Levels of Measurement in Statistics :-

- Nominal - ~~Odr~~ Ordinal } $\rightarrow$ Discrete

- Interval - Ratio. } $\rightarrow$ Continuous

Input image $\rightarrow f(i,j)$ Output image $\rightarrow G(i,j)$

Kernel $\rightarrow h(i,j)$

If $h(i,j)$ is an averaging kernel. with some K.

$$G(i,j) = \frac{1}{(2K+1)^2} \sum_{u=-K}^{K} \sum_{v=-K}^{K} f(i+u, j+v)$$

$\hookrightarrow$ uniform weights

a kernel with some K has a size of $(2K+1) \times (2K+1)$.

for a general filter.

$$G(i,j) = \sum_{u=-K}^{K} \sum_{v=-K}^{K} h(u,v) \times f(i+u, j+v)$$

$\hookrightarrow$ non-uniform weights

$\downarrow$

Called cross correlation.

A lot of information is lost, mainly of the edge of the image.
$\hookrightarrow$ this is due to the fact that we have to get the edge pixels of the output image, hence we increase the input image by 1 row + 1 col. to obtain it. This was the case for K=1
For larger K we would lose more information from the edge.

Median filtering :- Instead of averaging we use median
- reduces noise and preserves edge.
- there are no kernels; just filter width.

Cross-Correlation :-

$$G(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h(u,v)\, F(i+u, j+v)$$

$$G = h \otimes F.$$

Filtering an image → replace each pixel with L.C. of its neighbourhood pixels. Used to do with a kernel/mask.

Box filter : → • Constant filters pixel values.
• Blurry images / edges.

Gaussian filter :- • Bright in the middle and dark towards the edges. like the gaussian distribution in the heat map.
• Detailed edges.

Normal
↓
parameter → σ → tells us the 'pointiness' of the heat map.
↓
variance of kernel.

more σ; more blurry.

Applying a kernel on an impulse image, the result we get is basically a 2 times flipped version of the kernel in the image.

Convolution :- $G = h * F$.

Cross correlation v/s convolution .

(+)

(−)

h = a b c

d e f

g h i

-1, -1	0, -1	1, -1
-1, 0	0, 0	1, 0
-1, 1	0, 1	1, 1

K = 1

Convolution :- $F * G = G * F$

$$(F * G) * H = F * (G * H)$$

$f = [. . . 0, 0, 1, 0, 0 . .]$

$f * f = f$

Separable → b/w rows & columns.

Linear Filters :-

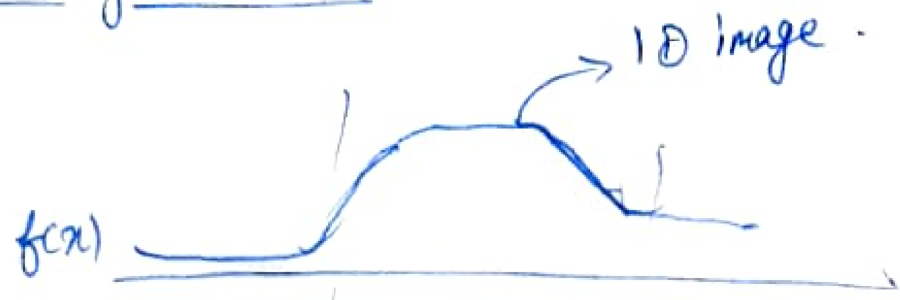
Sharpening: → ~~convolut~~ convolutions - averaging .

Edge Detection :

→ a rapid change in image intensity .

Edge operator should give us edge position , magnitude , orientation .

1D edge detection :-



local extrema in 1st derivative tells us where the edges are. Height of the peak tells us the strength of the edges.

2D images :- we use gradients.

$$\nabla I = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right)$$

Vertical edge.

$$\nabla I = \left(\frac{\partial I}{\partial x}, 0\right)$$

Horizontal edge

$$\nabla I = \left(0, \frac{\partial I}{\partial y}\right)$$

Slant edge :-

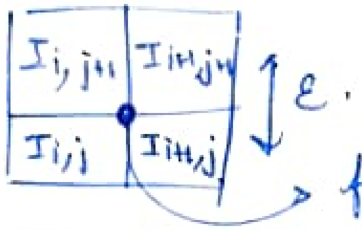
$$\nabla I = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right).$$

magnitude of gradient is strength of the edge :-

$$S = \|\nabla I\| = \sqrt{\frac{\partial I}{\partial x}^2 + \frac{\partial I}{\partial y}^2}$$

Orientation $\theta = \tan^{-1}\left(\frac{\partial I/\partial y}{\partial I/\partial x}\right).$

Discrete Images (we basically focus on discrete images since pixel values are discrete and not continuous).



finding $\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}$.

$$\frac{\partial I}{\partial x} \approx \frac{1}{2\varepsilon}\left[(I_{i+1,j+1} - I_{i,j+1}) + (I_{i+1,j} - I_{ij})\right]$$

similarly for $\frac{\partial I}{\partial y}$.

Kernel = $\begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}$

$$\frac{\partial}{\partial x} \approx \frac{1}{2\varepsilon}\begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} \qquad \frac{\partial}{\partial y} \approx \frac{1}{2\varepsilon}\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}$$

⇒ Hence now we have 3 edge operators.

	Robert's	Prewitt	Sobel.
$\frac{\partial}{\partial x}$	$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$
$\frac{\partial}{\partial x}$	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

Good localization → Poor localization.

Noise ineffective — Noise effective

Poor detection — Good detection.

Edge threshold :-

Standard Single (T)

edge: $||\nabla I(x,y)|| > T$

no edge: $||\nabla I(x,y)|| < T$.

Hysteresis Based : Two thresholds T_0 & T_1.

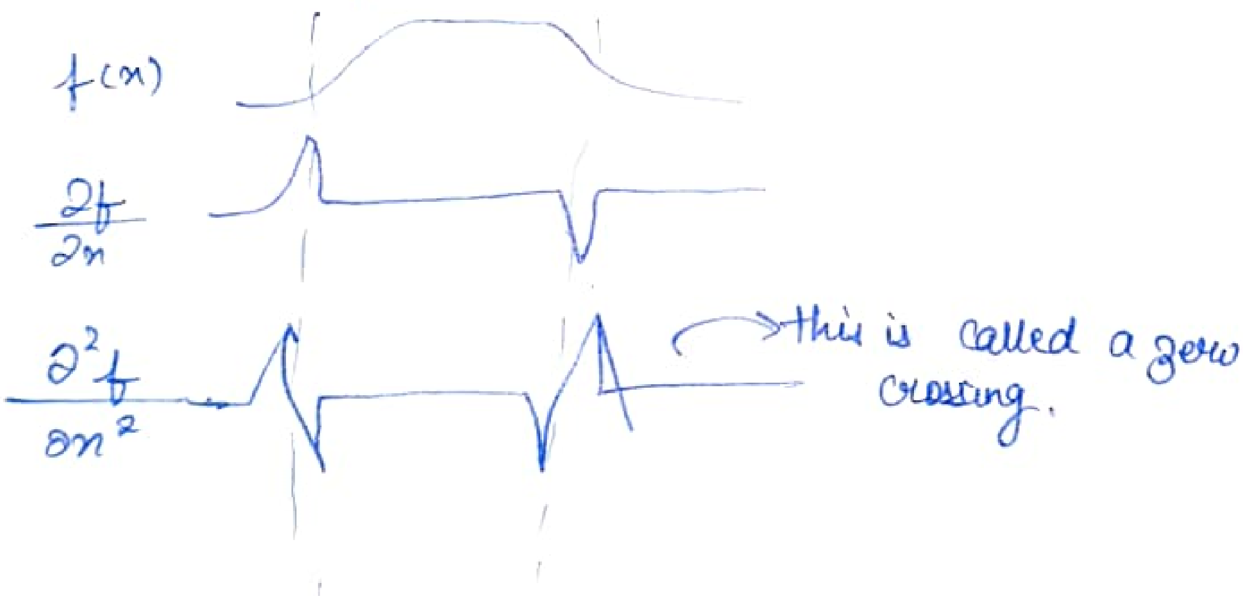
$T_0 < T_1$

$||\nabla I(x,y)|| < T_0$ No edge.

$||\nabla I(x,y)|| > T_1$ Def. edge.

$T_0 < I < T_1$ is an edge if neighbouring pixel is an edge.

Edge Detection Using Laplacian :-



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

Orientation is not given by L.O.

$$\frac{\partial^2 I}{\partial x^2} \approx \frac{1}{\epsilon^2}\left(I_{i-1,j} - 2I_{ij} + I_{i+1,j}\right)$$

$$\frac{\partial^2 I}{\partial y^2} \approx \frac{1}{\epsilon^2}\left(I_{i,j-1} - 2I_{i,j} + I_{i,j+1}\right)$$

$I_{i-1,j}$		
$I_{i-1,}$	$I_{i,j}$	
		$I_{i+1,j+1}$

How will this work well with Continuous signals?

$$\nabla^2 \approx \frac{1}{\epsilon^2}\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \text{or} \quad \nabla^2 \approx \frac{1}{6\epsilon^2}\begin{bmatrix} 1 & 4 & 1 \\ 4 & -20 & 4 \\ 1 & 4 & 1 \end{bmatrix}$$

How did this modification help us?

For reducing noise in the image for edge detection.
we first do $\nabla(n_\sigma)$ — gaussian convolution of width. then apply the filter to the image.
similarly we first do $\nabla^2(n_\sigma)$ then apply to the filter.
$\nabla^2(n_\sigma)$ → laplacian of the gaussian.

Canny Edge Detector :-

1. Smooth Image with Convoluting with Gaussian.
2. take out ∇ of the smoothed out image.

Binary Images

Dilation :- Expanding the 1's to increase the area.

↳ Filling holes and gaps.

Erosion :- Shrinking the shape. creating boundaries and holes.

Opening :- erosion then by dilation.

↳ is idempotent. use a disc or a square.

Closing :- dilation followed by erosion.

Input :- Camera, Lidar, → Images.

Output :- Cone position in car frame, Confidence, Class

Class → Big/Small

Big/Small → colour orange / blue/yellow

→ ~~Big orange, small~~

Lidar → gives depth of every pixel.

0	1	0
0	1	0
0	1	0

img = [[0,1,0], [0,1,0], [0,1,0]]

= [[1,0,0], [1,0,0], [1,0,0]] →

1	0	0
1	0	0
1	0	0

(2,1)

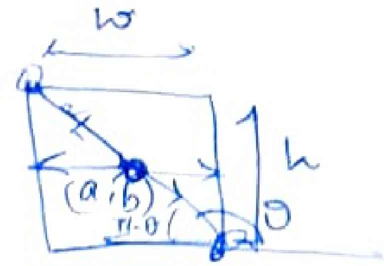
(x,y) in an image is y, x in array.

[[(255, ~~0,0~~ 255,255), (0,0,0), (0,0,0)],
[(255, ~~0,0~~ 255,255), (0,0,0), (0,0,0)],
[(255, ~~0,0~~ 255,255), (0,0,0), (0,0,0)]]

→

1	0	
1		
1		

(x, y, w, h)



$$\frac{\sqrt{w^2 + h^2}}{2}$$

$$m = \tan\theta = \frac{h}{w}$$

$$y = -\frac{h}{w}x + c \qquad c = b + \frac{ah}{w}$$

$$\left(a - \frac{w}{2},\ b + \frac{h}{2}\right) \qquad \left(a + \frac{w}{2},\ b - \frac{h}{2}\right)$$

$$\cancel{[0, 1, 0] - [0, 1,}$$

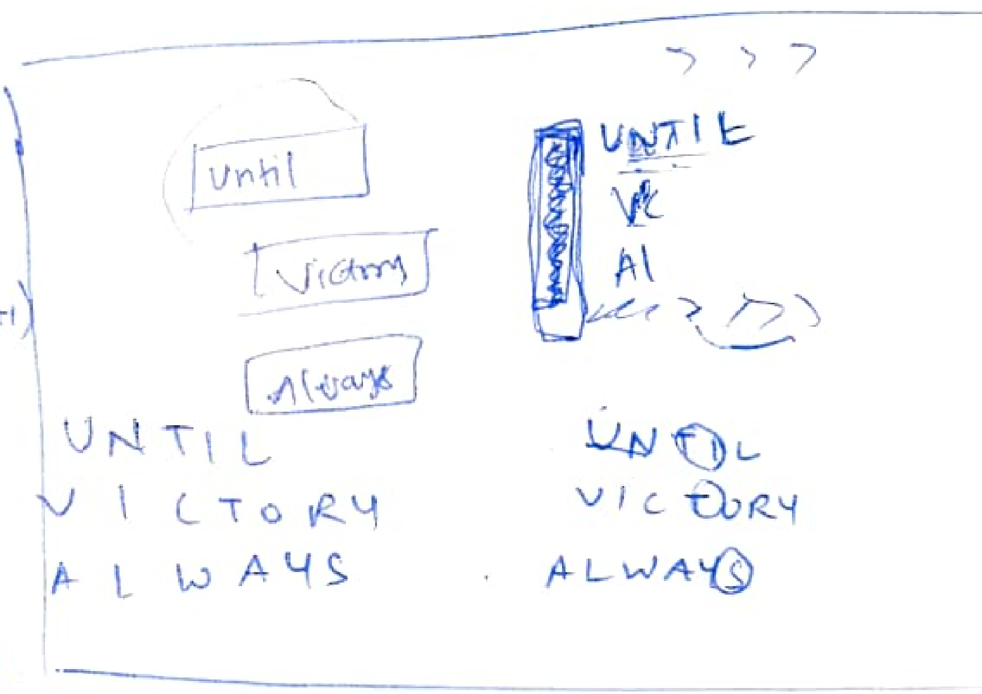
$$\left([0, 1, 0] - [0, 1, 2]/_2,\right. \qquad \left([0, 1, 1] + [0, 1, 3]/_2\right)$$

Zero Padding :-

- n×n image
  f×f filter

output image → $(n-f+1)\times(n-f+1)$

- adding zeroes to the ~~output~~ input image.
- In the above eg. we have to add f-1 rows/columns to the input image.



Padding → valid → no padding
→ same → output size is same as input size.

Max Pooling :- Scaling down the ~~size~~ dimensions of ~~the~~ image

Stride → how much the filter moves in a step.

Storing the max value in the filter. Not the dot product.

image → $n \times n$ filter → $f \times f$ stride → s.

Vanishing Gradient :-

Weighting :- Mean → 0 Var (z) = ?
σ → 1
→ Normally distributed

Bias:- basically a ~~thereshold~~ threshold for each neuron.
Bias determines the flexibility of the model.

$$\underbrace{g}_{\text{activation function}}\,(\underbrace{w_1x_1 + \dots + w_nx_n}_{\text{weighted sum}} + \underbrace{b}_{\text{bias}})$$

Learnable Parameters:-

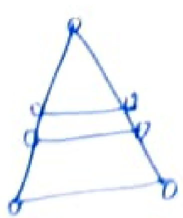
Monocular camera gives → left image.

Pointcloud → a set of points in space made by a ~~3d~~ 3D scan.
~~each~~
↳ gives us a sort of an image which can be moved in 3D.
Each point contains information like (x, y, z) of that point in the image frame, orientation of the point, ~~and someti~~ timestamp of the time when the scan was done, and sometimes the RGB value of the point (pixel). → Intensity too.

Stereo pipeline ~~Right i~~ Left image ↔ Right image
↓
disparity → distance b/w same objects in ~~&~~ the 2 images.

$$\text{Depth} = \frac{\text{focal length} \times \text{baseline}}{\text{disparity}}$$

baseline → dist. b/w left and right cameras.



SIFT - ?

Batch-normalization :-

Normalization.
↳ scaling
the data set down

Standardization
↳ $Z_i = \frac{x_i - \bar{x}}{\sigma}$

Generalizing

Supervised ~~Unsup~~ Unsupervised Deep

How to use semi-supervised learning :-

using a Blue cone we can change the RGB values

1500 Blue

300 Yellow.

Weighted loss function-

freezing a bunch of layers and then modifying other layers to

OpenCV :-

Learnable parameters :- weights and biases.

Bias → Each neuron has a bais.
↳ sort of a ~~the~~ threshold.

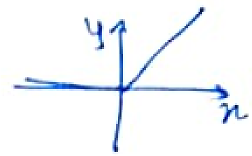
↓

$g\,(w_1x_1 + \dots + w_nx_n + \beta)$

β ↓ bias.

used to shift the threshold → see eg. of relu function. → to narrow down values ~~to~~ or congregate them.

relu



- in ANN.

L.P. = $\underbrace{\text{input} \times \text{output}}_{\text{amount of weights}}$ + biases ⇒ no. of nodes.

for a layer

output ↓ no. of nodes

- in CNN.

input × output + biases

input ↓ dense: no. of nodes

input ↓ ~~CN~~ CL: no. of filters

output ↓ filters × size

biases ↓ no. of filters.

Regularization :→ reduce overfitting. training ↓ generalization ↑.

L2

Batch-Size :→ no. of samples ~~o~~ passed through network at a time.

total = 1000 b.s. = 10 ⇒ 1 epoch = 100 batches.

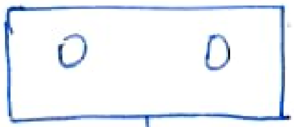
b.s. ↑ faster training ↑

Fine-tuning :→ Transfer learning. applications.
~~ts~~ trained on 1 Class. fine tune it to apply it to class 2 if they are similar.

Perception

- Stereo mono sensor fusion
- Stereo (colour depth)
- Monocular (PnP) depth = ? colour ✓
- LiDar
- LiDar - Stereo sensor fusion.
- LiDar Mono Sensor fusion.

Stereo :→



2 cameras, L.I. R.I. → depth → disparity b/w L.I. and R.I.

colour → RGB values

L.I. R.I.

↓

~~Imposing them on one another.~~

bounding boxes & key-points on L.I. using YOLO v5

↓

Key points on R.I. using propagation → patch regression

↓

using keypoints to draw bounding boxes on R.I.

↓

~~Calculating disparities~~ correcting them using SIFT.

YOLO v5 : Bounding box

RektNet : (x, y) of keypoints on every cone.

Left image

↓ YOLO v5

B B

↓ RektNet

Keypoints

- Perspective Transform
- Patch Regression

Can not run YOLO v5 on both images because the data given by YOLO is random. There is no data correlation.

sift

finds key point on both images and correlates the key-points.

then finds disparity between above key point and SIFT key points

Problems → time.

Mono → faster (only 1 image) low accuracy.

PnP

Mono using BB height.

Homogeneous Transformation Matrix.

global frame.

[ ] = [ ] [ ] [ ]

coordinates in Image frame

Cam. int. Matrix

homo

coord in global frame.