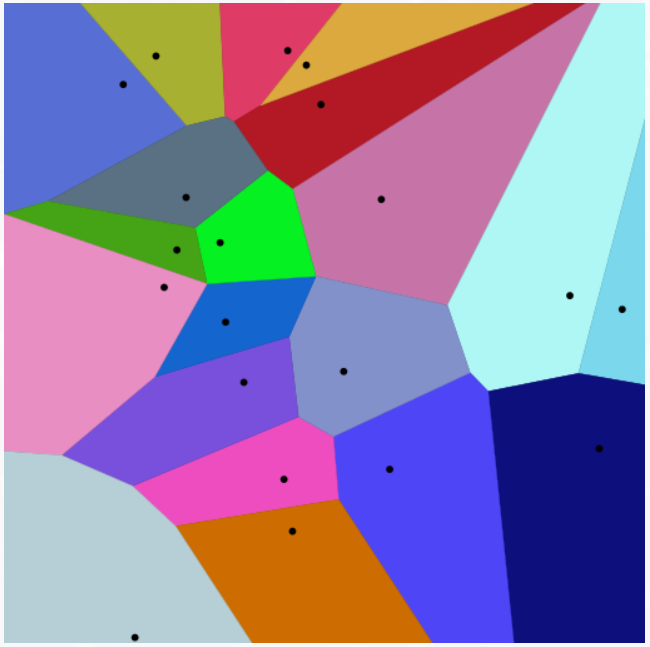
where η is a learning rate that gauges the importance of the previous frames.

Review Topic:
k-Means

k-Means

- k-means is a very popular (and simple) clustering algorithm used in ML and data science.

- *k*-means clustering aims to partition *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean, serving as a *prototype of the cluster*. This results in a partitioning of the data space into **Voronoi cells**.



Vornoi Tessellation; 20 points and their Voroni cells.

k-Means

- Given a set of observations ($\mathbf{x}_1$, $\mathbf{x}_2$, …, $\mathbf{x}_n$), where each observation is a d-dimensional real vector, k-means clustering aims to partition the n observations into k (≤ n) sets $\mathbf{S}=\{S_1, S_2, \ldots, S_k\}$ so as to **minimize the within-cluster sum of squares** (WCSS).

k-Means

- Given a set of observations ($\mathbf{x}_1$, $\mathbf{x}_2$, …, $\mathbf{x}_n$), where each observation is a d-dimensional real vector, k-means clustering endeavors to partition the n observations into k (≤ n) sets $\mathbf{S}=\{S_1, S_2, \ldots, S_k\}$ so as to **minimize the within-cluster sum of squares** (WCSS).

- Formally, the objective is to find:

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{x \in S_i} \left\| \mathbf{x} - \boldsymbol{\mu}_i \right\|^2 = \arg\min_{\mathbf{S}} \sum_{x \in S_i} |S_i| \operatorname{Var}(S_i)$$

where $\boldsymbol{\mu}_i$ is the mean of cluster S_i.

k-Means

- The algorithm itself works by iterative refinement, and is a variant of a more general algorithm, known as **EM** (**expectation-maximization**).

- Given an initial set of k means $m_1^{(1)}, \ldots, m_k^{(1)}$ (the subscript is the cluster identification, while superscript is the iteration number) k-means alternates between the following (2) steps:

k-Means

- The algorithm itself works by iterative refinement, and is a variant of a more general algorithm, known as **EM** (**expectation-maximization**).

- Given an initial set of k means $m_1^{(1)}, \ldots, m_k^{(1)}$ (the subscript is the cluster identification, while superscript is the iteration number) k-means alternates between the following (2) steps:

(I) Assignment Step (i.e., the expectation step):

Assign each observation to the cluster whose mean has the least squared Euclidean distance, this is intuitively the "nearest" mean. Mathematically, this means partitioning the observations according to the Voroni tessellation generated by the means.

$$S_i^{(t)} = \left\{ x_p : \left\| x_p - m_i^{(t)} \right\|^2 \leq \left\| x_p - m_j^{(t)} \right\|^2 \ \forall j, 1 \leq j \leq k \right\}$$

Where each datum x_p is assigned to exactly one cluster, $S^{(t)}$.

k-Means

- Given an initial set of k means $m_1^{(1)}, \ldots, m_k^{(1)}$ k-means alternates between the following (2) steps:

(I) Assignment Step (i.e., the expectation step):

Assign each observation to the cluster whose mean has the least squared Euclidean distance, this is intuitively the "nearest" mean. Mathematically, this means partitioning the observations according to the Voroni tessellation generated by the means.

$$S_i^{(t)} = \left\{ x_p : \left\| x_p - m_i^{(t)} \right\|^2 \leq \left\| x_p - m_j^{(t)} \right\|^2 \ \forall j, 1 \leq j \leq k \right\}$$

(II) Update Step (i.e., the parameter maximization step):

- Calculate the new means to be the **centroids** of the observations in the new clusters.

$$m_i^{(t+1)} = \frac{1}{\left| S_i^{(t)} \right|} \sum_{x_j \in S_i^{(t)}} x_j$$

- The algorithm has **converged** when the assignments no longer change. There is no guarantee that the optimum is found using this algorithm.

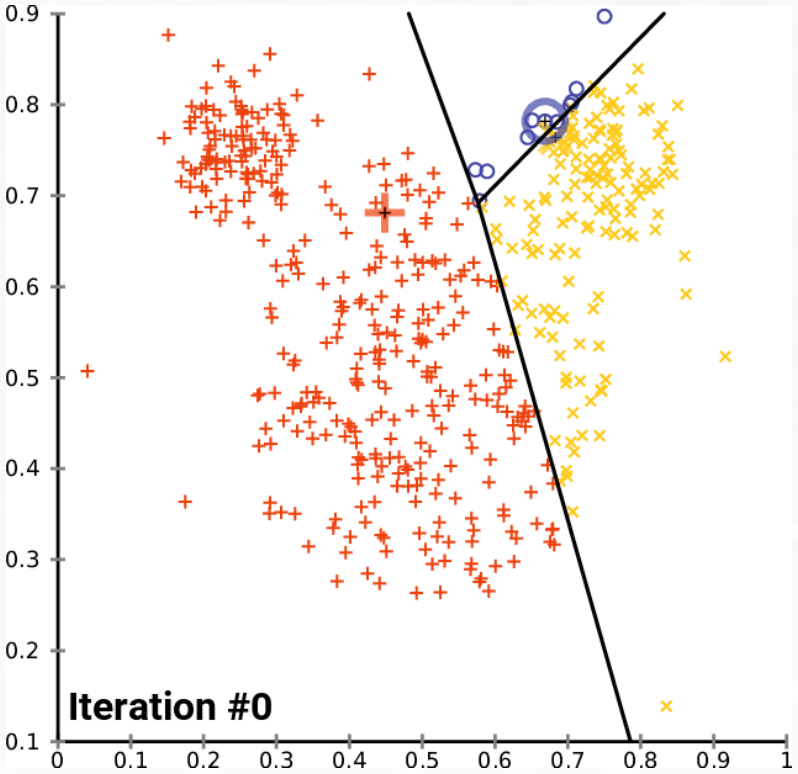
k-Means

(I) Assignment Step (i.e., the expectation step):

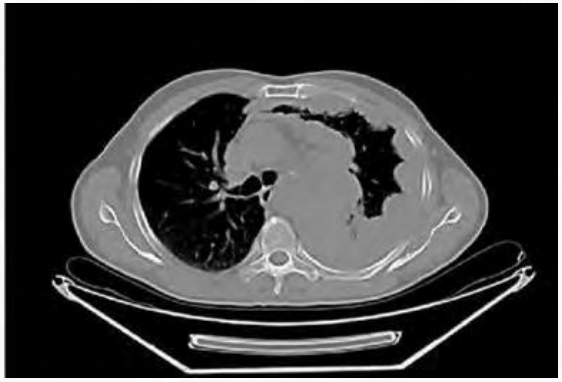
$$S_i^{(t)} = \left\{ x_p : \left\| x_p - m_i^{(t)} \right\|^2 \le \left\| x_p - m_j^{(t)} \right\|^2 \ \forall j, 1 \le j \le k \right\}$$

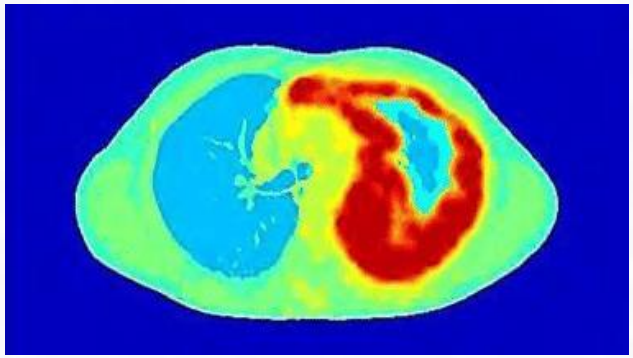
(II) Update Step (i.e., the parameter maximization step):

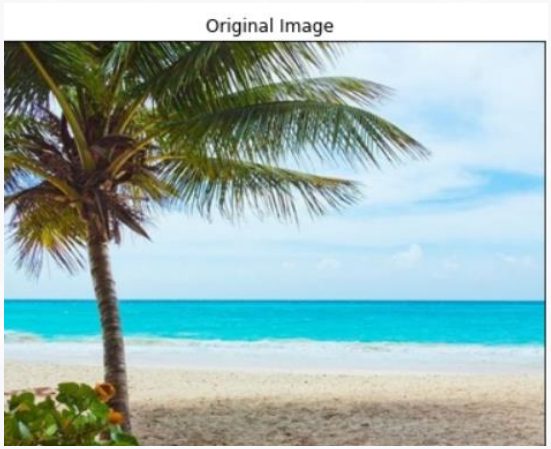
$$m_i^{(t+1)} = \frac{1}{\left| S_i^{(t)} \right|} \sum_{x_j \in S_i^{(t)}} x_j$$

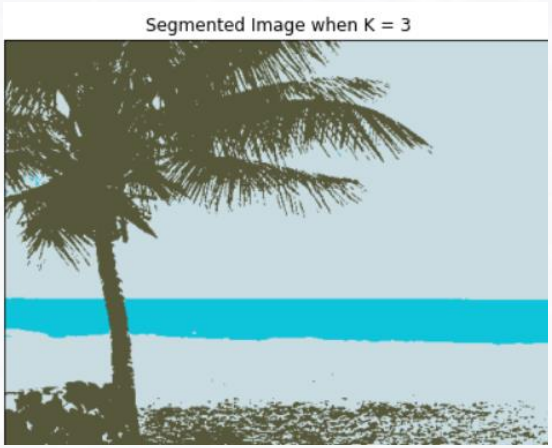


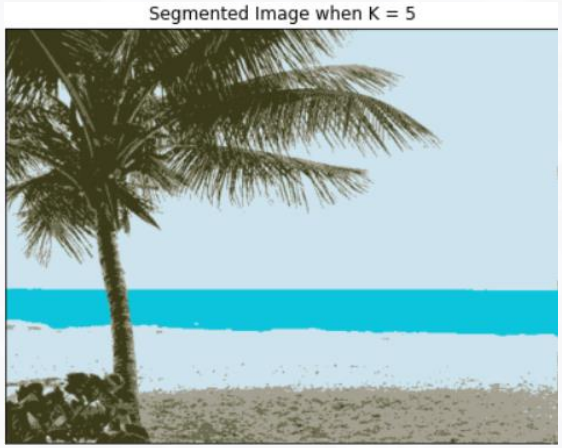
Example: Image segmentation by k-Means clustering







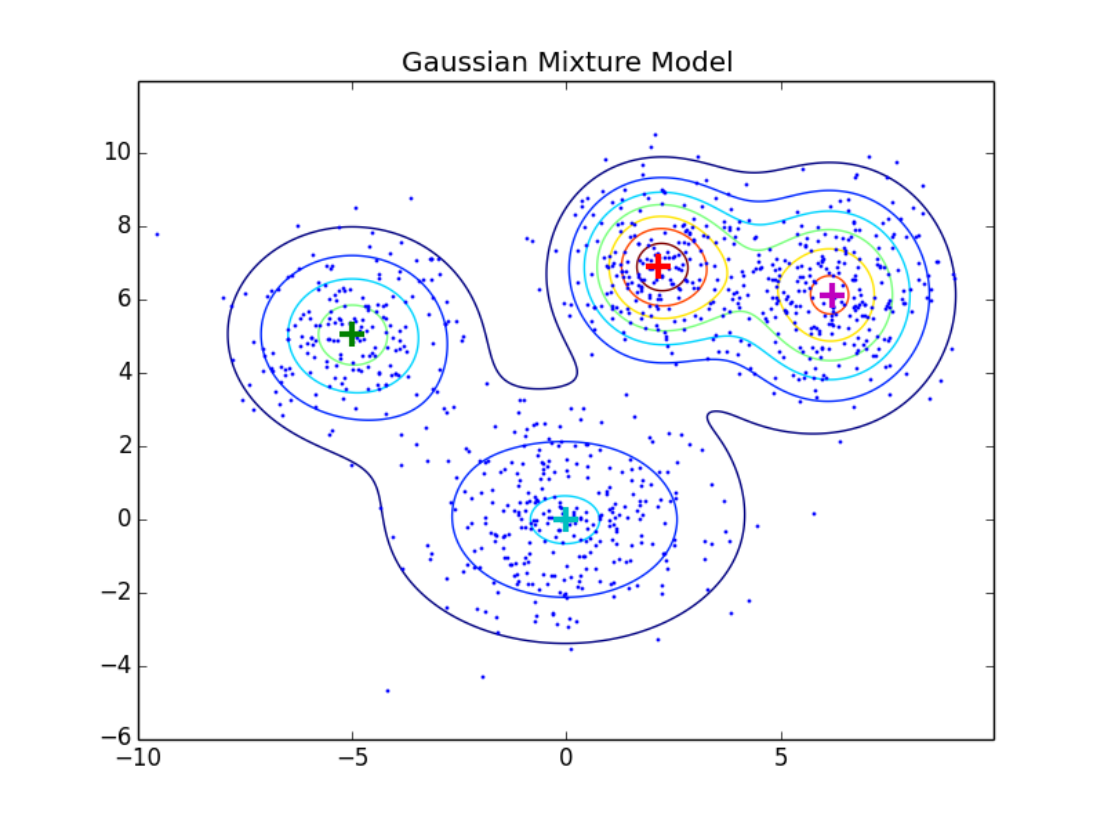




Review Topic:
GMMs

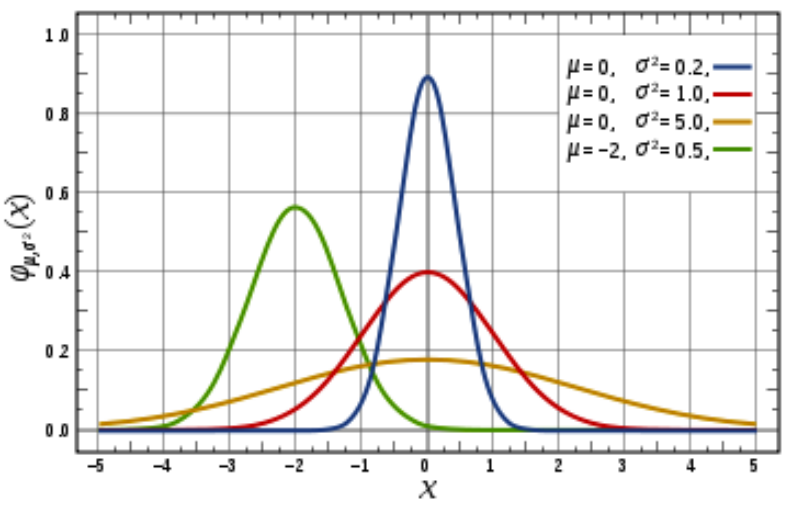
GMMs

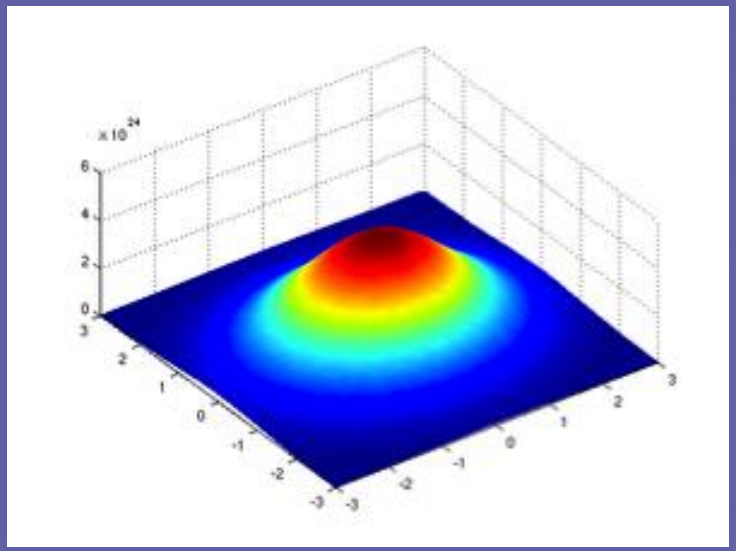
• A commonly used soft clustering model is the GMM (**Gaussian mixture model**); with GMMs, we assume (*a priori*) that the clusters resemble tightly-packed balls (i.e., Gaussian distributions).



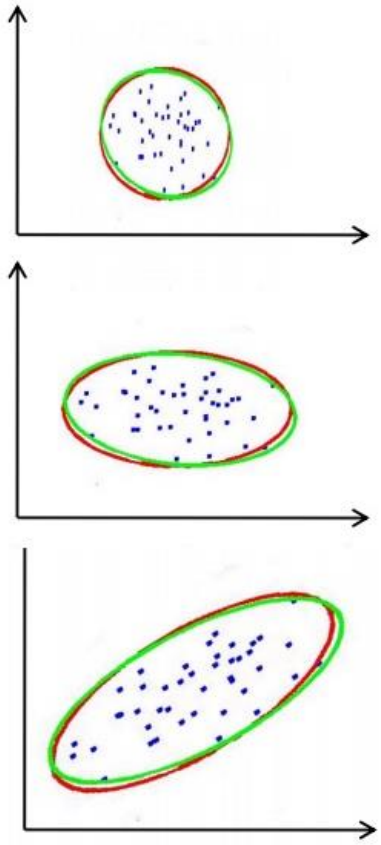
GMMs: Gaussian Distribution Review

$$N(x;\mu,\Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\}$$





GMMs: Gaussian Distribution Review



Covariance matrix Σ =

$$\begin{matrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{matrix}$$

Covariance matrix Σ =

$$\begin{matrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{matrix}$$

Covariance matrix Σ =

$$\begin{matrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{matrix}$$

Using different forms of covariance matrix allows for clusters of different shapes

GMMs

Main ideas for clustering using GMM:

- *Initialization*: given a data set, fix *k*, the number of clusters; initialize the mean (μ) and covariance matrices (Σ) for the k Gaussian clusters.

- Assign the data points to the k clusters (using a soft clustering) (**assignment step/E-step**)

- Update the parameters (i.e. μ, Σ) for each of the clusters. (**update step/M-step**)

…repeat until stopping condition/convergence

GMMs

Main ideas for clustering using GMM:

• *Initialization*: given a data set, fix *k*, the number of clusters; initialize the mean (μ) and covariance matrices (Σ) for the k Gaussian clusters.

• Assign the data points to the k clusters (using a soft clustering) (**assignment step/E-step**)

• Update the parameters (i.e. μ, Σ) and prior class estimates ($P(C_i|x)$ (for each of the clusters. (**update step/M-step**)

…repeat until stopping condition/convergence

What makes this problem challenging? There are, ostensibly, **many unknowns**!

• Strictly speaking, we don't know the cluster assignments nor any of the Gaussian distribution parameters.

GMMs

What makes this problem challenging? There are, ostensibly, **many unknowns**!

- Strictly speaking, we don't know the cluster assignments nor any of the Gaussian distribution parameters.

How can we simplify things?

A nice trick…**Solve each subproblem separately**!

GMMs

What makes this problem challenging? There are, ostensibly, **many unknowns**!

• Strictly speaking, we don't know the cluster assignments nor any of the Gaussian distribution parameters.

How can we simplify things?

A nice trick…Solve each subproblem separately!

(1) For instance, to find the optimal class assignments for each datum, use the current approximations for the Gaussian parameters distributions (i.e. treat μ and Σ as known for each cluster, as well as each class prior) and compute the class posterior: $P(C_i | x)$ using Bayes' Rule.

(2) Conversely, to find the optimal estimates for μ and Σ for each cluster, in addition to the class priors, use the current (soft) class posterior assignments and compute the MLE.

GMMs

(1) For instance, to find the optimal class assignments for each datum, use the current approximations for the Gaussian parameters distributions (i.e. treat μ and Σ as known for each cluster, as well as each class prior) and compute the class posterior: $P(C_i \mid x)$ using Bayes' Rule. (**assignment step/E-step**)

• Given the current estimates of both the parameters of each Gaussian cluster: $(\mu_1,\Sigma_1),\ldots,(\mu_k,\Sigma_k)$, and the prior for each cluster: $P(C_1)=\pi_1,\ldots, P(C_k)=\pi_k$, we compute the class posterior $P(C_i)$ using Bayes' Rule as follows:

$$P(C_i \mid x) = \frac{P(x \mid C_i)P(C_i)}{P(x)}$$

GMMs

(1) For instance, to find the optimal class assignments for each datum, use the current approximations for the Gaussian parameters distributions (i.e. treat μ and Σ as known for each cluster, as well as each class prior) and compute the class posterior: $P(C_i \mid x)$ using Bayes' Rule. (**assignment step/E-step**)

- Given the current estimates of both the parameters of each Gaussian cluster: $(\mu_1,\Sigma_1),\ldots,(\mu_k,\Sigma_k)$, and the prior for each cluster: $P(C_1)=\pi_1,\ldots, P(C_k)=\pi_k$, we compute the class posterior $P(C_i)$ using Bayes' Rule as follows:

$$P(C_i \mid x) = \frac{P(x \mid C_i)P(C_i)}{P(x)} \propto \pi_i \frac{1}{(2\pi)^{d/2}|\Sigma_i|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)\right]$$

GMMs

(2) To find the optimal estimates for μ and Σ for each cluster, in addition to the class priors, use the current (soft) class posterior assignments and compute the MLE. (**update step/M-step**)

- Observe that if we knew which points belong to, say cluster *i*, for a hard clustering, we can use the standard MLE estimates (from beginning statistics) to estimate the Gaussian parameters (μ and Σ) for each cluster, in addition to the cluster priors (e.g., $P(C_i)$). These standard parameter estimates are given as follows:

$$\hat{\pi}_i = \frac{n_i}{n} \qquad \hat{\mu}_i = \frac{1}{n_i} \sum_{\mathbf{x}^j \in C_i} \mathbf{x}^j \qquad \hat{\Sigma}_i = \frac{1}{n_i} \sum_{\mathbf{x}^j \in C_i} \left(\mathbf{x}^j - \hat{\mu}_i\right)\left(\mathbf{x}^j - \hat{\mu}_i\right)^T$$

cluster prior | cluster mean | cluster covariance matrix

where above, n_i denotes the size of the *i*th cluster.

GMMs: MLE Parameter Estimates

(2) To find the optimal estimates for μ and Σ for each cluster, in addition to the class priors, use the current (soft) class posterior assignments and compute the MLE. (**update step/M-step**)

• Observe that if we knew which points belong to, say cluster *i*, for a hard clustering, we can use the standard MLE estimates (from beginning statistics) to estimate the Gaussian parameters (μ and Σ) for each cluster, in addition to the cluster priors (e.g. $P(C_i)$. These standard parameter estimates are given as follows:

$$\hat{\pi}_i = \frac{n_i}{n} \qquad \hat{\mu}_i = \frac{1}{n_i} \sum_{\mathbf{x}^j \in C_i} \mathbf{x}^j \qquad \hat{\Sigma}_i = \frac{1}{n_i} \sum_{\mathbf{x}^j \in C_i} \left(\mathbf{x}^j - \hat{\mu}_i\right)\left(\mathbf{x}^j - \hat{\mu}_i\right)^T$$

cluster prior cluster mean cluster covariance matrix

where above, n_i denotes the size of the *i*th cluster.

(*) However, because we are executing a **soft clustering**, these parameter update formulae must incorporate the class posteriors: $P(C_i|x)$, for each i=1,…,k and for each data point x, respectively.

GMMs: Modified Parameter Estimates

(2) To find the optimal estimates for μ and Σ for each cluster, in addition to the class priors, use the current (soft) class posterior assignments and compute the MLE. (**update step/M-step**)

• Here are the parameter estimate formulas, updated to account for the soft clustering induced by the class posteriors: $P(C_i|x)$, for each i=1,…,k, for each data point:

$$\hat{\pi}_i = \frac{n_i}{n} \rightarrow \hat{\pi}_i = \frac{1}{n}\sum_{j=1,..,n} P\left(C_i \mid \mathbf{x}^j\right)$$

cluster prior modified formula

$$\hat{\mu}_i = \frac{1}{n_i}\sum_{\mathbf{x}^j \in C_i} \mathbf{x}^j \rightarrow \hat{\mu}_i = \frac{\sum_{j=1,..,n} \mathbf{x}^j P\left(C_i \mid \mathbf{x}^j\right)}{\sum_{j=1,..,n} P\left(C_i \mid \mathbf{x}^j\right)}$$

cluster mean modified formula

$$\hat{\Sigma}_i = \frac{1}{n_i}\sum_{\mathbf{x}^j \in C_i} \left(\mathbf{x}^j - \hat{\mu}_i\right)\left(\mathbf{x}^j - \hat{\mu}_i\right)^T \rightarrow \hat{\Sigma}_i = \frac{\sum_{j=1,..,n} P\left(C_i \mid \mathbf{x}^j\right)\left(\mathbf{x}^j - \hat{\mu}_i\right)\left(\mathbf{x}^j - \hat{\mu}_i\right)^T}{\sum_{j=1,..,n} P\left(C_i \mid \mathbf{x}^j\right)}$$

cluster covariance matrix modified formula

GMMs: Summary

Main ideas for clustering using GMM:

• *Initialization*: given a data set, fix *k*, the number of clusters; initialize the mean (μ) and covariance matrices (Σ) for the *k* Gaussian clusters, and cluster priors ($P(C_i)$).

(I) Assign the data points to the *k* clusters (using a soft clustering) (**assignment step/E-step**)

$$P(C_i \mid x) \propto \pi_i \frac{1}{(2\pi)^{d/2} |\Sigma_i|^{1/2}} \exp\left[-\frac{1}{2} (x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i) \right]$$

(II) Update the parameters (i.e. μ, Σ) for each of the clusters, including the cluster priors. (**update step/M-step**)

$$\hat{\pi}_i = \frac{1}{n} \sum_{j=1,..,n} P(C_i \mid \mathbf{x}^j)$$

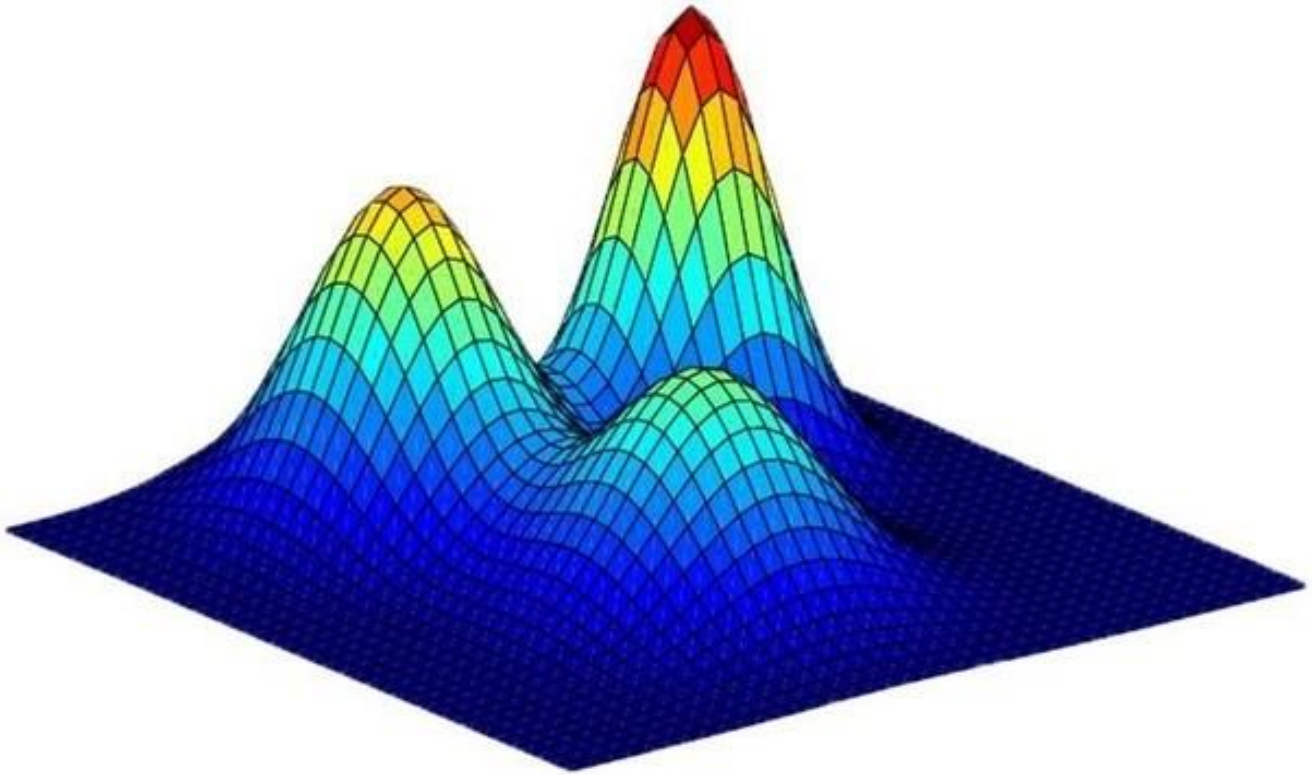
$$\hat{\mu}_i = \frac{\sum_{j=1,..,n} \mathbf{x}^j P(C_i \mid \mathbf{x}^j)}{\sum_{j=1,..,n} P(C_i \mid \mathbf{x}^j)}$$

$$\hat{\Sigma}_i = \frac{\sum_{j=1,..,n} P(C_i \mid \mathbf{x}^j)(\mathbf{x}^j - \hat{\mu}_i)(\mathbf{x}^j - \hat{\mu}_i)^T}{\sum_{j=1,..,n} P(C_i \mid \mathbf{x}^j)}$$

…repeat until stopping condition/convergence

GMMs

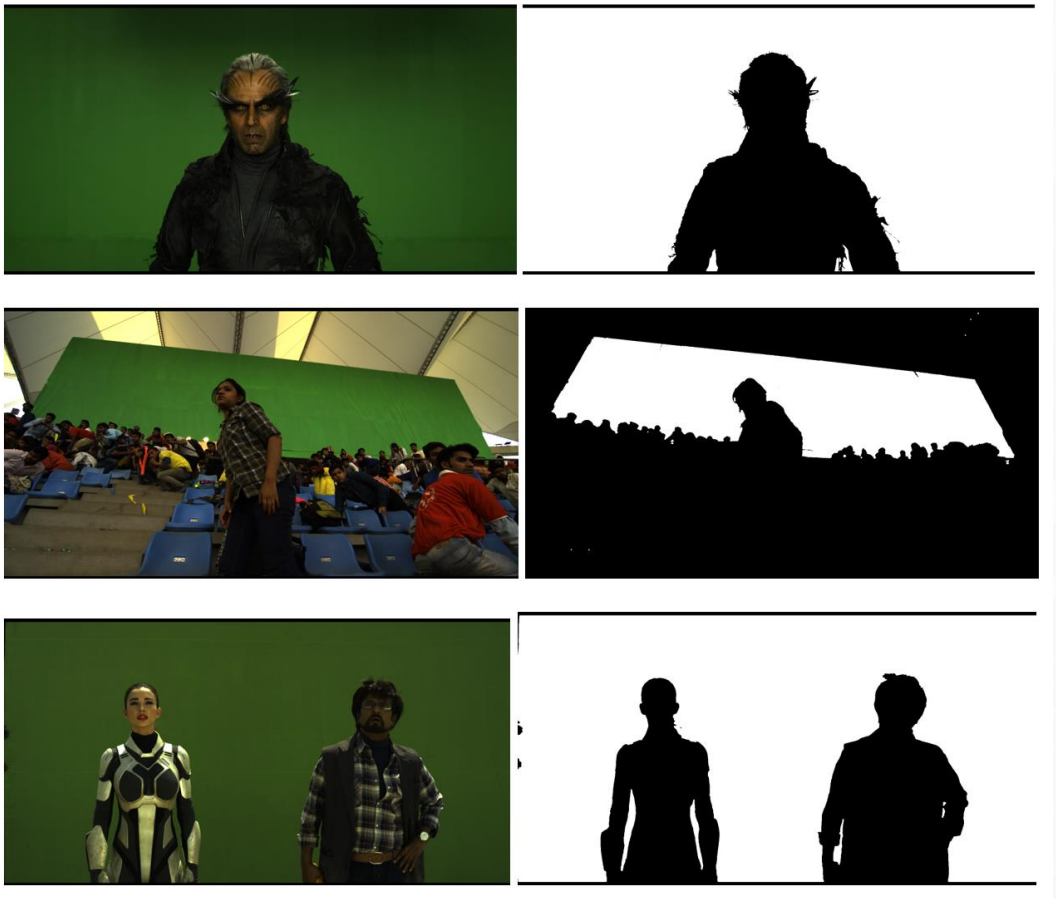
- Demo: https://lukapopijac.github.io/gaussian-mixture-model/



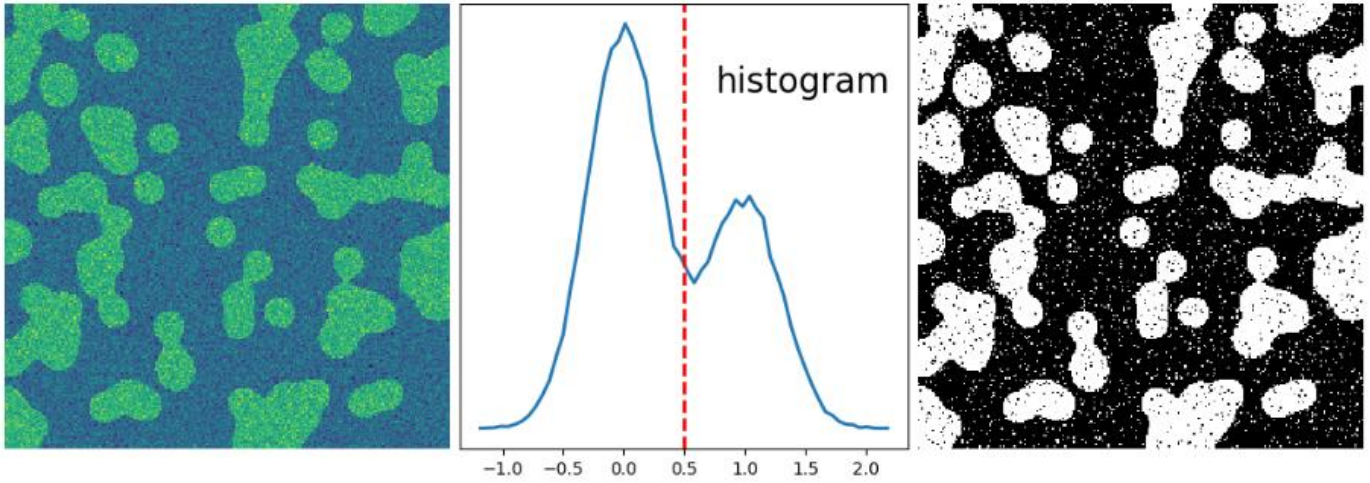
GMMs: Image Segmentation

Original image

Segmentation result using GMM with 3 components



GMMs: Image Segmentation




```python
import numpy as np
from scipy import ndimage
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture

np.random.seed(1)
n = 10
l = 256
im = np.zeros((l, l))
points = l*np.random.random((2, n**2))
im[(points[0]).astype(np.int), (points[1]).astype(np.int)] = 1
im = ndimage.gaussian_filter(im, sigma=l/(4.*n))

mask = (im > im.mean()).astype(np.float)


img = mask + 0.3*np.random.randn(*mask.shape)

hist, bin_edges = np.histogram(img, bins=60)
bin_centers = 0.5*(bin_edges[:-1] + bin_edges[1:])

classif = GaussianMixture(n_components=2)
classif.fit(img.reshape((img.size, 1)))

threshold = np.mean(classif.means_)
binary_img = img > threshold
```

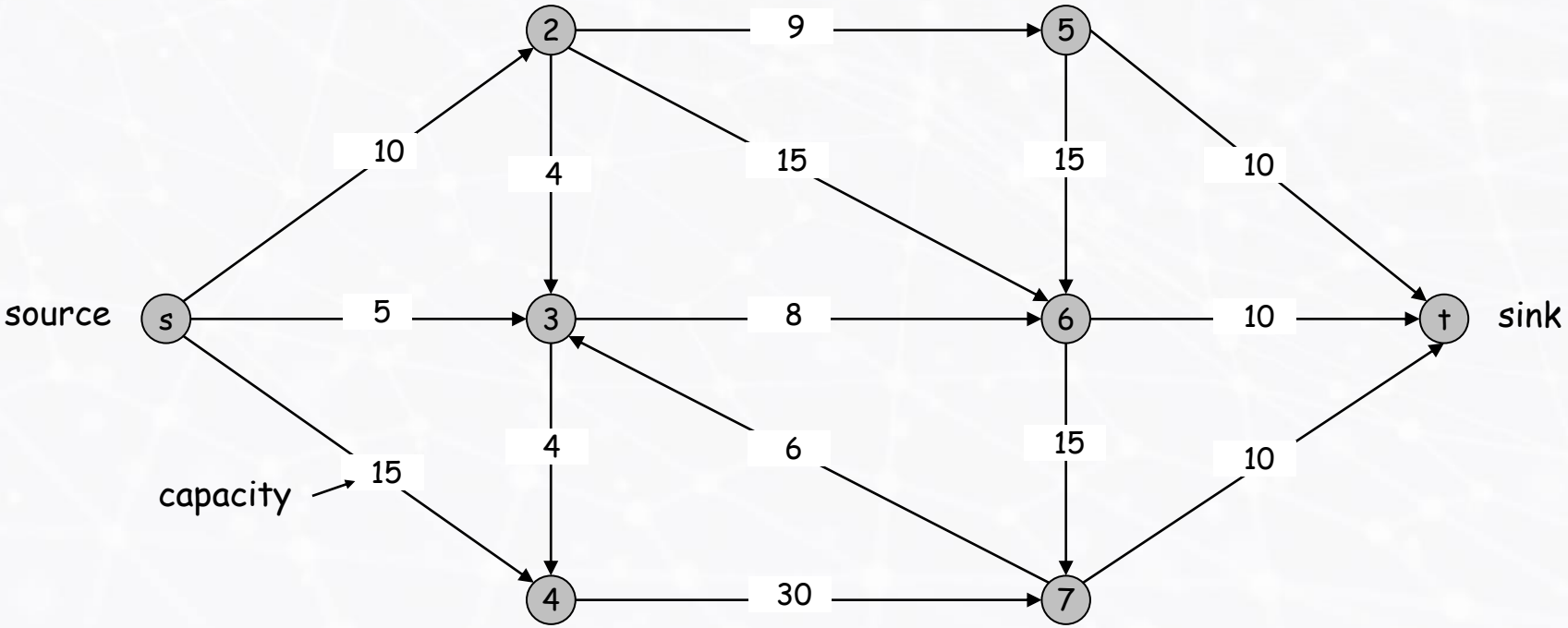
http://scipy-lectures.org/advanced/image_processing/auto_examples/plot_GMM.html

Review Topic:
Max Flow Min Cut
Ford-Fulkerson Algorithm

Minimum Cut Problem

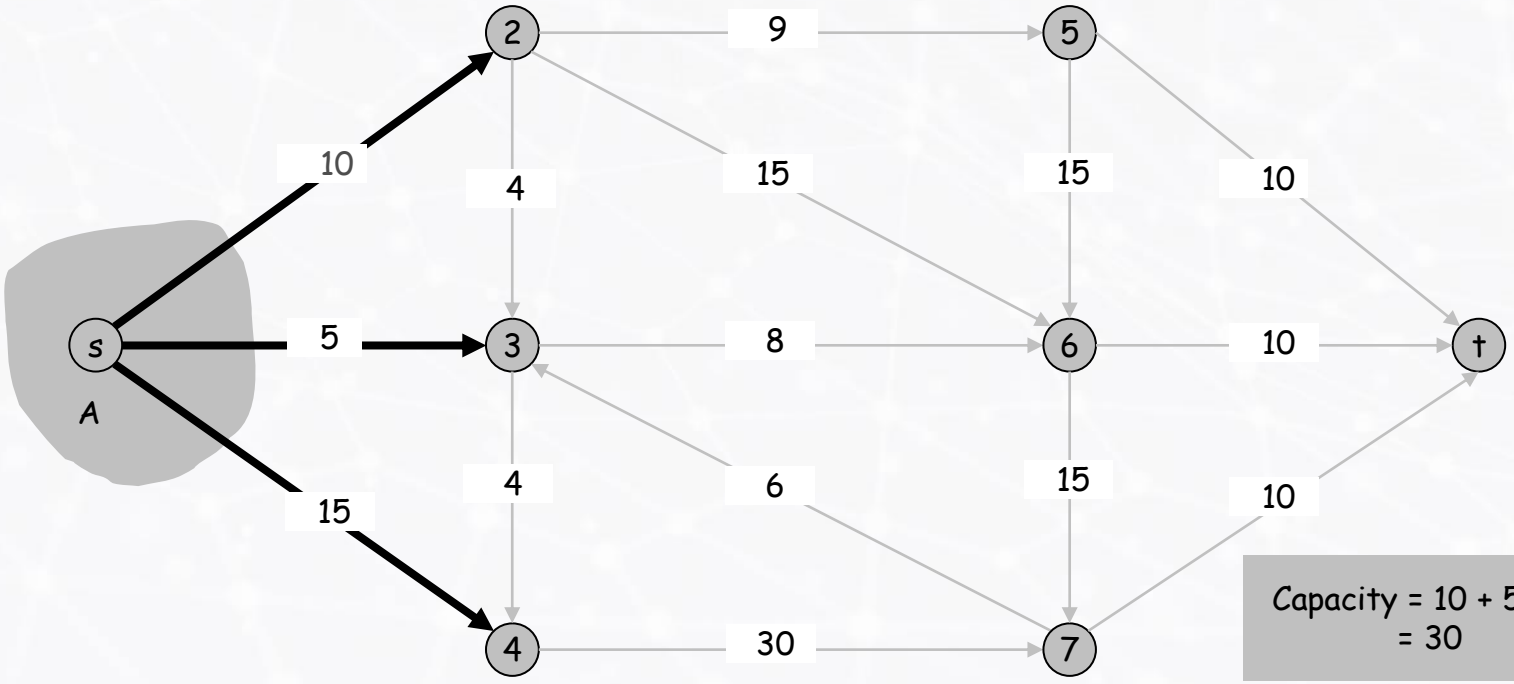
Flow network

- Abstraction for material flowing through the edges.
- G = (V, E) = directed graph, no parallel edges.
- Two distinguished nodes: s = source, t = sink.
- c(e) = capacity of edge e.



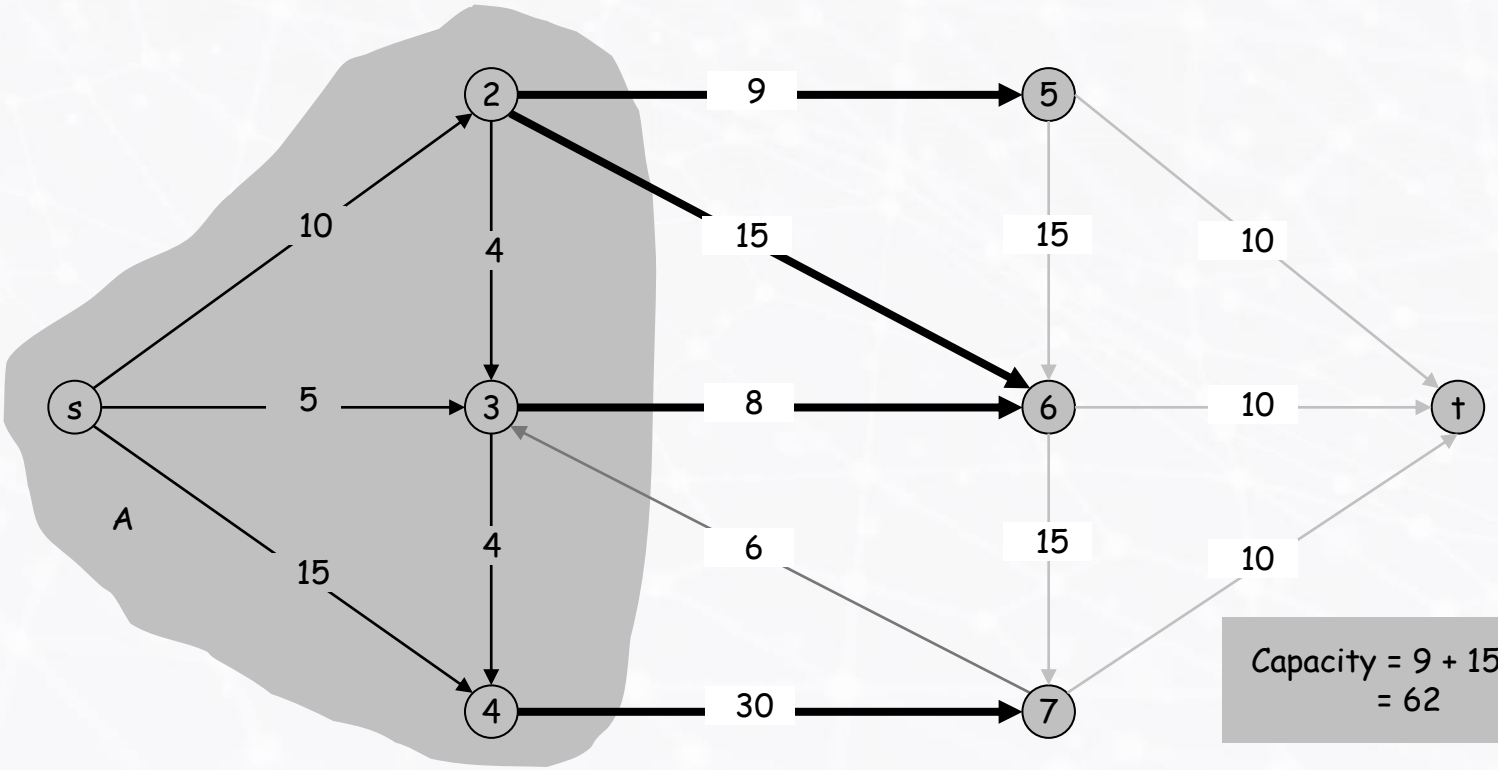
Minimum Cut Problem

- **Def.** An s-t cut is a partition (A, B) of V with $s \in A$ and $t \in B$.
- **Def.** The capacity of a cut (A, B) is: $cap(A, B) = \sum_{e \text{ out of } A} c(e)$



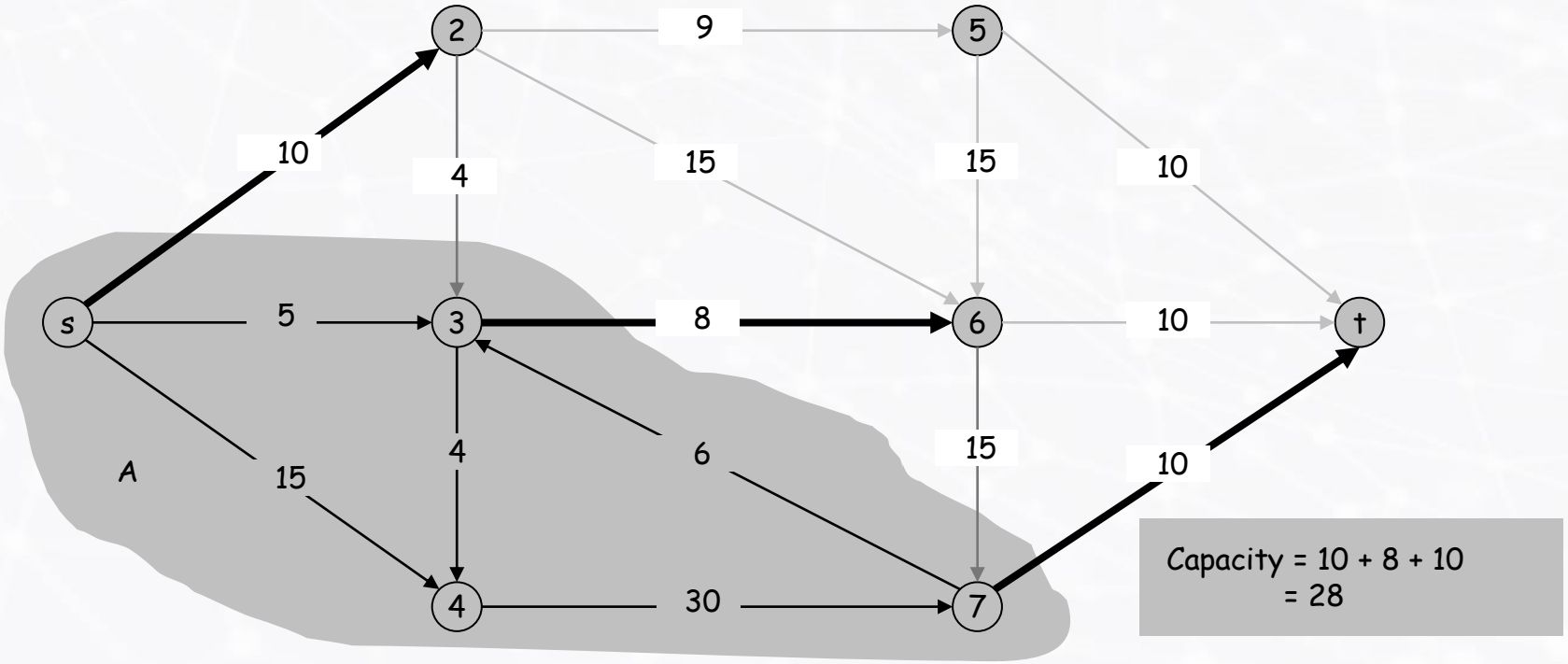
Minimum Cut Problem

- **Def.** An s-t cut is a partition (A, B) of V with $s \in A$ and $t \in B$.
- **Def.** The capacity of a cut (A, B) is: $cap(A, B) = \sum_{e \text{ out of } A} c(e)$



Minimum Cut Problem

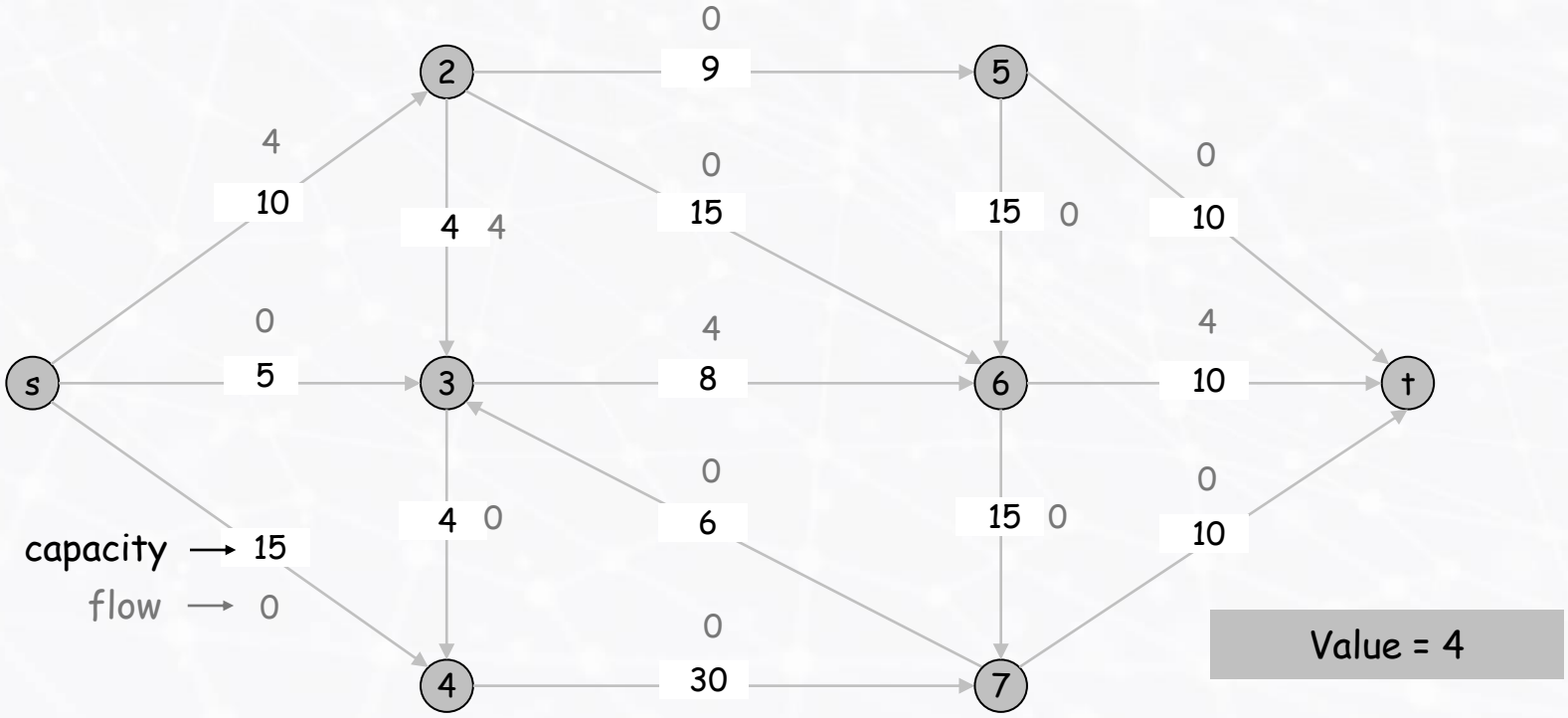
- **Min s-t cut problem.** Find an s-t cut of minimum capacity.



Max Flow Problem

- **Def.** An **s-t flow** is a function that satisfies:
  - For each e ∈ E: $0 \le f(e) \le c(e)$ [capacity]
  - For each v ∈ V – {s, t}: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [conservation]

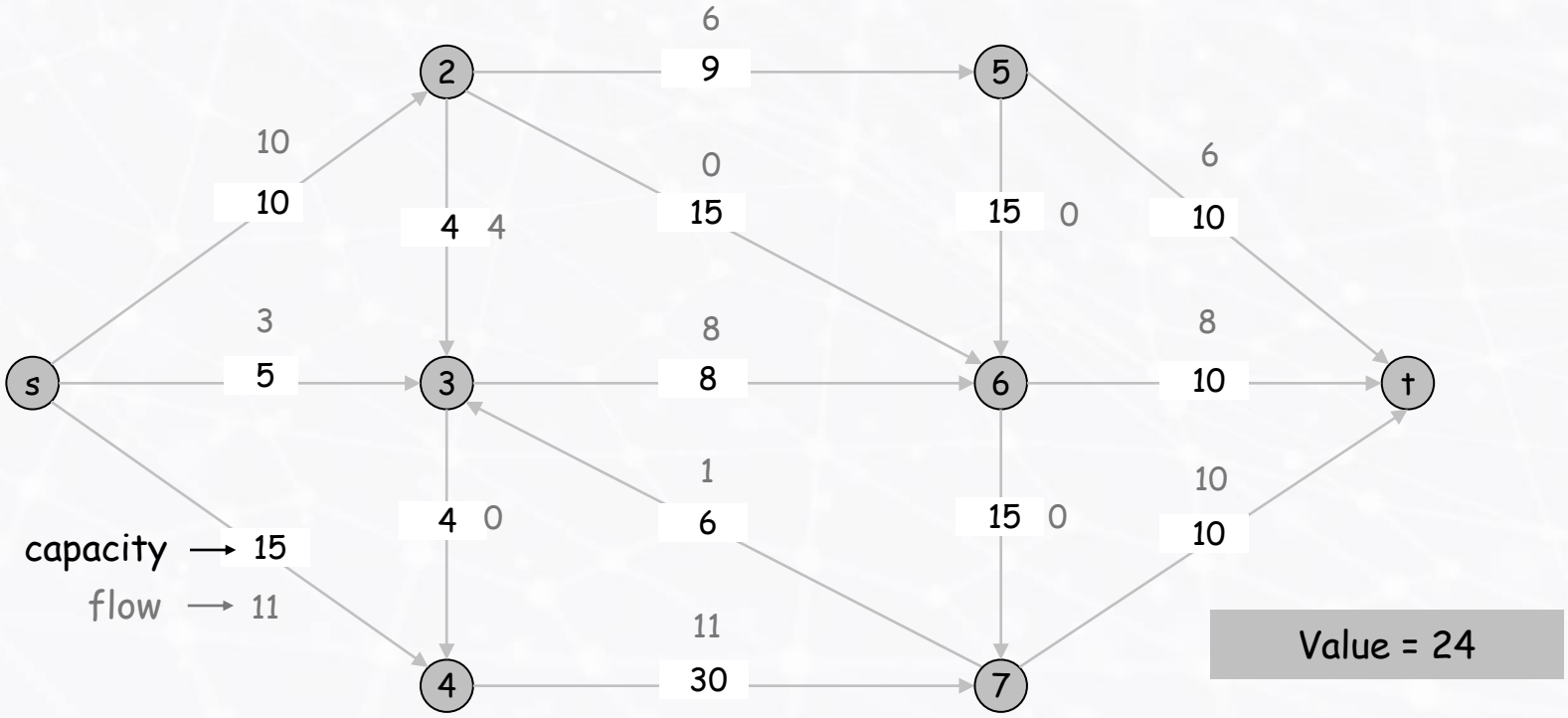
- **Def.** The **value** of a flow f is: $v(f) = \sum_{e \text{ out of } s} f(e)$.



Max Flow Problem

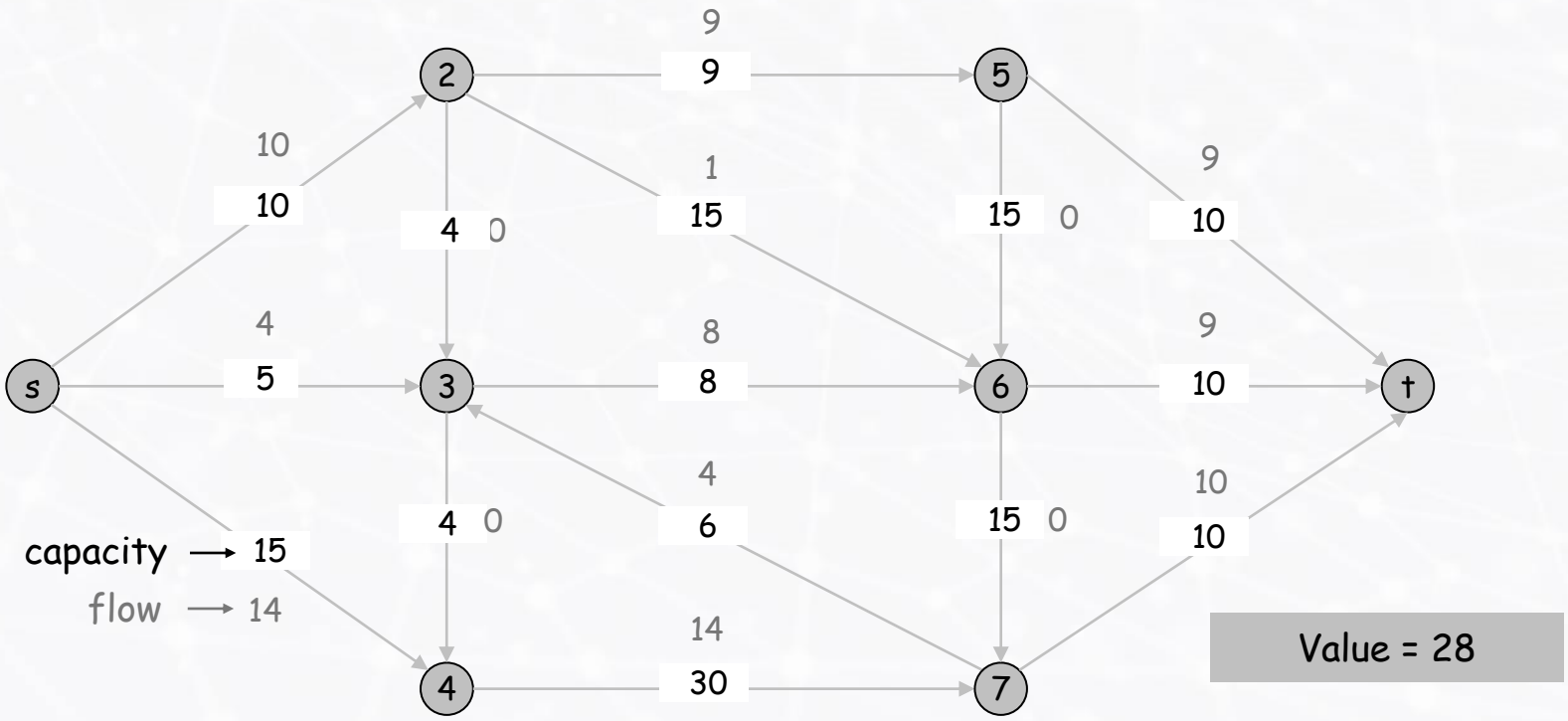
- **Def.** An **s-t flow** is a function that satisfies:
  - For each e ∈ E: $0 \le f(e) \le c(e)$ [capacity]
  - For each v ∈ V – {s, t}: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [conservation]

- **Def.** The **value** of a flow f is: $v(f) = \sum_{e \text{ out of } s} f(e)$.



Max Flow Problem

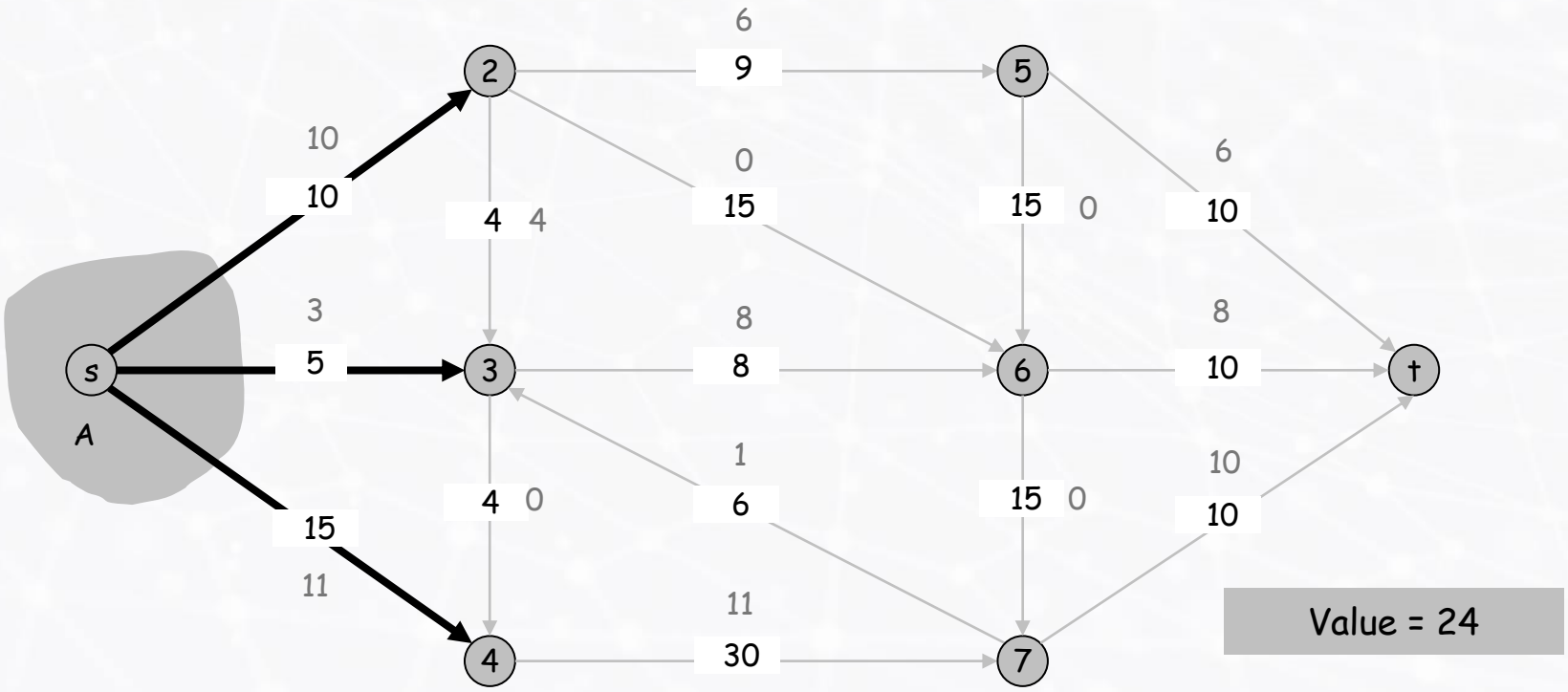
- **Max flow problem**. Find s-t flow of maximum value.



Max Flow Problem

• **Flow value lemma.** Let f be any flow, and let (A, B) be any s-t cut. Then, the net flow sent across the cut is equal to the amount leaving s.

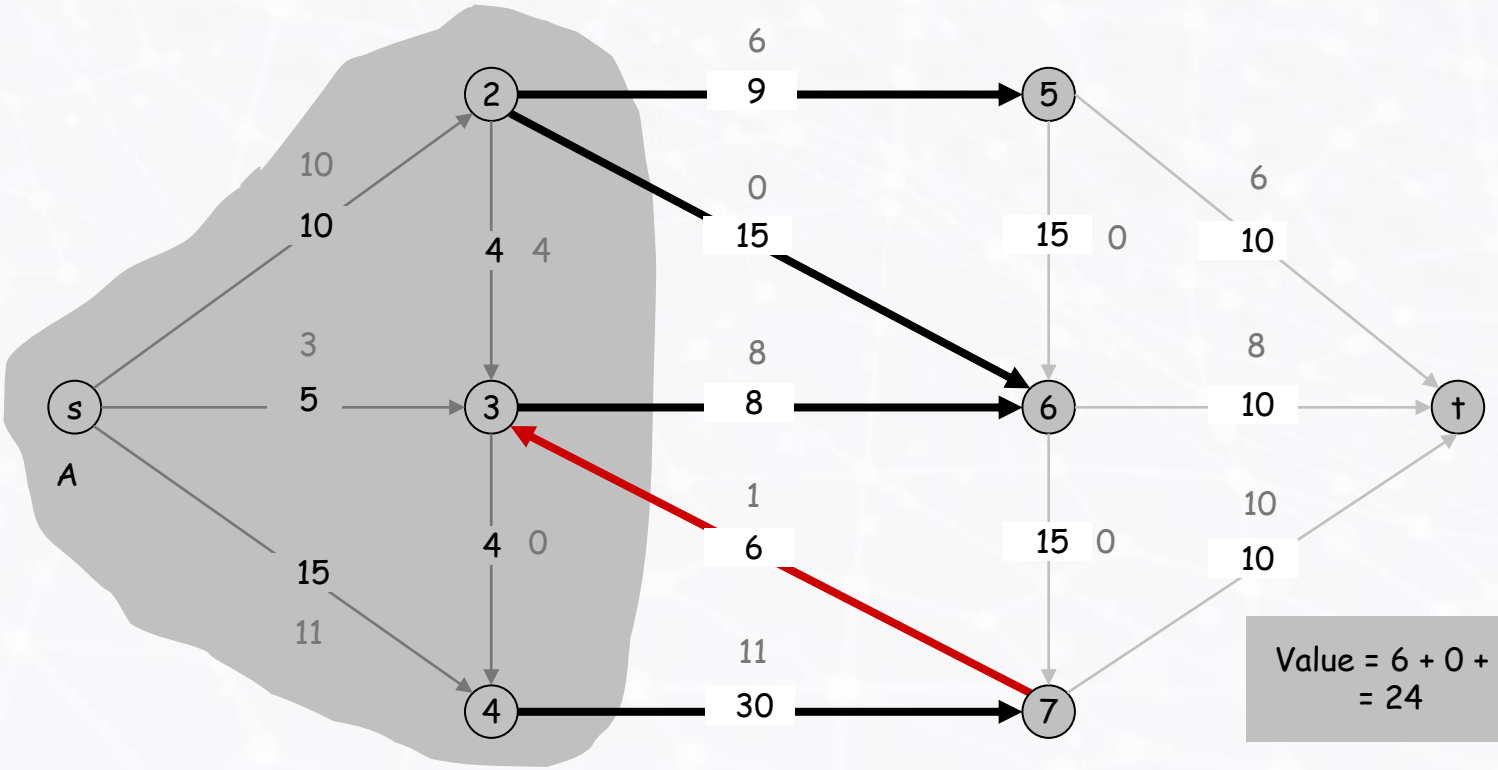
$$\sum_{e \text{ out of } A} f(e) \; - \sum_{e \text{ in to A}} f(e) \; = \; v(f)$$



Max Flow Problem

• **Flow value lemma.** Let f be any flow, and let (A, B) be any s-t cut. Then, the net flow sent across the cut is equal to the amount leaving s.

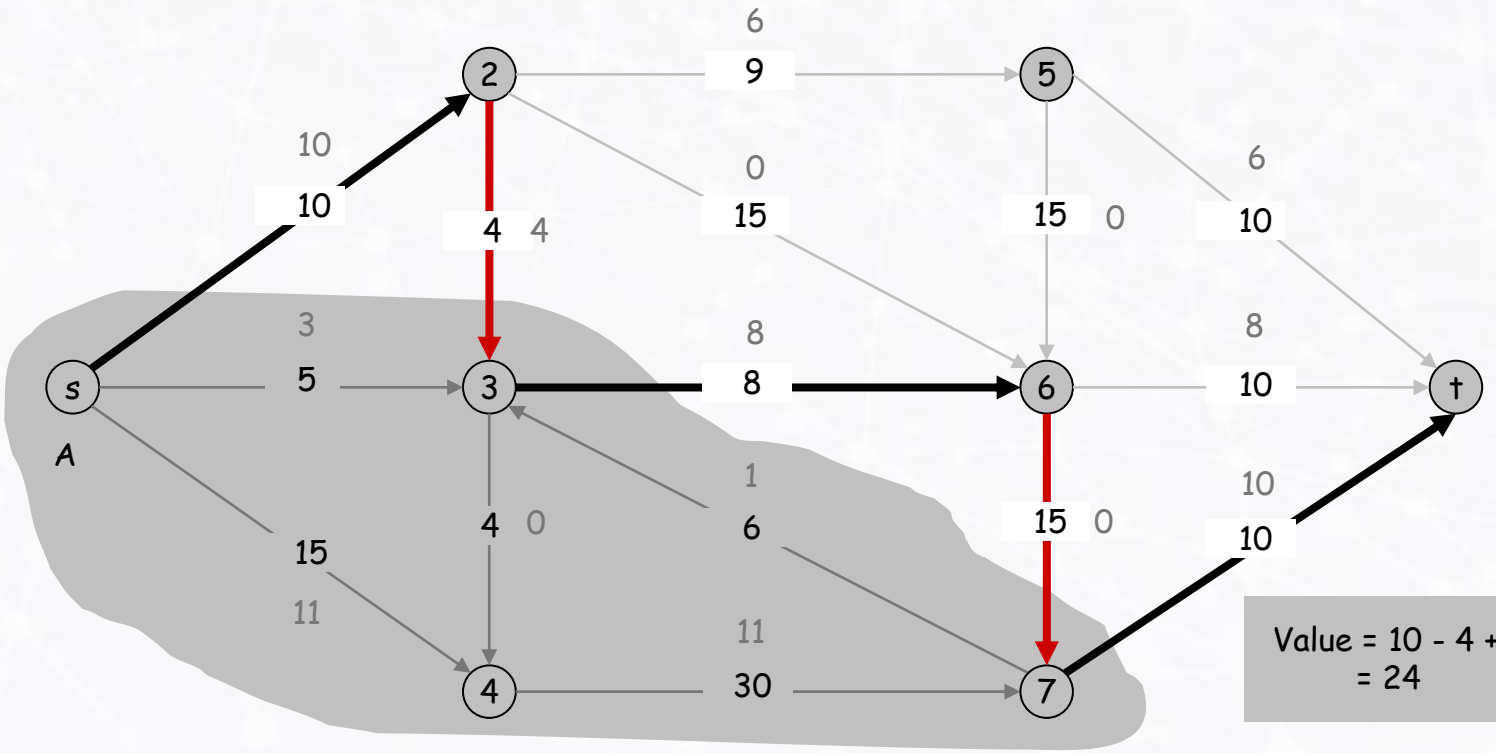
$$\sum_{e \text{ out of } A} f(e) \; - \sum_{e \text{ in to A}} f(e) \; = \; v(f)$$



Max Flow Problem

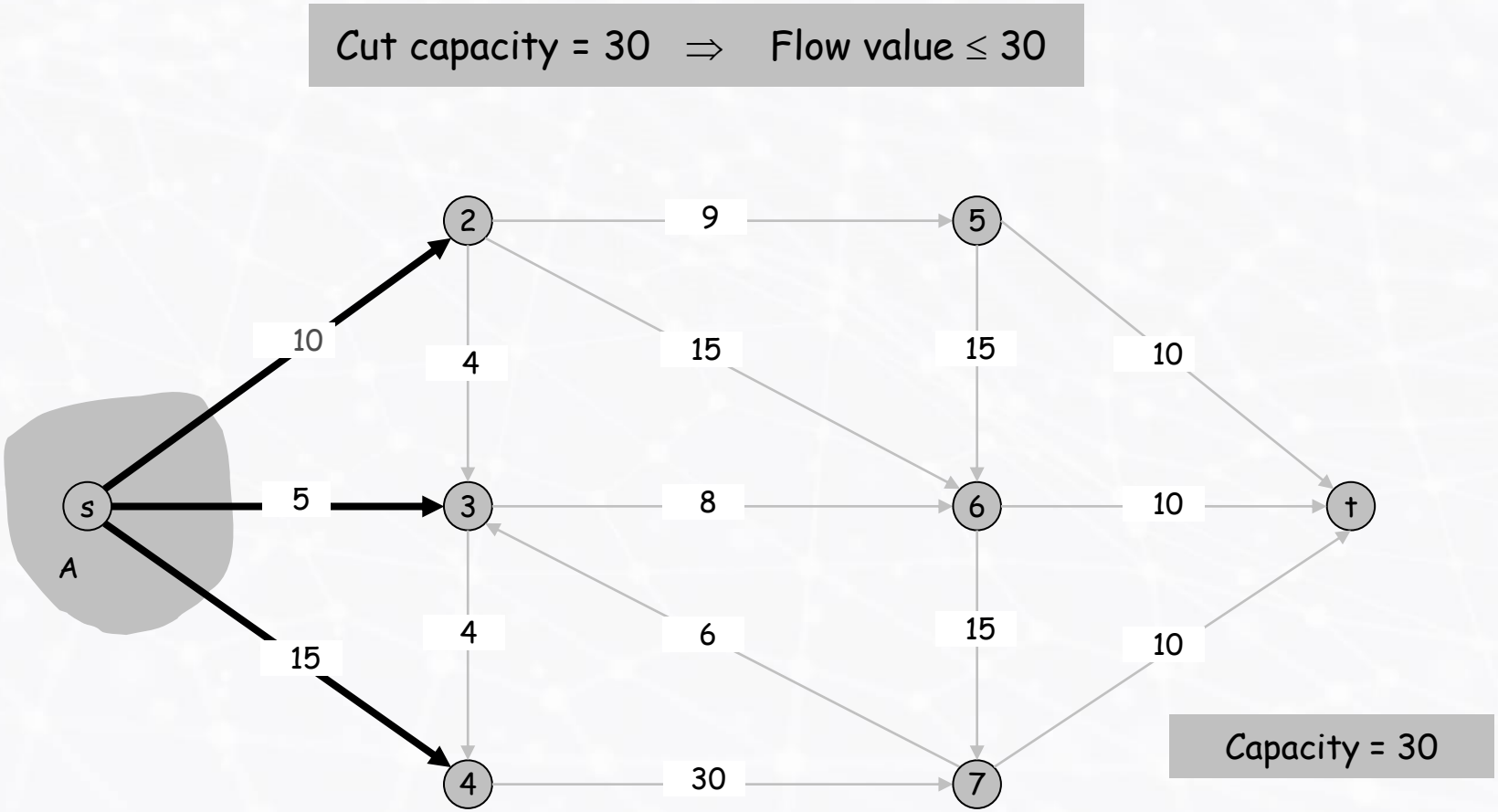
• **Flow value lemma**. Let f be any flow, and let (A, B) be any s-t cut. Then, the net flow sent across the cut is equal to the amount leaving s.

$$\sum_{e \text{ out of } A} f(e) \; - \sum_{e \text{ in to A}} f(e) \; = \; v(f)$$



Flows and Cuts

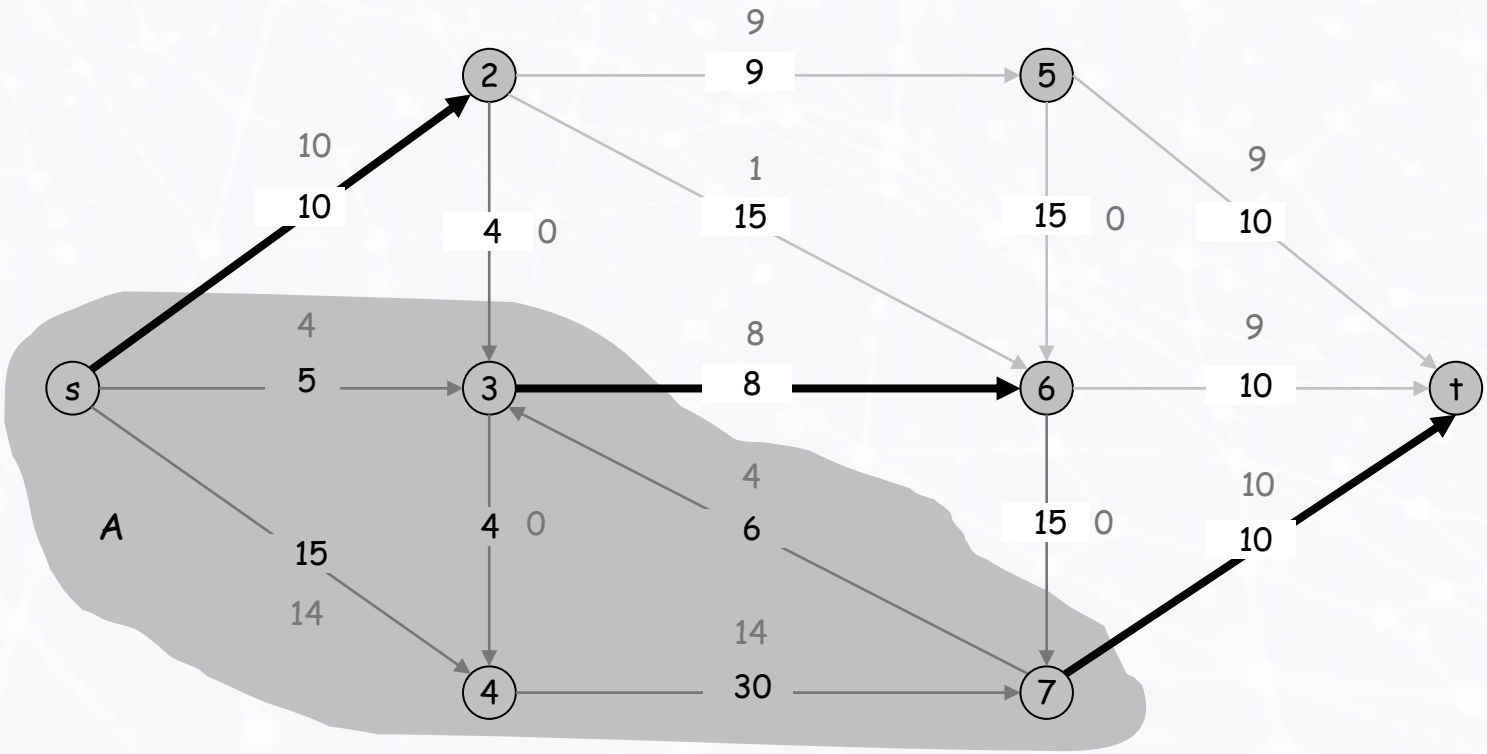
• **Weak duality**. Let f be any flow, and let (A, B) be any s-t cut. Then the value of the flow is at most the capacity of the cut.



Flows and Cuts

• **Corollary**. Let f be any flow, and let (A, B) be any cut.
If v(f) = cap(A, B), then f is a max flow and (A, B) is a min cut.

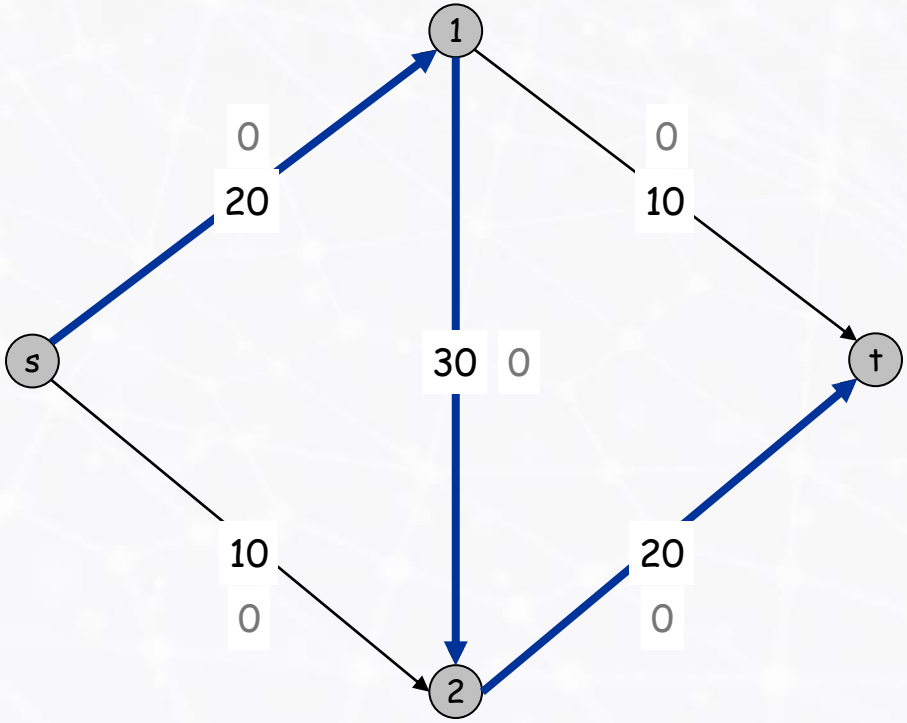
Value of flow = 28
Cut capacity = 28 ⇒ Flow value ≤ 28



Towards a Max Flow Algorithm

Greedy algorithm.

- Start with f(e) = 0 for all edge e ∈ E.
- Find an s-t path P where each edge has f(e) < c(e).
- Augment flow along path P.
- Repeat until you get stuck.

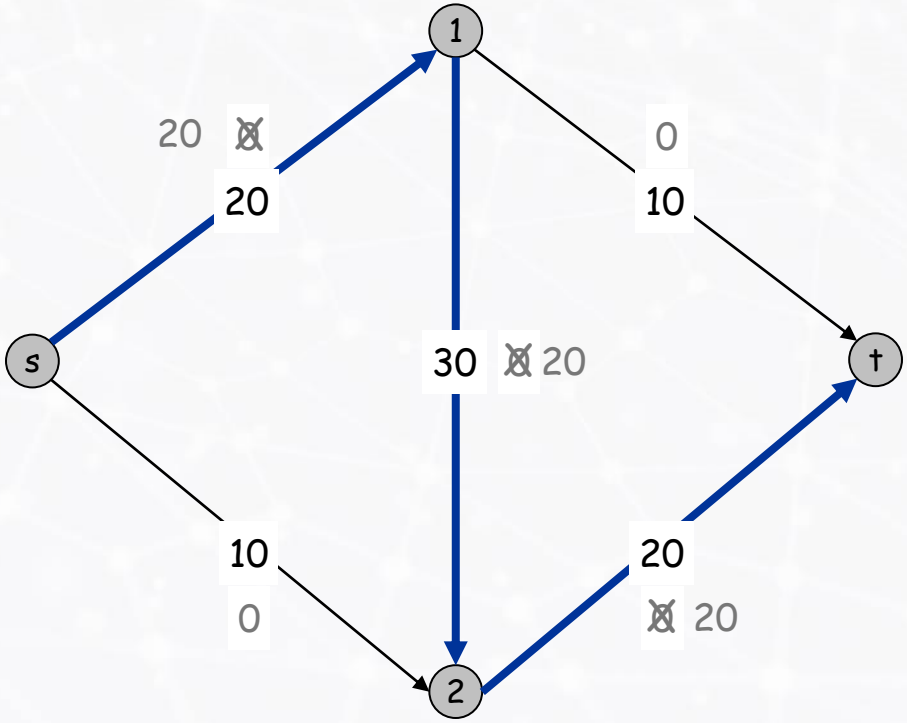


Flow value = 0

Towards a Max Flow Algorithm

Greedy algorithm.

- Start with f(e) = 0 for all edge e ∈ E.
- Find an s-t path P where each edge has f(e) < c(e).
- Augment flow along path P.
- Repeat until you get stuck.



Flow value = 20

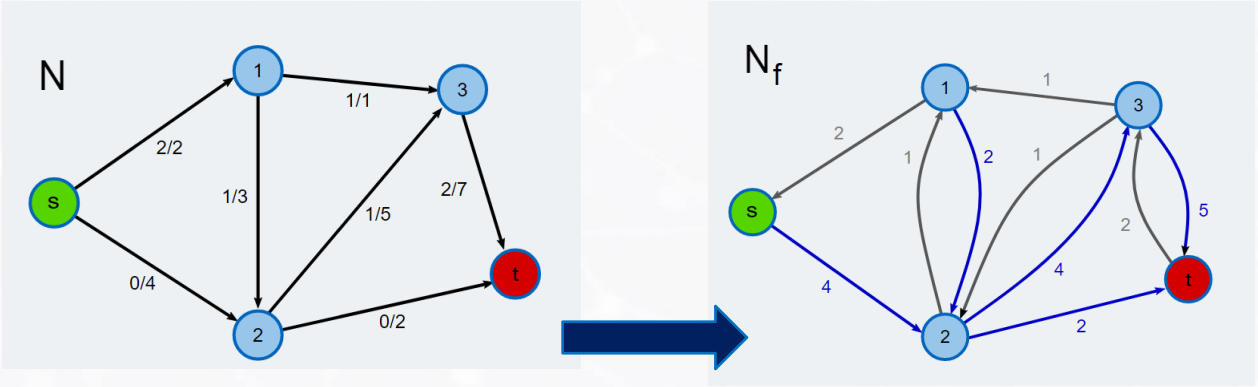
Ford-Fulkerson Algorithm

• The Ford-Fulkerson Algorithm (FFA) computes a **maximum flow** in an iterative manner by starting with a valid flow, and then making adjustments that fulfill the constraints and increase the flow.

Ford-Fulkerson Algorithm

• The Ford-Fulkerson Algorithm (FFA) computes a **maximum flow** in an iterative manner by starting with a valid flow, and then making adjustments that fulfill the constraints and increase the flow.

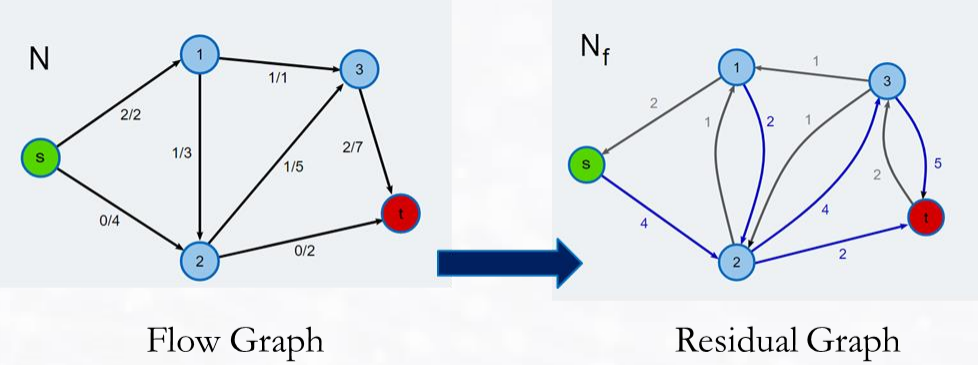
• To achieve this, FFA utilizes the **residual graph**. This is a graph generated by calculating how the flow along each edge can be modified - each edge in the network graph is replaced by up to two new edges, a forward edge with the same direction that that signifies how much the flow can be increased, and a backward edge storing how much the flow can be reduced.



Flow Graph

Residual Graph

Ford-Fulkerson Algorithm



Flow Graph Residual Graph

• The algorithm starts with an empty flow (which is always valid) and then **repeatedly finds paths in the residual graph from source to target**. Adding just enough flow along the path to saturate one edge (i.e., "bottlenecking"), which is the one with the lowest capacity, keeps the constraints on the flow fulfilled and strictly increases the flow. These are called **augmenting paths**.

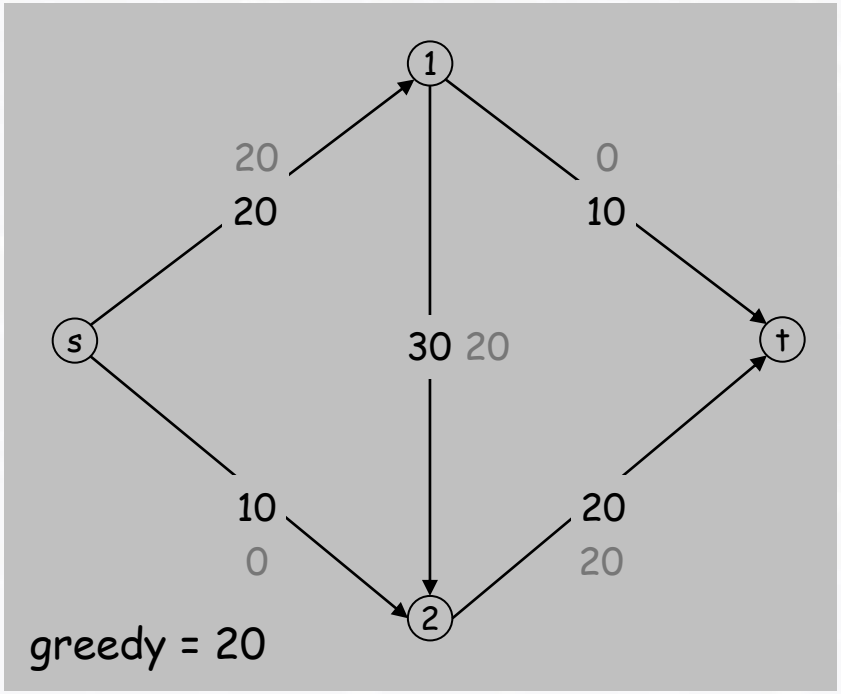
• The FFA does not explicitly state how to find the augmenting paths, and so the algorithm is agnostic to the mechanism used to find an augmenting path (in practice BFS is commonly used).

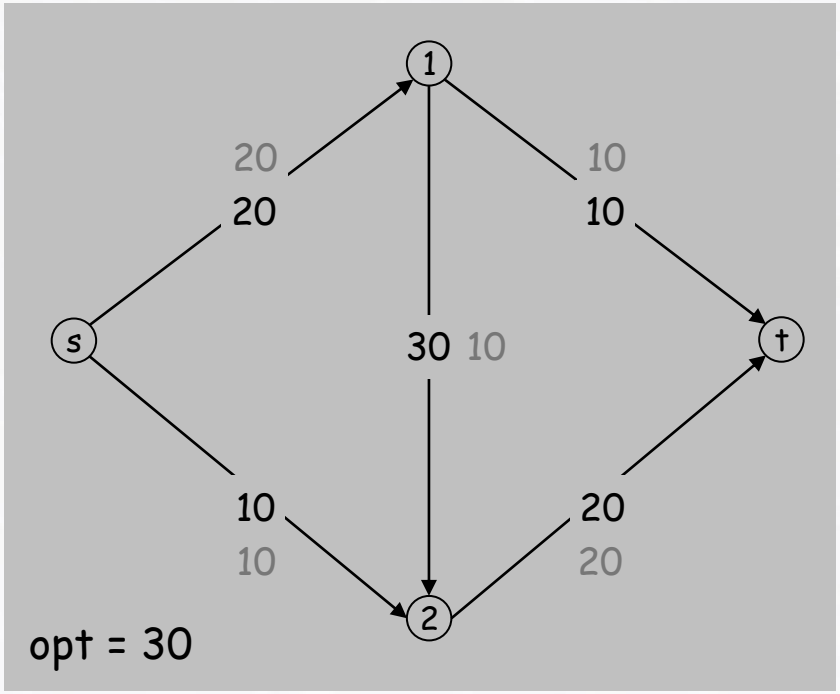
Ford-Fulkerson Algorithm

Greedy algorithm. (polynomial time solution)

- Start with f(e) = 0 for all edge e ∈ E.
- Find an s-t path P where each edge has f(e) < c(e).
- Augment flow along path P.
- Repeat until you get stuck.

locally optimality ⇏ global optimality

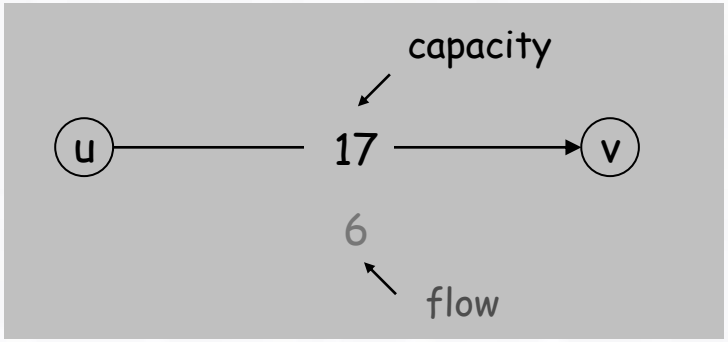




Ford-Fulkerson Algorithm

Original edge. $e = (u, v) \in E$.

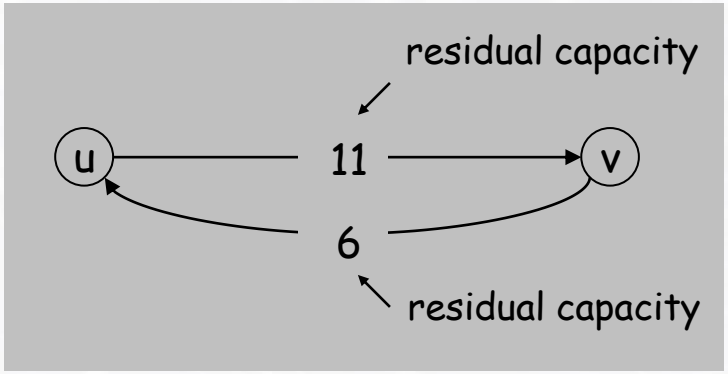
- Flow f(e), capacity c(e).



Residual edge.

- "Undo" flow sent.
- $e = (u, v)$ and $e^R = (v, u)$.
- Residual capacity:

$$c_f(e) = \begin{cases} c(e) - f(e) & \text{if } e \in E \\ f(e) & \text{if } e^R \in E \end{cases}$$

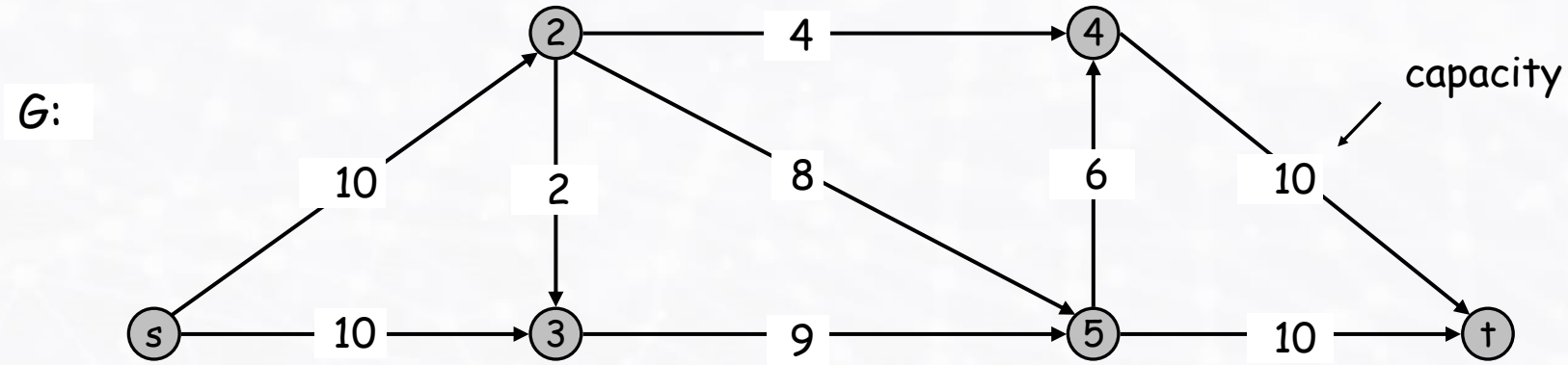


Residual graph: $G_f = (V, E_f)$.

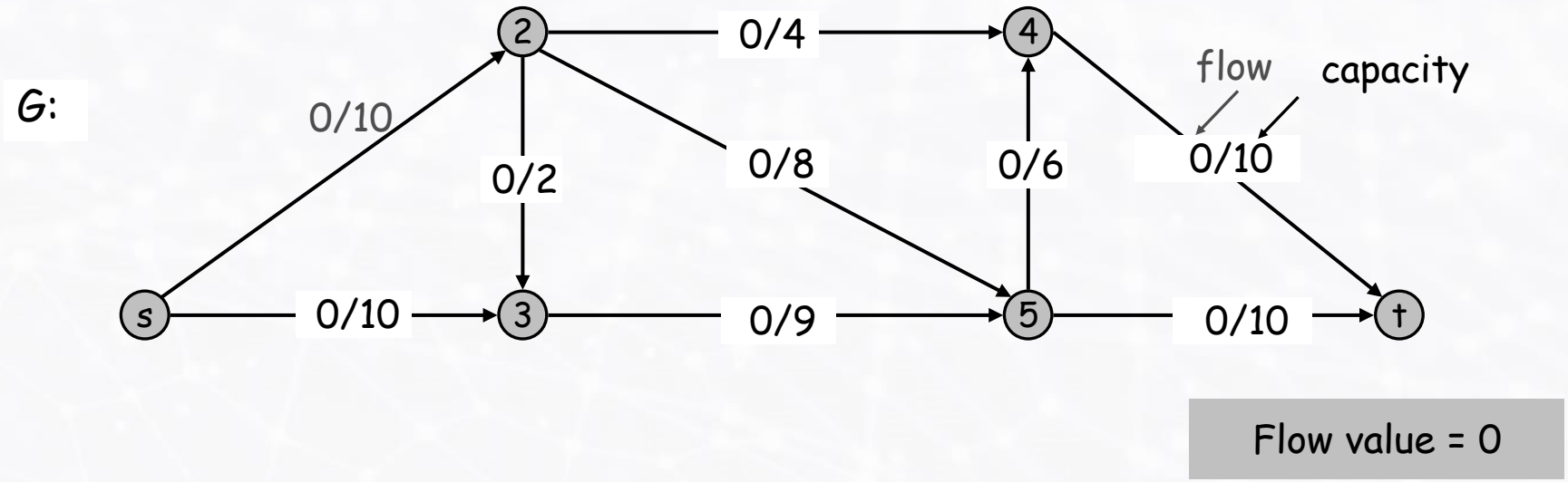
- Residual edges with positive residual capacity.
- $E_f = \{e : f(e) < c(e)\} \cup \{e^R : f(e) > 0\}$.

Ford-Fulkerson Algorithm Demo

Ford-Fulkerson Algorithm

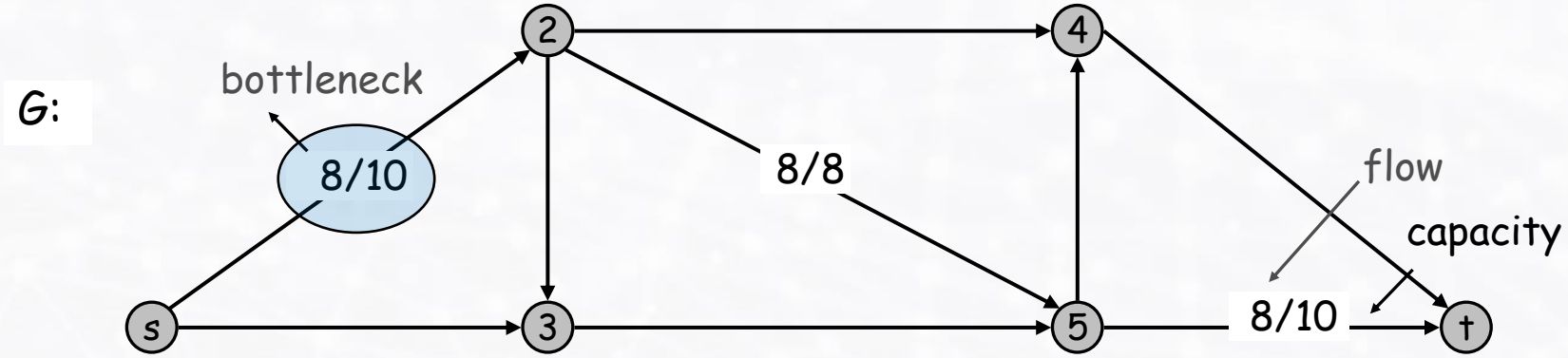


Ford-Fulkerson Algorithm

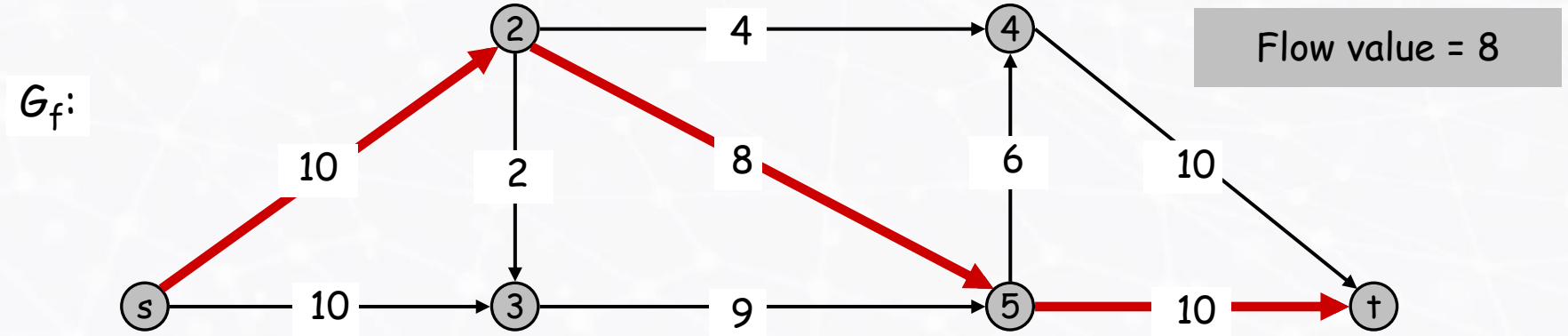


- FFA begins with an empty flow.

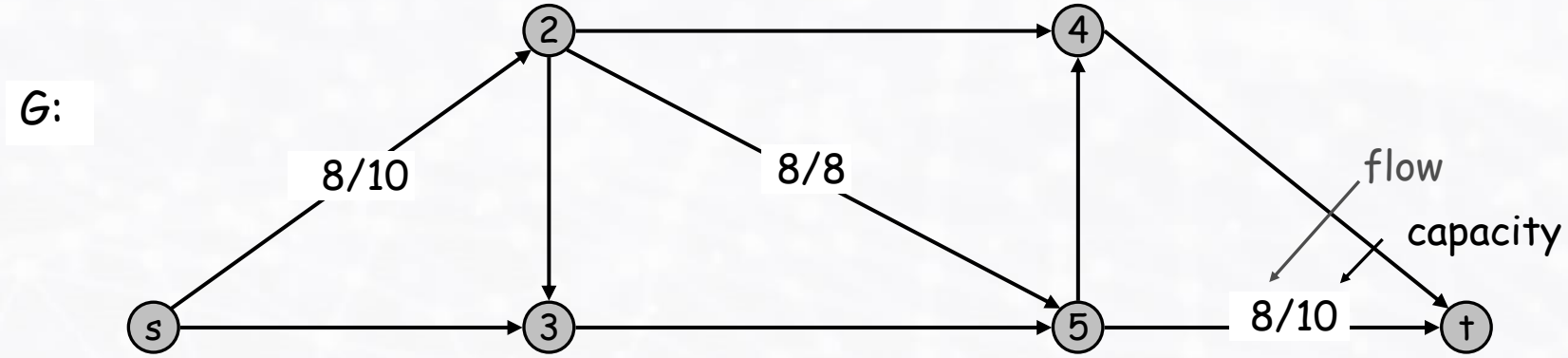
Ford-Fulkerson Algorithm



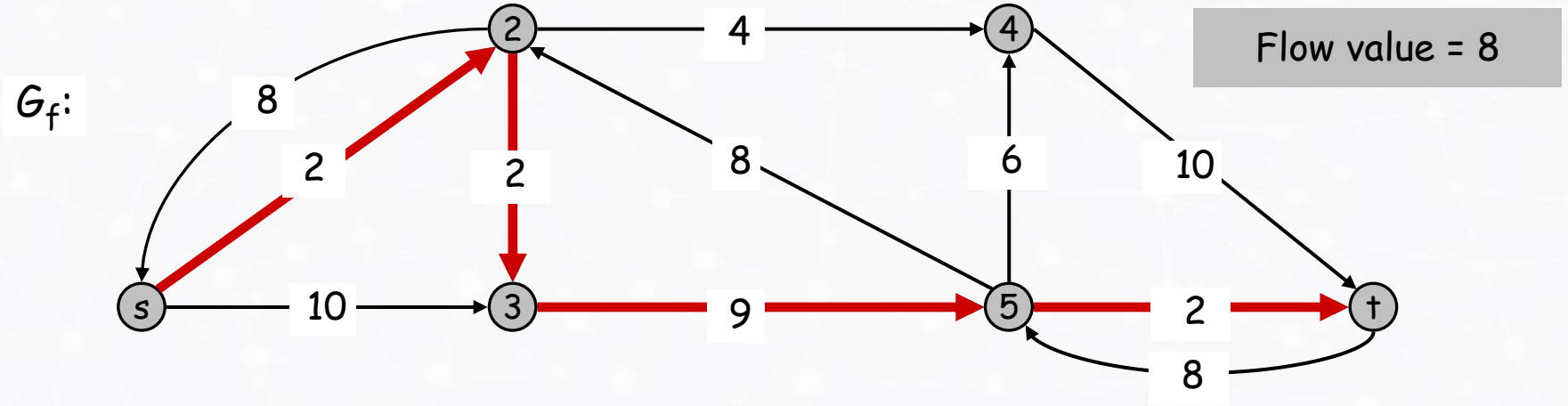
• Choose a valid s-t path. Notice that the path $s \rightarrow 2$ is a potential bottleneck, as it has an additional capacity of 2 this is currently unused.



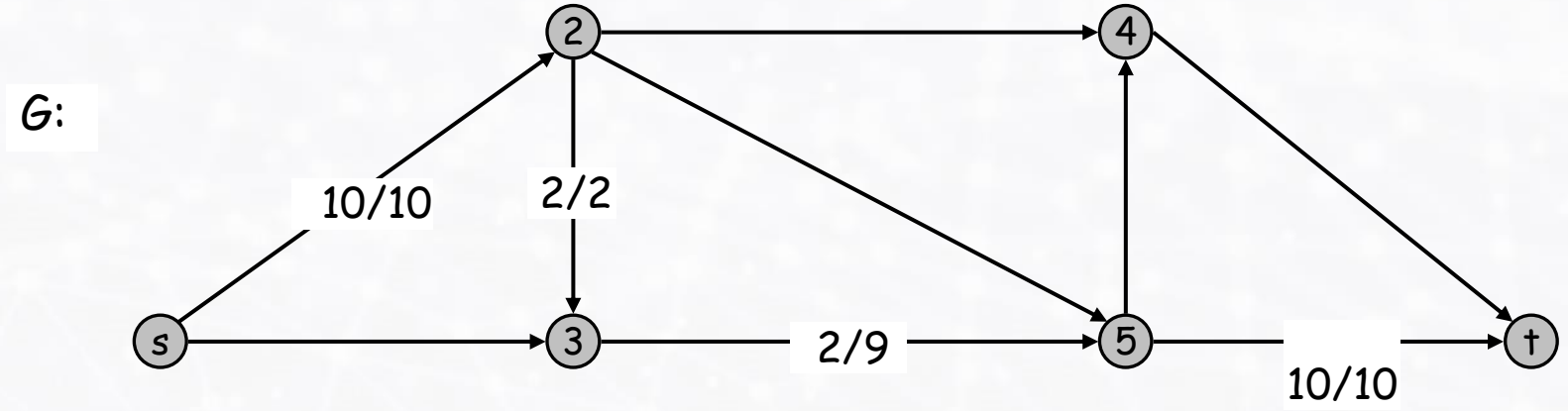
Ford-Fulkerson Algorithm

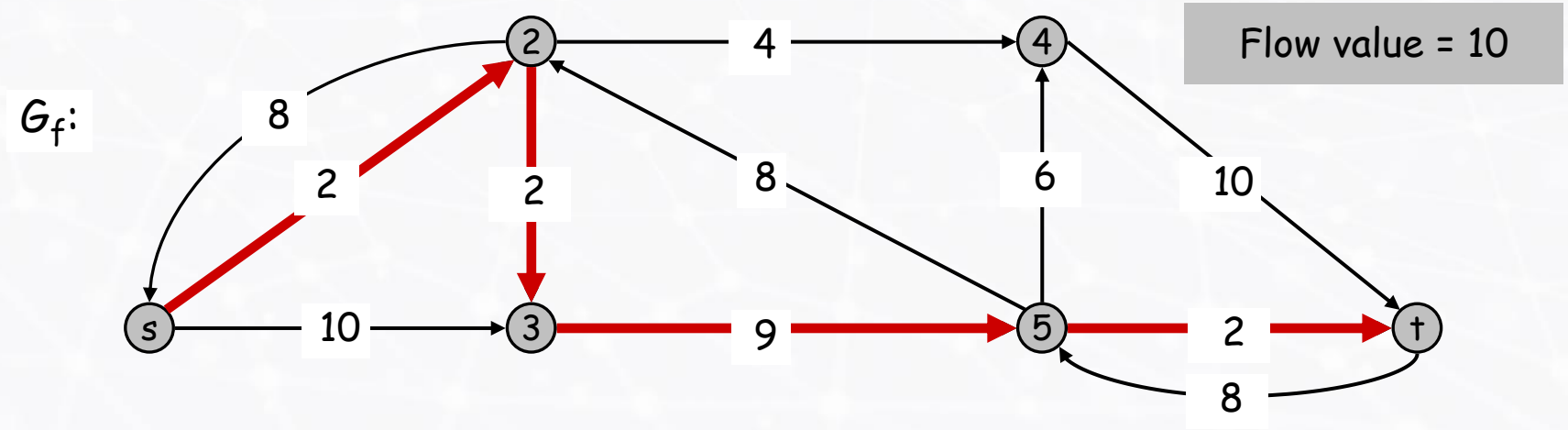


• The residual graph below shows an augmenting path allowing us to add 2 to the overall flow.



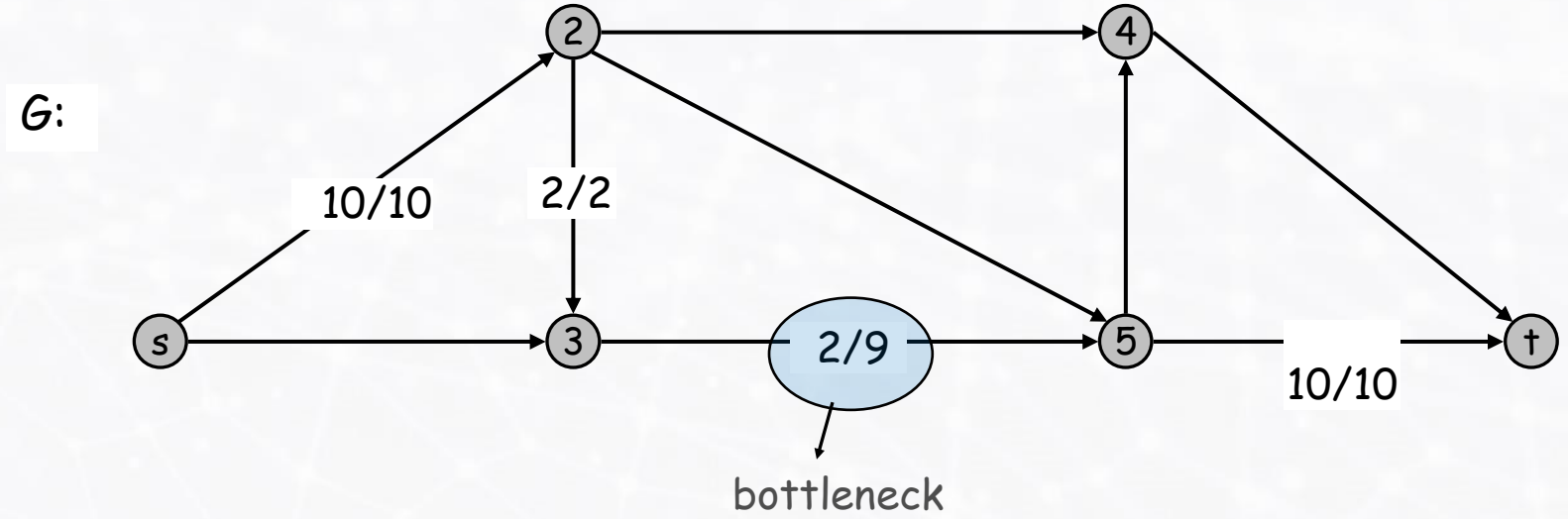
Ford-Fulkerson Algorithm

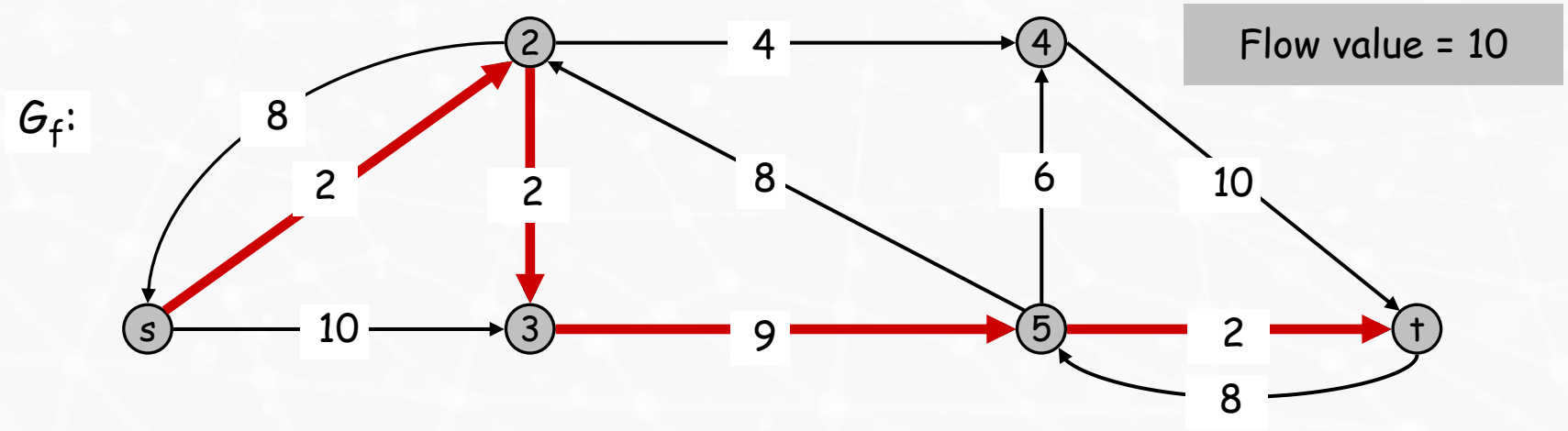




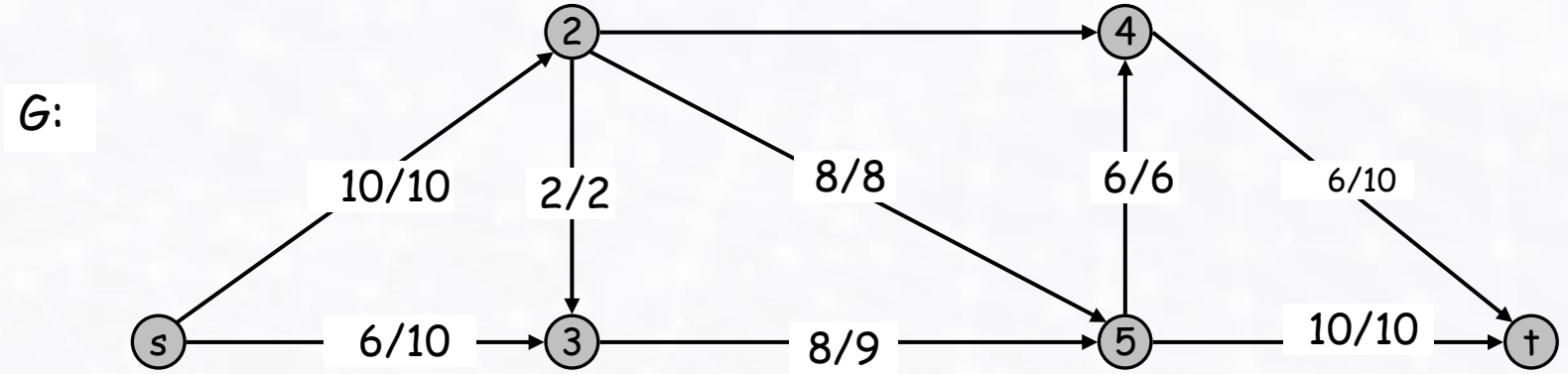
Flow value = 10

Ford-Fulkerson Algorithm



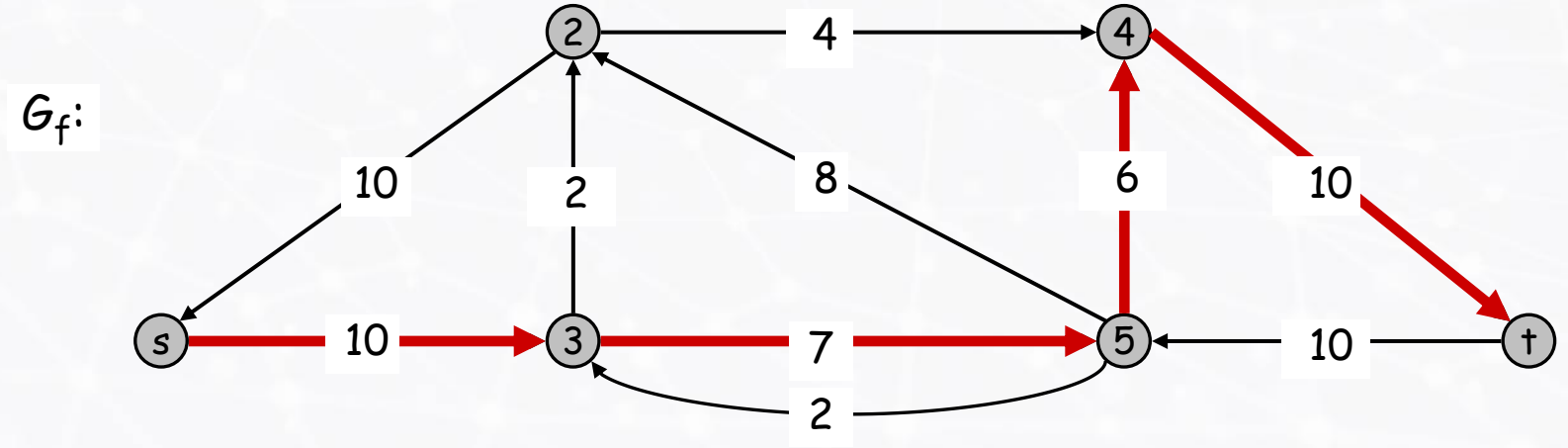


Ford-Fulkerson Algorithm

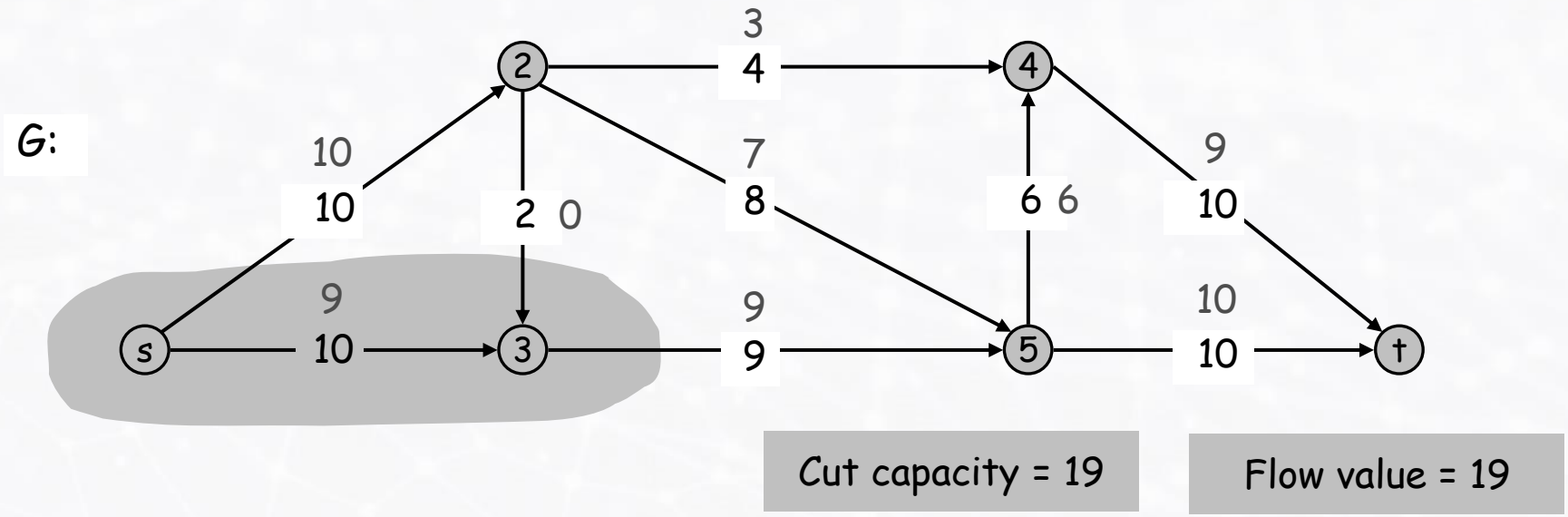


• The residual graph below shows an augmenting path allowing us to add 6 to the overall flow.

Flow value = 16

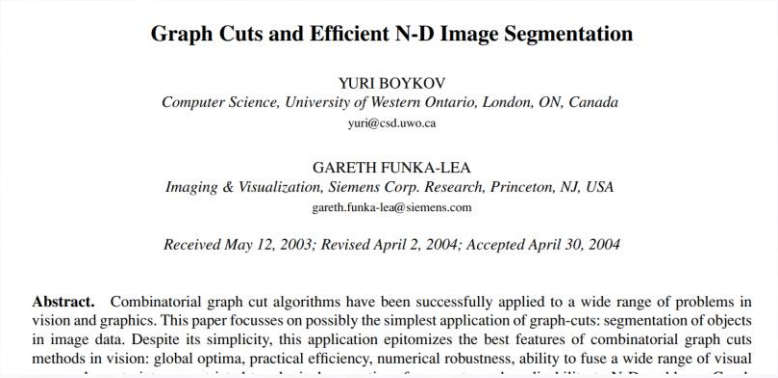


Ford-Fulkerson Algorithm



• Continuing this process of adding augmenting paths, we arrive at a flow= 19. One can show that **this is the maximum flow achievable** by appealing to *max-flow/min-cut duality*. Because there exists a cut with capacity also equal to 19, this proves optimality.

Graph Cuts Image Segmentation



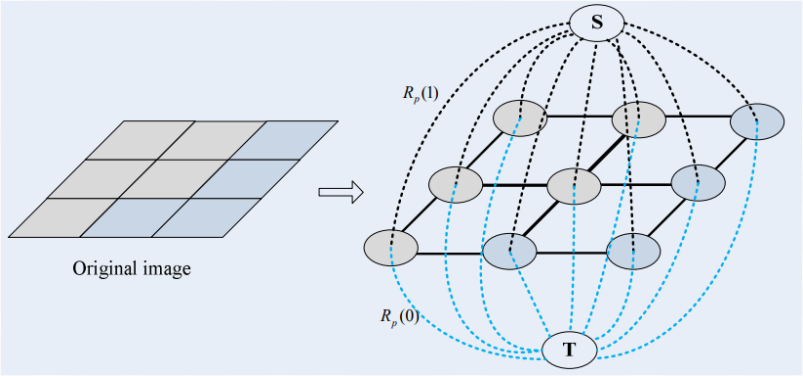
Graph Cuts and Efficient N-D Image Segmentation

YURI BOYKOV
Computer Science, University of Western Ontario, London, ON, Canada
yuri@csd.uwo.ca

GARETH FUNKA-LEA
Imaging & Visualization, Siemens Corp. Research, Princeton, NJ, USA
gareth.funka-lea@siemens.com

Received May 12, 2003; Revised April 2, 2004; Accepted April 30, 2004

Abstract. Combinatorial graph cut algorithms have been successfully applied to a wide range of problems in vision and graphics. This paper focusses on possibly the simplest application of graph-cuts: segmentation of objects in image data. Despite its simplicity, this application epitomizes the best features of combinatorial graph cuts methods in vision: global optima, practical efficiency, numerical robustness, ability to fuse a wide range of visual



Graph Cuts

• The graph cuts segmentation algorithm (2004) leverages the max-flow min-cut theorem and Ford –Fulkerson algorithm for image segmentation. In this way, segmentation is regarded as a pixel labeling problem.

• Boykov et al. define a graph based on an image that includes two types of edges: (1) **n-links** connecting neighboring pixels vertices in a 4-neighborhood system; (2) **t-links** that connect the source and sink vertices with all other pixel vertices.

Graph Cuts Image Segmentation

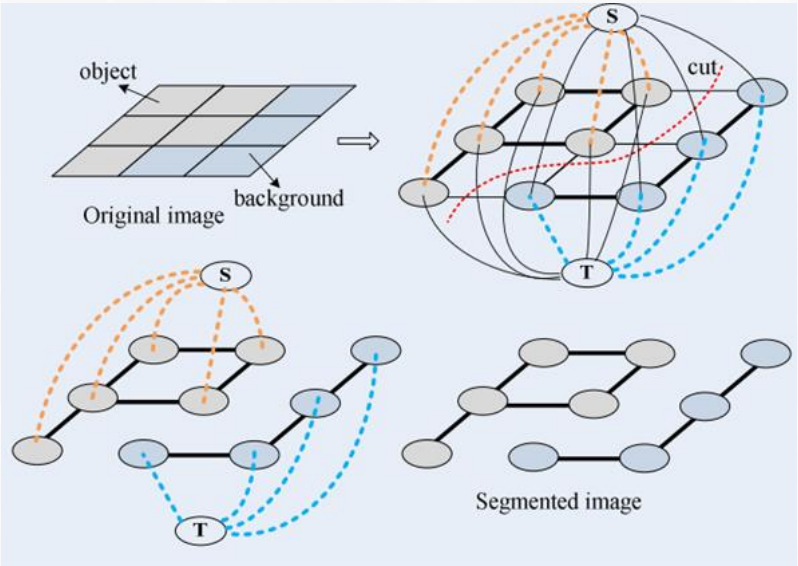
Graph Cuts

- Finding the optimal segmentation/cut is tantamount to finding a **minimum energy solution**:

$$E(L) = \alpha R(L) + B(L)$$

total energy — regional term — boundary term

where L denotes a binary labelling of pixels (i.e. a cut), $R(L)$ represents a **regional term** incorporating t-link connections, $B(L)$ connotes a **boundary term** incorporating s-link connections; α provides a "smoothness" parameter.



Graph Cuts Image Segmentation

Graph Cuts

$$E(L) = \alpha R(L) + B(L)$$

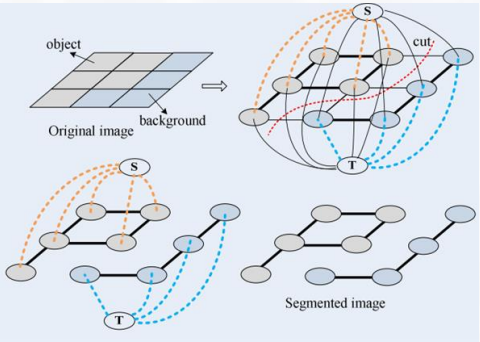
total energy — regional term — boundary term

• **t-links** connect the terminal nodes (s and t nodes) with all other nodes in the graph.

$$R(L) = \sum_{p \in P} R_p\left(l_p\right)$$

$$R_p(1) = -\ln P(I_p \mid \textit{foreground})$$

$$R_p(0) = -\ln P(I_p \mid \textit{background})$$



• $R_p\left(l_p\right)$ is the penalty for assigning the label l_p to pixel p. The weight of $R_p\left(l_p\right)$ can be obtained by comparing the intensity of pixel p with the histogram of the of the "object" and "background" (reflected by the current object/background segmentation).

Graph Cuts Image Segmentation

Graph Cuts

$$E(L) = \alpha R(L) + B(L)$$

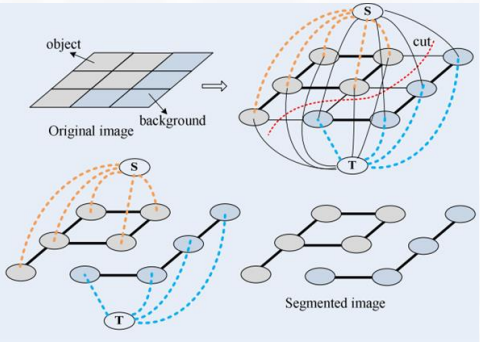
total energy — regional term — boundary term

• **t-links** connect the terminal nodes (s and t nodes) with all other nodes in the graph.

$$R(L) = \sum_{p \in P} R_p\left(l_p\right)$$

$$R_p(1) = -\ln P(I_p \mid foreground)$$

$$R_p(0) = -\ln P(I_p \mid background)$$



• $R_p\left(l_p\right)$ is the penalty for assigning the label l_p to pixel p. The weight of $R_p\left(l_p\right)$ can be obtained by comparing the intensity of pixel p with the histogram of the of the "object" and "background" (reflected by the current object/background segmentation).

• Thus, for instance, if $P(I_p|foreground)$ is larger than $P(I_p|background)$, then $R_p(1)$ will be smaller than $R_p(0)$. This means when the pixel is more likely to be the object, the penalty for identifying that pixel as "object" should be smaller than if we identified it as "background".

• In this way, when all pixels have been correctly separated into two subsets, the regional term would be minimized.

Graph Cuts Image Segmentation

Graph Cuts

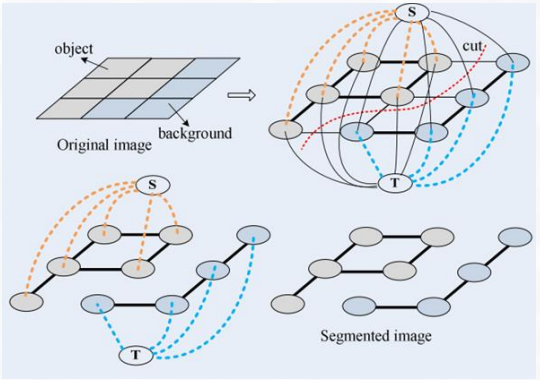
$$E(L) = \alpha R(L) + B(L)$$

total energy — regional term — boundary term

• **n-link** edge weights reflect inter-pixel similarities (i.e. in a cohesive image, neighboring pixels are likely to have similar hue/brightness values). In more detail: the weight of an edge should be large when pixels are similar and small when they are different.

$$B(L) = \sum_{\{p,q\} \in N} B_{pq} \cdot \delta\left(l_p, l_q\right)$$

$$B_{pq} \propto e^{-\frac{\left(I_p - I_q\right)^2}{2\sigma^2}}, \quad \{p,q\} \in \mathrm{N}, \quad \delta\left(l_p, l_q\right) = \begin{cases} 0 \text{ if } l_p = l_q \\ 1 \text{ if } l_p \neq l_q \end{cases}$$



Graph Cuts Image Segmentation

Graph Cuts

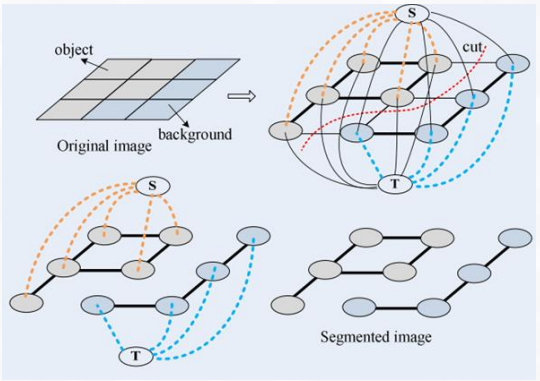
$$E(L) = \alpha R(L) + B(L)$$

total energy — regional term — boundary term

• **n-link** edge weights reflect inter-pixel similarities (i.e. in a cohesive image, neighboring pixels are likely to have similar hue/brightness values). In more detail: the weight of an edge should be large when pixels are similar and small when they are different.

$$B(L) = \sum_{\{p,q\} \in N} B_{pq} \cdot \delta\left(l_p, l_q\right)$$

$$B_{pq} \propto e^{-\frac{\left(I_p - I_q\right)^2}{2\sigma^2}}, \quad \{p,q\} \in \mathrm{N}, \quad \delta\left(l_p, l_q\right) = \begin{cases} 0 \text{ if } l_p = l_q \\ 1 \text{ if } l_p \neq l_q \end{cases}$$



• $\mathrm{B}(L)$ energy is computed for all neighboring pixels in the graph $(\{p,q\} \in N);$), l_i is the label (i.e. "foreground" or "background" for each pixel i in the graph); δ is zero if neighboring labels agree and one otherwise. Finally, when the labels of adjacent pixels disagree, we compute their **inter-pixel similarity based on a Gaussian function**: B_{pq} (where I_i is the intensity of the ith pixel).

*__Basic idea for B(L) energy term__: Terms that contribute to the $\mathrm{B}(L)$ energy are positive when neighboring pixels have different labels; the energy of a particular edge contribution in this case is proportional to their similarity. **Near a boundary, this energy will be minimal**.

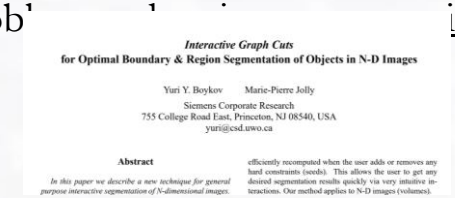
Graph Cuts Image Segmentation

Interactive Graph Cuts

• The preceding problem formulation can be solved (efficiently, in polynomial-time) using the aforementioned *Ford-Fulkerson algorithm* (or related variant). In particular, we aim to minimize energy with respect to the binary graph labelling (foreground vs background). On its own, the **graph cuts** algorithm provides an effective baseline "unsupervised" image segmentation algorithm.

• However, one can easily expand (as Boykov *et al.* have done) the graph cut framework to encompass a broader set of **interactive image segmentation** prob[illegible]ides a set of foreground/background labels (via clicks, lines, etc.).

https://www.csd.uwo.ca/~yboykov/Papers/iccv01.pdf



Interactive Graph Cuts
for Optimal Boundary & Region Segmentation of Objects in N-D Images

Yuri Y. Boykov Marie-Pierre Jolly
Siemens Corporate Research
755 College Road East, Princeton, NJ 08540, USA
yuri@csd.uwo.ca

Abstract

In this paper we describe a new technique for general purpose interactive segmentation of N-dimensional images.

efficiently recomputed when the user adds or removes any hard constraints (seeds). This allows the user to get any desired segmentation results quickly via very intuitive interactions. Our method applies to N-D images (volumes).

How do we adapt the previous graph cuts algorithm to incorporate interactivity?

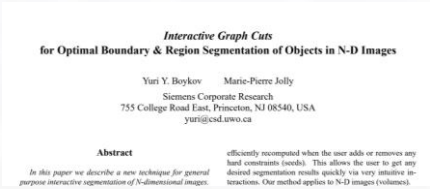
Graph Cuts Image Segmentation

Interactive Graph Cuts

• The preceding problem formulation can be solved (efficiently, in polynomial-time) using the aforementioned *Ford-Fulkerson algorithm* (or related variant). On its own, the **graph cuts** algorithm provides an effective baseline "unsupervised" image segmentation algorithm.

• However, one can easily expand the (as Boykov *et al.* have done) graph cut framework to encompass a broader set of **interactive image segmentation** problems, wherein a user provides a set of foreground/background labels (via clicks, lines, etc.).

https://www.csd.uwo.ca/~yboykov/Papers/iccv01.pdf



Interactive Graph Cuts
for Optimal Boundary & Region Segmentation of Objects in N-D Images

Yuri Y. Boykov Marie-Pierre Jolly
Siemens Corporate Research
755 College Road East, Princeton, NJ 08540, USA
yuri@csd.uwo.ca

Abstract

In this paper we describe a new technique for general purpose interactive segmentation of N-dimensional images. efficiently recomputed when the user adds or removes any hard constraints (seeds). This allows the user to get any desired segmentation results quickly via very intuitive interactions. Our method applies to N-D images (volumes).

How do we adapt the previous graph cuts algorithm to incorporate interactivity? Simply:

(1) The calculation of the regional term can now be based on the histogram of foreground and background labelled pixels.

$$R(L) = \sum_{p \in P} R_p(l_p)$$

$$R_p(1) = -\ln P(I_p \mid foreground)$$

$$R_p(0) = -\ln P(I_p \mid background)$$

(2) The overall energy function can be amended to include "hard constraints" reflecting the user provided labels (e.g., for every mislabel produced by the algorithm, the energy function incurs a penalty of K):

$$E(L) = \alpha R(L) + B(L) + K \cdot N(misclassified)$$

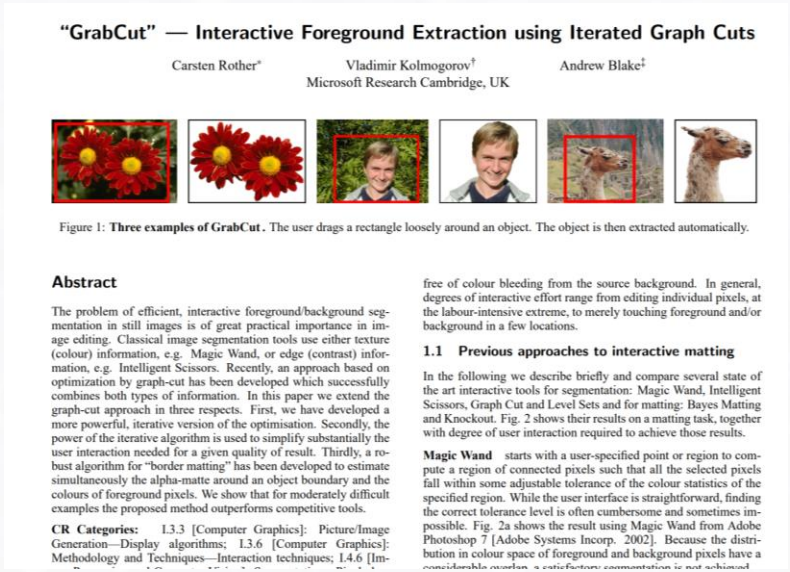
total energy — regional term — boundary term

where N(misclassified) symbolizes the number of misclassified pixels in the final segmentation with respect to the user labels.

Graph Cuts Image Segmentation

GrabCut (2004)

https://docs.opencv.org/3.4/d8/d83/tutorial_py_grabcut.html



"GrabCut" — Interactive Foreground Extraction using Iterated Graph Cuts

Carsten Rother* Vladimir Kolmogorov† Andrew Blake‡
Microsoft Research Cambridge, UK

Figure 1: **Three examples of GrabCut.** The user drags a rectangle loosely around an object. The object is then extracted automatically.

Abstract

The problem of efficient, interactive foreground/background segmentation in still images is of great practical importance in image editing. Classical image segmentation tools use either texture (colour) information, e.g. Magic Wand, or edge (contrast) information, e.g. Intelligent Scissors. Recently, an approach based on optimization by graph-cut has been developed which successfully combines both types of information. In this paper we extend the graph-cut approach in three respects. First, we have developed a more powerful, iterative version of the optimisation. Secondly, the power of the iterative algorithm is used to simplify substantially the user interaction needed for a given quality of result. Thirdly, a robust algorithm for "border matting" has been developed to estimate simultaneously the alpha-matte around an object boundary and the colours of foreground pixels. We show that for moderately difficult examples the proposed method outperforms competitive tools.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Display algorithms; I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques; I.4.6 [Im-

free of colour bleeding from the source background. In general, degrees of interactive effort range from editing individual pixels, at the labour-intensive extreme, to merely touching foreground and/or background in a few locations.

1.1 Previous approaches to interactive matting

In the following we describe briefly and compare several state of the art interactive tools for segmentation: Magic Wand, Intelligent Scissors, Graph Cut and Level Sets and for matting: Bayes Matting and Knockout. Fig. 2 shows their results on a matting task, together with degree of user interaction required to achieve those results.

Magic Wand starts with a user-specified point or region to compute a region of connected pixels such that all the selected pixels fall within some adjustable tolerance of the colour statistics of the specified region. While the user interface is straightforward, finding the correct tolerance level is often cumbersome and sometimes impossible. Fig. 2a shows the result using Magic Wand from Adobe Photoshop 7 [Adobe Systems Incorp. 2002]. Because the distribution in colour space of foreground and background pixels have a

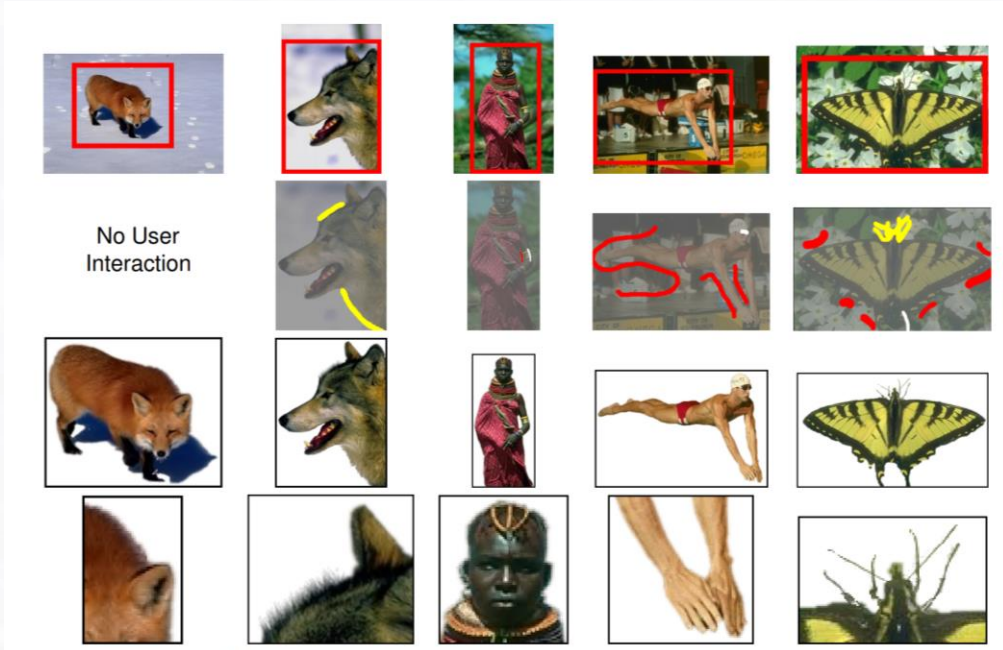


Figure 8: **Results using GrabCut.** The first row shows the original images with superimposed user input (red rectangle). The second row displays all user interactions: red (background brush), white (foreground brush) and yellow (matting brush). The degree of user interaction increases from left to right. The results obtained by GrabCut are visualized in the third row. The last row shows zoomed portions of the respective result which documents that the recovered alpha mattes are smooth and free of background bleeding.

Interactive Graph Cuts

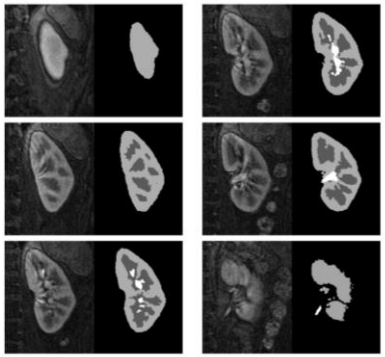


Figure 6. Kidney in a 3D MRI angio data.

Review Topic: OLS Regression

OLS Regression

- We consider an equivalent – but more elegant – approach to OLS by appealing to linear algebra/geometric intuition.

Consider the problem of solving the previous system of linear equations in the **"overdetermined" case** (i.e. $m > n$, where m is the number of equations/measurements, n is the number of variables).

OLS Regression

- We consider an equivalent – but more elegant – approach to OLS by appealing to linear algebra/geometric intuition.

Consider the problem of solving the previous system of linear equations in the **"overdetermined" case** (i.e. $m > n$, where m is the number of equations/measurements, n is the number of variables): Ax=b.

x	y
0	1
1	5
2	10
3	22
4	38

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \\ 10 \\ 22 \\ 38 \end{bmatrix}$$

OLS Regression

- An overdetermined system:

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \\ 10 \\ 22 \\ 38 \end{bmatrix} \quad \Rightarrow \quad \beta_0 \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \beta_1 \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \\ 10 \\ 22 \\ 38 \end{bmatrix}$$

$$A\mathbf{x} = \mathbf{b}$$

Q: Are we always guaranteed that such a system has a solution (say using Gaussian elimination)?

OLS Regression

- An overdetermined system:

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \\ 10 \\ 22 \\ 38 \end{bmatrix} \quad \Rightarrow \quad \beta_0 \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \beta_1 \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \\ 10 \\ 22 \\ 38 \end{bmatrix}$$

$$A\mathbf{x} = \mathbf{b}$$

Q: Are we always guaranteed that such a system has a solution (say using Gaussian elimination)?

Definitely not! *Short answer: because we <u>cannot guarantee</u> that the vector **b** resides in the **column space of A** (*col*(A))!

Next, let's consider this situation from a geometric perspective.

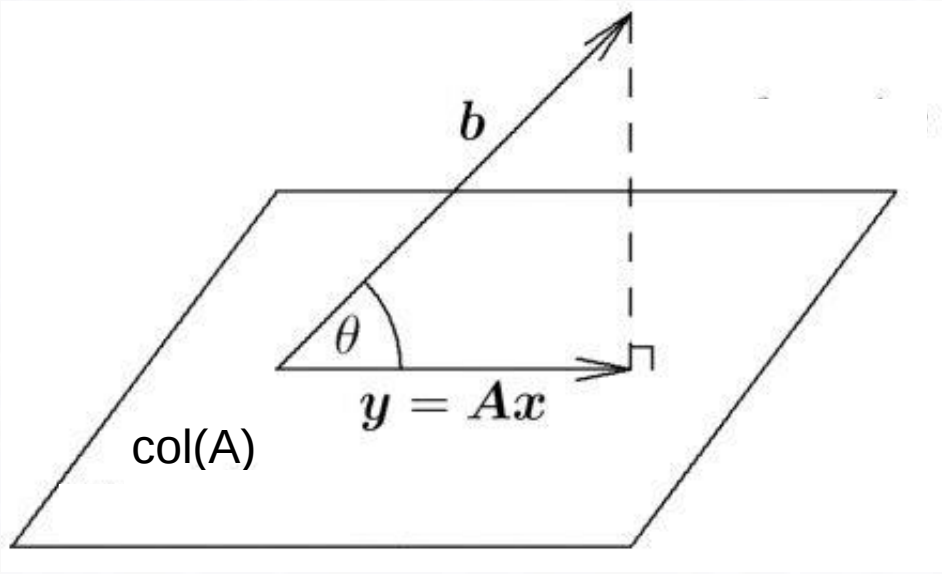
(*) Recall that the **col(A):= the span of the column vectors of A.**

OLS Regression

- An overdetermined system:

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \\ 10 \\ 22 \\ 38 \end{bmatrix} \quad \Rightarrow \quad \beta_0 \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \beta_1 \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \\ 10 \\ 22 \\ 38 \end{bmatrix}$$

$$A\mathbf{x} = \mathbf{b}$$

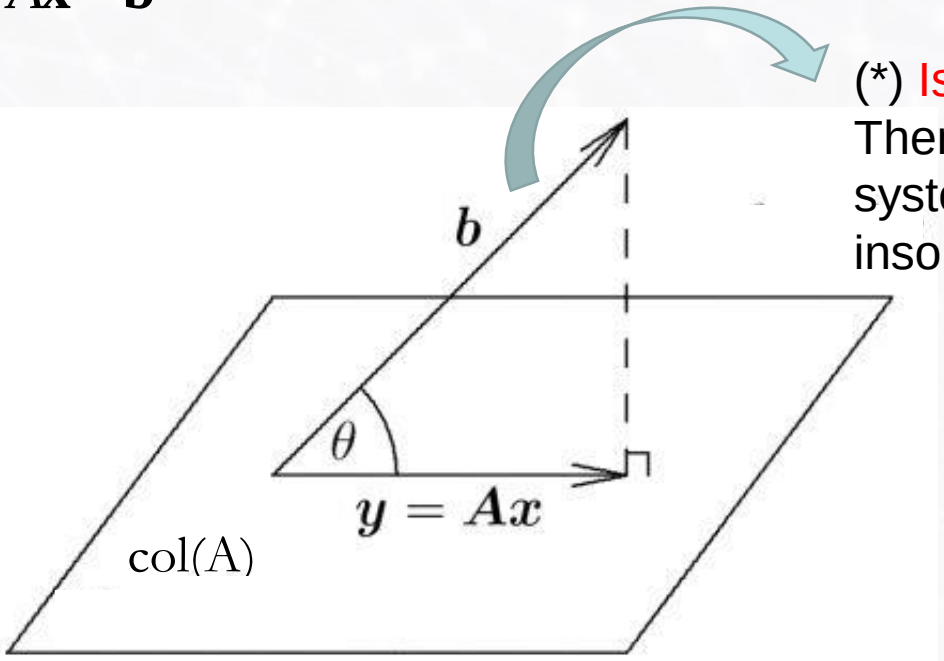


OLS Regression

- An overdetermined system:

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \\ 10 \\ 22 \\ 38 \end{bmatrix} \quad \Rightarrow \quad \beta_0 \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \beta_1 \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \\ 10 \\ 22 \\ 38 \end{bmatrix}$$

$$A\mathbf{x} = \mathbf{b}$$



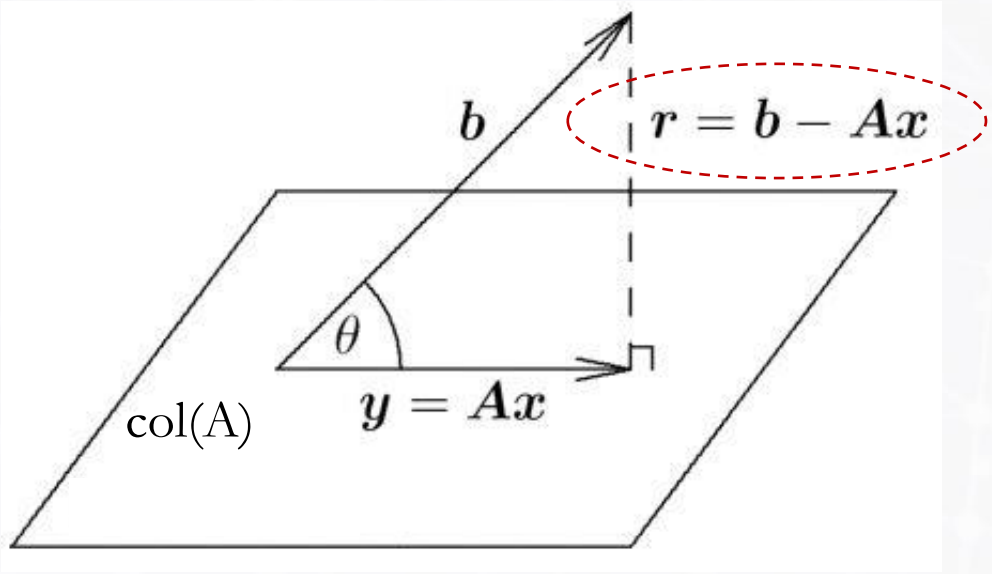
(*) Issue: if $\mathbf{b} \notin col(A)$ Then overdetermined system: Ax=b is insoluble.

OLS Regression

- An overdetermined system:

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \\ 10 \\ 22 \\ 38 \end{bmatrix} \quad \Rightarrow \quad \beta_0 \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \beta_1 \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \\ 10 \\ 22 \\ 38 \end{bmatrix}$$

$$A\mathbf{x} = \mathbf{b}$$



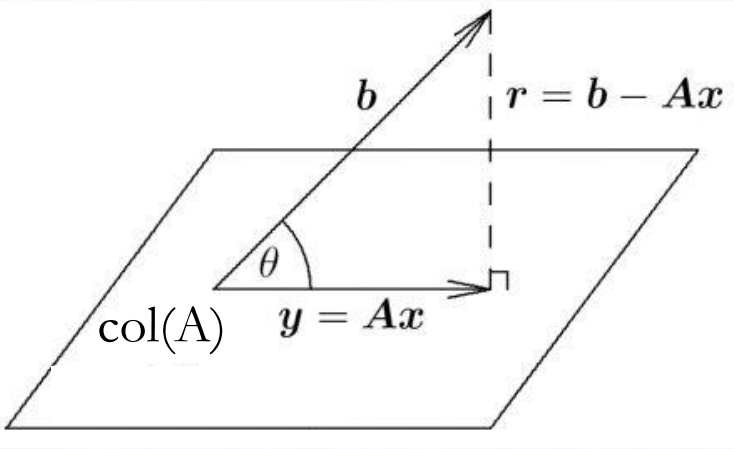
(*) A resolution: The best we can hope to do is to minimize the distance r (i.e. the *residual*) between **b** and any vector in col(A).

Namely, we want:

$$\underset{\mathbf{x}}{\operatorname{argmin}} \|\mathbf{r}\|^2 = \underset{\mathbf{x}}{\operatorname{argmin}} \|\mathbf{b} - A\mathbf{x}\|^2$$

(this formula should look familiar)

OLS Regression



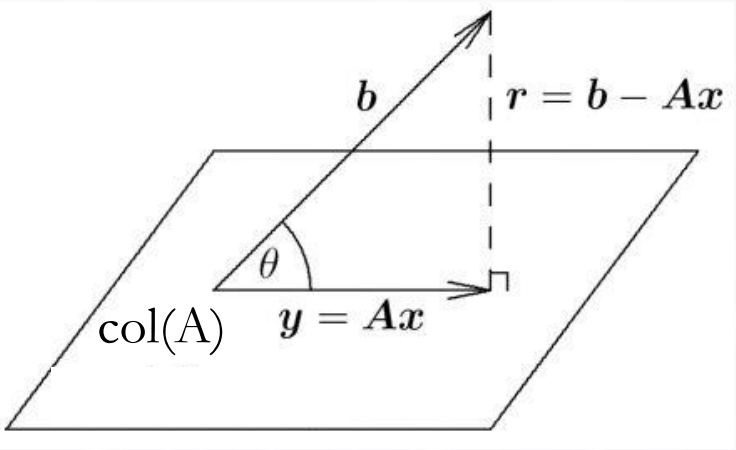
(*)We want:

$$\underset{\mathbf{x}}{\operatorname{argmin}} \left\| \mathbf{b} - A\mathbf{x} \right\|^2$$

(Let's denote the solution vector x*)

An astute observation: The residual vector r achieves a minimum when it is orthogonal to col(A)!

OLS Regression



(*)We want:

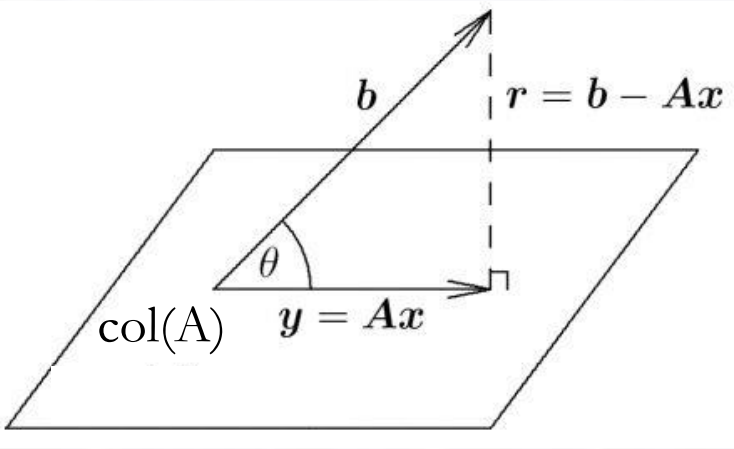
$$\underset{\mathbf{x}}{\operatorname{argmin}} \left\| \mathbf{b} - A\mathbf{x} \right\|^2$$

(Let's denote the solution vector x*)

An astute observation: The residual vector r achieves a minimum when it is orthogonal to col(A)!

This implies: $\left(A\mathbf{x}\right)^T \left(\mathbf{b} - A\mathbf{x}^*\right) = 0 \qquad (\textit{for all } \mathbf{x})$

OLS Regression



(*)We want:

$$\underset{\mathbf{x}}{\operatorname{argmin}} \left\| \mathbf{b} - A\mathbf{x} \right\|^2$$

(Let's denote the solution vector x*)

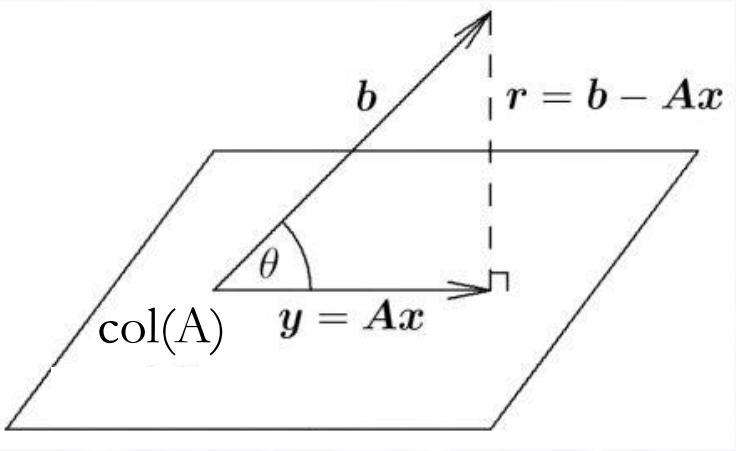
An astute observation: The residual vector r achieves a minimum when it is orthogonal to col(A)!

This implies:

$$\left(A\mathbf{x}\right)^T \left(\mathbf{b} - A\mathbf{x}^*\right) = 0 \qquad (\textit{for all } \mathbf{x})$$

$$\mathbf{x}^T A^T \left(\mathbf{b} - A\mathbf{x}^*\right) = 0 \qquad (\textit{for all } \mathbf{x})$$

OLS Regression



(*)We want:

$$\underset{\mathbf{x}}{\operatorname{argmin}} \left\| \mathbf{b} - A\mathbf{x} \right\|^2$$

(Let's denote the solution vector x*)

An astute observation: The residual vector r achieves a minimum when it is orthogonal to col(A)!

This implies:

$$\left(A\mathbf{x}\right)^T \left(\mathbf{b} - A\mathbf{x}^*\right) = 0 \qquad (\textit{for all } \mathbf{x})$$

$$\mathbf{x}^T A^T \left(\mathbf{b} - A\mathbf{x}^*\right) = 0 \qquad (\textit{for all } \mathbf{x})$$

This indicates that the vector: $A^T(\mathbf{b}\text{-}A\mathbf{x}^*)$ is perpendicular to every vector **x** in the space. What can we claim about this vector?

OLS Regression

$$(A\mathbf{x})^T(\mathbf{b}-A\mathbf{x}^*)=0 \qquad (\textit{for all } \mathbf{x})$$

$$\mathbf{x}^T A^T(\mathbf{b}-A\mathbf{x}^*)=0 \qquad (\textit{for all } \mathbf{x})$$

This indicates that the vector: $A^T(\mathbf{b}\text{-}A\mathbf{x}^*)$ is perpendicular to <u>every vector **x**</u> in the space. What can we claim about this vector?

(*) Consequently: $A^T(\mathbf{b}-A\mathbf{x}^*)=0$

OLS Regression

Now we solve for **x***.

$$A^T\left(\mathbf{b} - A\mathbf{x}^*\right) = 0$$

OLS Regression

Now we solve for **x***.

$$A^T\left(\mathbf{b}-A\mathbf{x}^*\right)=0$$

$$A^T\mathbf{b}-A^TA\mathbf{x}^*=0$$

OLS Regression

Now we solve for **x***.

$$A^T\left(\mathbf{b} - A\mathbf{x}^*\right) = 0$$

$$A^T\mathbf{b} - A^T A\mathbf{x}^* = 0$$

$$A^T\mathbf{b} = A^T A\mathbf{x}^*$$

OLS Regression

Now we solve for **x***.

$$A^T\left(\mathbf{b} - A\mathbf{x}^*\right) = 0$$

$$A^T\mathbf{b} - A^T A\mathbf{x}^* = 0$$

$$A^T\mathbf{b} = A^T A\mathbf{x}^*$$

(These are the *normal equations* in matrix form!)

OLS Regression

Now we solve for **x***.

$$A^T\left(\mathbf{b} - A\mathbf{x}^*\right) = 0$$

$$A^T\mathbf{b} - A^T A\mathbf{x}^* = 0$$

$$A^T\mathbf{b} = A^T A\mathbf{x}^*$$

$$\mathbf{x}^* = \left(A^T A\right)^{-1} A^T\mathbf{b}$$

(*) This implies that OLS has a <u>unique, closed form solution when A^TA is **non-singular**</u> (i.e. *invertible*).

(*) When A^TA is singular, it is common practice to use the *Moore-Penrose pseudoinverse*: A^+.

OLS Regression

Now we solve for **x***.

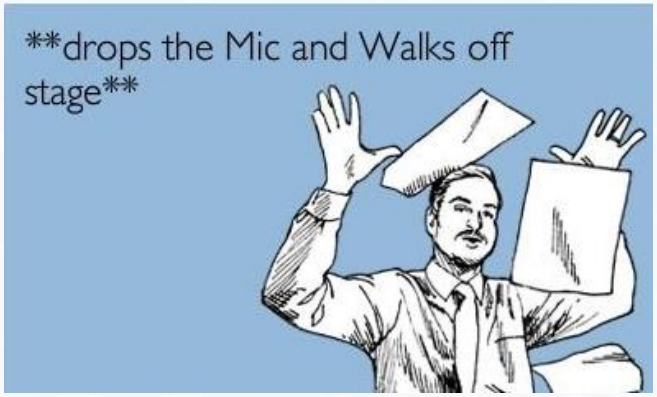
$$A^T\left(\mathbf{b} - A\mathbf{x}^*\right) = 0$$

$$A^T\mathbf{b} - A^T A\mathbf{x}^* = 0$$

$$A^T\mathbf{b} = A^T A\mathbf{x}^*$$

$$\boxed{\mathbf{x}^* = \left(A^T A\right)^{-1} A^T\mathbf{b}}$$

(*) This completes our derivation of the OLS solutions using linear algebra!



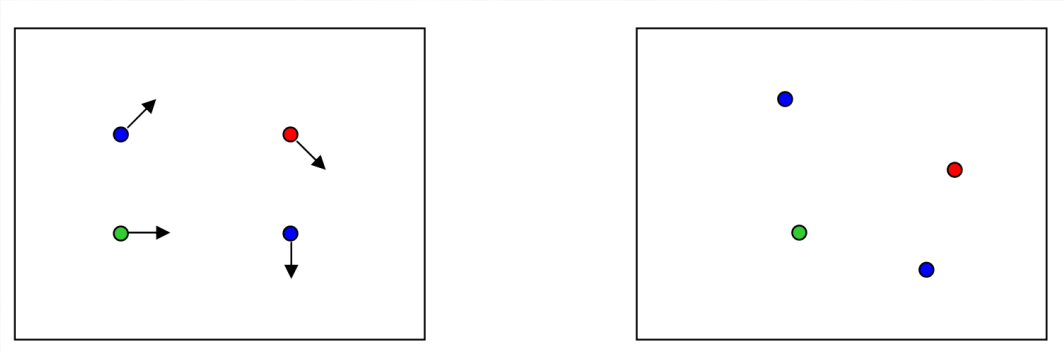
Motion Estimation: Optical Flow

• Motion is an intrinsic property of the world, and an essential aspect of our visual experience. Motion estimation can be used successfully in a wide variety of CV-related applications, including: object tracking, camera stabilization, scene understanding, and 3D scene reconstruction.

Motion Estimation: Optical Flow

• The goal of **optical flow estimation*** (OF) is to compute an approximation to the motion field from time-varying image intensities.

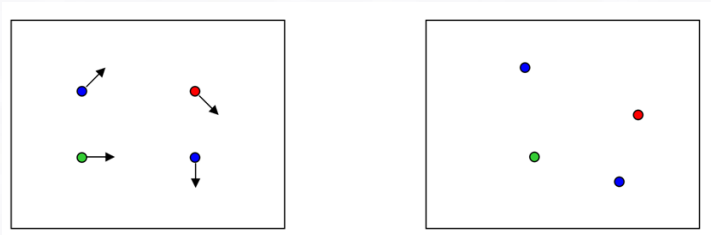
• Next, we present the classic optical flow estimation algorithm* (Beauchemin et al., 1995).



$I(x, y, t)$ $I(x, y, t + \Delta t)$

*https://dl.acm.org/doi/abs/10.1145/212094.212141

Motion Estimation: Optical Flow



$I(x, y, t)$ $I(x, y, t + \Delta t)$

• A common starting point for OF is to **assume** that pixel intensities are translated (without alteration) from one frame to the next, so that:

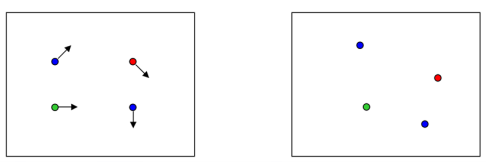
$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t)$$

holds, where $I(x, y, t)$ is the image intensity at time t, $\langle u, v \rangle$ is a displacement vector. Naturally, this brightness constancy assumption rarely holds exactly, but is nevertheless plausible under stable conditions.

With this assumption, we wish to estimate (dense) optical flow at each pixel (x, y): $\frac{\Delta x}{\Delta t}$ and $\frac{\Delta y}{\Delta t}$.

*The version of OF given here is for grayscale images, but the method is easily adapted for RGB images.

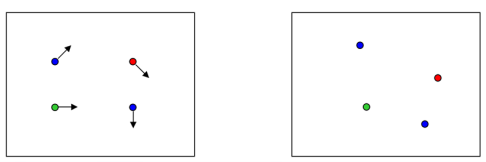
Motion Estimation: Optical Flow



$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t)$$

•Next, we use the multi-variate Taylor series approximation to calculate the linearization of $I(\Delta x, \Delta y, t)$:

Motion Estimation: Optical Flow



$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t)$$

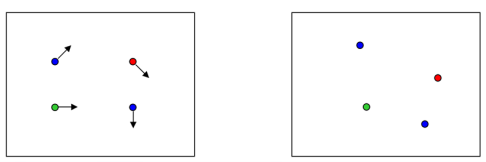
•Next, we use the multi-variate Taylor series approximation to calculate the linearization of $I(\Delta x, \Delta y, t)$:

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y \frac{\partial I}{\partial t}\Delta t + \text{higher order terms...}$$

This yields:

$$0 = I(x + \Delta x, y + \Delta y, t + \Delta t) - I(x, y, t) \approx \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y \frac{\partial I}{\partial t}\Delta t$$

Motion Estimation: Optical Flow



$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t)$$

•Next, we use the multi-variate Taylor series approximation to calculate the linearization of $I(\Delta x, \Delta y, t)$:

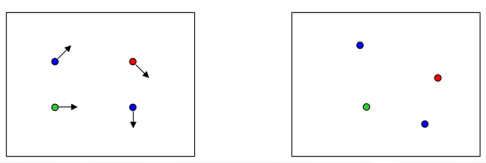
$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y \frac{\partial I}{\partial t}\Delta t + \text{higher order terms...}$$

This yields:

$$0 = I(x + \Delta x, y + \Delta y, t + \Delta t) - I(x, y, t) \approx \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y \frac{\partial I}{\partial t}\Delta t$$

Dividing through by Δt, we have: $\frac{\partial I}{\partial x}\underbrace{\frac{\Delta x}{\Delta t}}_{V_x} + \frac{\partial I}{\partial y}\underbrace{\frac{\Delta y}{\Delta t}}_{V_y} + \frac{\partial I}{\partial t} = 0$

Motion Estimation: Optical Flow



$$I(x,y,t) = I(x+\Delta x, y+\Delta y, t+\Delta t)$$

•Next, we use the multi-variate Taylor series approximation to calculate the linearization of $I(\Delta x, \Delta y, t)$:

$$I(x+\Delta x, y+\Delta y, t+\Delta t) = I(x,y,t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y \frac{\partial I}{\partial t}\Delta t + \text{higher order terms...}$$

This yields:

$$0 = I(x+\Delta x, y+\Delta y, t+\Delta t) - I(x,y,t) \approx \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y \frac{\partial I}{\partial t}\Delta t$$

Dividing through by Δt, we have: $\frac{\partial I}{\partial x}\underbrace{\frac{\Delta x}{\Delta t}}_{V_x} + \frac{\partial I}{\partial y}\underbrace{\frac{\Delta y}{\Delta t}}_{V_y} + \frac{\partial I}{\partial t} = 0$

This equation can be notated equivalently:

$$I_x V_x + I_y V_y = -I_t$$

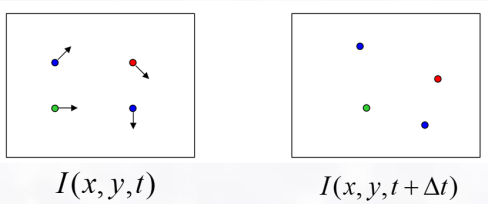
$$\nabla I \cdot V = -I_t$$

where **we wish to solve for OF** $\langle V_x, V_y \rangle$;
I_t denotes the *image derivative* which can be approximated using the **Sobel transformation**.

$$\begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} V_x \\ V_y \end{bmatrix} = -I_t$$

Motion Estimation: Optical Flow

$$\begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} V_x \\ V_y \end{bmatrix} = -I_t$$



• We wish to solve the equation above for OF $\langle V_x, V_y \rangle$; however, this requires solving for two unknowns (with only one equation), an **underdetermined system**.

• In the **Lucas-Kanade method** (1981) for approximating optical flow, we consider 3x3 patches of pixels around the current pixel. This gives rise to a system of 9 equations and 2 unknowns, an **overdetermined system**:

$$\underbrace{\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ \vdots & \vdots \\ I_x(p_9) & I_y(p_9) \end{bmatrix}}_{9\times 2} \underbrace{\begin{bmatrix} V_x \\ V_y \end{bmatrix}}_{2\times 1} = -\underbrace{\begin{bmatrix} I_t(p_1) \\ \vdots \\ I_t(p_9) \end{bmatrix}}_{9\times 1}, \quad I_t \approx \text{Sobel Transform (or other derivative estimate)}$$

• We can approximate the solution to this system using the standard **least-squares solution**.

Motion Estimation: Optical Flow

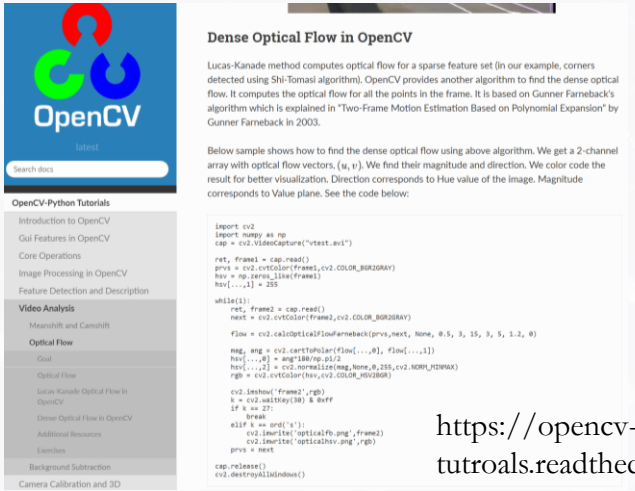
• We can approximate the solution to this system using the standard **least-squares solution**.

$$\underbrace{\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ \vdots & \vdots \\ I_x(p_9) & I_y(p_9) \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} V_x \\ V_y \end{bmatrix}}_{x} = -\underbrace{\begin{bmatrix} I_t(p_1) \\ \vdots \\ I_t(p_9) \end{bmatrix}}_{b}$$

$$\rightarrow \underbrace{\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}}_{A^T A} \underbrace{\begin{bmatrix} V_x \\ V_y \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} -\sum I_x I_t \\ -\sum I_y I_t \end{bmatrix}}_{A^T b}$$

$$\rightarrow \begin{bmatrix} V_x \\ V_y \end{bmatrix}_x = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}^{-1} \begin{bmatrix} -\sum I_x I_t \\ -\sum I_y I_t \end{bmatrix}$$

Motion Estimation: Optical Flow



https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_video/py_lucas_kanade/py_lucas_kanade.html

Review Topic: PCA/SVD

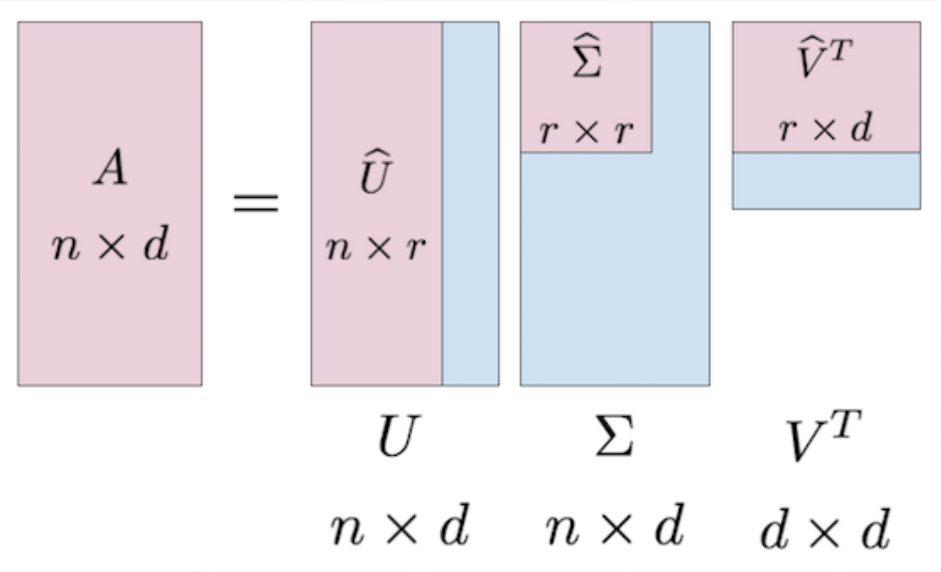
SVD

- Definition: Let A be an *m x n* matrix with singular values, $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$ and $\sigma_{r+1} = \sigma_{r+2} = \ldots = \sigma_n = 0$. Then there exist an *m x n* orthogonal matrix **U**, and *n x n* orthogonal matrix **V**, and an *m x n diagonal* matrix **Σ** of the form:

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

Note: the columns of **U** are called *left singular vectors* of **A**, and the columns of **V** are called *right singular vectors* of **A**. The matrices **U** and **V** are <u>not</u> uniquely determined by **A**.

(*) *NB*: **rank(A) = r**.



SVD

- Definition: Let A be an *m x n* matrix with singular values, $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$ and $\sigma_{r+1} = \sigma_{r+2} = \ldots = \sigma_n = 0$. Then there exist an *m x n* orthogonal matrix **U**, and *n x n* orthogonal matrix **V**, and an *m x n diagonal* matrix **Σ** of the form:

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

- *Every* matrix has a singular value decomposition!

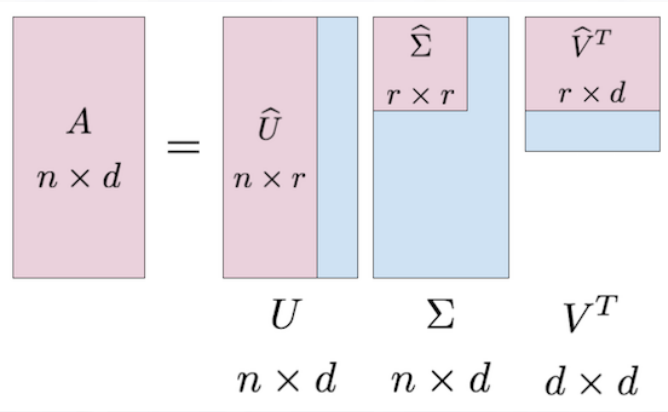
Definition: For an *m x n* matrix **A**, the *singular values* of **A** are the square roots of the eigenvalues of $\mathbf{A}^T\mathbf{A}$. They are denoted:

$$\sigma_1, \ldots, \sigma_n$$

It is conventional to arrange the singular values in decreasing order, whence: $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n$

SVD

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



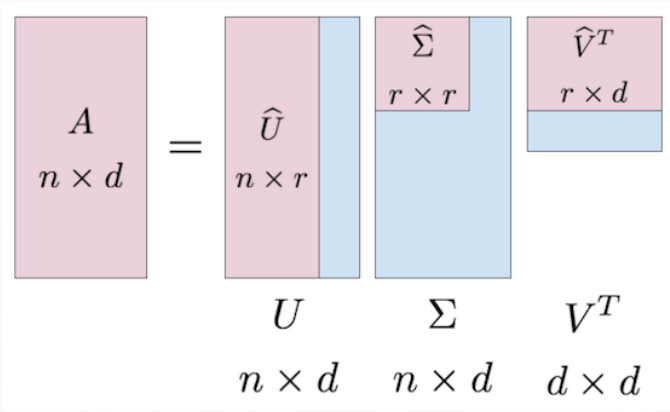
Example:

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

$$A^T A = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

SVD

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



Example:

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

$$A^T A = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$eigenvalues\left(A^T A\right): \lambda_1 = 2, \lambda_2 = 1, \lambda_3 = 0$$

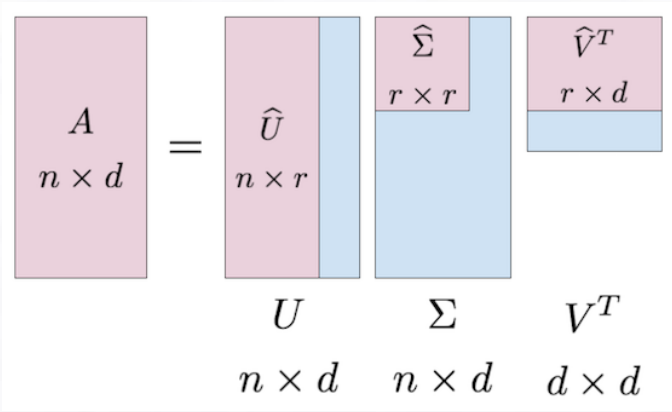
has eigenvalues λ_1=3 and λ_2=1. Consequently, the singular values of **A** are:

$$\sigma_1 = \sqrt{\lambda_1} = \sqrt{3}$$

$$\sigma_2 = \sqrt{\lambda_2} = 1$$

SVD

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



Example:

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

$$A^T A = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

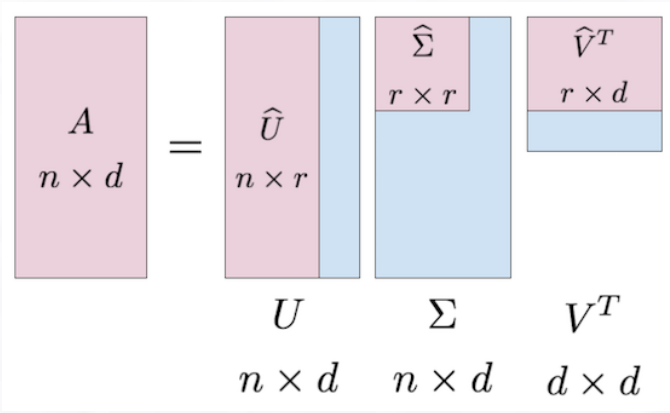
$$eigenvalues\left(A^T A\right): \lambda_1 = 2, \lambda_2 = 1, \lambda_3 = 0 \qquad eigenvectors\left(A^T A\right): \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}$$

These vectors are orthogonal, so now we normalize them:

$$V = \begin{bmatrix} 1/\sqrt{2} & 0 & -1/\sqrt{2} \\ 1/\sqrt{2} & 0 & 1/\sqrt{2} \\ 0 & 1 & 0 \end{bmatrix}, \Sigma = \begin{bmatrix} \sqrt{2} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

SVD

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



Example: $A = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

$eigenvalues(A^T A): \lambda_1 = 2, \lambda_2 = 1, \lambda_3 = 0$

$eigenvectors(A^T A): \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}$

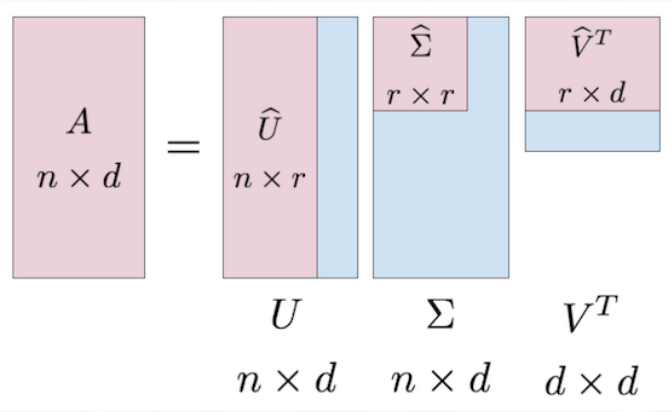
$$V = \begin{bmatrix} 1/\sqrt{2} & 0 & -1/\sqrt{2} \\ 1/\sqrt{2} & 0 & 1/\sqrt{2} \\ 0 & 1 & 0 \end{bmatrix}, \Sigma = \begin{bmatrix} \sqrt{2} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

To find **U** we compute:

$$u_1 = \frac{1}{\sigma_1} A v_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad u_2 = \frac{1}{\sigma_2} A v_2 = \frac{1}{1} \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

SVD

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



Example:

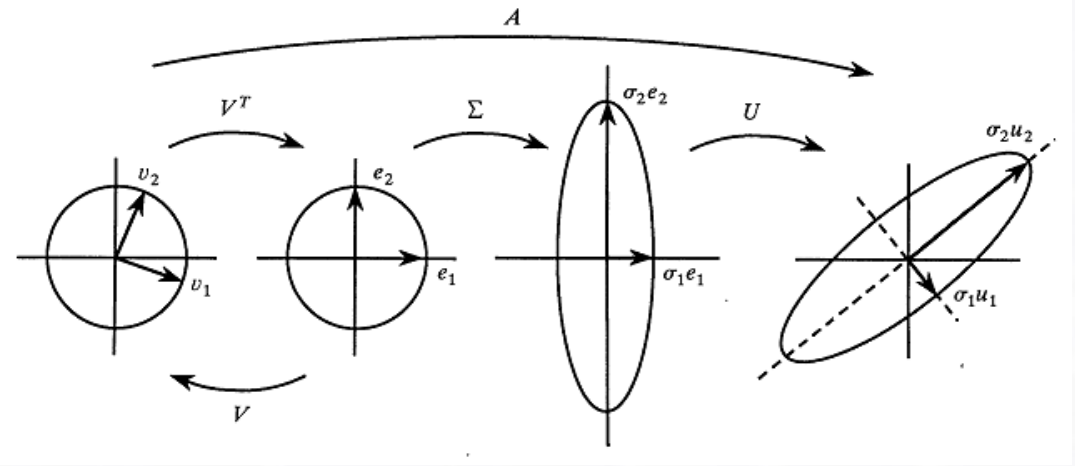
$$A = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \sqrt{2} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \end{bmatrix} = U\Sigma V^T$$

SVD

Geometric Interpretation: In general, **Σ** can be regarded as a scaling matrix, and **U**, **V** can be viewed as rotation matrices.

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

Thus the expression **UΣV** can be intuitively interpreted as a **composition of three successive geometrical transformations**: a rotation or reflection, a scaling and another rotation or reflection.



As shown in the figure, the singular values can be interpreted as the **semiaxes of an ellipse in 2D**. This concept can be generalized to *n*-dimensional *Euclidean space*, with the singular values of any $n \times n$ square matrix being viewed as the semiaxes of an *n*-dimensional ellipsoid.

As in PCA, these coordinate axes provide a natural framework for determining a dimensionality reduction scheme that captures maximal variation.

SVD: Outer Product Form

- SVD factorization yields a useful method for "low rank" approximations/dimensionality reduction of data.

Theorem: For a given SVD decomposition of an *m x n* matrix **A**, we can express **A** in the so-called **outer product form**:

$$\mathbf{A} = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \ldots + \sigma_r \mathbf{u}_r \mathbf{v}_r^T$$

Where $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$ denote the singular values of **A**; **u** and **v** are the corresponding *left singular* and *right singular vectors*.

(*) Note that the **condition number** of a matrix **A** is defined as the ratio of the largest and the smallest singular values of **A**. Matrices with large condition numbers are called **ill-conditioned** (this has a significant impact on the stability of many different kinds of numerical algorithms in linear algebra).

$$cond(A) = \frac{\sigma_{\max}}{\sigma_{\min}}$$

SVD: Outer Product Form

$$\mathbf{A} = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \ldots + \sigma_r \mathbf{u}_r \mathbf{v}_r^T$$

Example:

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \sqrt{2} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \end{bmatrix} = U \Sigma V^T$$

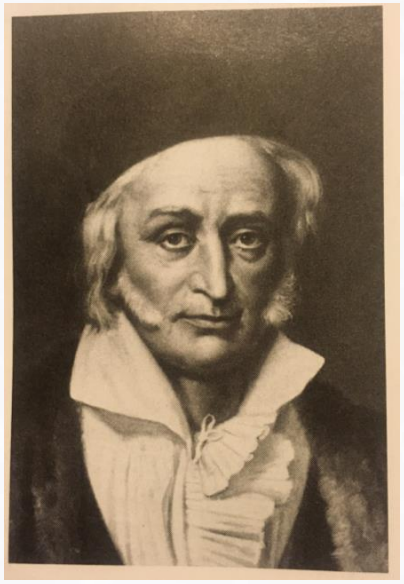
$$A = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \sqrt{2} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \end{bmatrix} + 1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$$

SVD: Outer Product Form for Image Compression

• Consider the task of compressing a grayscale image of dimension *340 x 280*; each pixel is in the range [0, 255].

• We can store this image in a *340 x 280* dimension matrix, but transmitting and manipulating these 95,200 numbers is very expensive.

• Let's use SVD for efficient image compression. Recall that the small singular values in the SVD of a matrix correspond with "less informative" data features.



SVD: Outer Product Form for Image Compression

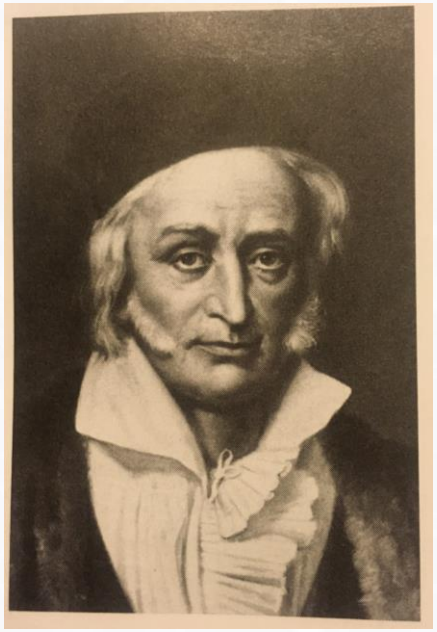
- Suppose we have the SVD of A expressed in outer product form:

$$\mathbf{A} = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \ldots + \sigma_r \mathbf{u}_r \mathbf{v}_r^T$$

- For the original *340 x 280* image shown, we have r = 280 (why?).

- Define: $\mathbf{A}_k = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \ldots + \sigma_k \mathbf{u}_k \mathbf{v}_k^T, \; k \le r$

  as the *k-rank approximation to* ***A***.



SVD: Outer Product Form for Image Compression

- Suppose we have the SVD of A expressed in outer product form:

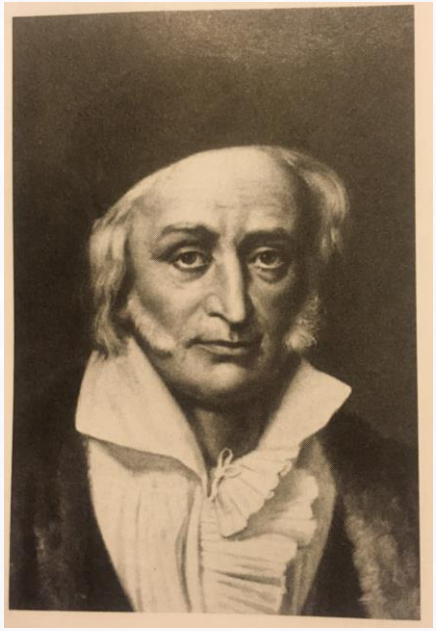
$$\mathbf{A} = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \ldots + \sigma_r \mathbf{u}_r \mathbf{v}_r^T$$

- For the original *340 x 280* image shown, we have r = 280 (why?).

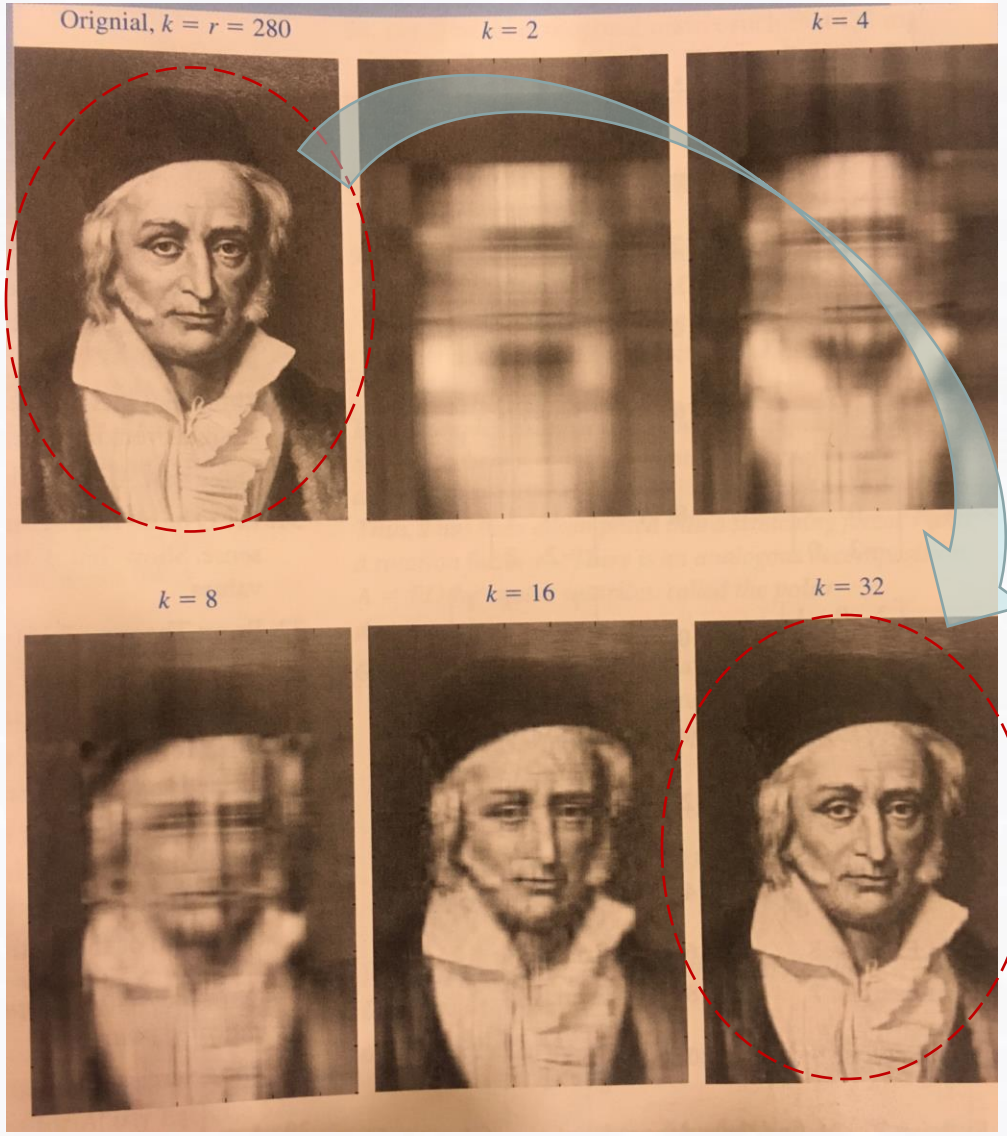
- Define: $\mathbf{A}_k = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \ldots + \sigma_k \mathbf{u}_k \mathbf{v}_k^T, \; k \le r$

  as the *k-rank approximation to* ***A***.

(*) If for example, we use a **k = 20** *rank approximation* for **A** (i.e. we use the largest 20 singular values), the storage/ computational overhead is reduced from 95,200 numbers to 12,420!



SVD: Outer Product Form for Image Compression



$$\mathbf{A}_k = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \ldots + \sigma_k \mathbf{u}_k \mathbf{v}_k^T, \quad k = 32$$

(*) Here, using the SVD-based, low-rank approximation to **A**, the fidelity of the image is very strong – even after discarding roughly 85% of the image data!

PCA

- Here is the **PCA algorithm**:

(1) Write N data points $x_i=(x_{1i},x_{2i},\ldots,x_{Mi})$ as row vectors.

(2) Put these vectors into the data matrix **X** (of size N x M).

(3) **Center the data** by subtracting off the mean of each column, place into matrix **B**.

(4) Computer the covariance matrix: $\mathbf{C} = \frac{1}{N}\mathbf{B}\mathbf{B}^T$

(5) Computer the *eigenvalues* and *eigenvectors* of **C**, so: $\mathbf{C} = \mathbf{V}\mathbf{D}\mathbf{V}^T$

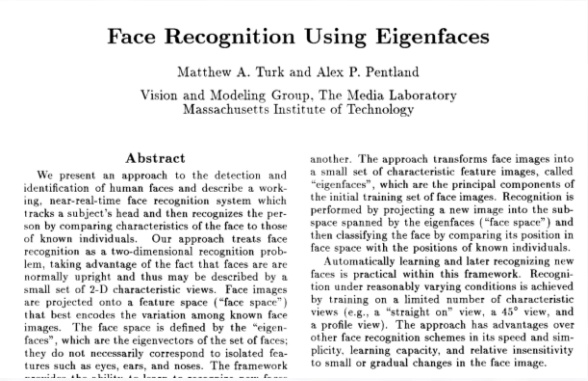
where **D** is the diagonal matrix of eigenvalues; V is the matrix of corresponding eigenvectors.

(6) Sort of the columns of D into order of decreasing eigenvalues, and apply the same order to the columns of V.

(7) Reject those with eigenvalues less than some given threshold, leaving L dimensions in the data.

Facial Recognition: EigenFace

• Sirovich and Kirby (1987) developed an early (now classic) algorithm for facial recognition: "Face recognition using eigenfaces" **(Eigenface)**.



Face Recognition Using Eigenfaces

Matthew A. Turk and Alex P. Pentland

Vision and Modeling Group, The Media Laboratory
Massachusetts Institute of Technology

Abstract

We present an approach to the detection and identification of human faces and describe a working, near-real-time face recognition system which tracks a subject's head and then recognizes the person by comparing characteristics of the face to those of known individuals. Our approach treats face recognition as a two-dimensional recognition problem, taking advantage of the fact that faces are are normally upright and thus may be described by a small set of 2-D characteristic views. Face images are projected onto a feature space ("face space") that best encodes the variation among known face images. The face space is defined by the "eigenfaces", which are the eigenvectors of the set of faces; they do not necessarily correspond to isolated features such as eyes, ears, and noses. The framework

another. The approach transforms face images into a small set of characteristic feature images, called "eigenfaces", which are the principal components of the initial training set of face images. Recognition is performed by projecting a new image into the subspace spanned by the eigenfaces ("face space") and then classifying the face by comparing its position in face space with the positions of known individuals.

Automatically learning and later recognizing new faces is practical within this framework. Recognition under reasonably varying conditions is achieved by training on a limited number of characteristic views (e.g., a "straight on" view, a 45° view, and a profile view). The approach has advantages over other face recognition schemes in its speed and simplicity, learning capacity, and relative insensitivity to small or gradual changes in the face image.

It is a very simple yet effective algorithm (simplified version):

(1) Determine the SVD of the mean-centered covariance matrix of the (training) dataset of face images (all front facing) – i.e., perform PCA. We only retain the eigenvectors associated with the largest eigenvalues (usually ~100 or so).

(2) The eigenvectors produced from SVD form a **basis set** of for the training data.

(3) For facial recognition – given a test datum (i.e., new face image); we project this image into the eigenspace spanned by the basis set. From the weights produced by this projection, we compare the weights (wrt basis set) of all training images; the nearest neighbor (per L2, etc.) of the test image in the training set corresponds with the recognized face.

https://sites.cs.ucsb.edu/~mturk/Papers/mturk-CVPR91.pdf

Facial Recognition: EigenFace

- In more detail:

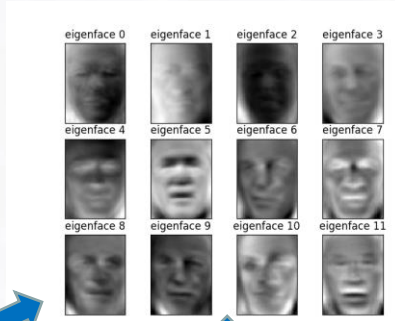
To create a set of eigenfaces, one must:

1. Prepare a training set of face images. The pictures constituting the training set should have been taken under the same lighting conditions, and must be normalized to have the eyes and mouths aligned across all images. They must also be all resampled to a common pixel resolution ($r \times c$). Each image is treated as one vector, simply by concatenating the rows of pixels in the original image, resulting in a single column with $r \times c$ elements. For this implementation, it is assumed that all images of the training set are stored in a single matrix **T**, where each column of the matrix is an image.
2. Subtract the mean. The average image **a** has to be calculated and then subtracted from each original image in **T**.
3. Calculate the eigenvectors and eigenvalues of the covariance matrix **S**. Each eigenvector has the same dimensionality (number of components) as the original images, and thus can itself be seen as an image. The eigenvectors of this covariance matrix are therefore called eigenfaces. They are the directions in which the images differ from the mean image. Usually this will be a computationally expensive step (if at all possible), but the practical applicability of eigenfaces stems from the possibility to compute the eigenvectors of **S** efficiently, without ever computing **S** explicitly, as detailed below.
4. Choose the principal components. Sort the eigenvalues in descending order and arrange eigenvectors accordingly. The number of principal components k is determined arbitrarily by setting a threshold ε on the total variance. Total variance $v = (\lambda_1 + \lambda_2 + \ldots + \lambda_n)$, n = number of components.
5. k is the smallest number that satisfies $\frac{(\lambda_1 + \lambda_2 + \ldots + \lambda_k)}{v} > \epsilon$

Training dataset

PCA

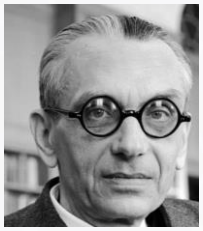
$$\mathbf{C} = \frac{1}{N}\mathbf{B}\mathbf{B}^T \rightarrow C = \mathbf{V}\mathbf{D}\mathbf{V}^T$$



Basis set

$\langle w_1, w_2, w_3, \ldots, w_k \rangle$

$\langle w'_1, w'_2, w'_3, \ldots, w'_k \rangle$



Nearest neighbor training image (in Eigenspace)

Test image

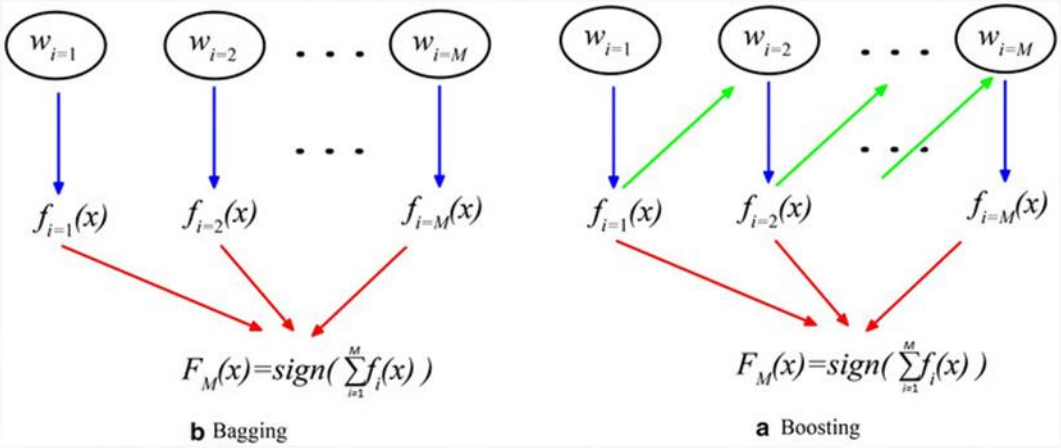
https://sites.cs.ucsb.edu/~mturk/Papers/mturk-CVPR91.pdf

Topic Review:
Boosting and Adaboost

Model Combination Schemes: Boosting

• With **boosting**, we *actively* try to generate complementary base-learners by training the learners *sequentially*, so that the next learner trains on the mistakes of the previous learners.

• Boosting combines complementary *weak learners* (meaning their accuracy is above chance, but they are nonetheless relatively inexpensive to train).



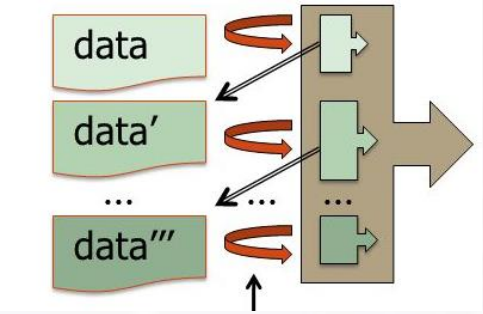
"Intro to Boosting": https://cseweb.ucsd.edu/~yfreund/papers/IntroToBoosting.pdf

Model Combination Schemes: Boosting

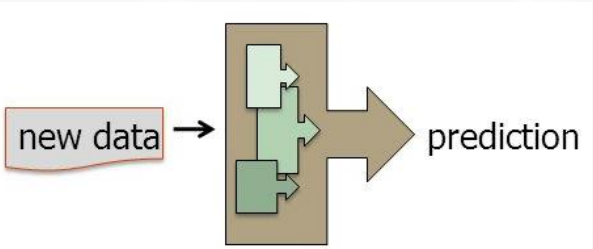
• As a basic schematic for boosting, consider a boosting algorithm (this is how the original 1990 Schapire paper worked) that combines three weak learners to generate a strong learner.

• Given a training set, we randomly partition it into three subsets: X_1, X_2 and X_3; use X_1 to train d_1. Then take X_2 and feed it to d_1. Next, we use every instance misclassified by d_1 in combination with many instances on which d_1 is correct from X_2, and together form the training set for d_2.

• Lastly, we take X_3 and feed it to d_1 and d_2; the instances on which d_1 and d_2 disagree form the training set of d_3.



(*) During testing, we take a datum and give it to d_1 and d_2; if they agree this is the prediction; otherwise, the response of d_3 is taken as the output.



Model Combination Schemes: AdaBoost

• A very popular boosting method known as **AdaBoost*** (short for *adaptive boosting*) was developed by Freund and Schapire in 1996 (later won the *Gödel prize*).

(*) Adaboost uses the same training set over and over and thus the data set need not be large, but the classifiers should be simple so that they do no overfit. AdaBoost can also combine an arbitrary number of base learners – not just three.

(*) Adaboost combines different weak learners (i.e. hypotheses), where the training error is close but less than 50%, to produce a strong learner (i.e. with training error close to zero).

*https://cseweb.ucsd.edu/~yfreund/papers/IntroToBoosting.pdf

AdaBoost: Algorithm Sketch

Given examples S and learning algorithm L, with $|S| = N$

- Initialize probability distribution over examples $\mathbf{w}_1(i) = 1/N$.
- Repeatedly run L on training sets $S_t \subset S$ to produce $h_1, h_2, \ldots, h_K$.
  - At each step, derive S_t from S by choosing examples probabilistically according to probability distribution $\mathbf{w}_t$. Use S_t to learn h_t.
- At each step, derive $\mathbf{w}_{t+1}$ by giving more probability to examples that were misclassified at step t.
- The final ensemble classifier H is a weighted sum of the h_t's, with each weight being a function of the corresponding h_t's error on its training set.

AdaBoost: Algorithm

- Given $S = \{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_N, y_N)\}$ where $\mathbf{x} \in \boldsymbol{X}, y_i \in \{+1, -1\}$
- Initialize $\mathbf{w}_1(i) = 1/N$. (Uniform distribution over data)

AdaBoost: Algorithm

- For $t = 1, ..., K$:
  - Select new training set S_t from S with replacement, according to $\mathbf{w}_t$
  - Train L on S_t to obtain hypothesis h_t
  - Compute the training error ε_t of h_t on $\boldsymbol{S}$:

$$e_t = \sum_{j=1}^{N} \mathbf{w}_t(j)\, d(y_j \neq h_t(\mathbf{x}_j)), \text{ where}$$

$$d(y_j \neq h_t(\mathbf{x}_j)) = \begin{cases} 1 \text{ if } y_j \neq h_t(\mathbf{x}_j) \\ 0 \text{ otherwise} \end{cases}$$

  - Compute coefficient

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$

AdaBoost: Algorithm

– Compute new weights on data:

For $i = 1$ to N

$$\mathbf{w}_{t+1}(i) = \frac{\mathbf{w}_t(i)\ \exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t}$$

where Z_t is a normalization factor chosen so that $\mathbf{w}_{t+1}$ will be a probability distribution:

$$Z_t = \sum_{i=1}^{N} \mathbf{w}_t(i)\ \exp(-\alpha_t y_i h_t(\mathbf{x}_i))$$

AdaBoost: Algorithm

- At the end of K iterations of this algorithm, we have

  $h_1, h_2, \ldots, h_K$

  We also have

  $\alpha_1, \alpha_2, \ldots, \alpha_K$, where

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$

- Ensemble classifier:

$$H(\mathbf{x}) = \operatorname{sgn}\sum_{t=1}^{K} a_t h_t(\mathbf{x})$$

- Note that hypotheses with higher accuracy on their training sets are weighted more strongly.

AdaBoost: Data Example

$$S = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8,\}$$

where $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$ are class +1

$\{\mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8\}$ are class −1

$t = 1$:

$\mathbf{w}_1 = \{1/8, 1/8, 1/8, 1/8, 1/8, 1/8, 1/8, 1/8\}$

$S_1 = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_2, \mathbf{x}_5, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8\}$ (notice some repeats)

Train classifier on S_1 to get h_1

Run h_1 on **S**. Suppose classifications are: {**1**, **−1**, **−1**, **−1**, **−1**, **−1**, **−1**, **−1**}

- Calculate error: $\epsilon_1 = \sum_{j=1}^{N} \mathbf{w}_t(j)\,\delta(y_j \neq h_t(\mathbf{x}_j)) = \frac{1}{8}(3) = .375$

AdaBoost: Data Example

$$S = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8,\}$$

where $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$ are class +1

$\{\mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8\}$ are class −1

$t = 1$:

$\mathbf{w}_1 = \{1/8, 1/8, 1/8, 1/8, 1/8, 1/8, 1/8, 1/8\}$

$S_1 = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_2, \mathbf{x}_5, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8\}$ (notice some repeats)

Train classifier on S_1 to get h_1

Run h_1 on **S**. Suppose classifications are: {**1**, **−1**, **−1**, **−1**, **−1**, **−1**, **−1**, **−1**}

- Calculate error: $\epsilon_1 = \sum_{j=1}^{N} \mathbf{w}_t(j)\,\delta(y_j \neq h_t(\mathbf{x}_j)) = ?$

Calculate α's:

$$\alpha_1 = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)=$$

Calculate new $\mathbf{w}$'s:

$$\mathbf{w}_{t+1}(i) = \frac{\mathbf{w}_t(i)\ \exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t}$$

$\hat{\mathbf{w}}_2(1) =$

$\hat{\mathbf{w}}_2(2) =$

$\hat{\mathbf{w}}_2(3) =$

$\hat{\mathbf{w}}_2(4) =$

$\hat{\mathbf{w}}_2(5) =$

$\hat{\mathbf{w}}_2(6) =$

$\hat{\mathbf{w}}_2(7) =$

$\hat{\mathbf{w}}_2(8) =$

$\mathbf{w}_2(1) =$

$\mathbf{w}_2(2) =$

$\mathbf{w}_2(3) =$

$\mathbf{w}_2(4) =$

$\mathbf{w}_2(5) =$

$\mathbf{w}_2(6) =$

$\mathbf{w}_2(7) =$

$\mathbf{w}_2(8) =$

$$Z_1 = \sum_i \hat{\mathbf{w}}_2(i) =$$

Calculate α's:

$$\alpha_1 = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right) = .255$$

Calculate new **w**'s:

$$\mathbf{w}_{t+1}(i) = \frac{\mathbf{w}_t(i)\ \exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t}$$

$$\hat{\mathbf{w}}_2(1) = (.125)\exp(-.255(1)(1)) = 0.1 \qquad \mathbf{w}_2(1) = 0.1/.98 = 0.102$$
$$\hat{\mathbf{w}}_2(2) = (.125)\exp(-.255(1)(-1)) = 0.16 \qquad \mathbf{w}_2(2) = 0.163$$
$$\hat{\mathbf{w}}_2(3) = (.125)\exp(-.255(1)(-1)) = 0.16 \qquad \mathbf{w}_2(3) = 0.163$$
$$\hat{\mathbf{w}}_2(4) = (.125)\exp(-.255(1)(-1)) = 0.16 \qquad \mathbf{w}_2(4) = 0.163$$
$$\hat{\mathbf{w}}_2(5) = (.125)\exp(-.255(-1)(-1)) = 0.1 \qquad \mathbf{w}_2(5) = 0.102$$
$$\hat{\mathbf{w}}_2(6) = (.125)\exp(-.255(-1)(-1)) = 0.1 \qquad \mathbf{w}_2(6) = 0.102$$
$$\hat{\mathbf{w}}_2(7) = (.125)\exp(-.255(-1)(-1)) = 0.1 \qquad \mathbf{w}_2(7) = 0.102$$
$$\hat{\mathbf{w}}_2(8) = (.125)\exp(-.255(-1)(-1)) = 0.1 \qquad \mathbf{w}_2(8) = 0.102$$

$$Z_1 = \sum_i \hat{\mathbf{w}}_2(i) = .98$$

t = 2

w_2 = {0.102, 0.163, 0.163, 0.163, 0.102, 0.102, 0.102, 0.102}

S_2 = {x_1, x_2, x_2, x_3, x_4, x_4, x_7, x_8}

Learn classifier on S_2 to get h_2

Run h_2 on S. Suppose classifications are: {1, 1, 1, 1, 1, 1, 1, 1}

Calculate error:

$$\varepsilon_2 = \sum_{j=1}^{N} \mathbf{w}_t(j)\,\delta(y_j \neq h_t(\mathbf{x}_j))$$

$$= (.102) \times 4 = 0.408$$

Calculate α's:

$$a_2 = \frac{1}{2}\ln\left(\frac{1-e_t}{e_t}\right) = .186$$

Calculate $\mathbf{w}$'s:

$$\mathbf{w}_{t+1}(i) = \frac{\mathbf{w}_t(i)\ \exp(-a_t y_i h_t(\mathbf{x}_i))}{Z_t}$$

$$\hat{\mathbf{w}}_3(1) = (.102)\exp(-.186(1)(1)) = 0.08$$
$$\hat{\mathbf{w}}_3(2) = (.163)\exp(-.186(1)(1)) = 0.135$$
$$\hat{\mathbf{w}}_3(3) = (.163)\exp(-.186(1)(1)) = 0.135$$
$$\hat{\mathbf{w}}_3(4) = (.163)\exp(-.186(1)(1)) = 0.135$$
$$\hat{\mathbf{w}}_3(5) = (.102)\exp(-.186(-1)(1)) = 0.122$$
$$\hat{\mathbf{w}}_3(6) = (.102)\exp(-.186(-1)(1)) = 0.122$$
$$\hat{\mathbf{w}}_3(7) = (.102)\exp(-.186(-1)(1)) = 0.122$$
$$\hat{\mathbf{w}}_3(8) = (.102)\exp(-.186(-1)(1)) = 0.122$$

$$\mathbf{w}_3(1) = 0.08/.973 = 0.082$$
$$\mathbf{w}_3(2) = 0.139$$
$$\mathbf{w}_3(3) = 0.139$$
$$\mathbf{w}_3(4) = 0.139$$
$$\mathbf{w}_3(5) = 0.125$$
$$\mathbf{w}_3(6) = 0.125$$
$$\mathbf{w}_3(7) = 0.125$$
$$\mathbf{w}_3(8) = 0.125$$

$$Z_2 = \sum_i \hat{\mathbf{w}}_2(i) = .973$$

t =3

w_3 = {0.082, 0.139, 0.139, 0.139, 0.125, 0.125, 0.125, 0.125}

S_3 = {x_2, x_3, x_3, x_3, x_5, x_6, x_7, x_8}

Run classifier on S_3 to get h_3

Run h_3 on S. Suppose classifications are: {1, 1, −1, 1, −1, −1, 1, −1}

Calculate error:

$$\varepsilon_3 = \sum_{j=1}^{N} w_t(i)\,\delta(y_j \neq h_t(x_j))$$

$$= (.139) + (.125) = 0.264$$

- Calculate α's:

$$\alpha_3 = \frac{1}{2}\ln\left(\frac{1-e_t}{e_t}\right) = .512$$

- Ensemble classifier:

$$H(\mathbf{x}) = \text{sgn}\sum_{t=1}^{K}\alpha_t h_t(\mathbf{x})$$

$$= \text{sgn}\left(.255 \times h_1(\mathbf{x}) + .186 \times h_2(\mathbf{x}) + .512 \times h_3(\mathbf{x})\right)$$

Example	Actual class	h_1	h_2	h_3
$\mathbf{x}_1$	1	1	1	1
$\mathbf{x}_2$	1	−1	1	1
$\mathbf{x}_3$	1	−1	1	−1
$\mathbf{x}_4$	1	1	1	1
$\mathbf{x}_5$	−1	−1	1	−1
$\mathbf{x}_6$	−1	−1	1	−1
$\mathbf{x}_7$	−1	1	1	1
$\mathbf{x}_8$	−1	−1	1	−1

Recall the training set:

$$S = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8,\}$$

where { x_1, x_2, x_3, x_4 } are class +1

{x_5, x_6, x_7, x_8 } are class −1

$$H(\mathbf{x}) = \operatorname{sgn}\sum_{t=1}^{T} a_t h_t(\mathbf{x})$$

$$= \operatorname{sgn}\left(.255\times h_1(\mathbf{x}) + .186\times h_2(\mathbf{x}) + .512\times h_3(\mathbf{x})\right)$$

AdaBoost: Summary

- Given S = $\{(x_1, y_1), ..., (x_N, y_N)\}$ where $x \in X$, $y_i \in \{+1, -1\}$
- Initialize $w_1(i) = 1/N$. (Uniform distribution over data)
- For $t = 1, ..., K$:

1. Select new training set S_t from S with replacement, according to $\mathbf{w}_t$
1. Train L on S_t to obtain hypothesis h_t
1. Compute the training error ε_t of h_t on $\boldsymbol{S}$:

$$\varepsilon_t = \sum_{j=1}^{N} \mathbf{w}_t(j)\, \delta(y_j \neq h_t(\mathbf{x}_j)),$$

$$\text{where } \delta(y_j \neq h_t(\mathbf{x}_j)) = \begin{cases} 1 \text{ if } y_j \neq h_t(\mathbf{x}_j) \\ 0 \text{ otherwise} \end{cases}$$

If $\varepsilon_t > 0.5$, abandon h_t and go to step 1

4. Compute coefficient:

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$

5. Compute new weights on data:

For $i = 1$ to N

$$\mathbf{w}_{t+1}(i) = \frac{\mathbf{w}_t(i)\ \exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t}$$

where Z_t is a normalization factor chosen so that $\mathbf{w}_{t+1}$ will be a probability distribution:

$$Z_t = \sum_{i=1}^{N} \mathbf{w}_t(i)\ \exp(-\alpha_t y_i h_t(\mathbf{x}_i))$$

- At the end of K iterations of this algorithm, we have $h_1, h_2, \ldots, h_K$, and $\alpha_1, \alpha_2, \ldots, \alpha_K$

- Ensemble classifier:

$$H(\mathbf{x}) = \mathrm{sgn}\sum_{t=1}^{K} \alpha_t h_t(\mathbf{x})$$

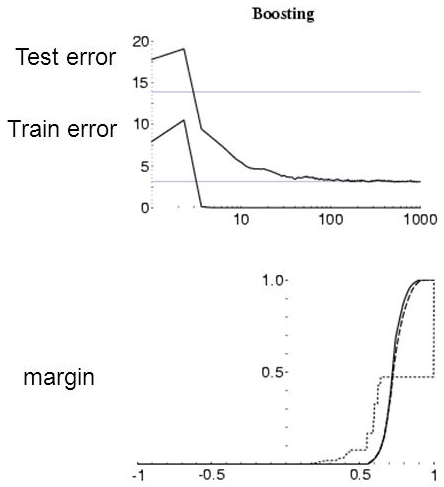
AdaBoost: Overview

• Adaboost seems to reduce both bias and variance and it does not seem to overfit for increasing K.

Why does it work?

Schapire et al. explain that the success of AdaBoost is *due to its property of increase the margin*. Recall from SVMs, that if the margin increases, the training instances are better separated, and an error is less likely.

Adaboost: Margin Maximizer



AdaBoost: Overview

•In AdaBoost, although different base-learners have slightly different training sets, this difference is not left to chance as in bagging, but is a function of the error of the previous base-learner. The actual performance of boosting on a particular problem is naturally dependent on the data and base-learner.

• In order to be effective, there should be enough training data and the base-learner should be weak but not too weak, as boosting is particularly susceptible to noise and outliers (since boosting focuses on examples are hard to classify).

• For this reason, boosting can be used to identify outliers and noise in a dataset.

(*) AdaBoost has also been generalized to regression.

Case Study of Adaboost: Viola-Jones Face Detection Algorithm

• P. Viola and M. J. Jones, *Robust real-time face detection.** International Journal of Computer Vision, 2004.

• First face-detection algorithm to work well in real-time (e.g., on digital cameras); it has been very influential in computer vision (16k+ citations); makes use of Adaboost.

Viola:
MIT/Amazon

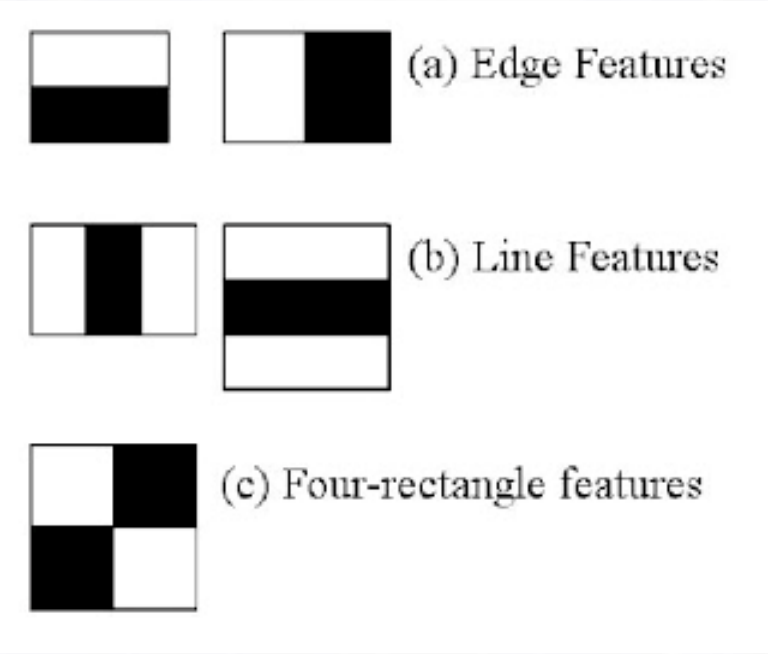
*https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf

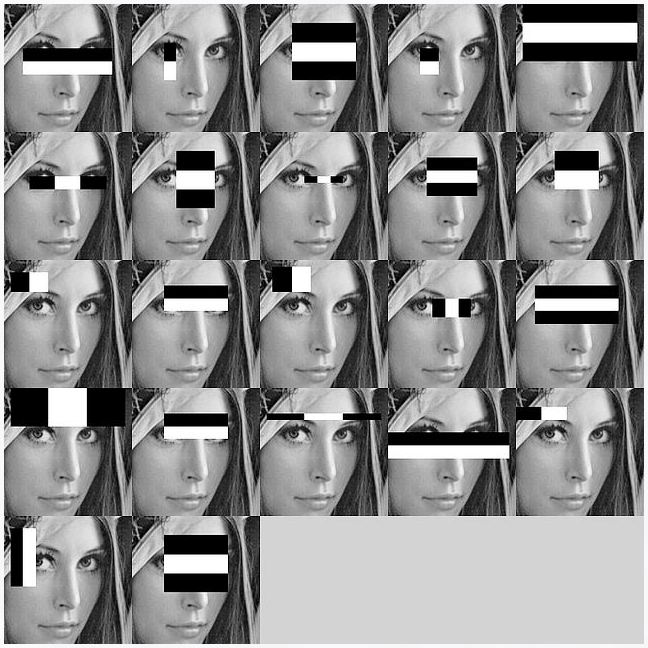
Viola-Jones: Training Data

- Positive: Faces scaled and aligned to a base resolution of 24 by 24 pixels.
- Negative: Much larger number of non-faces.

Figure 8. Example of frontal upright face images used for training.

Features





• Use rectangle features at multiple sizes and location in an image subwindow (candidate face).

From http://makematics.com/research/viola-jones/

For each feature f_j :

$$f_j = \sum_{b \in \text{black pixels}} \text{intensity(pixel } b\text{)} - \sum_{w \in \text{white pixels}} \text{intensity(pixel } w\text{)}$$

Possible number of features per 24 x 24 pixel subwindow > 180,000.

Detecting faces

Given a new image:

- Scan image using subwindows at all locations and at different scales
- For each subwindow, compute features and send them to an ensemble classifier (learned via boosting). If classifier is positive ("face"), then detect a face at this location and scale.

Viola-Jones Face Detection Algorithm

• **Preprocessing:** Viola & Jones use a clever pre-processing step that allows the rectangular features to be computed very quickly. (See their paper for description.)

• They use a variant of AdaBoost to both select a small set of features and train the classifier.

Viola-Jones Face Detection Algorithm

Base ("weak) classifiers:

For each feature f_j,

$$h_j = \left\{ \begin{array}{c} 1 \text{ if } p_j f_j(x) < p_j q_j \\ -1 \text{ otherwise} \end{array} \right\}$$

where x is a 24 x 24-pixel subwindow of an image, θ_j is the threshold that best separates the data using feature f_j, and p_j is either -1 or 1.

Such features are called **decision stumps**.

Viola-Jones Face Detection Algorithm

Boosting algorithm:

- Given example images $(x_1, y_1), \ldots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.

Viola-Jones Face Detection Algorithm

Boosting algorithm:

- For $t = 1, \ldots, T$:

  1. Normalize the weights,

  $$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$

  so that w_t is a probability distribution.

  2. For each feature, j, train a classifier h_j which is restricted to using a single feature. The error is evaluated with respect to w_t, $\epsilon_j = \sum_i w_i \left| h_j(x_i) - y_i \right|$.

  3. Choose the classifier, h_t, with the lowest error ϵ_t.

Viola-Jones Face Detection Algorithm

Boosting algorithm:

4. Update the weights:

$$w_{t+1,i} = w_{t,i}\beta_t^{1-e_i}$$

where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.

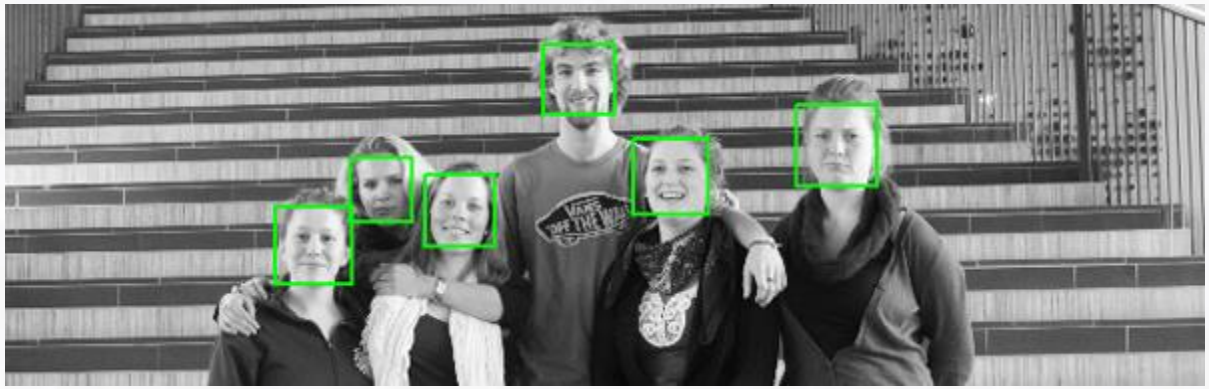
- The final strong classifier is:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

where $a_t = \ln \frac{1}{b_t}$

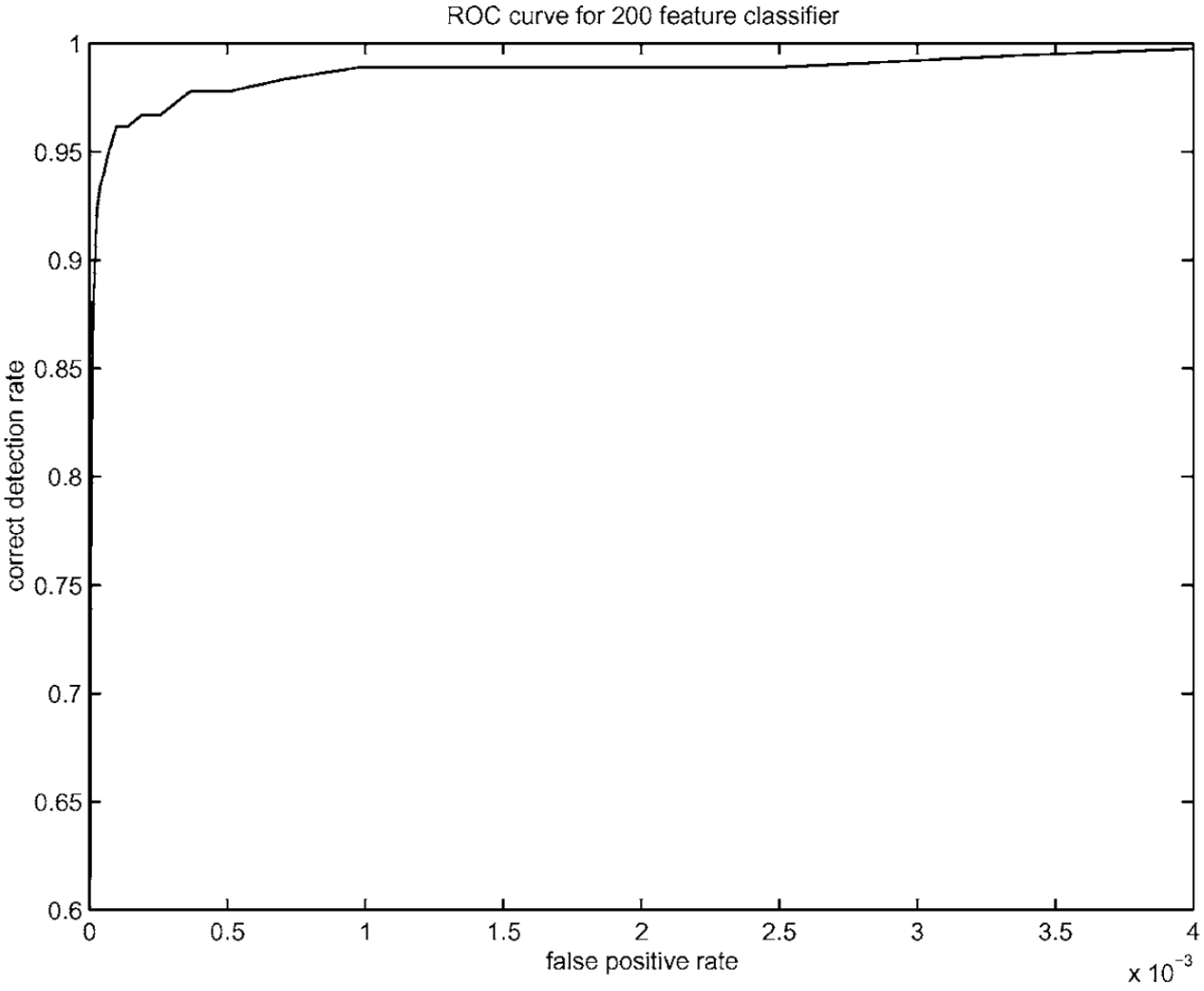
Note that only the top T features are used.

Viola-Jones Face Detection Algorithm



https://www.youtube.com/watch?v=k3bJUP0ct08

https://www.youtube.com/watch?v=c0twACIJYm8


ROC curve for 200 feature classifier
correct detection rate
false positive rate
x 10^-3
1
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0
0.5
1
1.5
2
2.5
3
3.5
4

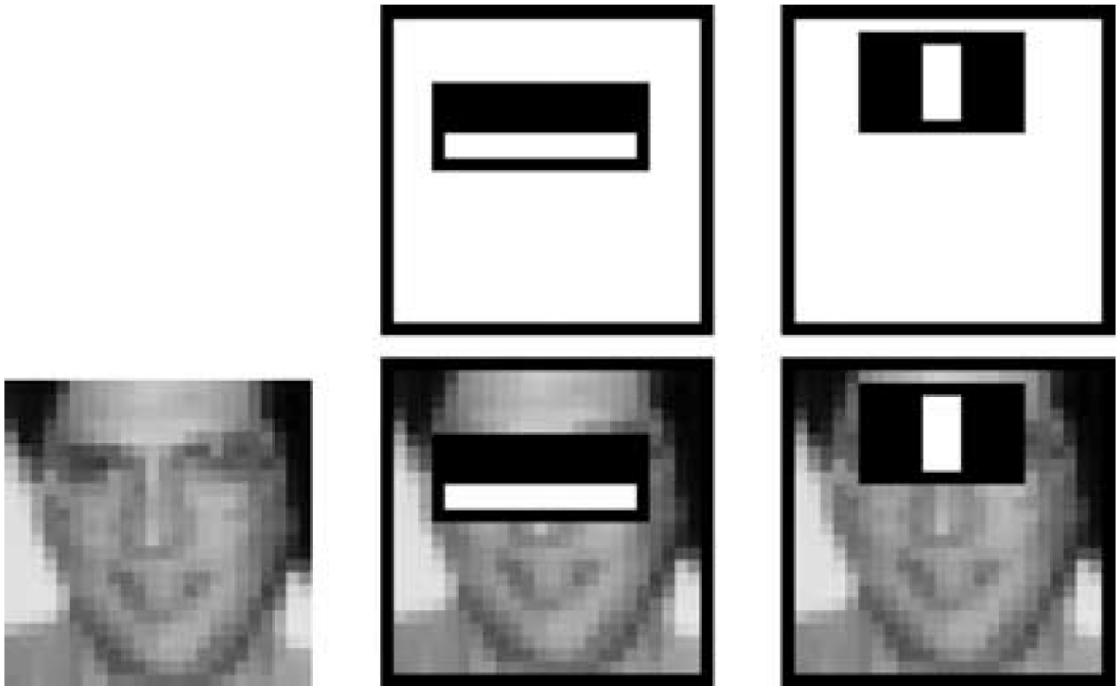


Figure 5. The first and second features selected by AdaBoost. The two features are shown in the top row and then overlayed on a typical training face in the bottom row. The first feature measures the difference in intensity between the region of the eyes and a region across the upper cheeks. The feature capitalizes on the observation that the eye region is often darker than the cheeks. The second feature compares the intensities in the eye regions to the intensity across the bridge of the nose.

Fin