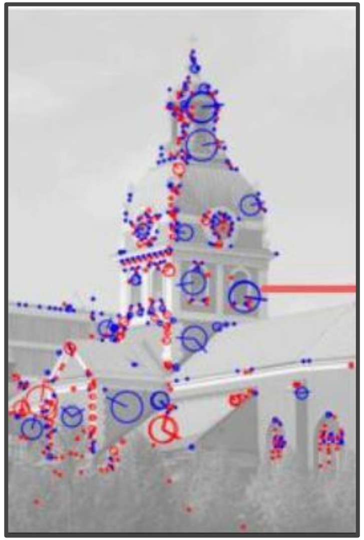


Computer Vision

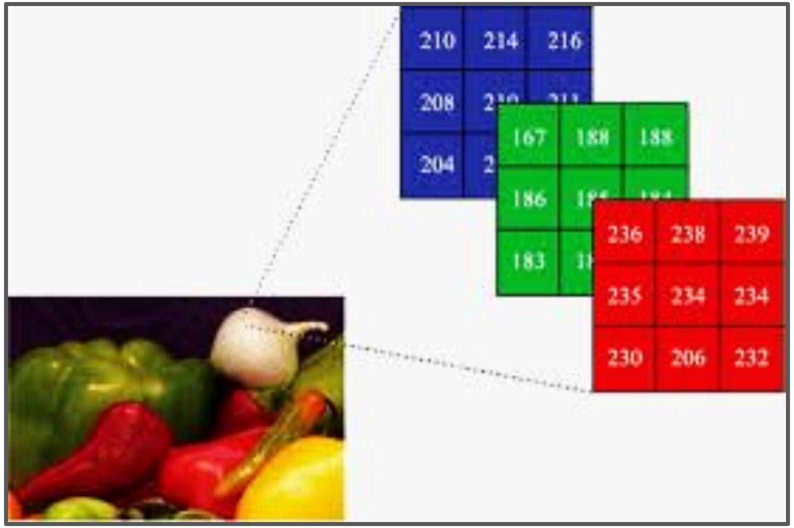
- Convolutions, Filters, Feature extraction
- Feature matching, SIFT
- Deep Learning (Convolutional Neural Networks)

Image Representation

- An *(m x n)*-pixel image is represented as an *(m x n x 3)* RGB matrix on a computer

- **R:** Intensities of Red
  G: Intensities of Green
  B: Intensities of Blue

- A grayscale image is represented as a single matrix where each value corresponds to the Pixel Intensity



Convolution Operation

$$\sum_i \sum_j h[x - i, y - j] f[i, j]$$

2*1 + 4*2 + 9*3 + 2*(-4) + 1*7 + 4*4 + 1*2 + 1*(-5) + 2*1 = 51

2	4	9	1	4
2	1	4	4	6
1	1	2	9	2
7	3	5	1	3
2	3	4	8	5

Image

X

1	2	3
-4	7	4
2	-5	1

Filter / Kernel

=

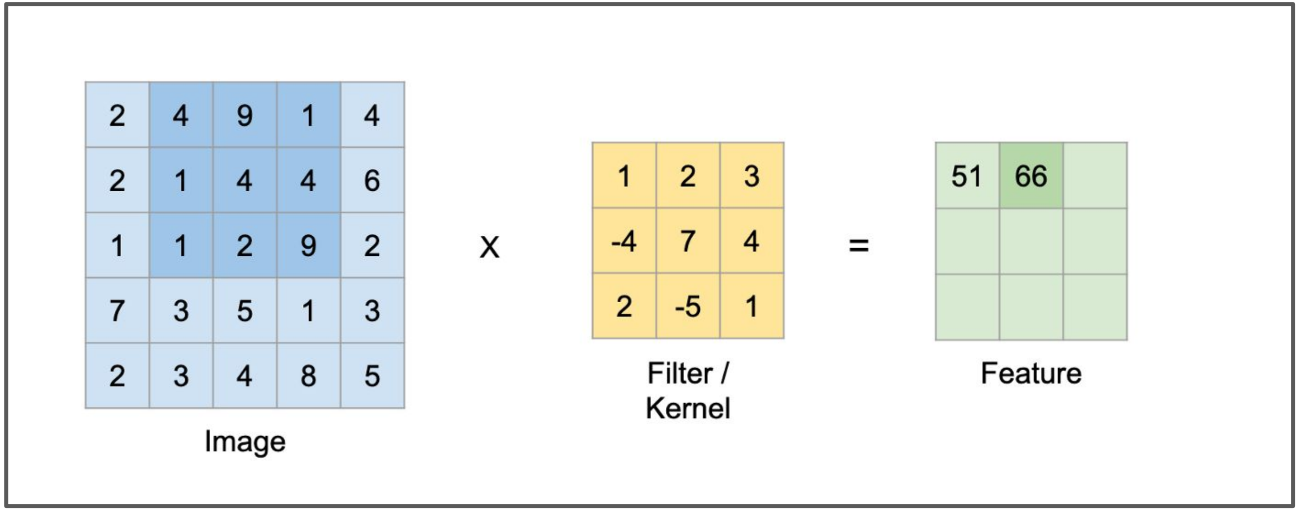
51		

Feature

Convolution Operation

$$\sum_i \sum_j h[x-i, y-j] f[i,j]$$

$4*1 + 9*2 + 1*3 + 1*(-4) + 4*7 + 4*4 + 1*2 + 2*(-5) + 9*1 = 66$



Convolution Operation

$$\sum_i \sum_j h[x-i, y-j] f[i,j]$$

2*1 + 9*2 + 2*3 + 5*(-4) + 1*7 + 3*4 + 4*2 + 8*(-5) + 5*1 = -2

2	4	9	1	4
2	1	4	4	6
1	1	2	9	2
7	3	5	1	3
2	3	4	8	5

Image

X

1	2	3
-4	7	4
2	-5	1

Filter / Kernel

=

51	66	20
31	49	101
15	53	-2

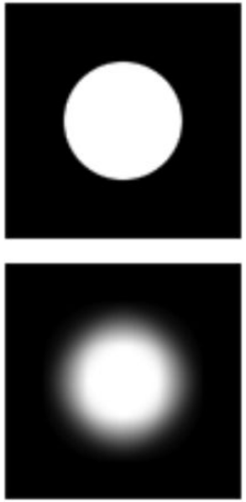
Feature

Blurring Operation - Average Filter

Idea - Adding a component of the intensity of the surrounding pixels, to every pixel

Kernel -
(Example - size:3)

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\1 & 1 & 1\\1 & 1 & 1\end{bmatrix}$$



100	100	100	0	0
100	100	100	0	0
100	100	100	0	0
100	100	100	0	0
100	100	100	0	0

Sharp edge

100	100	100	0	0
100	100	67	33	0
100	100	67	33	0
100	100	67	33	0
100	100	100	0	0

Gradient

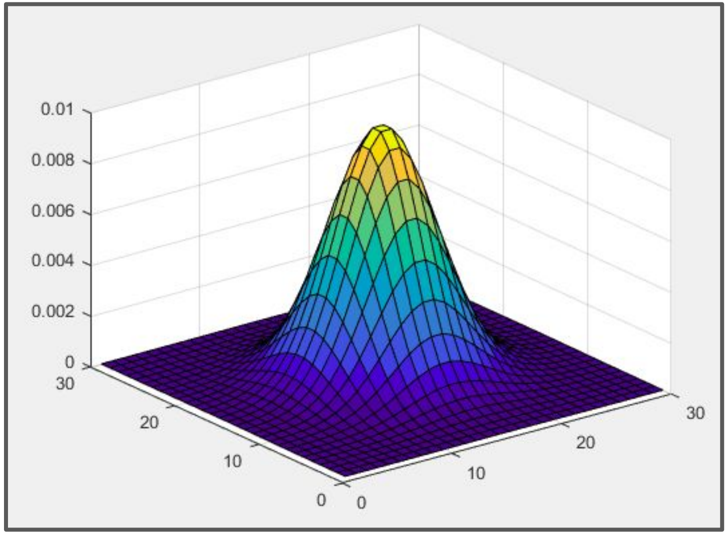
Gaussian Blur

Idea - Pixel values ‘nearby’ are more important than the ones far away

Hence, **more weight** needs to be given to surrounding pixel values in a kernel

Can be modeled by a **2D Gaussian** distribution

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



Gaussian Approximation

1/16

1	2	1
2	4	2
1	2	1

1/273

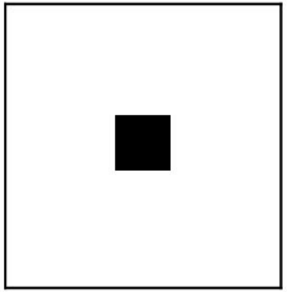
1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

(5 x 5) Gaussian kernel

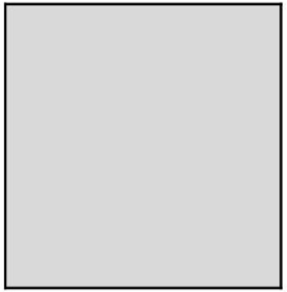
1/1003

0	0	1	2	1	0	0
0	3	13	22	13	3	0
1	13	59	97	59	13	1
2	22	97	159	97	22	2
1	13	59	97	59	13	1
0	3	13	22	13	3	0
0	0	1	2	1	0	0

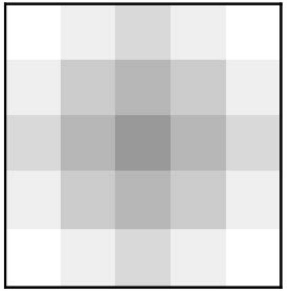
(7 x 7) Gaussian kernel



Image

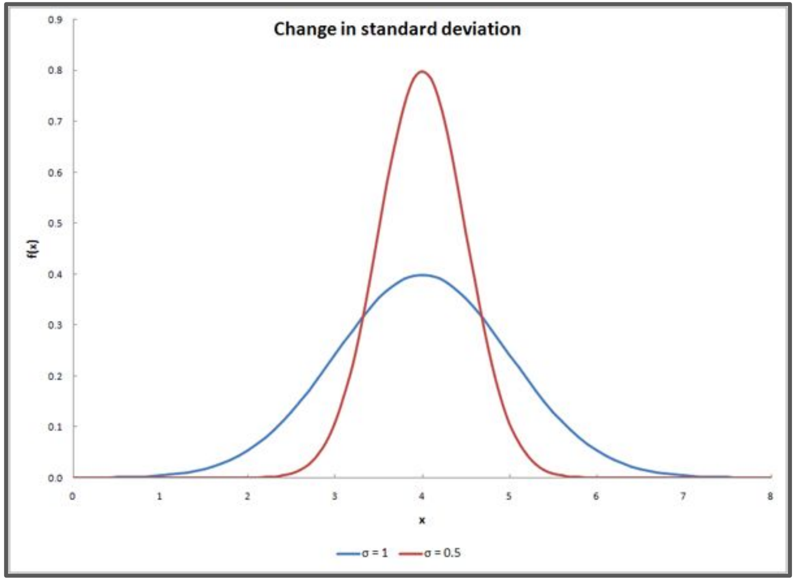


Average Blur



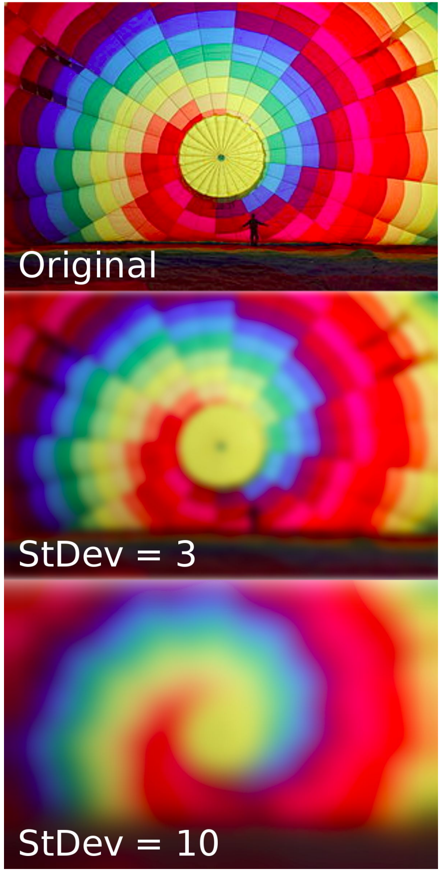
Gaussian Blur

Effect of Kernel Size



Kernel Size ↔ 'Spread' of Blur

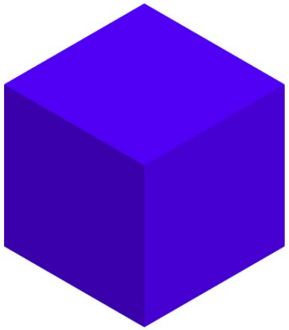
Larger spread⇒ More weight to surrounding pixels & less weight to central pixel
i.e. Larger **Standard deviation**



Edge Detection

Usual Method - How do humans do it?

What are the edges in this image? How do you know?



Answer: Areas of *sudden change of colour*

Formally:

Let ***f(i, j)* = *value at pixel in row i and column j.***

Edge of a particular direction ***d***

⇒ Sudden change in the value of ***f*** while moving along its perpendicular

⇒ ***(δf / δd)*** is large

Estimating derivatives for discrete functions

Using the Taylor Series,

1. $$f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + \frac{1}{3!}h^3 f'''(x) + O(h^4)$$

2. $$f(x-h) = f(x) - hf'(x) + \frac{1}{2}h^2 f''(x) - \frac{1}{3!}h^3 f'''(x) + O(h^4)$$

Ignoring higher order terms (>=2),

$$\boxed{\frac{f(x+h) - f(x-h)}{2h} = f'(x) + O(h^2)}$$

1. ***$O(h^2)$*** is very small
2. ***h*** should be as small as possible, i.e. **1 pixel** length

Edge Detection using Partial Derivatives

$$\frac{f(x+h)-f(x-h)}{2h} = f'(x)$$

Gradient in X-direction $= \delta f / \delta x$

$\alpha \; \left(f\,(i+1, j) - f\,(i-1, j)\right)$

-1	0	+1
-1	0	+1
-1	0	+1

G_x

Vertical edge detector

Gradient in X-direction $= \delta f / \delta y$

$\alpha \; \left(f\,(i, j+1) - f\,(i, j-1)\right)$

+1	+1	+1
0	0	0
-1	-1	-1

G_y

Horizontal edge detector

Example - Vertical Edge Detection Filter

$$
\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{bmatrix}
$$

(Single derivatives are noise-sensitive)
The filter can be made robust to noise by combining with a Gaussian Blur

Prewitt and Sobel Operators

Prewitt Filters:

- **Vertical**

1	0	-1
1	0	-1
1	0	-1

- **Horizontal**

1	1	1
0	0	0
-1	-1	-1

Sobel Filters: *Prewitt + Gaussian Blur*

- **Vertical**

1	0	-1
2	0	-2
1	0	-1

- **Horizontal**

1	2	1
0	0	0
-1	-2	-1

Directional Edge Detection: Robinson Masks

North

-1	0	1
-2	0	2
-1	0	1

South

1	0	-1
2	0	-2
1	0	-1

North-West

0	1	2
-1	0	1
-2	-1	0

South-East

0	-1	-2
1	0	-1
2	1	0

East

-1	-2	-1
0	0	0
1	2	1

West

1	2	1
0	0	0
-1	-2	-1

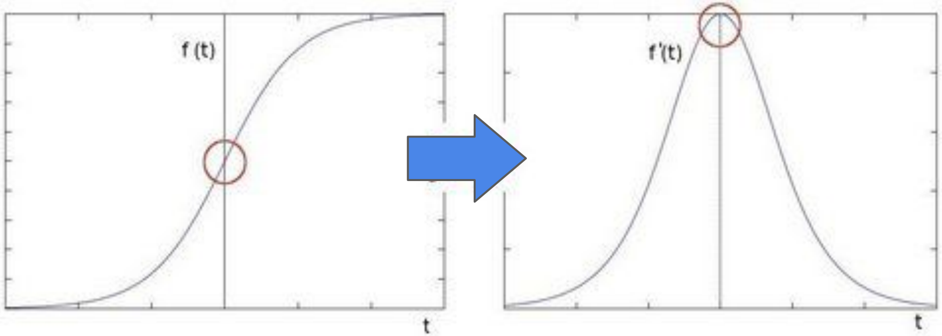
North-East

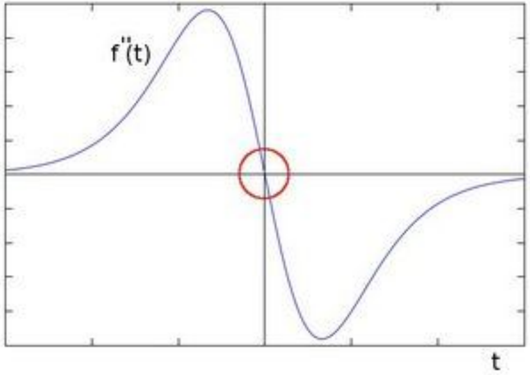
-2	-1	0
-1	0	1
0	1	2

South-West

2	1	0
1	0	-1
0	-1	-2

Sharp Edges - Double Derivatives





Single Derivative:
Difficult to localize exact edge

Double Derivative:
Double derivative = zero at edge

⇒ Edge conditions

1. High single derivative
2. Zero double derivative

Image Sharpening: Laplacian Operator

Laplacian Operator: $$\nabla = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

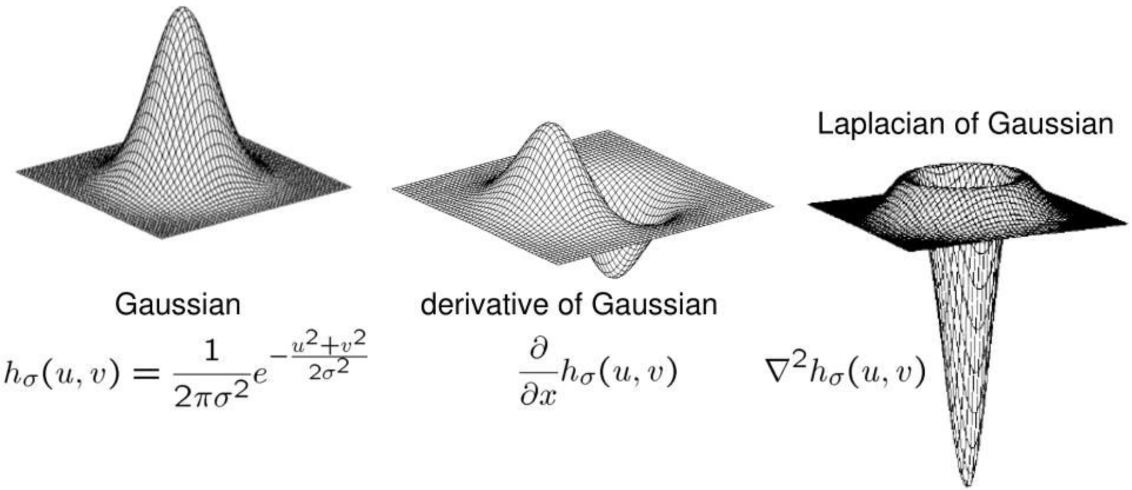
Effect: Edges become sharper

- Can be coupled with Gaussian smoothing for noise-removal
- Resultant Filter: '**Laplacian of Gaussian**' (LoG)

-1	-1	-1
-1	8	-1
-1	-1	-1

Edge detection: Laplacian of Gaussian (LoG Filter)

2D edge detection: *Gaussian for noise-removal → Laplacian for edge detection*



$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$

$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

$$\nabla^2 h_\sigma(u,v)$$

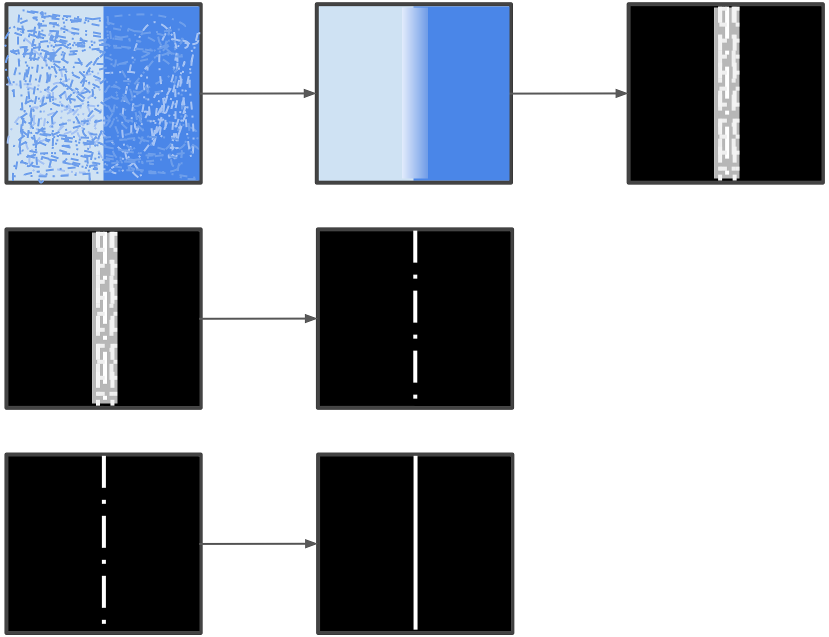
∇^2 is the **Laplacian** operator:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Canny Edge Detection

Steps:

1) Noise Removal &
   Preliminary edge detection
2) Localizing edges
3) Making edges continuous



Canny Edge Detection - Step 1

1. The image is converted to **grayscale**

2. Noise is removed through a **Gaussian Blur**

3. Horizontal and Vertical **Sobel operators** for magnitude of vertical and horizontal edge components ***(G_x and G_y)***

4. Create 2 matrices:
    1. Edge gradient **intensity** at every pixel
    2. Edge gradient **angle** at every pixel

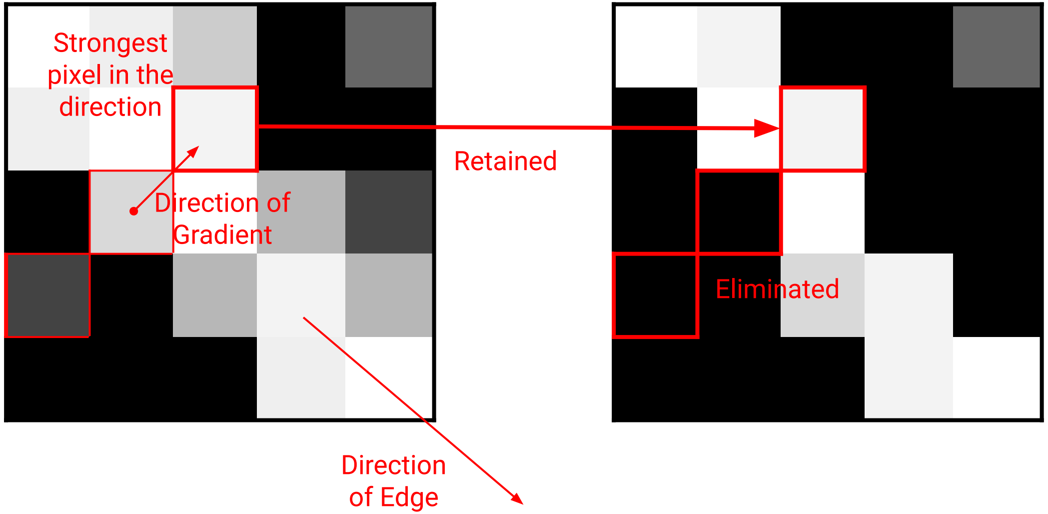
$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2}$$

$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

Canny Edge Detection - Step 2

Making all edges one pixel thick

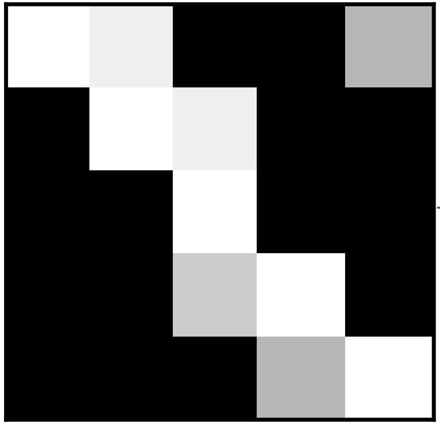
Idea - While travelling in the direction of the gradient, retain only the highest intensity pixel

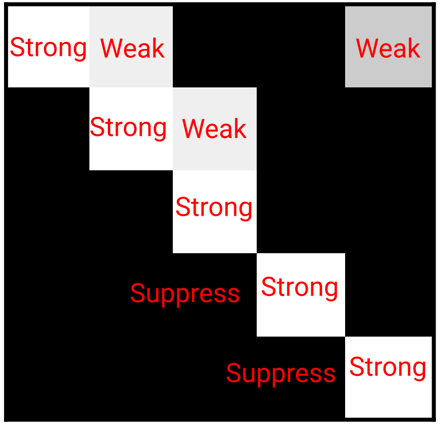


Canny Edge Detection - Step 3

Double Thresholding

1. Set two threshold values: **High** threshold and **Low** threshold
2. **Suppress** pixels **below** the **Low** threshold value
3. Mark the pixels **above** the **high** value as '**Strong**'
4. Mark the pixels **between** both values as '**Weak**'

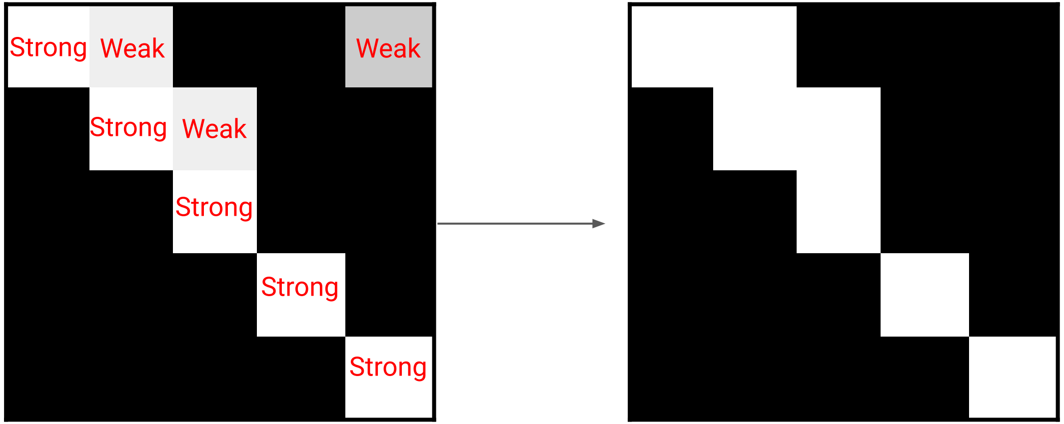




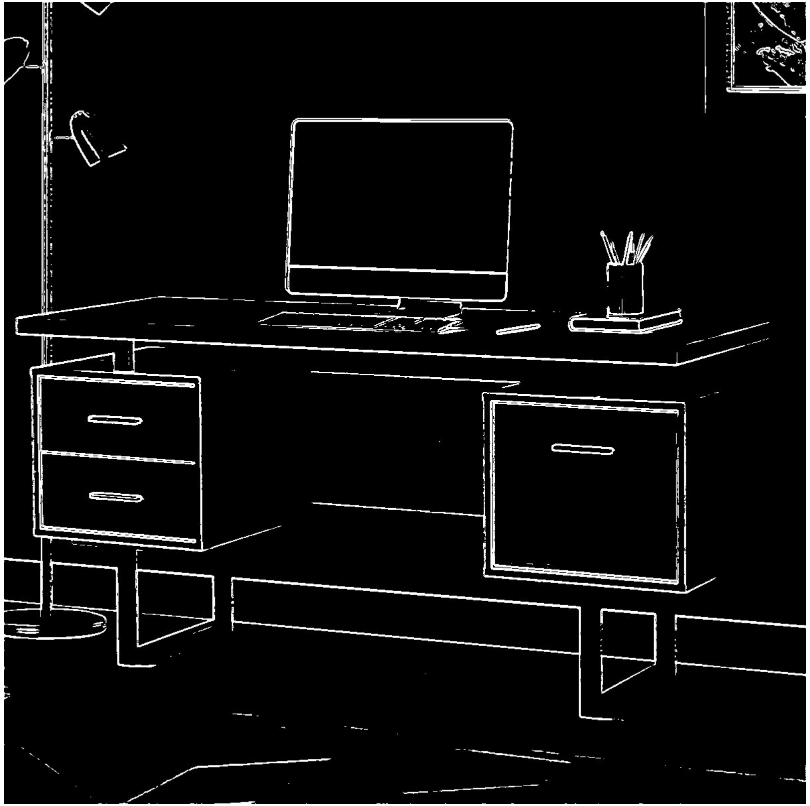
Canny Edge Detection - Step 4

Edge Tracking by Hysteresis *(Ensuring continuity of edges)*

1. Make all strong pixels 255 (max value)
2. Retain (make 255) the weak pixels connected to a strong pixel
3. Eliminate all remaining weak pixels



Example



Covariance (recap)

Covariance: A measure of how one variable varies with another

$$Cov(X,Y) = \frac{1}{N-1}\sum_{i=1}^{N}(X_i - \overline{X})(Y_i - \overline{Y})$$

where

$$\bar{X} = \frac{1}{N}\sum_{i=1}^{n} X_i \text{ and } \bar{Y} = \frac{1}{N}\sum_{i=1}^{N} Y_i$$

are the means of X, Y

Covariance Matrix:

$$\begin{pmatrix} Cov(X,X) & Cov(X,Y) \\ Cov(Y,X) & Cov(Y,Y) \end{pmatrix}$$

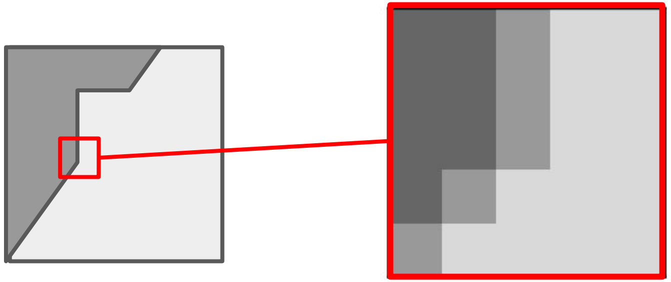
Analysing Features using Covariance

1. Take a small window of the image to analyze
2. Calculate X and Y gradients

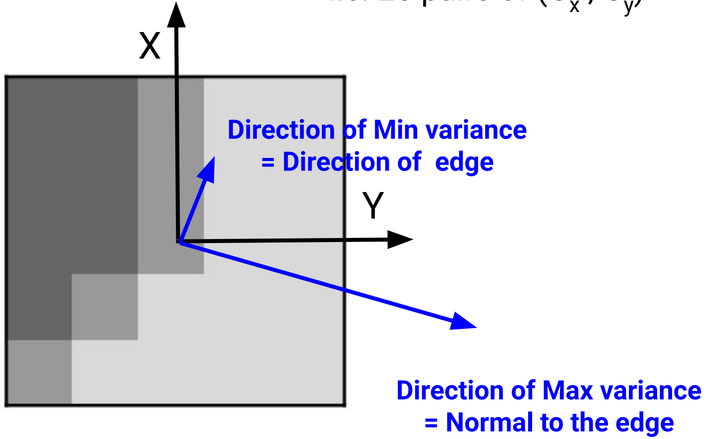
Idea -

- Direction of **maximum variance** is the **normal** to the edge
- Direction of **minimum variance** is the direction of the **edge**

- Requirement: **Change of Basis**
  Current Basis: X, Y
  New Basis: *Directions of* ***minimum*** *and* ***maximum variance***



(5x5) window
i.e. 25 pairs of (G_x , G_y)

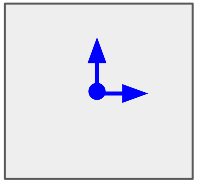


Information Gained from this:

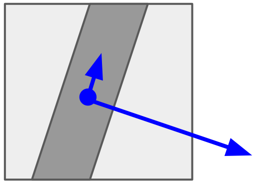
New basis of the window : The directions of maximum and next-maximum variance:

1. Both variances are low ⇒ Flat Region
2. One variance is high ⇒ Edge
3. Both variances are high ⇒ Corner

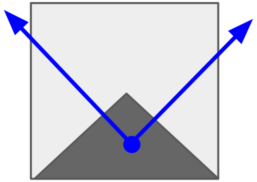
Additionally,
The directions of these new basis vectors tell us the orientation of the feature



Flat region



Edge



Corner

Harris Corner Detector

Idea-

1. Choose a small window

2. Find the two directions in which pixel intensity changes fastest

3. Analyze this rate of change to determine whether the window contains a corner

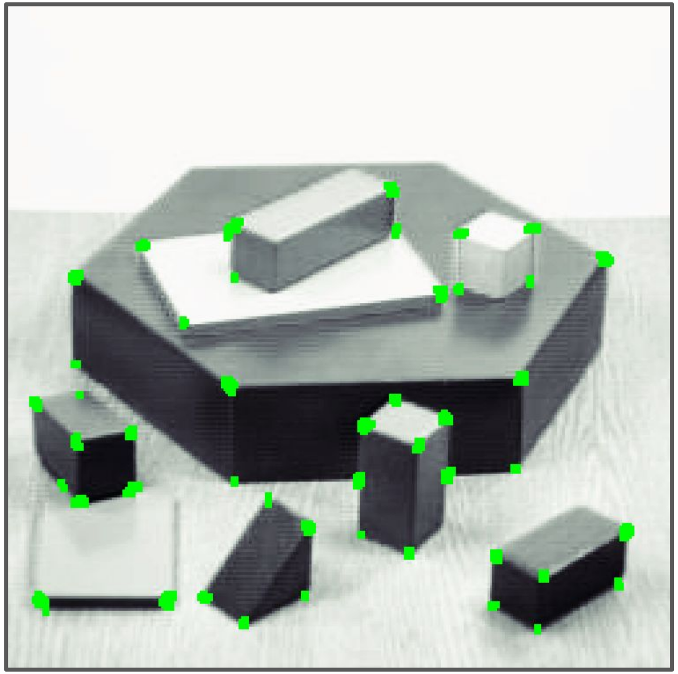
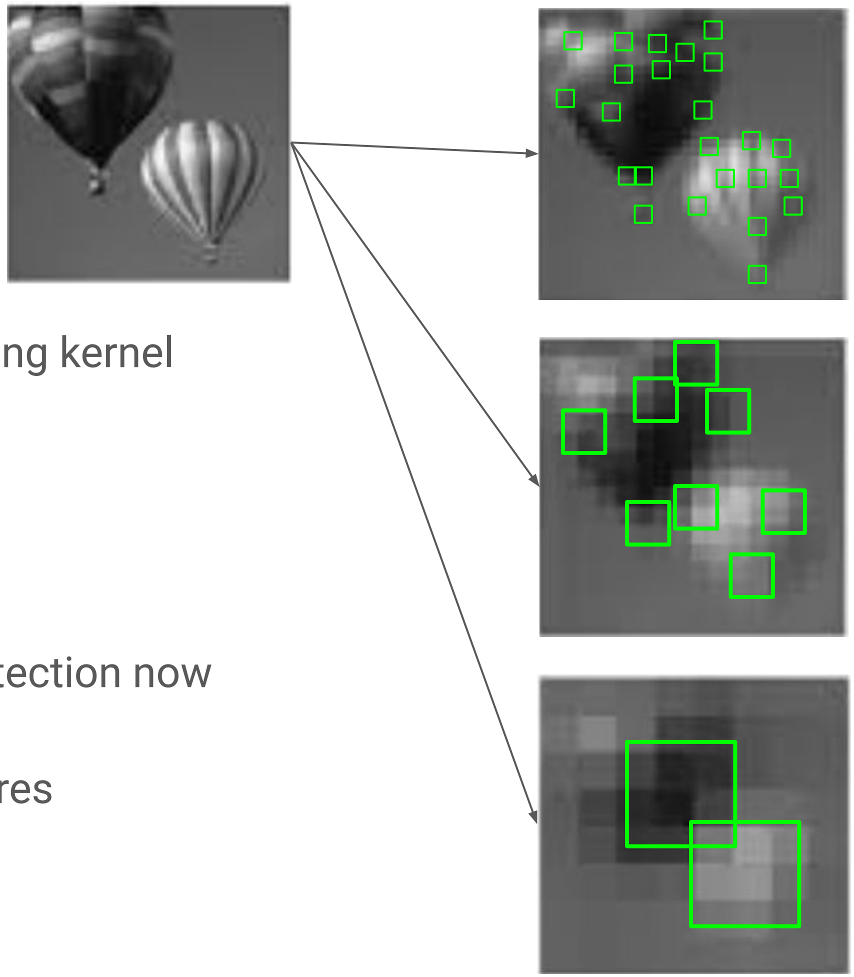


Image Downsampling



Gaussian blurring using a series of increasing kernel size, followed by reduction of resolution

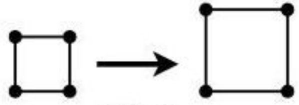
Result after downsampling:

- An (n x n) window used for feature detection now covers a larger portion of the image
- Larger Blobs will be detected as features

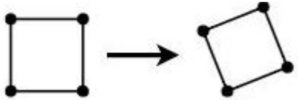
Image Transformations

How can an feature change, when seen from a different position?

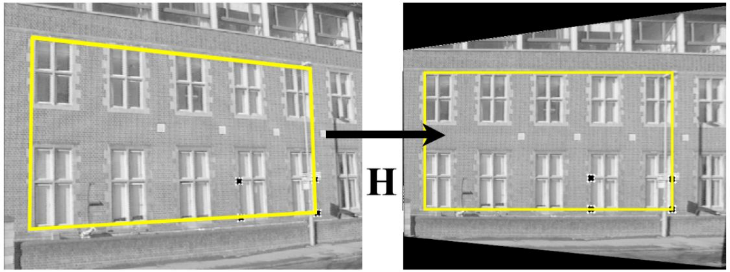
- Scaling



- Rotation & Displacement



- Affine/Projection

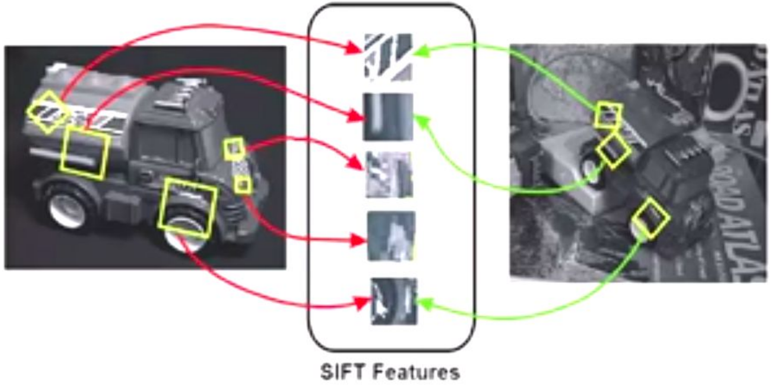


Aim:
To find a representation for features that is independent of these transformations

i.e **Feature descriptors** must be **Transformation-invariant**

Scale Invariant Feature Transform (SIFT)

An algorithm that detects features and provides **descriptors** to them such that the descriptors are invariant to transformations like scaling, rotating, sheer, luminance, etc.



SIFT: Method

- **Size** Invariance is achieved by **downsampling**
- **Luminance** invariance is achieved by **normalising** values in the window (i.e. divide the values to fall in a standard range)
- **Orientation** invariance is achieved using the **histogram** based descriptor (Will be explained shortly)
- Displacement invariance is achieved by the feature matching algorithm

SIFT: Assigning Scale

Detecting Features of different Scales

1. Downsample the image using LoG filters of different kernel sizes (sigma)

2. Apply Canny Edge Detection to all outputs

3. Compare points of interest with the same position in all LoG outputs.
   The output with the maximum value corresponds to the ‘scale’ of the feature

4. Use Harris Corner Detection to eliminate edges from points of interest and only retain corners

SIFT: Assigning Orientations

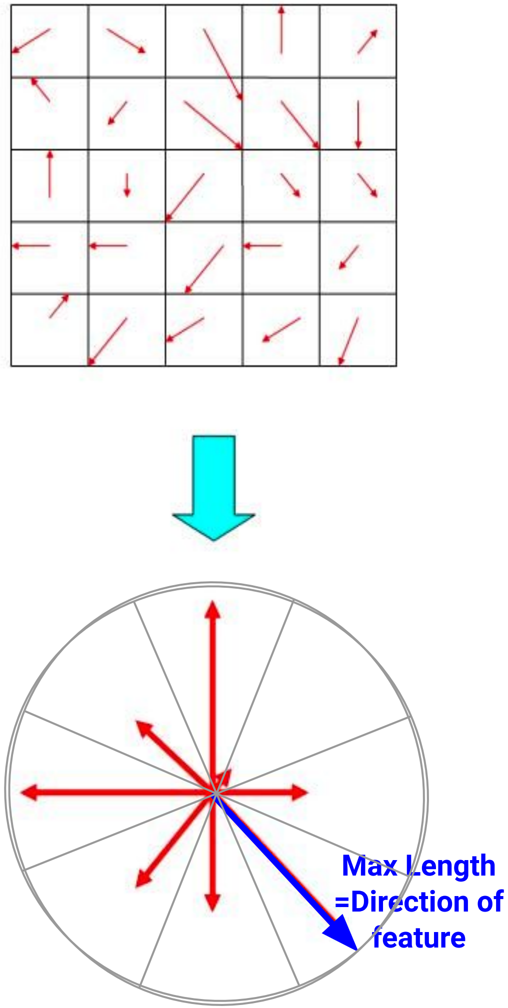
1. Choose an (n x n) window around a feature

2. For each pixel, calculate the direction in which the gradient is changing the most

3. Divide 360º into 36 **'bins'** of 10º each. Assign each pixel to its corresponding bin.

4. For each bin, add the magnitudes corresponding to that bin. Create a **histogram.**

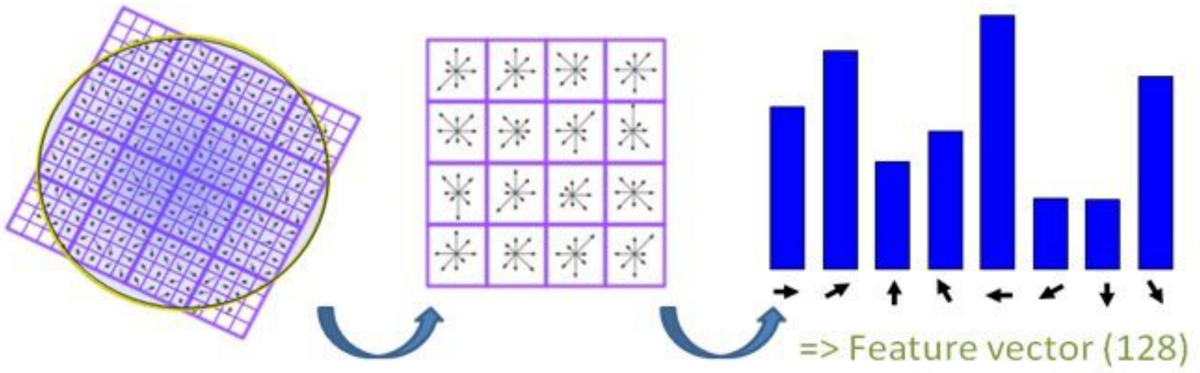
5. The bin with the maximum sum is the direction of the feature



SIFT: Descriptors

1. **Rotate** the feature so that **direction** is **upwards** → ***Direction invariance***
2. **Normalize** the values in the window → ***Luminance invariance***

3. Split the window into 16 parts (4x4) and create a histogram for each part



*These 16 vectors together form the **feature descriptor.***

SIFT (so far)

- **Step:** Features were chosen from the optimum LoG output
  Outcome: Different **scale** will correspond to a different LoG, but descriptor stays unaffected

- **Step:** Direction histograms were rotated to point upwards
  Outcome: Different **rotation** will give the same descriptor

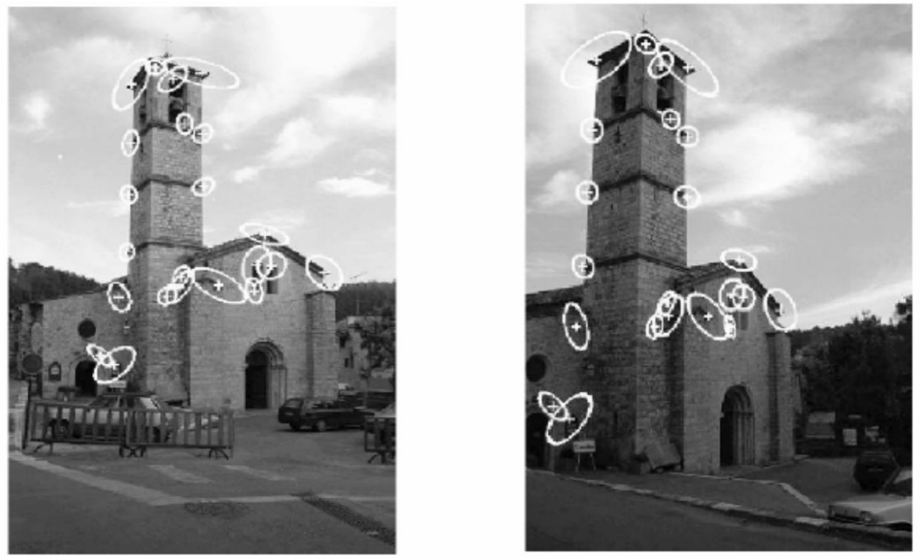
- **Step:** Values in the window were normalized
  Outcome: Change in **luminance** will not affect the descriptor

Final Outcome: ***The feature descriptor will not be affected by scale, rotation and luminance.***

SIFT: Feature Matching

Matching corresponding features in two images

Idea - Multiple features may give a similar descriptor. Hence, their neighbouring features decide if there is a match



SIFT: Feature Matching (2)

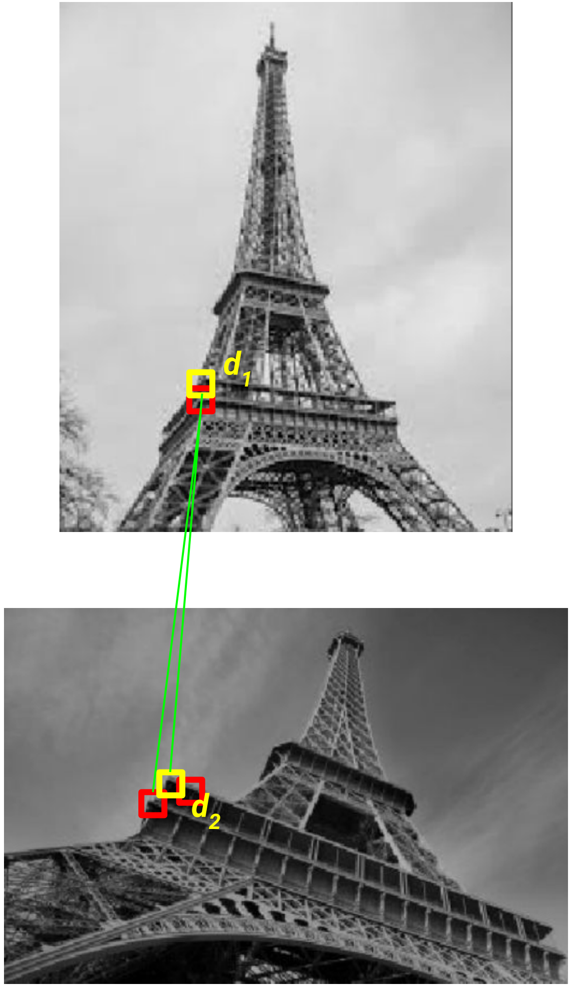
1. Choose a feature in image 1 *(d_1)*

2. Find the best matching descriptor in image 2 *(d_2)*

3. Select the nearest neighbour of d_1
   Select two nearest neighbours of d_2

4. Compare neighbours of d_2 with the neighbour of d_1

5. If the neighbour-descriptors are also nearly equal, then d_1 matches d_2



Fast Feature Detection: ORB

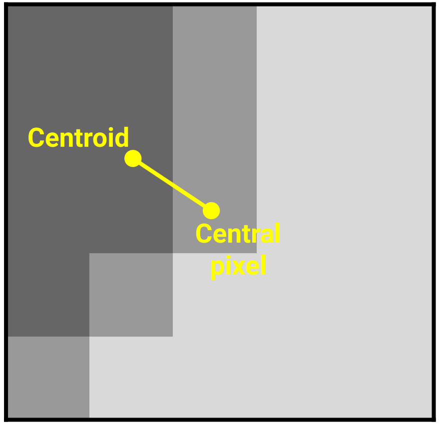
Efficient Alternative to SIFT

1. **Feature detection:**
   Downsampling → FAST algorithm → Harris Corners

2. **Orientation assignment:**
   Direction joining a pixel to the **centroid** of its window

3. **Descriptors:**
   ‘BRIEF’ descriptors

ORB: **Oriented-FAST + **R**otated **B**RIEF*



Centroid:

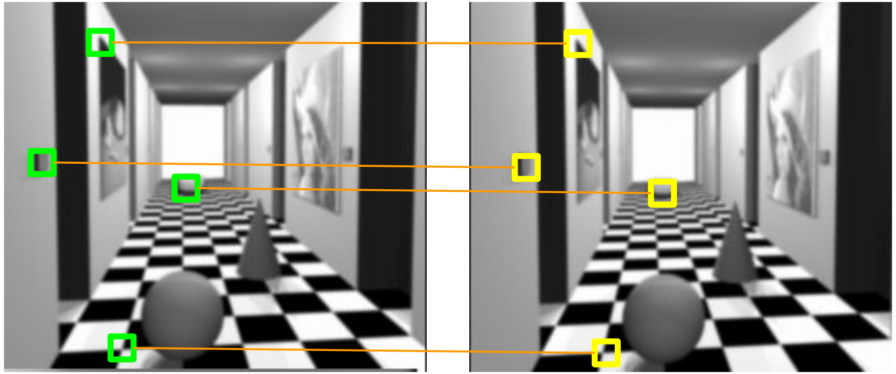
$$\frac{\Sigma[\ (pixel\ value) \times (position)]}{Number\ of\ pixels}$$

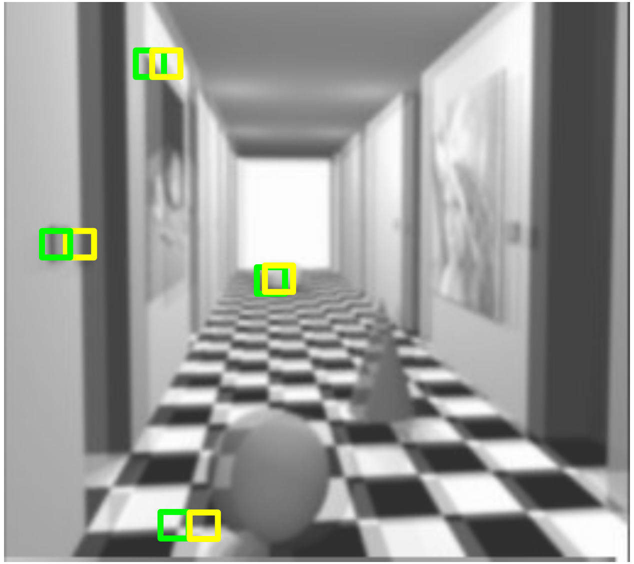
Image Disparity

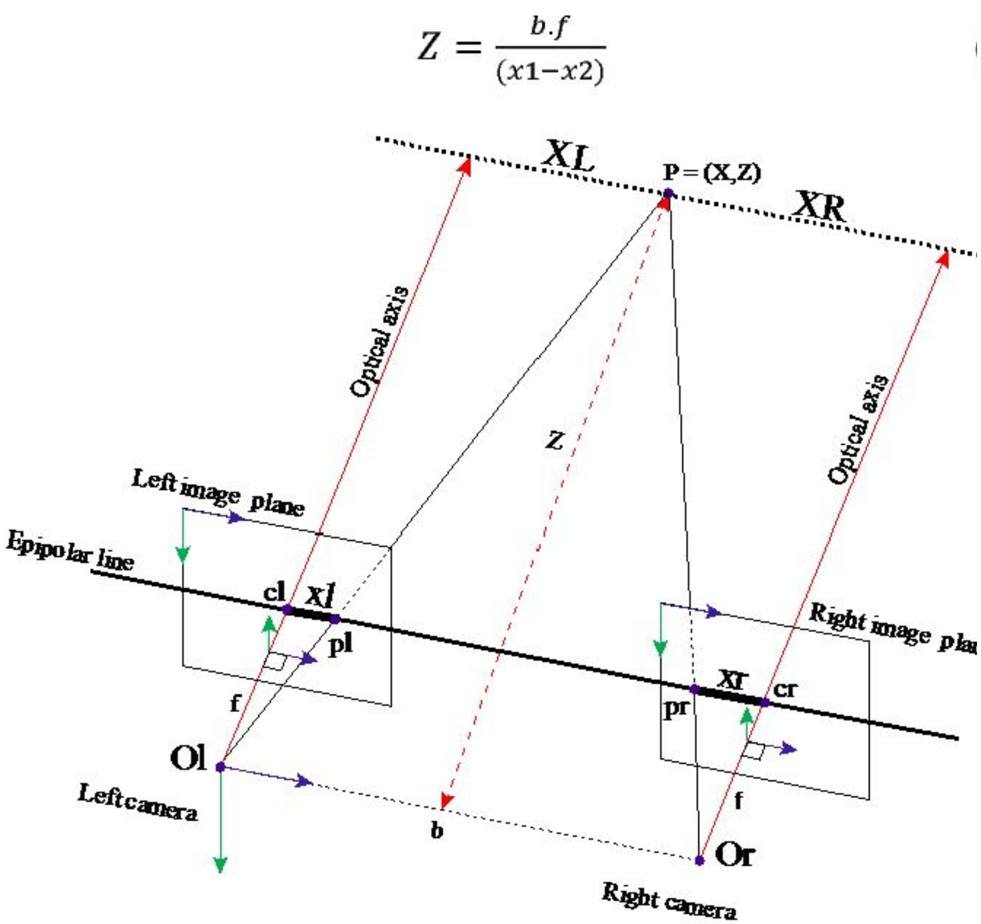
Disparity: Difference in the location of the same feature, when seen from two different perspectives (stereo vision)

Disparity can be directly used to calculate the distance of a feature

$$\textit{Depth} = \frac{\text{(dist. between cameras) x (focal length)}}{\text{disparity}}$$







Disparity to Depth

Disparity can be directly used to calculate the distance of a feature

Depth = $\dfrac{\underline{\textit{(dist. between cameras) x (focal length)}}}{\textit{disparity}}$

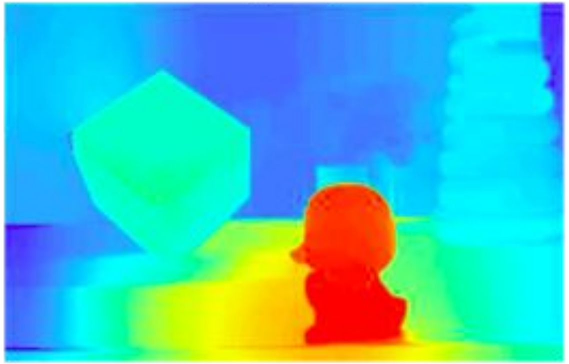
Thus,
Given enough features
disparity can be used to obtain a depth map

Depth Map

Left Image

Right Image

Grey Depth Map



Color Depth Map

Advanced Computer Vision: Deep Learning

Machine Learning -
Several tasks are too complicated to program;
Hence, we program the computer to *learn these tasks on its own*

Deep Learning -
A subset of machine learning that uses *‘Neural Networks’*

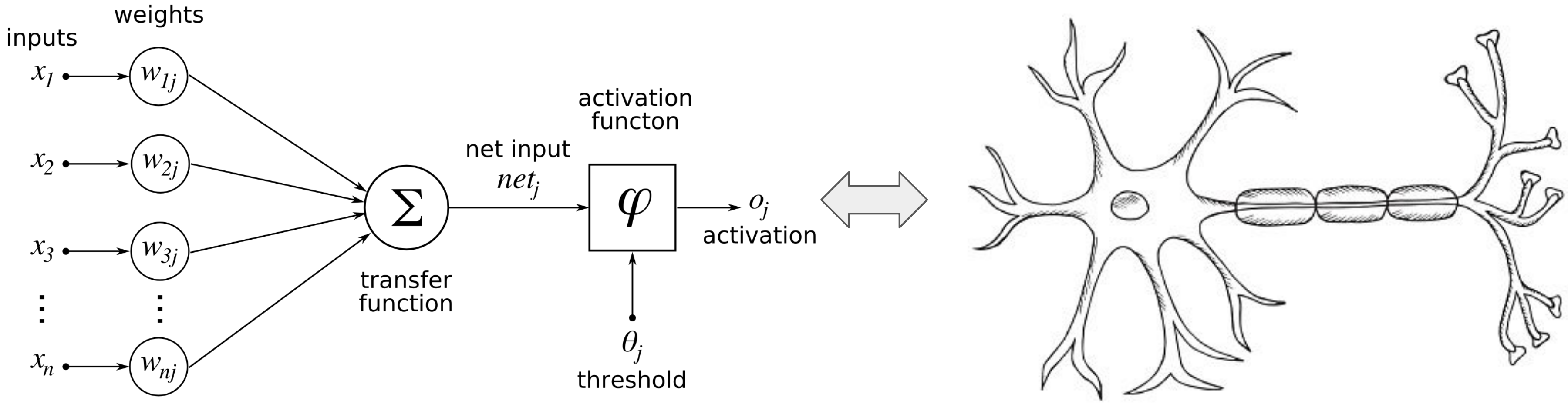
Neural Network -
A structure of interconnected virtual ‘Neurons’, similar to the brain.

With the combined power of all neurons, a neural network is able to learn highly complex tasks

Perceptron

Perceptron - A single multivariable nonlinear regression unit

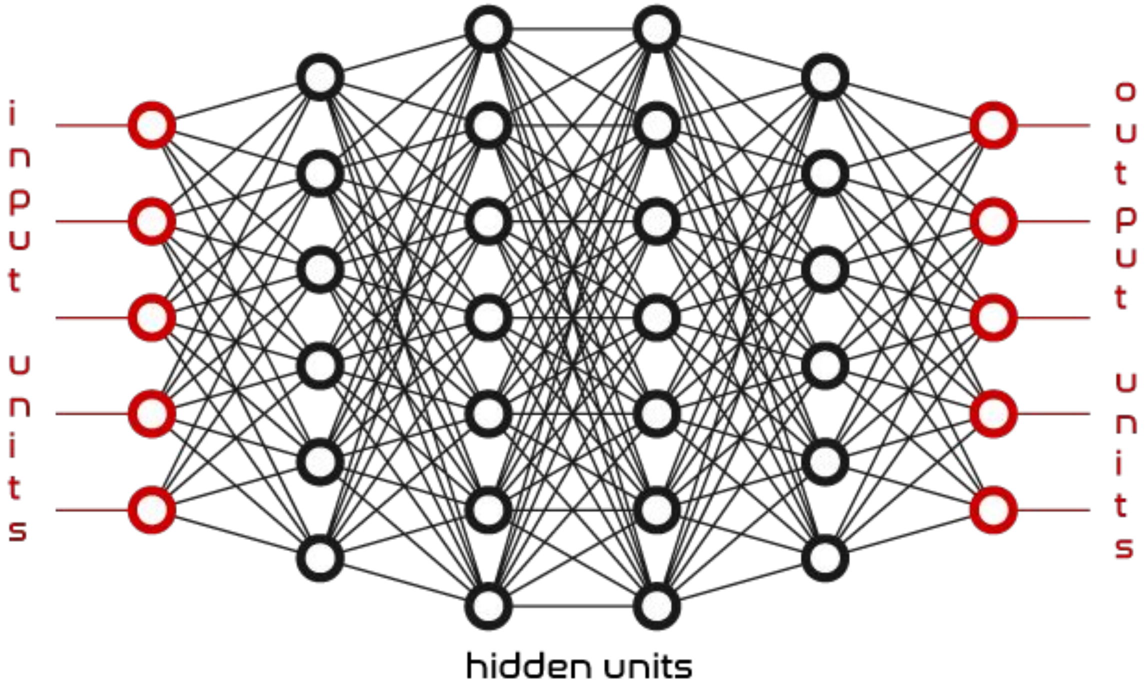
A perceptron learns some transformation (of its choice) on the input, and gives one or more outputs



Multi-Layer Perceptron (MLP)

Multiple interconnected layers of perceptrons form a simple neural network

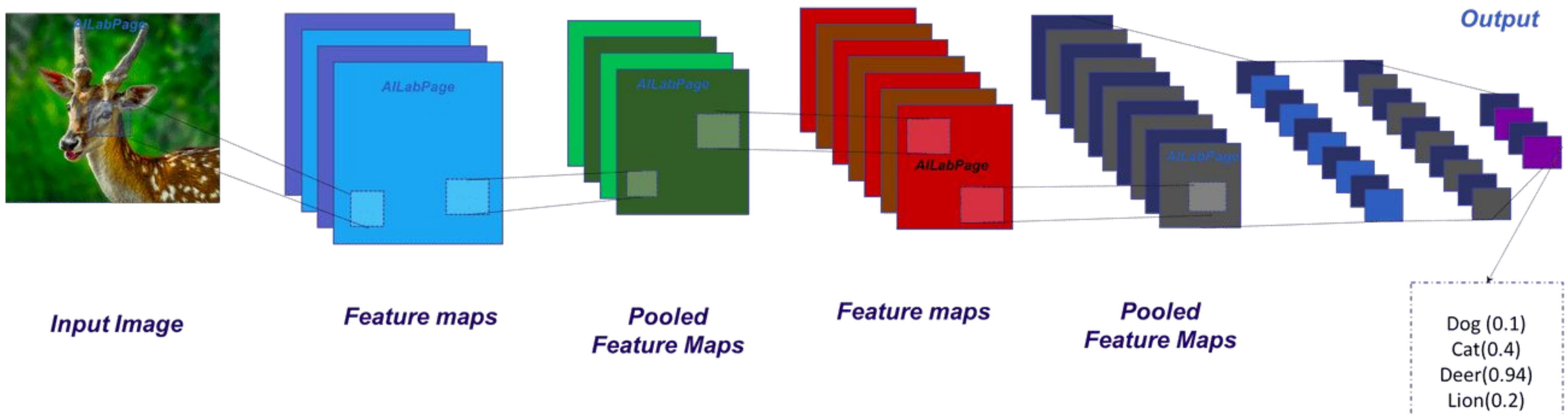
Each perceptron learns a different 'activation function'



Convolutional Neural Networks (CNN)

A special type of neural network that learns values in a convolution filter.

- Each layer consists of multiple **learned filters**
- **Each filter** is capable of detecting a different **type of feature**
- Filters in **further layers** can detect more and more **complex features**
- **Final layers** can detect and localize **entire objects**



Object Recognition & Detection

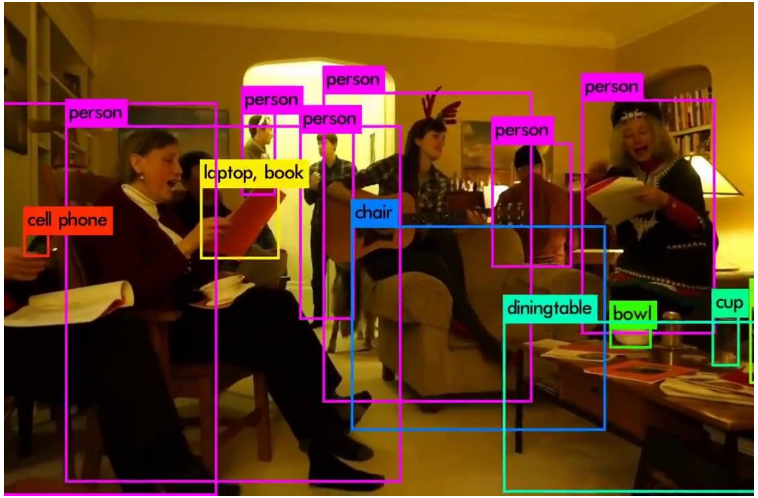
CNNs can detect the presence of different objects in the image (Cars, trees, humans, etc.)

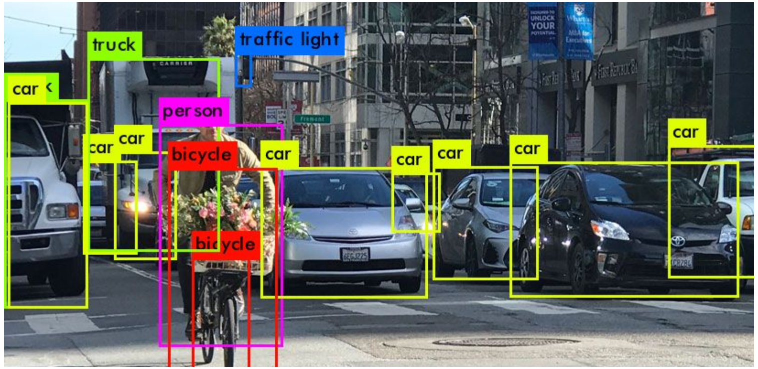
Locating Objects in an image:

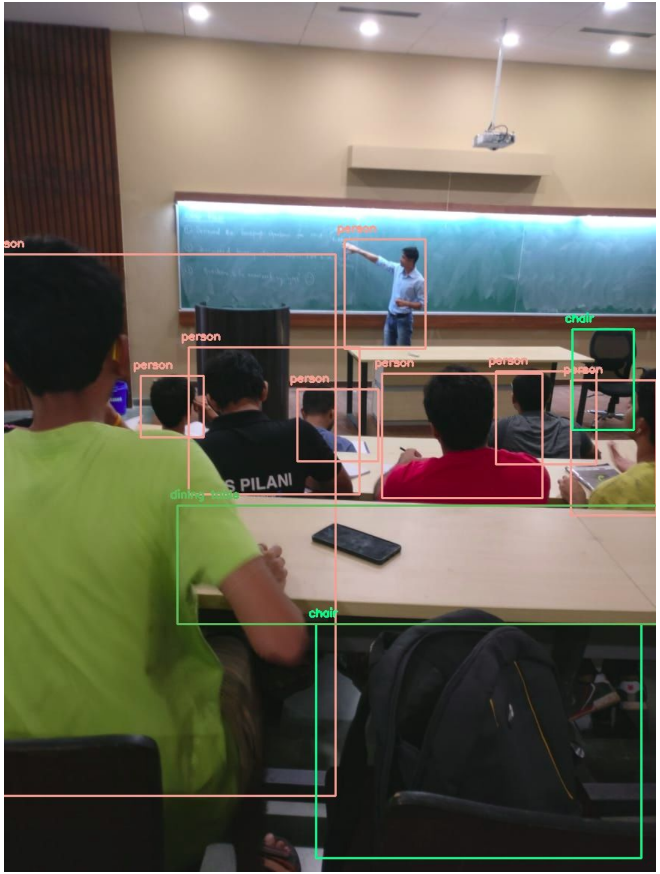
- **Conventional Approach:** Slide a window over the image, run the trained network on every window.
- **Issue:** Very slow. Cannot be used with videos, even with the best GPUs

- **Modern Approach:**
  A special neural network called **YOLO (You Only Look Once)** can detect, locate and identify multiple objects in an image, by a single glance on the image
  YOLO can be used on realtime image data

YOLOv3 results







Next Class...

Implementation of Computer Vision Algorithms in Python & ROS

OpenCV - Open-source Computer Vision library

CV Bridge - Bridge/Interface between ROS images and OpenCV images

Install:

1. cv2
2. cv_bridge