TSM_CompVis – Machine Learning in Computer Vision

Pascal Baumann, Selvin Blöchlinger

23. Januar 2024

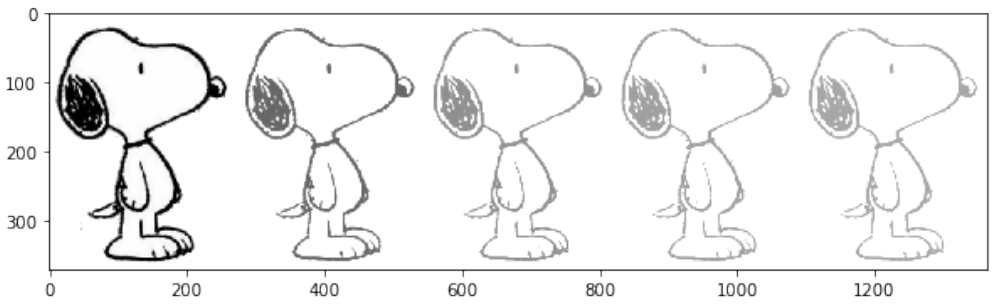
Contents

1 Image Handling in Python

```
plt.imshow(np.hstack((im_float, im_float)), cmap="gray", vmin=0, vmax=1);
```

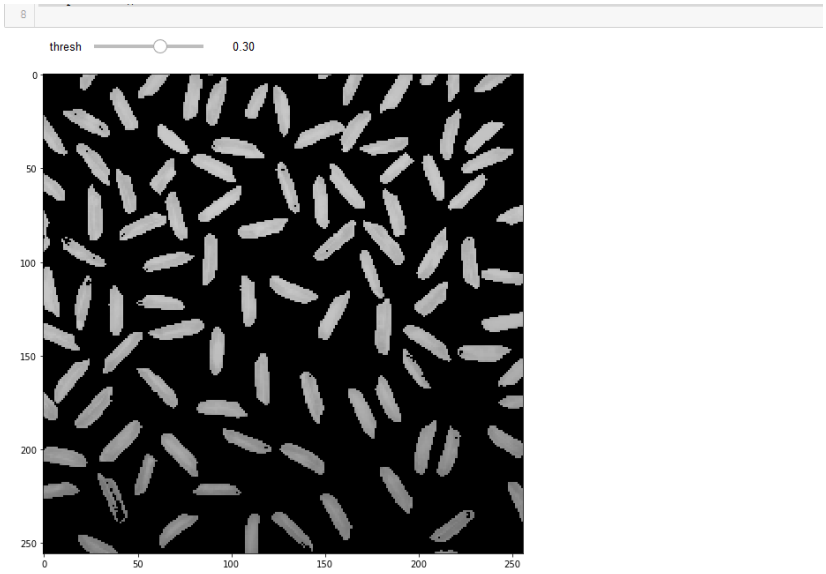
1.1 Fading an Image

```
im = skimage.io.imread("data/snoopy.png")
im = im/255 # tip: use floating point values

ims = []

im_temp = np.fliplr(im)
ims.append(im_temp)
for i in range(1,5):
  im_temp = im_temp + (0.4)**i
  ims.append(im_temp)

plt.imshow(np.hstack(ims),vmin=0, vmax=1, cmap="gray")
```



1.1.1 IPyWidgets

```
@interact(thresh=widgets.FloatSlider(min=0.0, max=0.5, step=0.01, value=0.2))
def threshold(thresh):
  im_thresh = []
  for line in im:
    im_thresh.append([pixel if pixel > np.min(line)
      + thresh else float(0) for pixel in line])
  plt.imshow(im_thresh, cmap="gray", vmin=0, vmax=1)
  plt.show()
```



2 Local Filtering and Edge Detection

2.1 Filtering

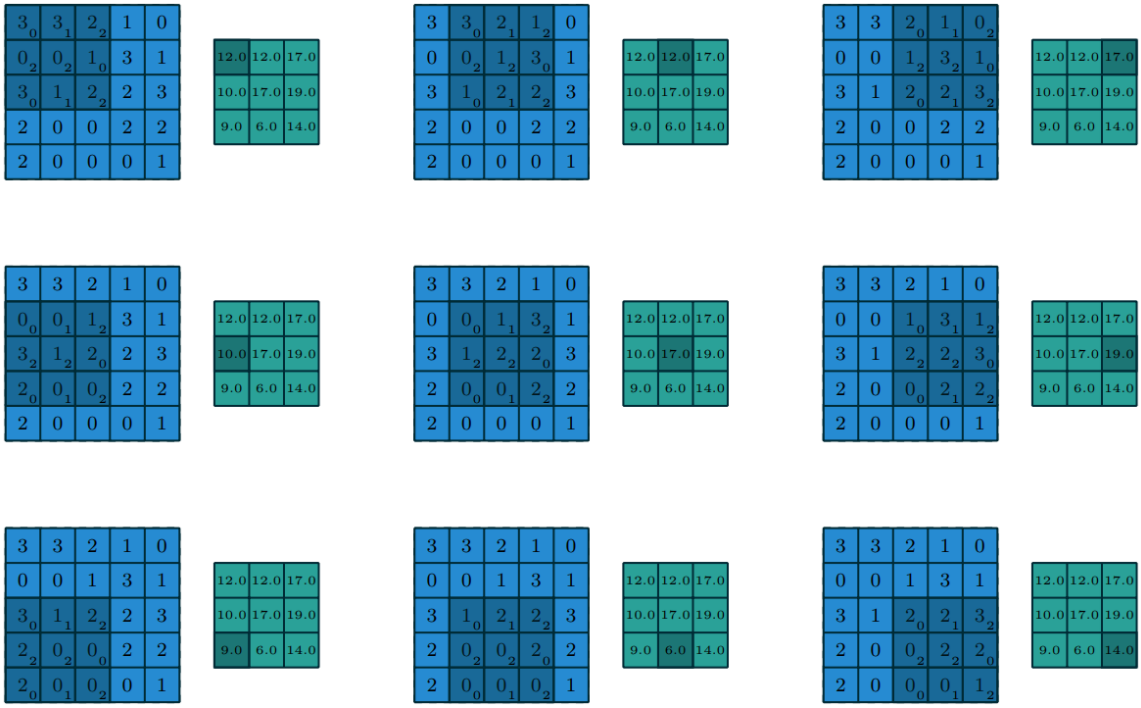
The naive approach of local filtering is taking just a moving average of the pixels in the neighbourhood.

2.2 Convolution

Convolve an input matrix (the input image) with a **kernel**, which is a matrix defining a weight for every element of the neighbourhood. The convolution can be considered a moving weighted average of the pixels.

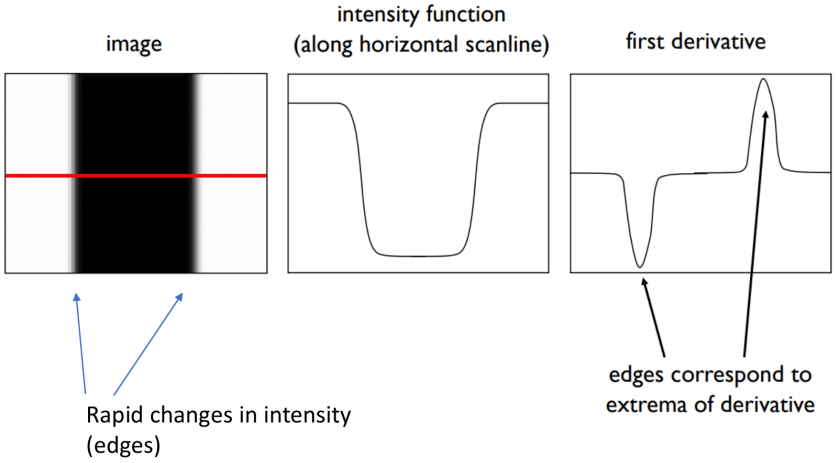
0	1	2
2	2	0
0	1	2

This kernel slides across the input matrix. At each location, the product between each element of the kernel and the input element it overlaps is computed and the results are summed up to obtain the output in the current location. In general, if the input matrix has i rows and the kernel has k rows, the output will have $i - k + 1$ rows, the same applies to columns.



2.3 Edge Detection

Although intuitive for humans to detect, edge detection was a hard problem in image processing. And edge is defined by a rapid change in the intensity, which can be exploited by calculating the derivative.



In two dimensions the derivative corresponds to the gradient $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$, which points from the edge towards the increase in intensity or the lighter side.

Gradient

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right] \tag{1}$$

Gradient Direction

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial x} / \frac{\partial f}{\partial y}\right) \tag{2}$$

Gradient Magnitude (strength of the edge)

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \tag{3}$$

But not all important edges have strong gradients, nor are all strong gradients important edges.

2.3.1 Computing the Gradient on an Image

Approximate Gradient in direction of $\frac{\partial f}{\partial x}$: Convolution with kernel

-1	1

Approximate Gradient in direction of $\frac{\partial f}{\partial y}$: Convolution with kernel

-1
1

The drawback of the gradient is, that it is very sensitive to noise:

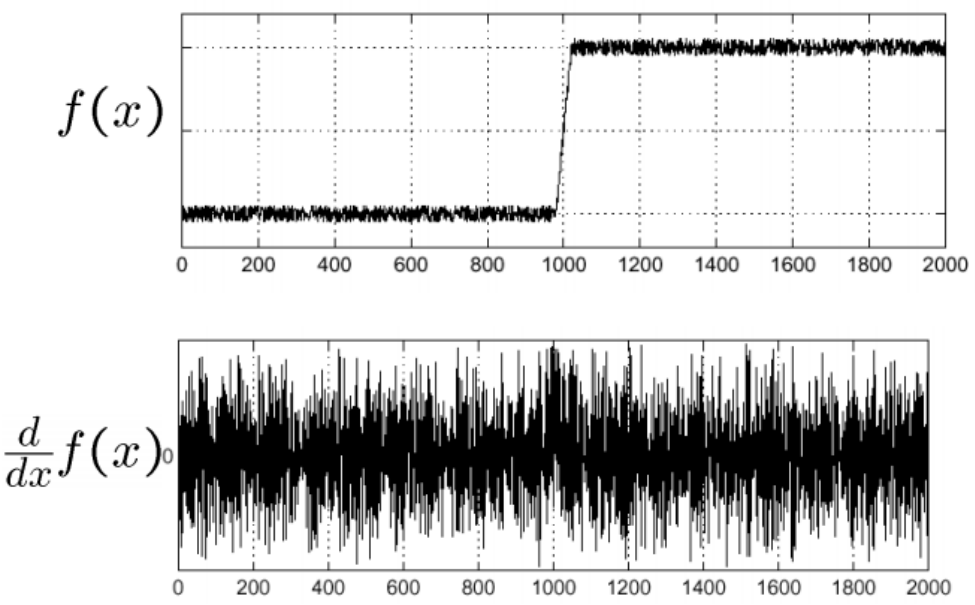


Figure 1: Gradient of a 1D function, where it is clear how it amplifies noise and may conceal a weak signal

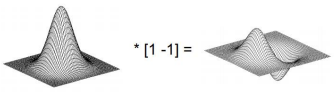
The solution to this is, to first smooth the function and then apply the gradient.

The larger the value of σ the more smoothing is applied and different kind of features can be identified.

large value of σ larger scale or stronger edges detected

smaller value of σ finer details detected

In practice a convolution with a derivative of Gaussian filter is calculated to compute the gradients.
This is equivalent to smoothing with a Gaussian and then taking the derivative.



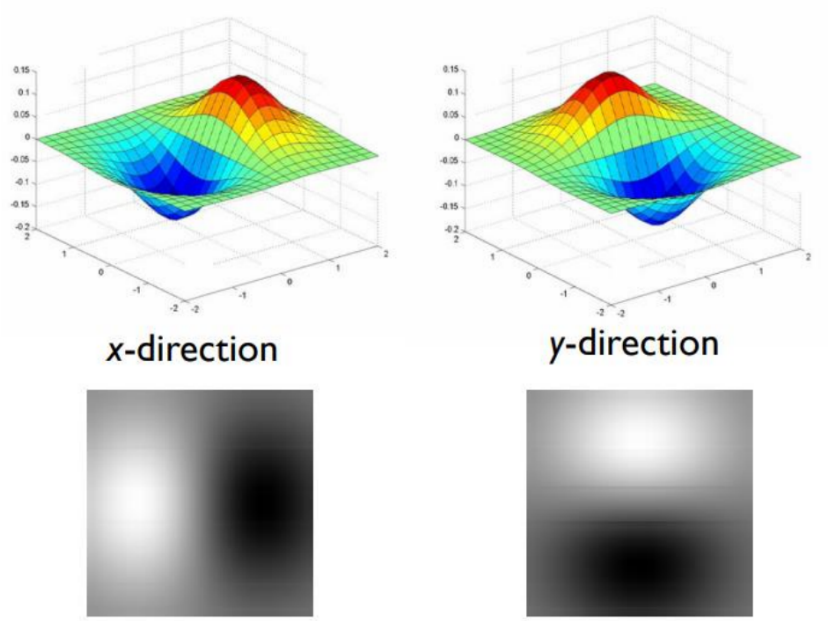


Figure 2: Derivative of Gaussian Filters

- A Gaussian smoothing filter removes high-frequency components
- Values of a Gaussian smoothing filter sum to one
- Derivative filters contain some negative value and values sum to zero
- Derivative filters yield large responses at points with high contrast

2.4 Canny Edge Detection

Same base approach but enhanced with edge thinning and hysteresis thresholding.[1]

1. Approximate gradients along axes by derivative of Gaussian filters
2. Compute gradient magnitude
3. Make edge one pixel wide, thinning through **non-maxima suppression** along perpendicular direction to the edge
4. Only keep strong edges through hysteresis thresholding

2.5 Hysteresis Thresholding

This stage decides which are all edges are really edges and which are not. For this, two threshold values, minVal and maxVal, are needed. Any edges with intensity gradient more than maxVal are sure to be edges and those below minVal are sure to be non-edges, and are thus discarded. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to "sure-edge" pixels, they are considered to be part of edges. Otherwise, they are also discarded.

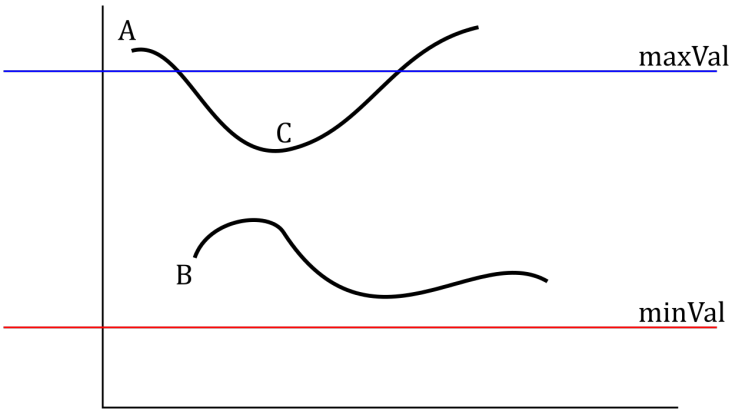


Figure 3: Hysteresis thresholding with A) sure edge, B) non-edge and C) a non-edge connected to a sure edge

3 Model Fitting and Line and Circle Detection

Edge detection is usually not enough as there are usually more than one line in an image and many edges that do not belong to a line. There might be noise in the edges belonging to a line which brings them out of alignment. Lines might also not be complete.

3.1 Model Fitting

Model fitting is an algorithm that finds or fits a high-level explanation or model that explains the observations well. In this case the edges are the observations and the model are one or more line.

[1] A good explanation with sample code can be found under OpenCV Canny Edge Detection Tutorial

3.2 Voting Algorithms

Voting algorithms are a general technique for decision methods.

1. Every feature casts votes for all models that are compatible with it
2. Choose models that accumulate a lot of votes

Clutter and noise will cast a lot of votes, but are inconsistent. Instead, features belonging to a model will concentrate a lot of votes for that model.

3.3 Hough Transform

1. Every **edge point** casts votes for **all lines that are compatible** with it
2. Choose **lines** that accumulated a lot of votes

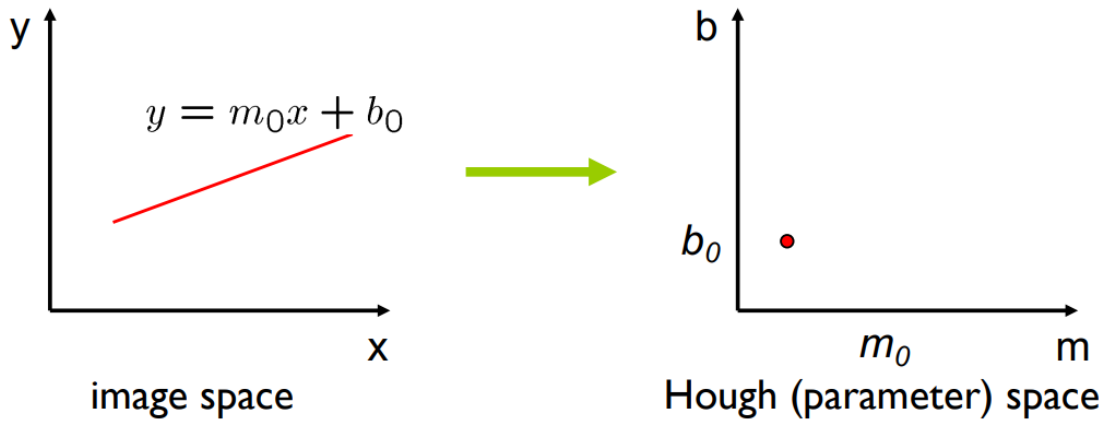


Figure 4: Line representation in Hough space

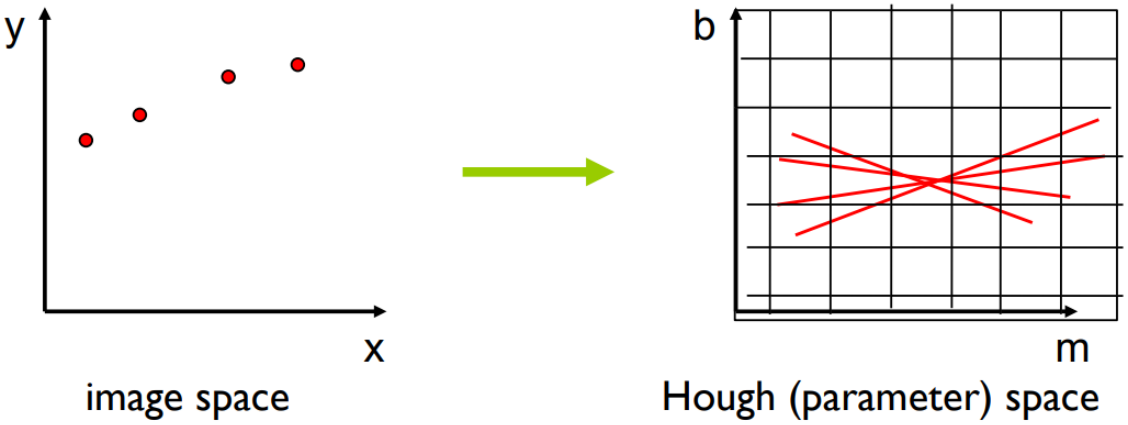


Figure 5: Points on a line in normal space almost intersect in Hough space

The problem with this approach is that

a. the parameter space (m, b) is **not bounded**

b. **vertical lines** can not be represented

3.4 Representation of Lines in Polar Coordinates

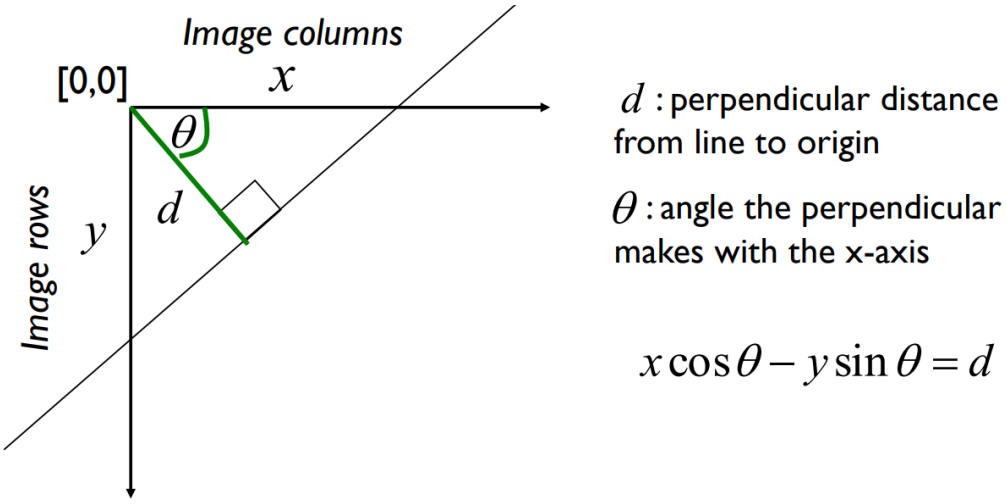


Figure 6: Line representation with polar parameters

This approach solves both the problem of unboundedness and representation. The parameter θ is bounded by $[0, 2\pi]$ and d by $[0, \text{imagewidth}]$.

For every point in normal space there will be a Sinusoid the polar Hough space.

3.5 Hough Transformation Algorithm

The algorithm transforms each edge point in the image to the Hough space, where an Accumulator array gets filled with votes. The bin size of the accumulator is an important hyperparameter, if the size is too small, there will be many weak peaks due to noise, if too large accuracy of locating a line drops and many votes from clutter might end up in the same bin. A solution is to keep the bin size small but also count votes from neighbours.

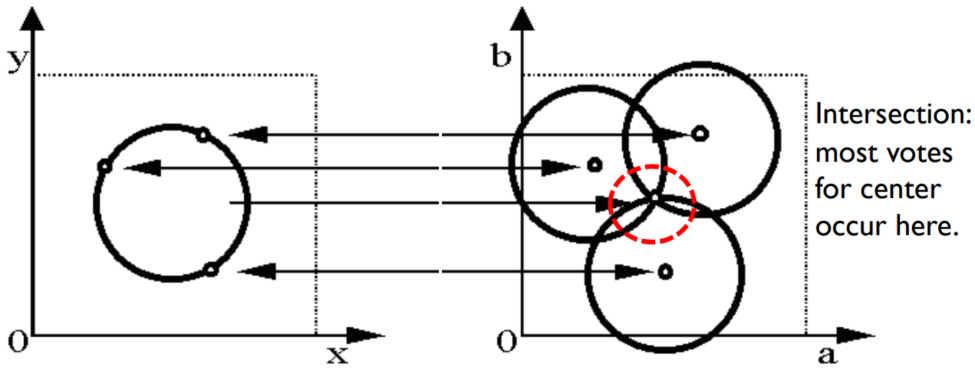
3.5.1 Voting Algorithm for Finding Lines

1. Every edge point casts votes for all circles that are compatible with it
2. We choose circles that accumulated a lot of votes

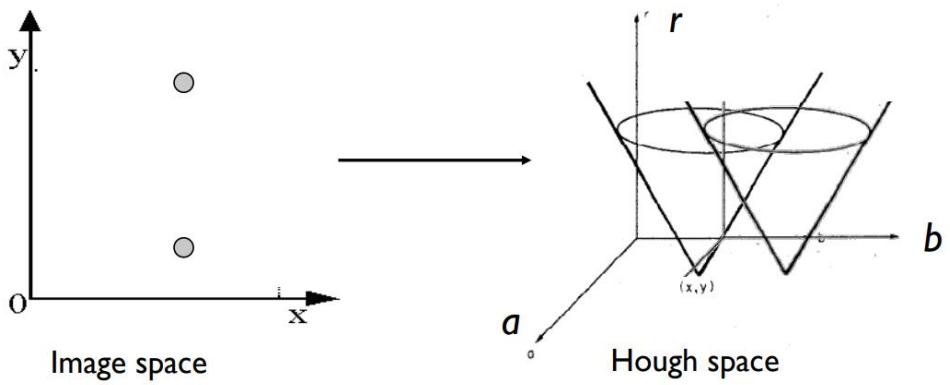
Theorem 1. *Parametrisation of a circle*

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

With centre (a, b) *and radius* r



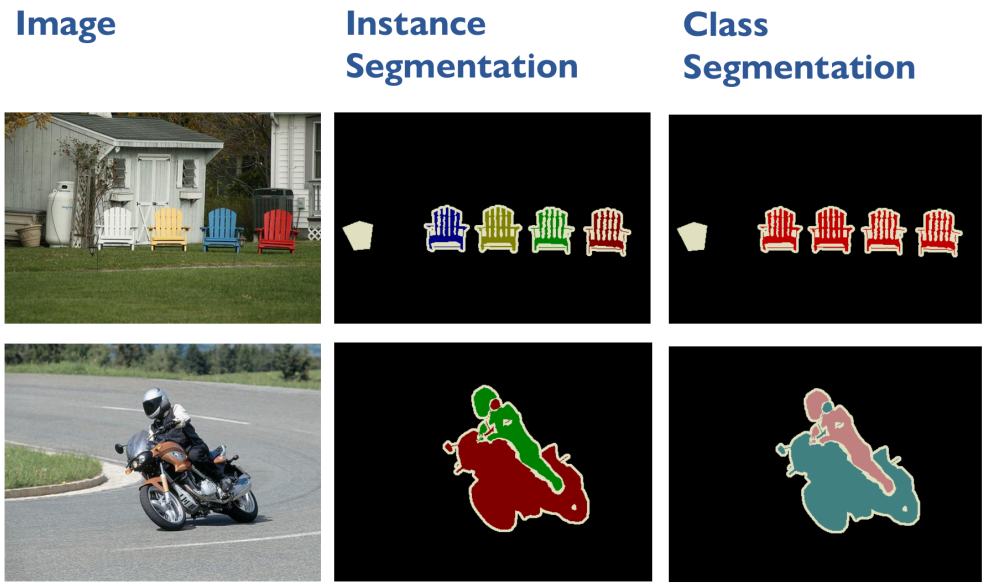
3.6 Hough Space for Circles With Unknown Radius



$$y = a \cdot x + \underline{b}$$

4 Semantic Segmentation

Identify pixels in images belonging to a specific class, sometimes additionally identifying pixels to belong to a specific instance of a class. It can thus be understood as supervised learning at the pixel level.



Non-semantic segmentation tries to find regions in images without any additional information like classes or similar. It is thus similar to unsupervised learning and often a bottom-up approach. Segmentation in general tries to find a compact representation of a scene in terms of regions which share common properties.

4.1 k-Means Clustering

Is not an optimal solution to cluster an image. Lloyds algorithm is one possibility to do clustering

- Initialise cluster centers (for example randomly)

1. Calculate which samples belong to which clusters
2. Calculate mean values of clusters as new centres

- Iterate until clusters don't change

Problems:

- Number of clusters must be given
- Cluster shapes are given (spherical)

4.2 Mean Shift Clustering

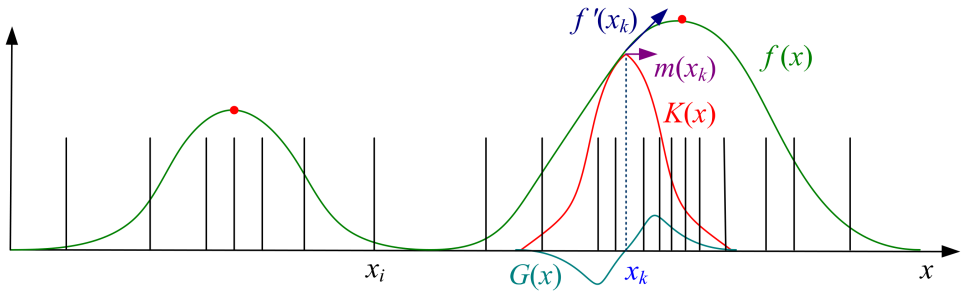
The shape of the clusters should be variable and not be determined by the method. Mean shift clustering can be achieved by the algorithm

1. Search for the maximum of the feature densitity from a given position in the image
2. Define cluster as the set of positions that converge to the same maximum

The problem to search for local maxima of the features without having a feature distribution available directly. The solution is to use a Kernel Density Estimation to describe the distribution, for example using the Gaussian kernel. Or using the Derivative of the Gaussian to get an approximation directly.

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

$$K\left(\frac{x - x_i}{h}\right) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-x_i)^2}{2h^2}}$$



4.2.1 Mean Shift Algorithm

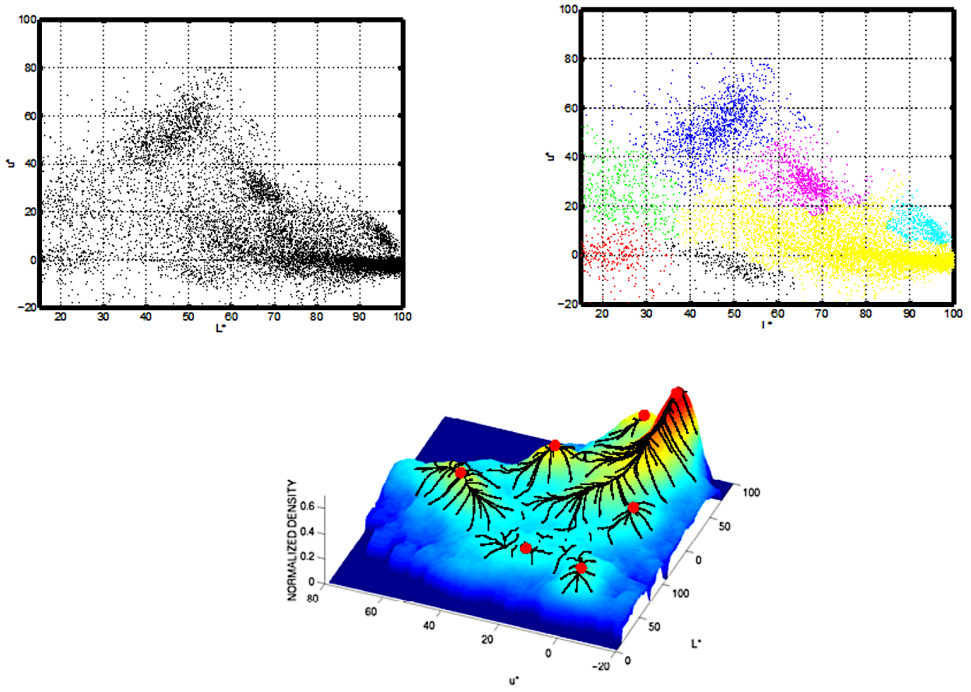
For all positions x in the image

- Calculate the mean $\overline{x}$ in the environment $N(x)$

$$\overline{x} = \frac{\sum_{x_i \in N(x)} K(x_i - x) x_i}{\sum_{x_i \in N(x)} K(x_i - x)}$$

- Set x to the mean $\overline{x}$
- Iterate until x does not change

Attraction basins are regions from which all trajectories lead to the same node.



4.3 Segmentation by Graph Cuts

Images are seen as graph with a node for every pixel with edges between nodes with an affinity weight for each edge. This weight measures similarity, for example inversely proportional to difference in colour and position.

Definition 4.1. Let $G = (V, E)$ be a graph. Each edge (u, v) has a weight $w(u, v)$ that represents the similarity between u and v. Cut G into two disjoint graphs with node sets A and B by removing all edges between those sets. Assign a value to the cut as follows

$$\text{cut}(A, B) = \sum_{u \in A, v \in B}^{w(u,v)}$$

Minimal value cuts often cuts too small groups. A better approach is to use normalised cuts:

$$\text{Ncut}(A, B) = \frac{\text{cut}(A, B)}{\text{assoc}(A, V)} + \frac{\text{cut}(A, B)}{\text{assoc}(B, V)}$$

$$\text{assoc}(A, V) = \sum_{u \in A, t \in V}^{w(u,t)}$$

where V is the set of all vertices.

4.4 Superpixels

The problem is that images contain many pixels and it is thus inefficient for graph calculations. The solution is to calculate *superpixels* by grouping similar pixels first. This is cheap but leads to over-segmentation. As soon as the superpixels are calculated, normalised graph cuts can be applied.

4.5 Features

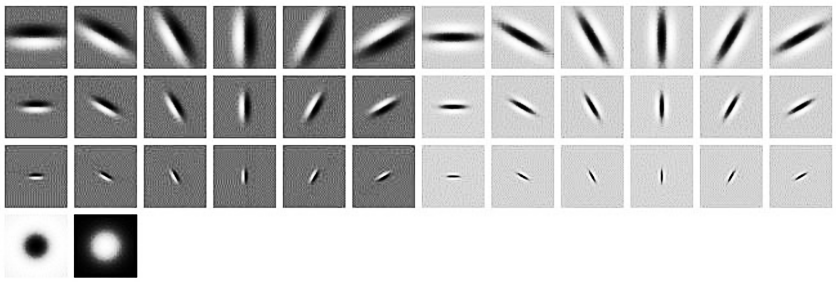
Usually, Raw Input pixel values are not used as input features, as this makes the input space too big. Instead a feature vector is calculated for each pixel or set of pixels. These feature calculate certain properties of an image or image region. Besides classification tasks, these vectors are also used to find subregion or image stitching, and more.

4.5.1 Feature Properties

Features should be invariant to

- Translation
- Rotation
- Scaling
- Illumination, colour changes
- Transformation (Skewing)

4.5.2 Filter Banks



- RFS Filters: edge and line filters at six orientations and three scales, Gaussian and Laplacian of Gaussian
- MR8 Filters: Maxima of RFS Filters at each scale

4.5.3 Texture Characteristics

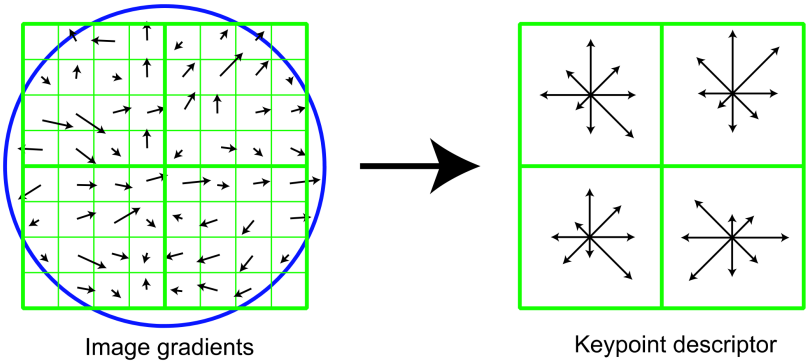
Textons use the texture descriptor as vectors of filter bank outputs. Textons found by clustering are called a dictionary generation. For comparison, use the similarity between regions in the Texton histograms directly.

4.5.4 Grey Level Co-occurence Matrices (GLCMs)

Measurement of how often a specific combination of a colour value between a pixel and its neighbours occur. This results in a 256×256 matrix for the 256 grey values. Additional features like entropy, energy, homogeneity, contrast or dissimilarity can be calculated from the matrix.

4.5.5 Scale Invariant Feature Transform (SIFT)

Popular feature descriptor for various tasks such as object matching, image stitching and more. It is a combination of detector and descriptor. The descriptor uses the histogram of the gradient. SIFT results in a 128-dimensional vector.

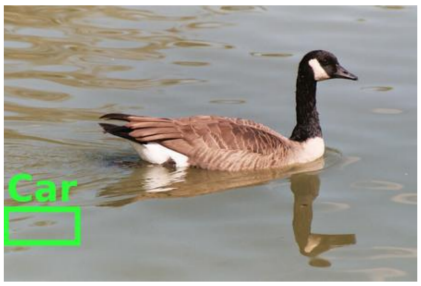


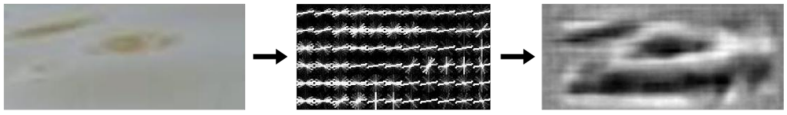
4.5.6 Histogram of Oriented Gradients (HOG)

Computes the gradient images in x and y, and divides the image into cells. Compute histogram of gradient orientation in cell, normalise and flatten it into a feature vector.

4.5.7 HOGles

"Invert" the HOG features to simulate the image that generates it. This is a helpful tool to understand bad detections.





5 CNN and Machine Learning in General

For this chapter I'd like to refer you to my summaries in Deep Learning and Machine Learning where the desired information is already written down in much more detail.

6 Finding Multiple Objects

The problem at the core is how to scan many positions efficiently for possible faces. Possible approaches are

- Features
- Classifier
- Sliding Window
- Haar Features
- Boosting
- Cascaded Classifiers

6.1 Boosting

For an input training data with weights, at each step $t \in 1..T$ train a weak classifier, let it classify the training data and increase the weight on incorrectly classified samples. Use the weighted combination of all weak classifiers for the final, strong classifier. The final strong classifier $F(x)$ can be understood as a weighted, linear combination of the weak classifiers.

$$F(x) = \alpha_1 f_1(x) + \alpha_2 f_2(x) + \alpha_2 f_2(x) + \dots$$

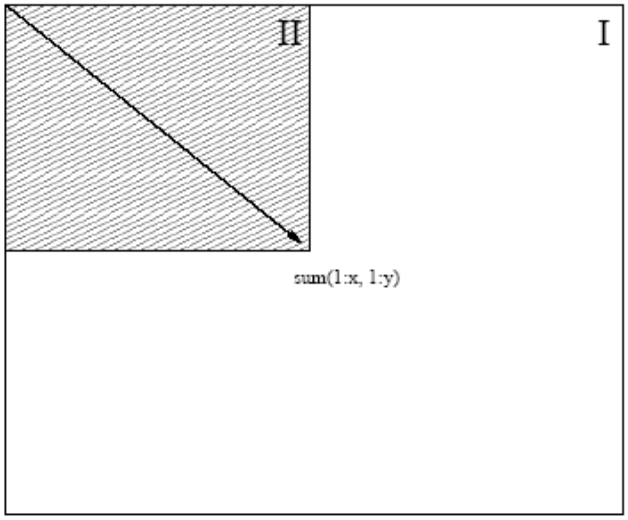
Weak classifier used with f feature, θ threshold and p_j parity

$$h_j(\theta) = \begin{cases} 1 & p_j f_j < p_j \theta_j \\ 0 & \text{otherwise} \end{cases}$$

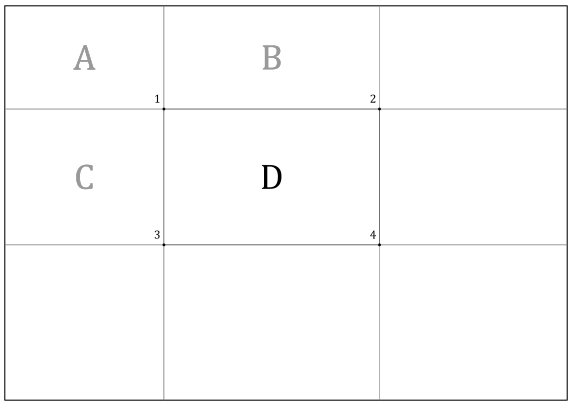
6.2 Haar Features

Filter responses, that is convolutions, are computationally intensive. The idea of haar features is to only use simple filters of different sizes and shapes with only -1 or 1 in the masks. Calculate the features on sliding sub windows (for example 24×24 pixels) and take all possible combinations of filters as feature candidates. This can be done in a fast manner using integral images, where each pixel of the integral image contains the sum of all values in the rectangle spanned to the upper left corner (see image 7a). The sum of pixel D in figure 7b is $4 + 1 - (2 + 3)$ or in terms of the named rectangles

$$D = (A + B + C + D) + A - ((A + B) + (A + C))$$



(a) Visualisation of an integral image sum

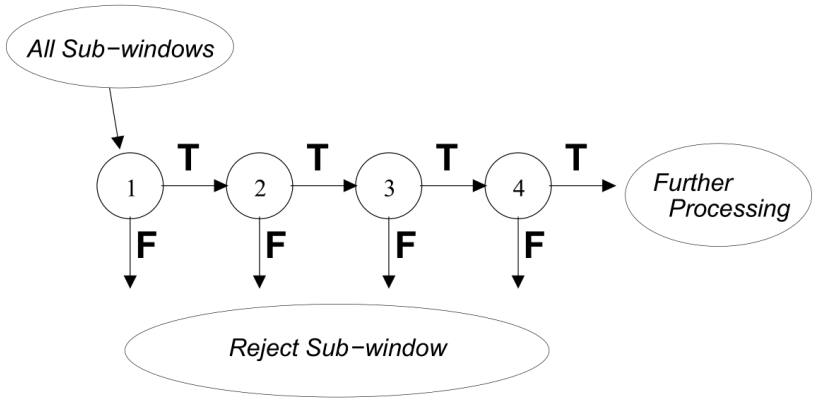


(b) Calculating Haar features using an integral image

The humongous size of the features per sub window can be evaluated by treating each feature as a weak classifier and training important features using adaboost.

6.3 Face Detection

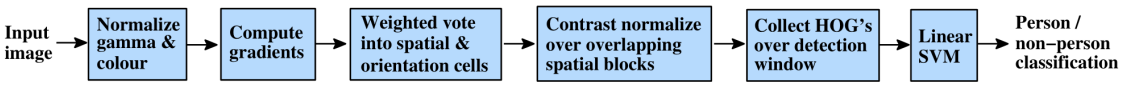
The problem is that there are thousands of possible position or scale combinations in the image which need to be evaluated, while true faces are rare in the image. The idea is thus to dismiss the regions without a face efficiently. The goal for a classifier is to have a high *true* positive rate (85%-95%) and a low *false* positive rate ($10^{-5} - 10^{-6}$). Use this classifier in a cascade which multiplies false and true positive rates. False results are rejected early in the processing.



6.4 HOG for Human Detection

Detection Pipeline

- Calculate HOG Descriptors
- Collect HOGs in Detection Window
- Use Linear SVM for Person or Non-Person Classification



6.5 OverFeat

Combine Classification, Localisation and Detection.

Classification

- CNN
- Sliding Windows, apply at every possible pixel and at multiple scales
- Subsampling factor of 12

Localisation

- Generate Bounding Box Prediction, as a Regression Problem
- Uses the same first five layers in the CNN

Detection

- Combine Classification and Localisation
- Merge and accumulate bounding boxes

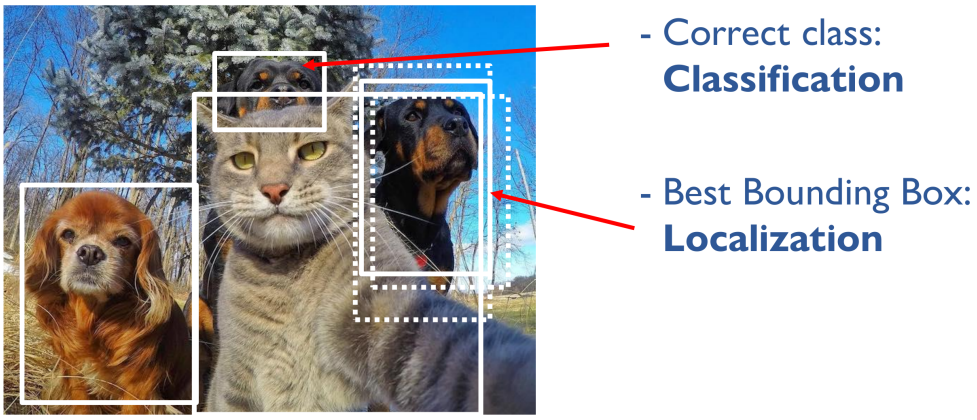
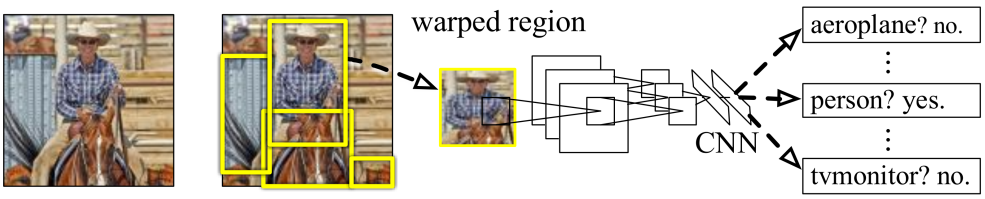


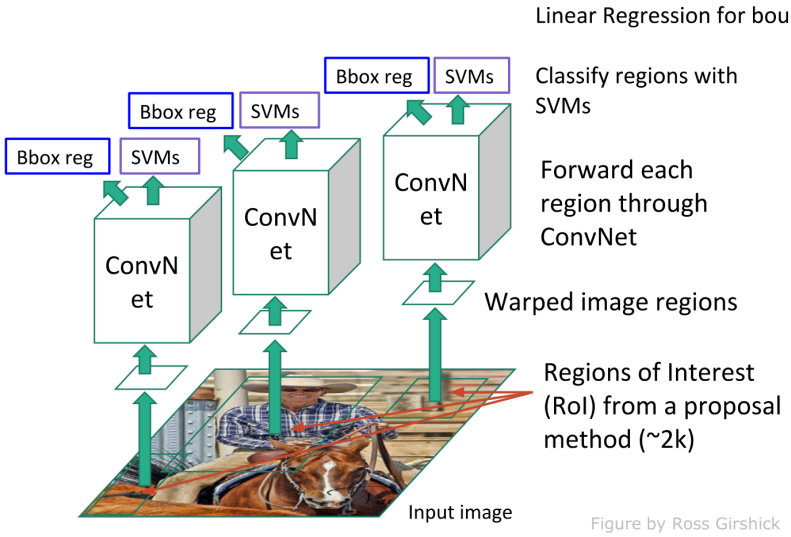
Figure 8: Classification and Localisation visualised

Sliding windows have the drawback of the huge numbers of different positions and sizes that need to be evaluated. A better idea would be to find image regions that are likely to contain an object, which are called region proposals.

6.6 R-CNN

Extract region proposals, compute CNN features on these, classify the regions using SVM.





6.6.1 Feature Extraction

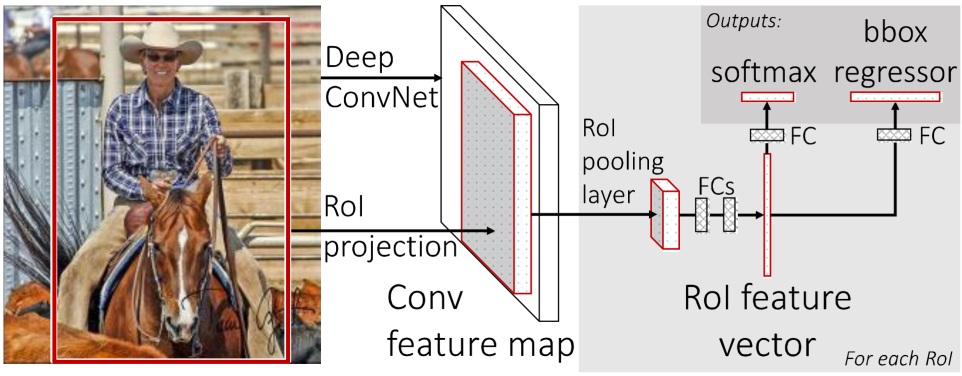
- Proposal regions are warped to 227×227 pixel images
- 4096 Features from 227×227 RGB Image using five Convolutional and two fully connected layers from AlexNet Architecture
- Supervised pre-training on the ILSVRC2012 classification data set
- Domain-specific fine-tuning is done by continuing training on warped region proposals

6.6.2 Classifier

Use one linear SVM per class (one true class versus all others), consider regions with overlap of 30% a positive example and treat the bounding box position as regression problem.

6.6.3 FAST R-CNN

Replace SVM of R-CNN with fully connected neural network for classification of objects and refined bounding boxes



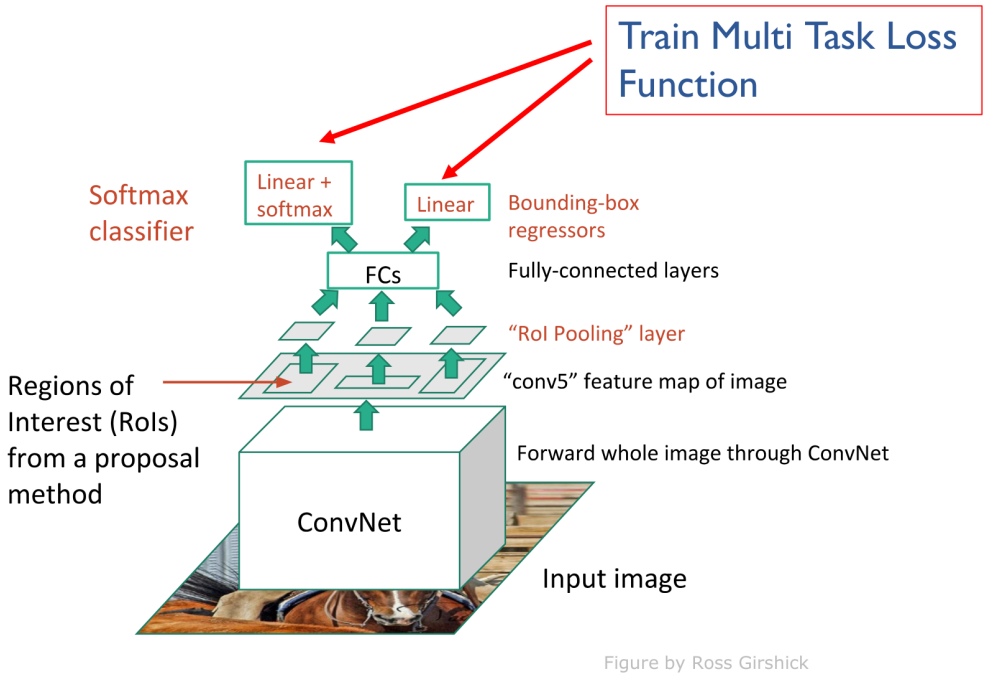


Figure by Ross Girshick

6.7 You Only Look Once (YOLO)

YOLO is a Single Neural Network that predicts bounding boxes and classification probabilities, which is very Fast and has good classification but not so good localization properties. The approach is

- Divide Image into $S \times S$ grid
- Each grid cell predicts B bounding boxes with confidence scores $(x, z, w, h, \text{score})$
- Each grid cell predicts class probabilities (independent of bounding boxes)
- Multiply confidence score and probabilities at test time

6.7.1 Improvements

- Add Batch Normalisation
- Use higher resolution (448×448)
- Use Anchor Bounding Boxes from k-Means Analysis on the training data
- Use new base net (darknet19 and darknet 58)

7 Generating Images

7.1 Conditional GANs for Image to Image Transfers

A GAN learns the mapping from random noise z to output image y

$$G : z \to y$$

A conditional GAN should learn the mapping from input image x and noise z to output image y

$$G : \{x, z\} \to y$$

In practice adding noise as input proves ineffective, as the network learns to ignore it. One solution is to use dropout as noise, both in training and during testing.

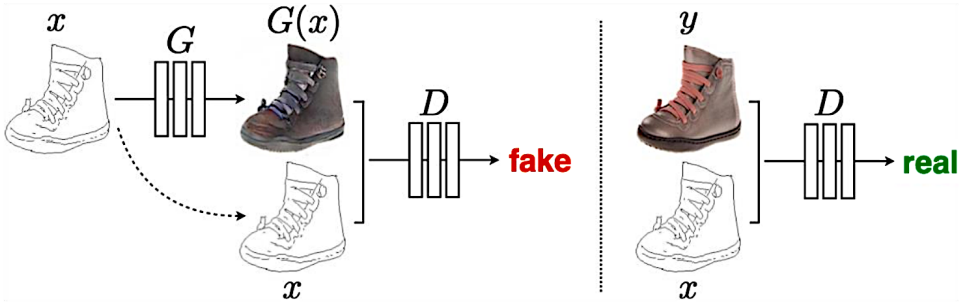


Figure 9: In conditional GANs both the Generator G and the Discriminator D input the edge map

7.1.1 Conditional GANs

Generator Architectures

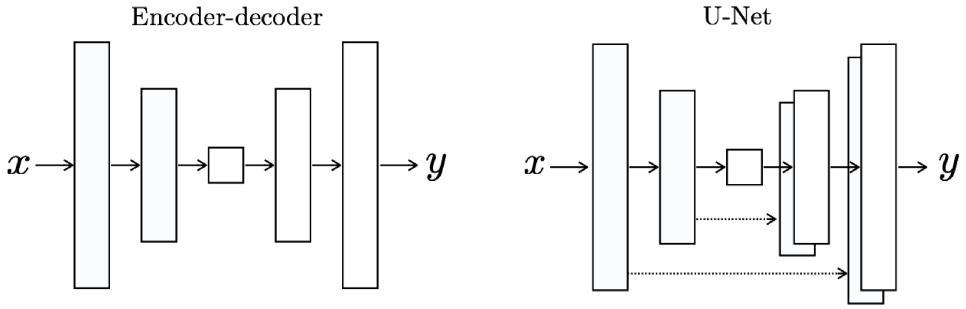


Figure 10: U-Net architectures with connections between the layers seem to work better

7.2 Neural Texture Synthesis

Given a low resolution texture image can a larger image be generated? For this there are two main approaches

- Replicate pixels and patches using a suitable algorithm
- Find a model of the texture and generate new textures from the model

7.2.1 Texture Synthesis Using CNNs

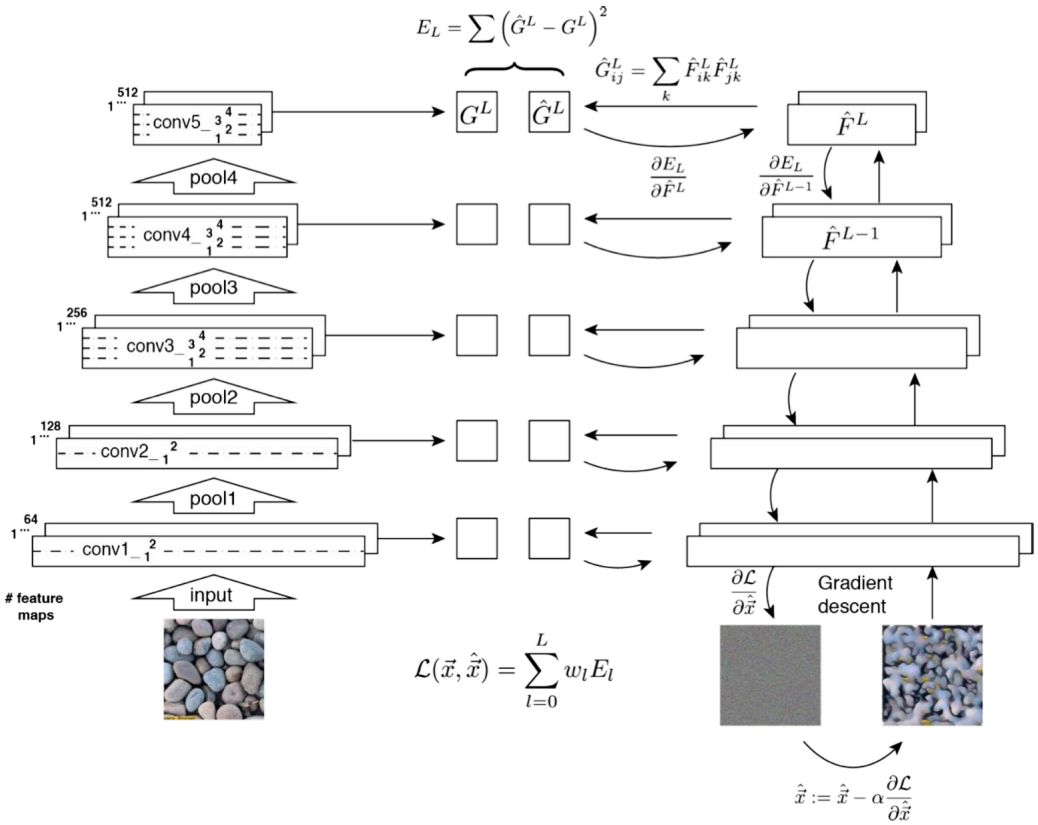
Pass image$x$ through the network, the activations from each layer are then the feature maps $F^l \in \mathbb{R}^{N \times M}$ for the texture. Calculate the Gram matrix to compute correlations between feature maps (on the same layers)

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}$$

These Gram matrices describe the texture model.

7.2.2 Generating Textures

1. Start with noise image
2. Use gradient descend to find another image that matches the Gram-matrix representation of the original image
3. Optimize my minimizing the mean-squared distance between the matrices from the original and generated image



7.3 Neural Style Transfer

7.3.1 Content Representation

Each input image generates filter responses $F^l \in \mathbb{R}^{N \times M}$ at each layer. Run gradient descend on noise images to find an image that generates the same response. If $F^l \in \mathbb{R}^{N \times M}$ is the response from the original image and $P^l \in \mathbb{R}^{N \times M}$ the response from the generated image, the loss is

$$Ł_{\text{content}}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} (F^l_{ij} - P^l_{ij})^2$$

Later layers in the network define the content

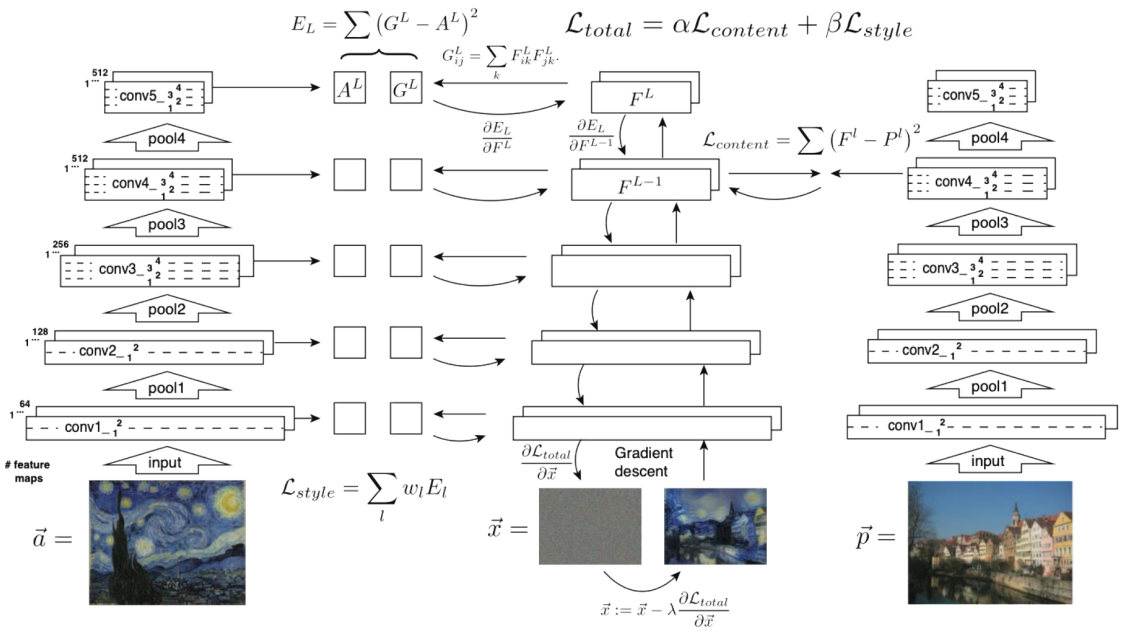
7.3.2 Style Representation

Each input image generates filter responses $F^l \in \mathbb{R}^{N \times M}$ at each layer, calculate Gram matrices from the features. If A^l is the matrix from the original image and G^l the matrix from the generated image, the loss for one layer is

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G^l_{ij} - A^l_{ij})^2$$

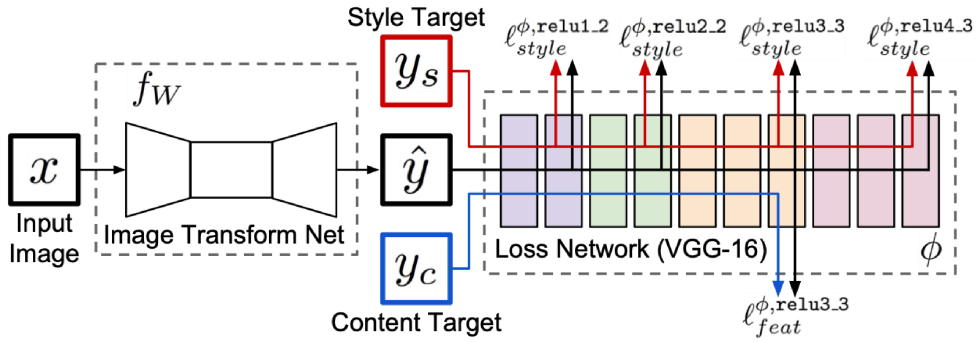
and including several layers

$$Ł_{\text{style}}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l$$



7.3.3 Fast Style Transfer

The problem is that style transfer is slow as it requires many passes through VNN, the solution is to train a neural network to perform style transfer. Train feedforward network for each style, use pretrained CNN with losses as before, and After training, stylize image with single forward pass.

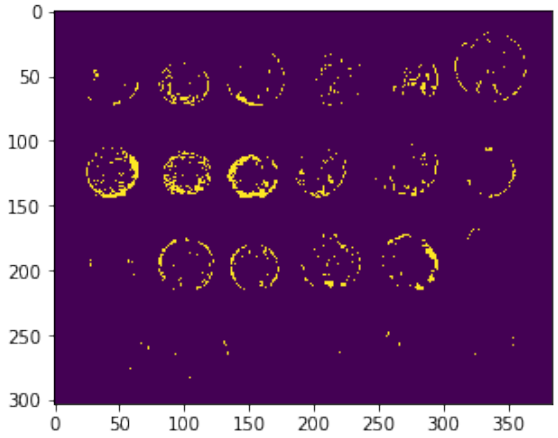


8 Python Code

8.1 Binarisation and Filtering

8.1.1 Binary Mask

```
mask = im > 0.8 # bright pixels will be True in the mask, others will be False
plt.imshow(mask)
```



8.1.2 Connected Component Analysis

```
import skimage.measure
labels = skimage.measure.label(im > 0.5)
print(labels.shape, labels.dtype)
print("Unique values in labels:", np.unique(labels))
> (303, 384) int64
> Unique values in labels: [  0   1   2   3   4   5   6   7   8   9  10
> 11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28
> 29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46
> 47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64
> 65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82
> 83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
> 101 102 103 104 105 106 107  108 109 110 111 112 113 114 115 116 117 118 119]

regions = skimage.measure.regionprops(labels)

large_regions = [r for r in regions if r.area > 100]

# equivalent to
large_regions = []
for r in regions:
        if r.area > 100:
                large_regions.append(r)

print(f"There are {len(large_regions)} large regions")
> There are 25 large regions
```

The `regionprops` (documentation) function can compute properties for each connected component. Some important properties are the following:

`area : int` Number of pixels of the region.

`bbox : tuple` Bounding box (`min_row, min_col, max_row, max_col`). Pixels belonging to the bounding box are in the half-open interval `[min_row; max_row)` and [min_col; max_col).

`centroid : array` Centroid coordinate tuple (row, col).

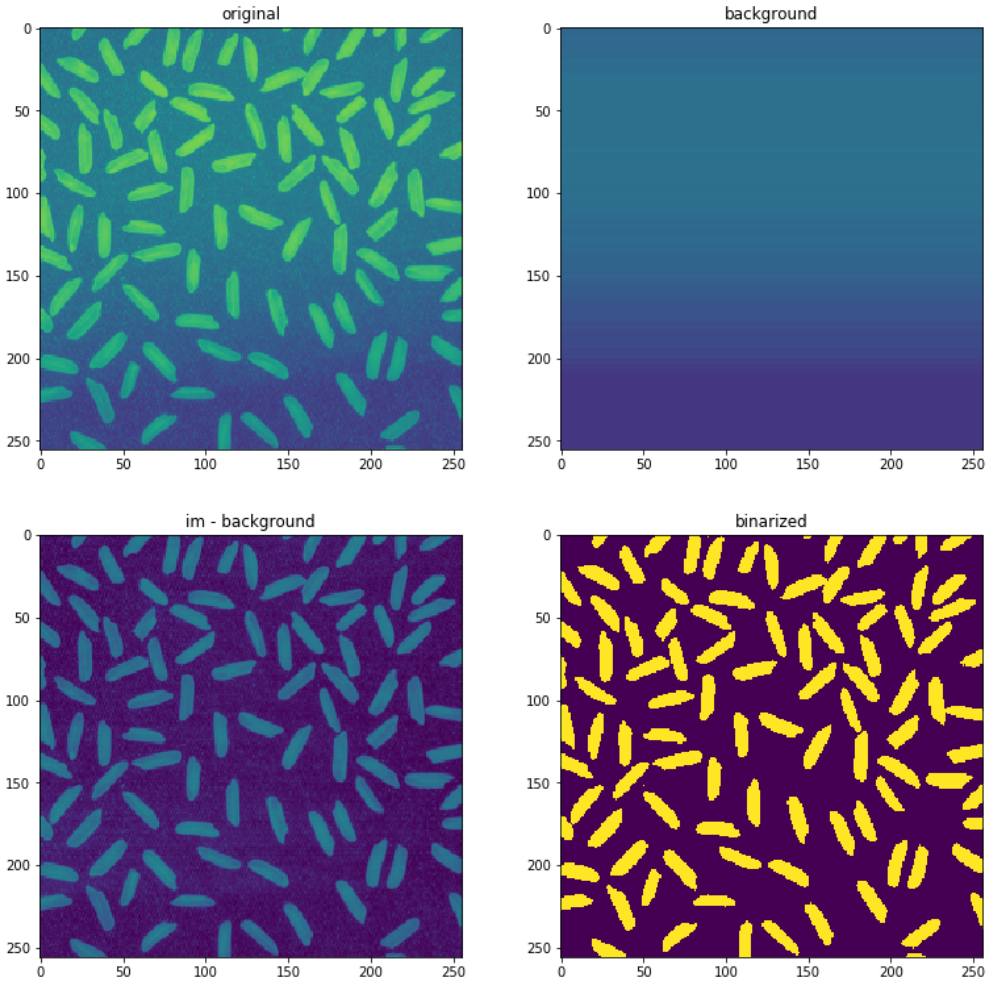
`convex_area : int` Number of pixels of convex hull image, which is the smallest convex polygon that encloses the region.

`label : int` The label in the labeled input image.

```
import matplotlib.patches as patches
fig, ax = plt.subplots()
ax.imshow(im, cmap="gray")
for r in large_regions:
```

```
        (min_row, min_col, max_row, max_col) = r.bbox
        width = max_col - min_col
        height = max_row - min_row
        rect = patches.Rectangle((min_col,min_row),width,height,
        linewidth=1,edgecolor='b',facecolor='none')
        ax.add_patch(rect)
```

```
background = (np.min(im,axis=1,keepdims=True) * np.ones((1,im.shape[1])))

fig, axs = plt.subplots(ncols=2, nrows=2, figsize = (13,13))
axs[0,0].imshow(im, vmin=0, vmax=1)
axs[0,0].set(title="original")
axs[0,1].imshow(background, vmin=0, vmax=1)
axs[0,1].set(title="background")
axs[1,0].imshow(im - background, vmin=0, vmax=1)
axs[1,0].set(title="im - background")
mask = (im - background) > 0.25
axs[1,1].imshow(mask, vmin=0, vmax=1)
axs[1,1].set(title="binarized")
```



8.2 Model Fitting Hough Lines

```
import skimage.feature
import skimage.transform.hough_transform as ht
import matplotlib.pyplot as plt
import numpy as np
```

```
import skimage.data
import skimage.feature

im = skimage.data.camera()
imedges = skimage.feature.canny(im)

lines = ht.probabilistic_hough_line(imedges, threshold=100)
len(lines)

# Draw detected lines
fig,ax = plt.subplots(figsize=(10,10))

ax.imshow(im,cmap="gray")
for ((x0,y0),(x1,y1)) in lines:
ax.plot([x0,x1],[y0,y1],'b-')
```

8.3 Superpixel Segmentation

```
# import the necessary packages
from skimage.segmentation import slic
from skimage.segmentation import mark_boundaries
from skimage.util import img_as_float
from skimage import io
import matplotlib.pyplot as plt
import argparse

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required = True, help = "Path to the image")
args = vars(ap.parse_args())

# load the image and convert it to a floating point data type
image = img_as_float(io.imread(args["image"]))

# loop over the number of segments
for numSegments in (100, 200, 300):

# apply SLIC and extract (approximately) the supplied number
# of segments
segments = slic(image, n_segments = numSegments, sigma = 5)

# show the output of SLIC
fig = plt.figure("Superpixels -- %d segments" % (numSegments))
ax = fig.add_subplot(1, 1, 1)
ax.imshow(mark_boundaries(image, segments))
plt.axis("off")

# show the plots
plt.show()
```