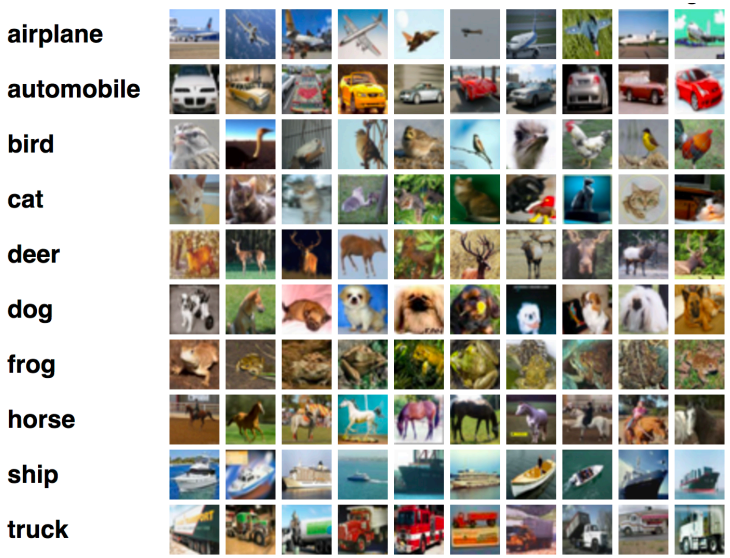
Computer vision:
basics of image processing

Materials for this section:

- scikit-image.org/docs/0.18.x/user_guide.html
- opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html
- N. Galatsanos et al., *Digital Image Enhancement* in Encyclopedia of Optical Engineering (2003).
- scikit-learn.org/stable/modules/clustering.html

Computer vision: typical tasks

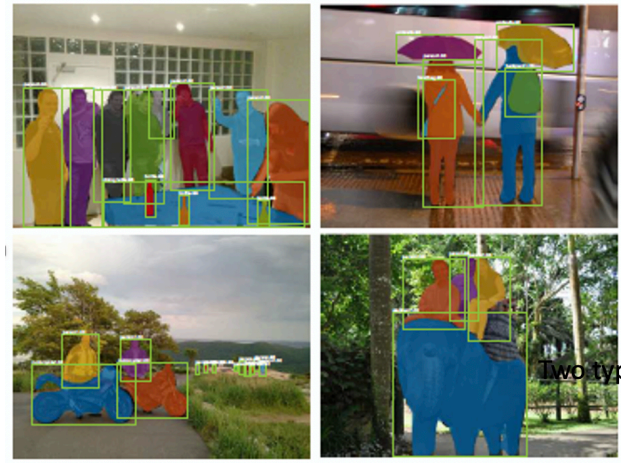
Image classification



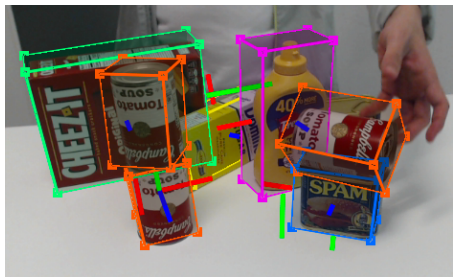
Object Detection

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 779-788).

Object segmentation

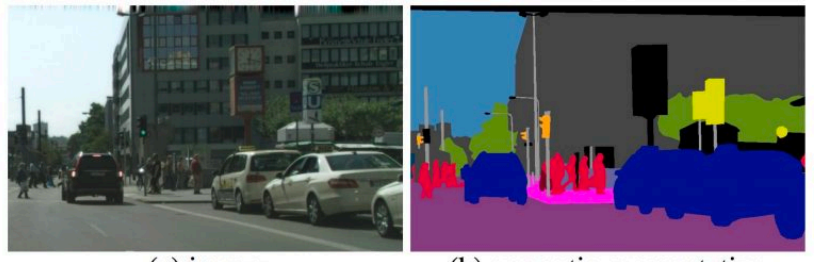


https://arxiv.org/abs/1703.06870

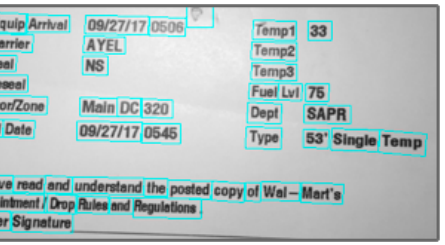


research.nvidia.com

Types of segmentation:



learnopencv.com



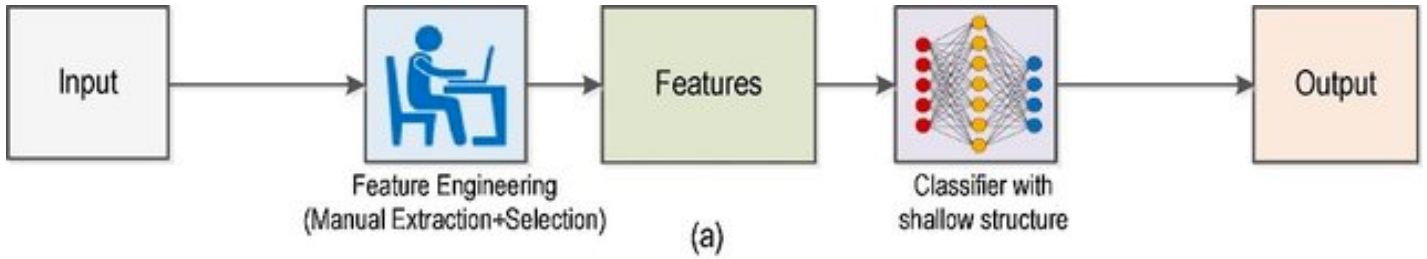
- 6D Pose estimation
- Optical character recognition (OCR)
- Face or fingerprint identification

Computer vision: approaches

Traditional classifiers:
SVM, DT (or ensembles)
or MLP

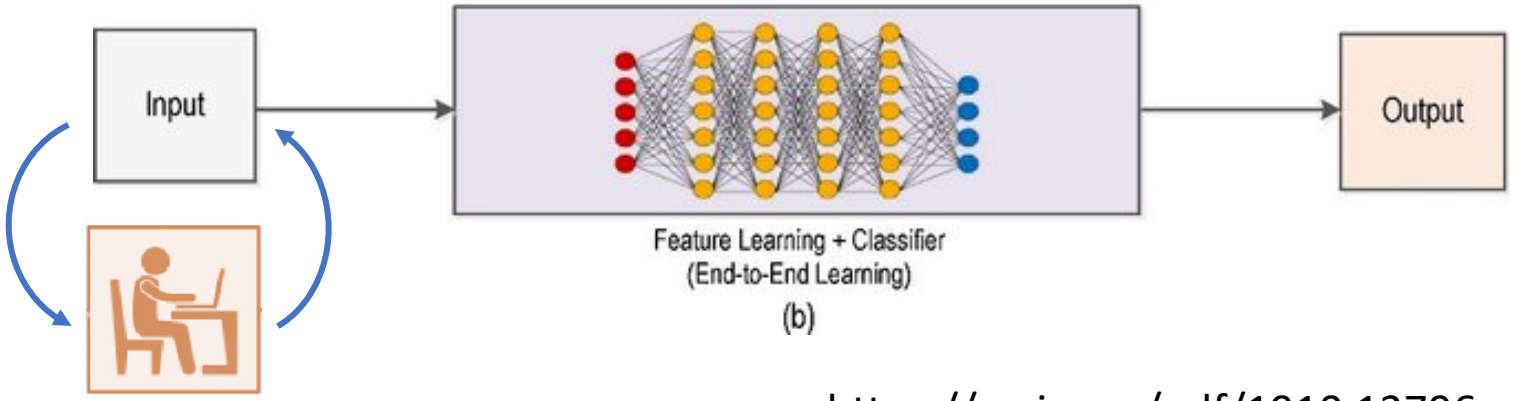
Traditional methods (developed since 1970s)

- Require feature extraction algorithms with many hand-tuned parameters
- Less power consuming than DL
- **New domain: tweaking parameters by hand**



Deep-learning (last 5-10 years)

- End-to-end approach: automatic feature extraction (feature map!)
- Nowadays: most of state-of-the-art approaches in CV employs deep-learning
- **New domain, problems with generalization: new dataset and/or domain adaptation methods**



https://arxiv.org/pdf/1910.13796.pdf

- **image preprocessing**
- **dataset augmentation**

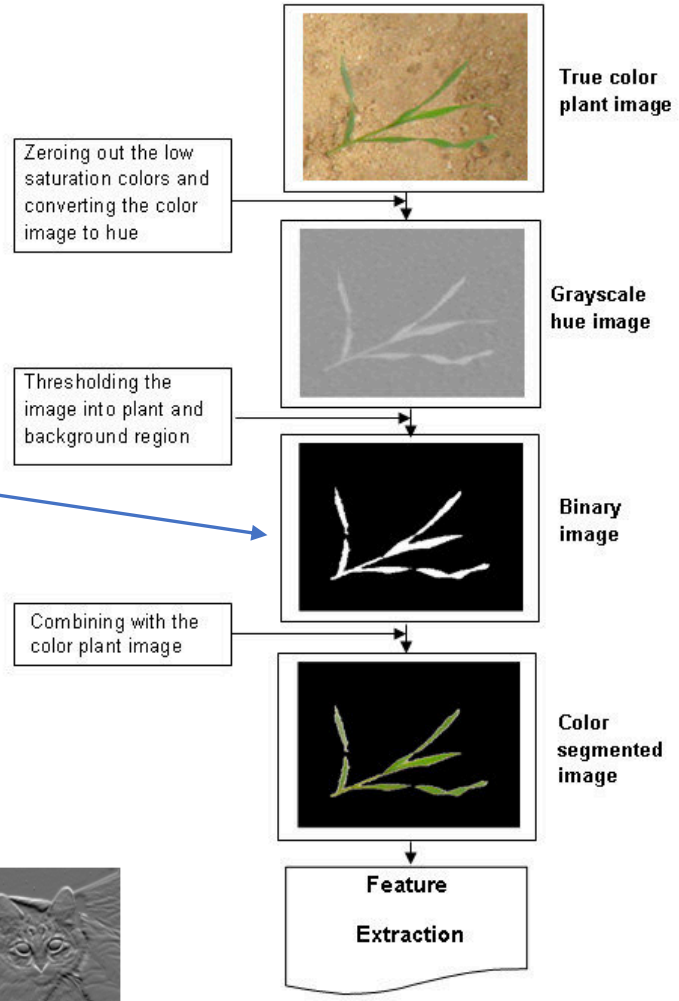
Features learned from a deep neural net are **specific** to training dataset which,
if not well constructed, probably won't perform well for images **different** from the training set

Feature extraction

Features in computer vision: vector of numbers:

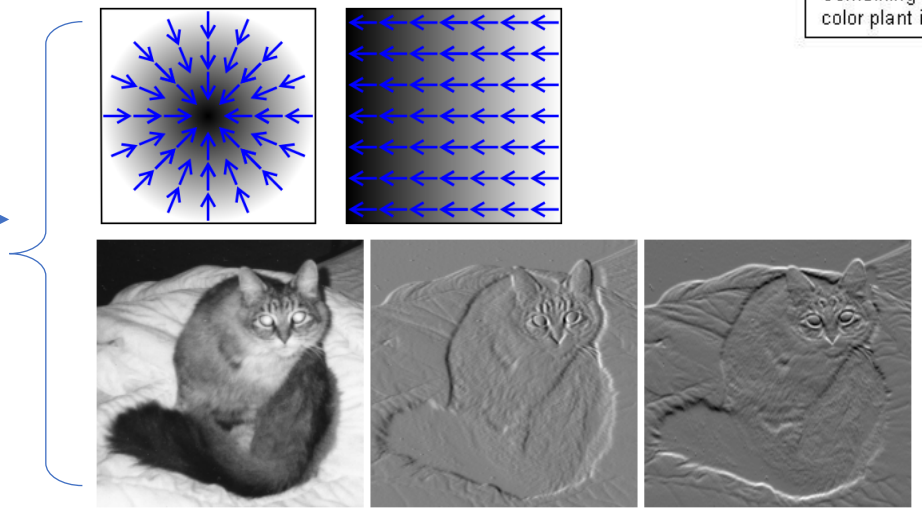
- characterizing whole image
  (image classification)
- describing interesting regions on images
  (e.g. keypoints or edges) with certain properties
  (object detection)
- Note that in CV community regions of interest (ROIs) itself are sometimes called *features* and numbers describing given region as *feature vector*

binary mask of interesting regions



Region-feature vector extraction:
(number calculated for each region)

- average of color (channel) magnitude
- pixel gradient or higher derivatives, e.g. Hessian matrix
- (local) histograms of pixel magnitudes
- region geometry attributes, such as the edge orientation



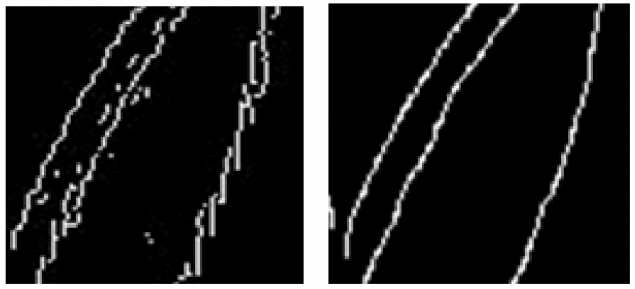
Plant Methods **7,** 28 (2011).

Feature extraction

How to detect Interesting regions (or *features*)?

Edge detectors:

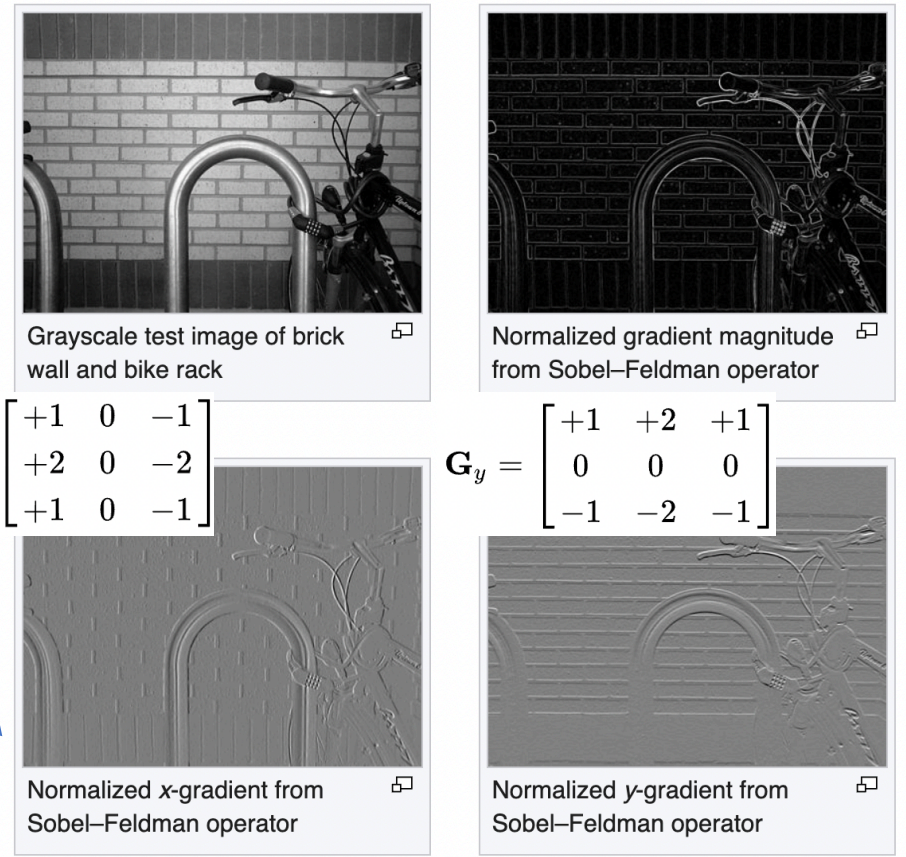
- **Sobel**
  - calculate approximation of the image gradient using convolution with Sobel kernels
  - apply threshold (intuition: regions with high gradient are likely edges)
  - other operators: Roberts, Prewitt
- **Canny**
  - multistage (denoising, edge operator, e.g. Sobel, then chose best pixel representing edge)
  - Canny edges are less noisy and straighter than ones from simple operators



(a) (b)

Figure 3.0: The edge generated by Sobel (a) and Canny Edge (b) detection algorithms.

Othman, Z. et al. "Comprison of canny and Sobel edge detection in MRI images." (2009).

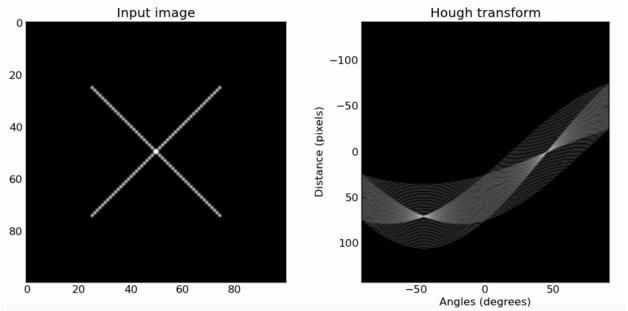


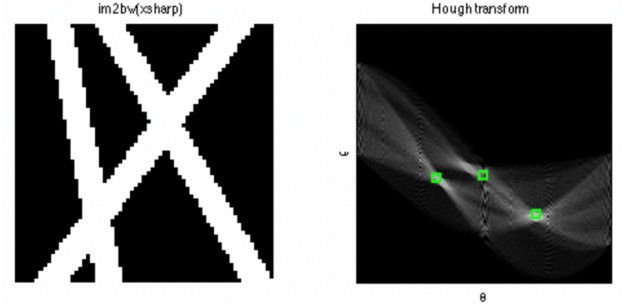
Convolution with kernel that approximates (minus) x-derivative

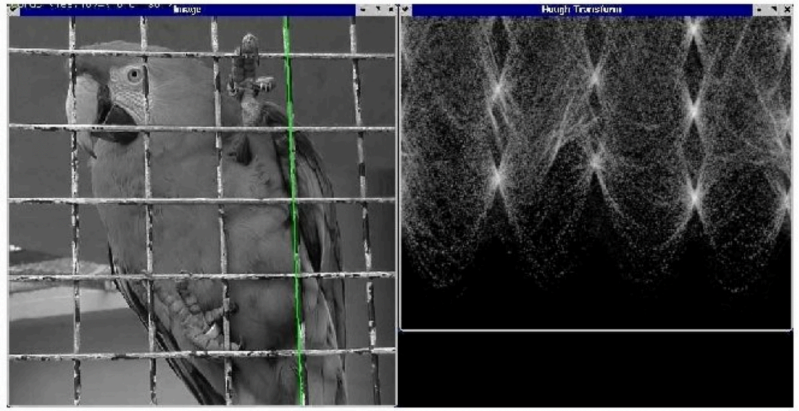
Feature extraction

How to detect Interesting regions (or *features*)?

Straight line detection: **Hough transform**
Variation: circle/ellipse shape detection



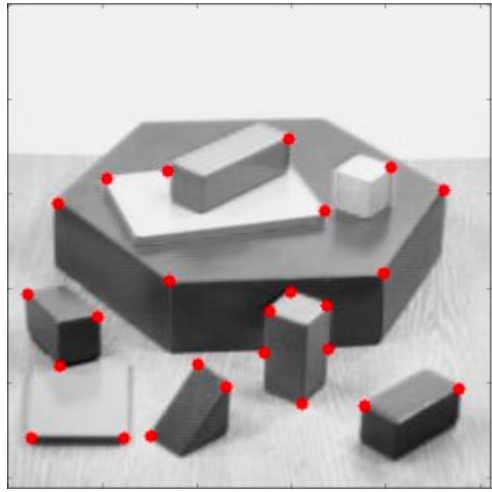




https://slidetodoc.com/fitting-the-hough-transform-voting-schemes-let-each/

Corner detection

- Plethora of algorithms
- can be interpreted as the junction of two edges
- Idea: analyze local differences (possible edges) in ortogonal directions at once



Simple example -- Harris detector:

- analyze *structure tensor* (*I* = pixel intensity map)

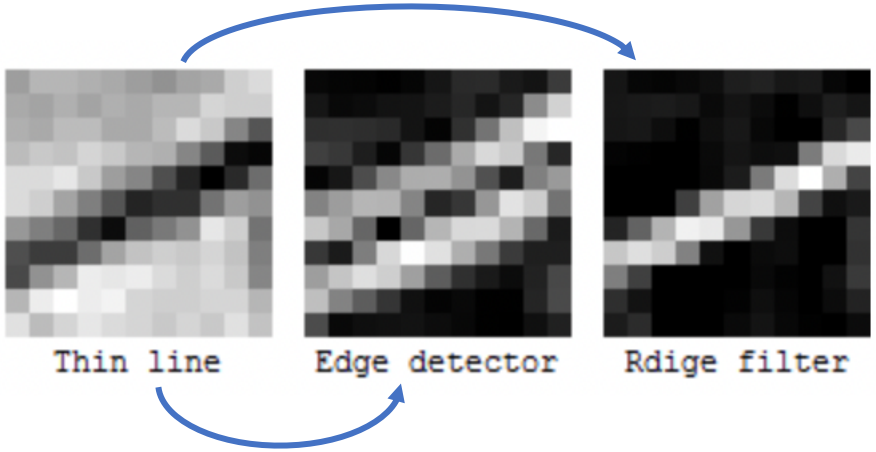
$$\begin{bmatrix} \sum_{(x,y)\in W} I_x^2 & \sum_{(x,y)\in W} I_x I_y \\ \sum_{(x,y)\in W} I_x I_y & \sum_{(x,y)\in W} I_y^2 \end{bmatrix}$$

- partial derivatives of Image with respect to x and y in corner: both eigenvalues are large

Feature extraction

How to detect Interesting regions (or *features*)

Ridge detector



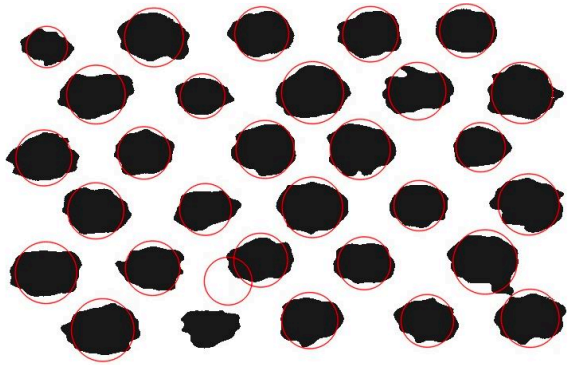
https://dsp.stackexchange.com/questions/1714/best-way-of-segmenting-veins-in-leaves

edges = borders between extended areas of high value,
ridges = thin lines darker or brighter than their neighborhood),

- edge filter will give two flanks, one on each side of the elongated object, and a low response in the middle of it,
- ridge detector gives single mark that might be easer to analyze

Blob detection

a blob is a region of an image in which some properties are constant or approximately constant

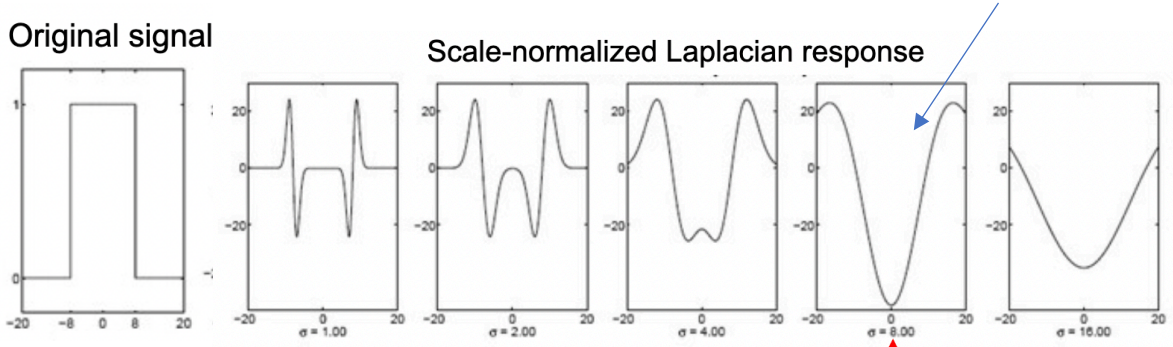


https://answers.opencv.org/

$$L(x, y; t) = g(x, y, t) * f(x, y)$$

$$\nabla^2_{norm} L = t\,(L_{xx} + L_{yy}) \quad g(x, y, t) = \frac{1}{2\pi t} e^{-\frac{x^2+y^2}{2t}}$$

Characteristic scale



www.di.ens.fr/willow/teaching/recvis10/assignment1/

Feature extraction

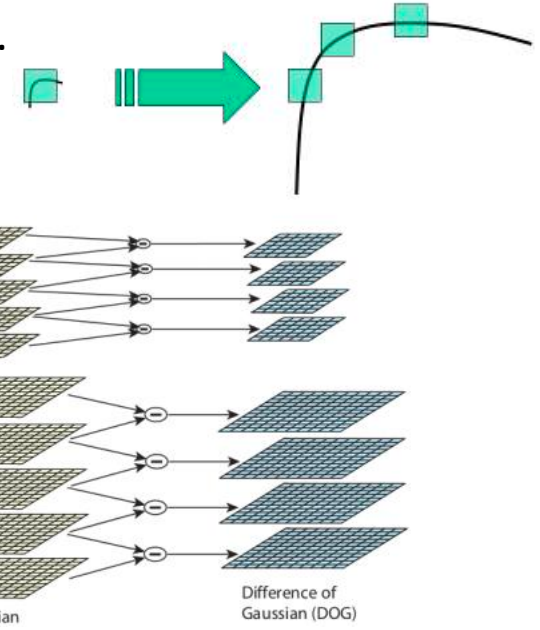
Multiscale Feature descriptors:

- **SIFT**
- Histograms of image gradient at key positions on image (*keypoints*)
- **SURF**
- Faster than SIFT
- Feature descriptor uses wavelet decomposition than gradients
- **BRIEF**
- Very fast regions of interest descriptor
- Needs additional method to extract ROIs
- and many others

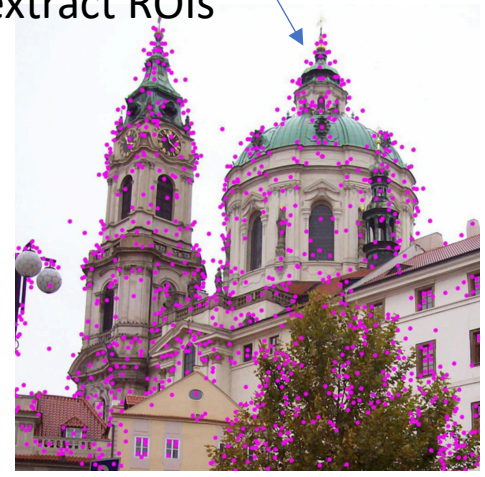
<u>SIFT algorithm</u>

- keypoint detection at different scales:
- detect blobs at different scales using Laplacian operator (approximated by difference of Gaussian convolutions with different sigmas)
- then local exteremas on DoG maps are founded as keypoints
- keypoint descriptor: histogram of grad.

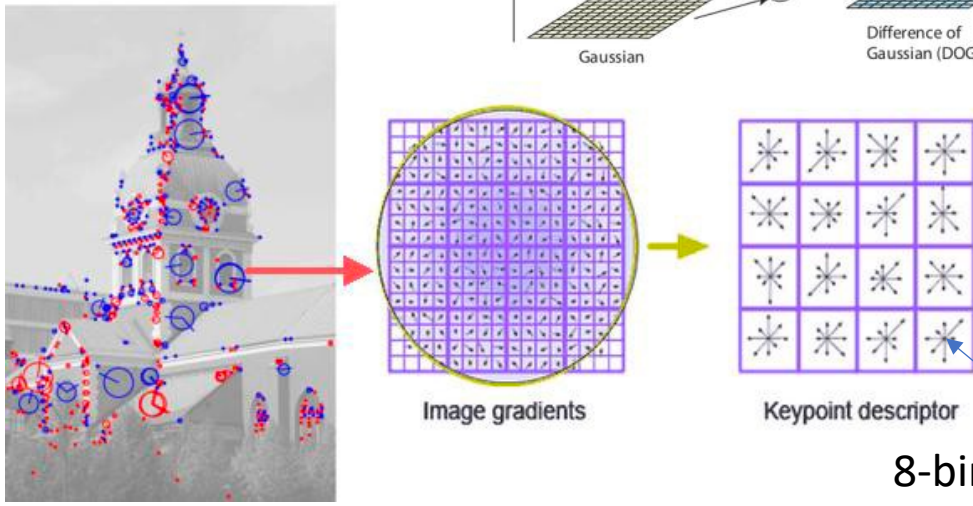
Scaling problem: corner here is not scale invariant.



opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials



medium.com

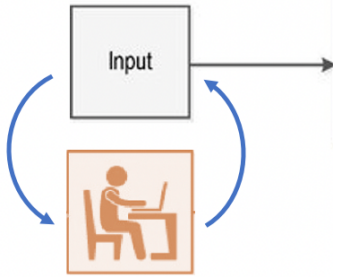


en.wikipedia.org/wiki/Scale-invariant_feature_transform

8-bin histograms of gradients

Image preprocessing

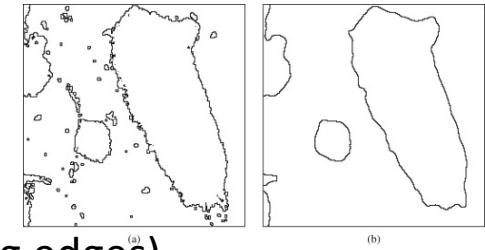
Preprocessing



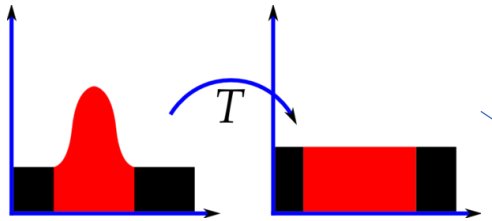
- simple: morphological filtering
- histogram equalization
- colors intensity normalization (important in deep-learning):

Filtering:

- median (usable for salt & pepper noise)
- morphological (for enhancing binary images/masks)
- anisotropic diffusion (strong denoising while preserving edges)



N. Galatsanos et al., "Digital Image Enhancement" (2003).



en.wikipedia.org/wiki/Histogram_equalization

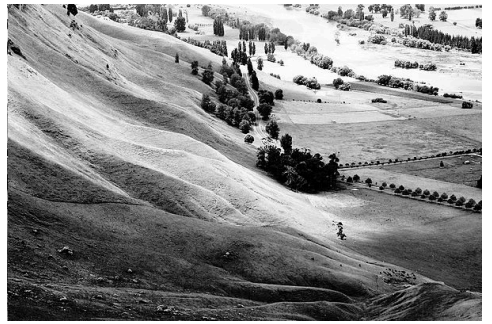
TORCHVISION.MODELS

All pre-trained models expect input images normalized in the same way, i.e. mini-batches of 3-channel RGB images of shape (3 x H x W), where H and W are expected to be at least 224. The images have to be loaded in to a range of [0, 1] and then normalized using `mean = [0.485, 0.456, 0.406]` and `std = [0.229, 0.224, 0.225]`. You can use the following transform to normalize:

```
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])
```

ImageNet parameters commonly used in CV to normalize images

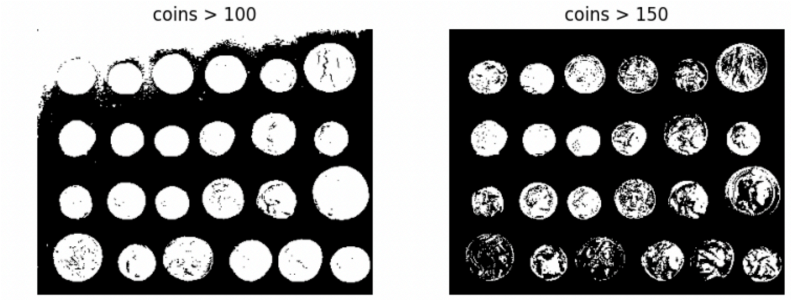
An example of such normalization can be found in the imagenet example here



Segmentation

Segmentation: partitioning image into multiple segments (objects)
Traditional approaches:

- thresholding, e.g. Otsu method (automatic threshold)
- edge detection
- watershed
- K-means



Thresholding text segments

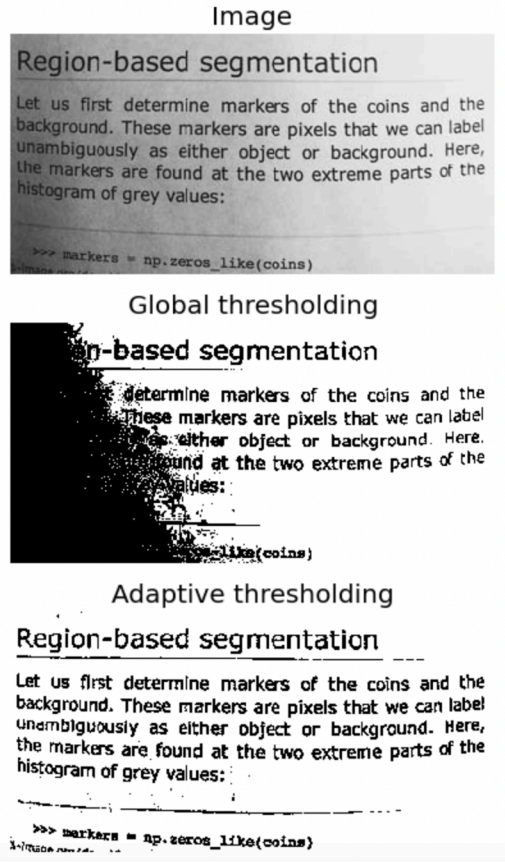
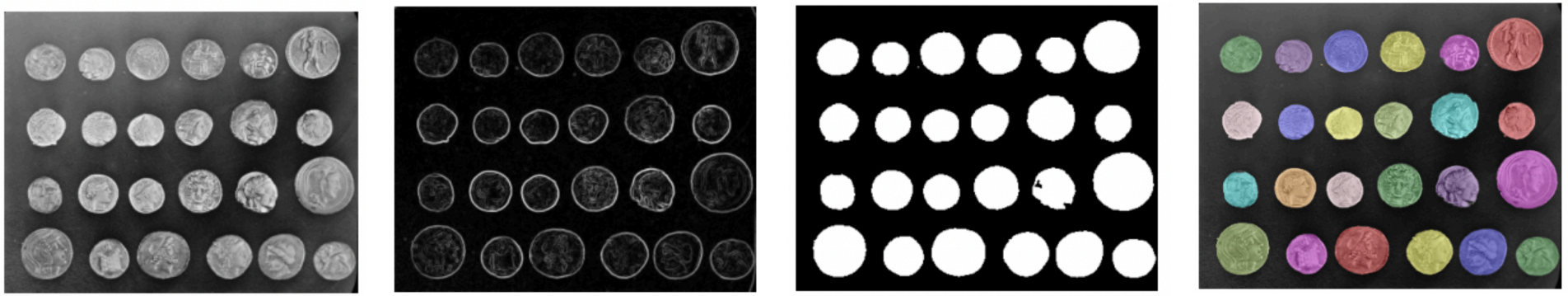


image → Sobel (relief map) → watershed → instances



https://scikit-image.org/docs/dev/user_guide/tutorial_segmentation.html

Segmentation

Segmentation: partitioning image into multiple segments (objects)
Traditional approaches:

- thresholding, e.g. Otsu method (automatic threshold)
- edge detection
- watershed
- **k-means**

Not to be confused with *k-nearest neighbors supervised classifier*

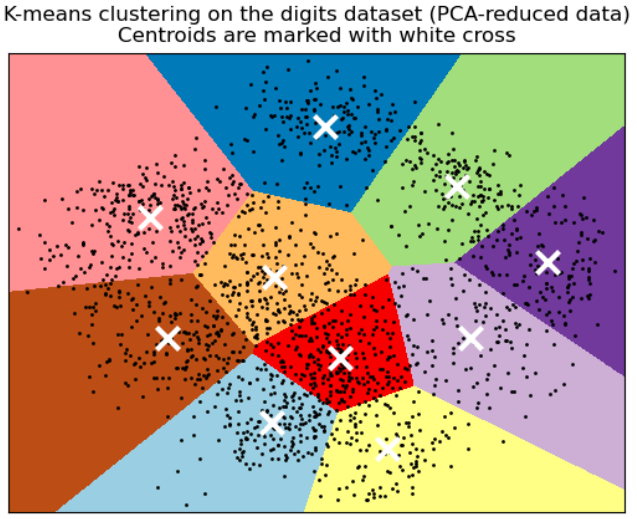
Note: this is not just simple geometrical distance

The K-means algorithm is an iterative technique that is used to partition an image into *K* clusters.[19] The basic algorithm is

1. Pick *K* cluster centers, either randomly or based on some heuristic method, for example K-means++
2. Assign each pixel in the image to the cluster that minimizes the distance between the pixel and the cluster center
3. Re-compute the cluster centers by averaging all of the pixels in the cluster
4. Repeat steps 2 and 3 until convergence is attained (i.e. no pixels change clusters)

In this case, distance is the squared or absolute difference between a pixel and a cluster center. The difference is typically based on pixel color, intensity, texture, and location, or a weighted combination of these factors. *K* can be selected manually, randomly, or by a heuristic. This algorithm is guaranteed to converge, but it may not return the optimal solution. The quality of the solution depends on the initial set of clusters and the value of *K*.

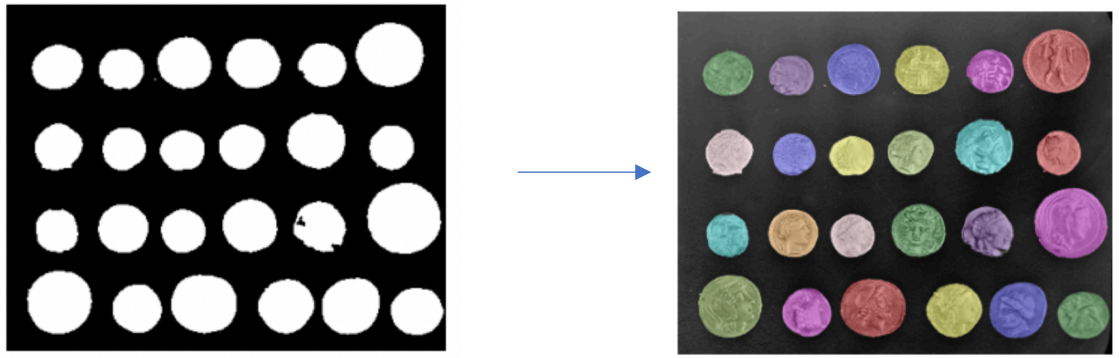
Voronoi tessellation with specifically defined metric



en.wikipedia.org/wiki/Image_segmentation

Clustering

How to color these segments?



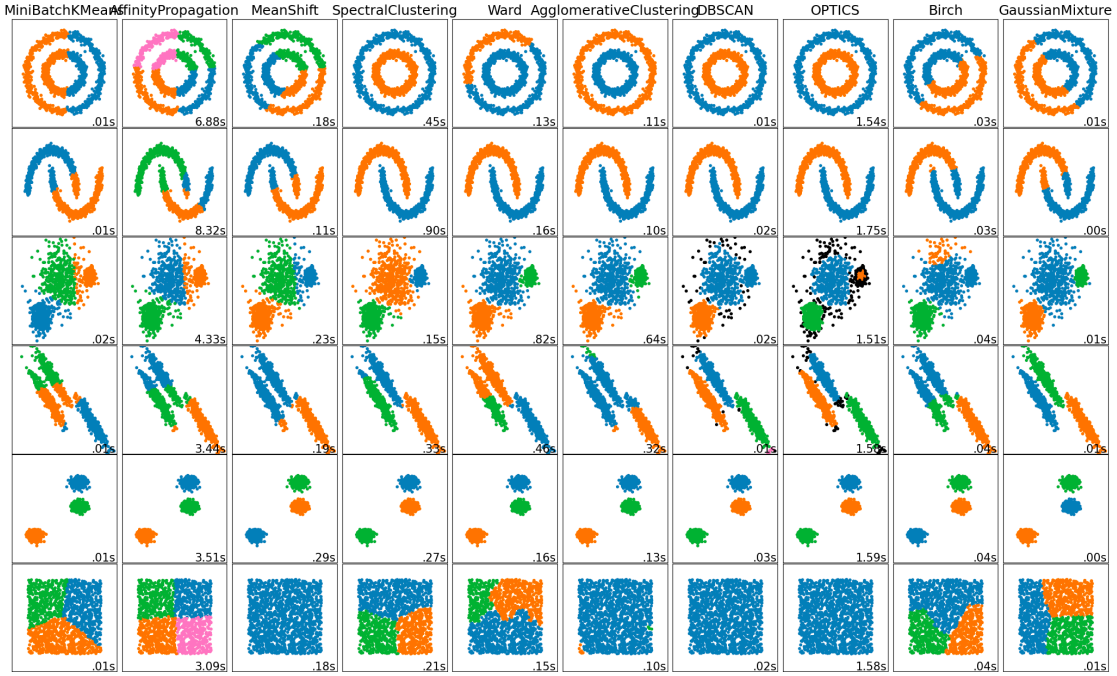
Cluster analysis, some popular algorithms:

- k-means
- spectral clustering
- Hierarchical clustering
- DBSCAN/OPTICS

Generalization to n-D space:
clustering of unlabeled data is type of **unsupervised learning**

Plethora of algorithms
Crucial parameter: number of clusters

- Few vs dozens (e.g. spectral vs hierarchical clustering)
- # of clusters to be specified in advance or not (e.g. DBSCAN don't need it)



scikit-learn.org/stable/modules/clustering.html