COMPUTER VISION
BSE: 6th SEMESTER

4/2/25

⊙ Deep Learning (Neural Network) for Computer Vision

∴ → Playlists of Topics, Every week Quiz and Assignment of video or topic.

→ **Computer Vision:**

⊙ The algorithm analyze data and classifies data.

⊙ Describe the objects / images.

→ ⊙ Automatically extracting meaningful information

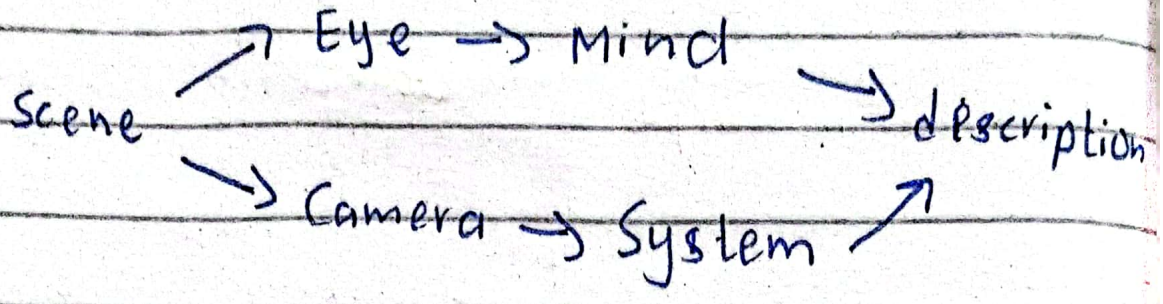
: Abilility of computers to visualize data.

→ where Quantitalive Analysis is required Human Vision System lacks, so Computer Vision is useful there.

⊙ Basic Elements:

(i) Lighting
(ii) Camera
(iii) Vision System Processing.

⊙



6/2/25

$$z = w^T x$$

→ Matrix Multiplication:

⊙ Pros:

(i) Visualization better
(ii) Computation fast.

→ Vector:

⊙ Operation
⊙ Use

→ Matrix:

⊙ operation
⊙ Use

⊙ PCA:

:SVD (Singular Value Decomposition)

→ Transformation:

- Translation
- Rotation
- Scaling

: Vectors ↓ Matrix

① Vector:

$$x \in R^N$$

$$X = \begin{bmatrix} 200 & 201 & \cdots & 280 \\ 281 & 282 & \cdots & 350 \\ \vdots & & & \end{bmatrix}_{28 \times 28}$$

$$X = \begin{bmatrix} 200 \\ 201 \\ \vdots \\ 280 \\ 281 \\ 282 \\ \vdots \\ 350 \end{bmatrix}_{764 \times 1}$$

② Eucleadian Algorithm:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$\|X\|_2 = (x_1^2 + x_2^2 + x_3^2)^{1/2}$$

$$MSE = \frac{1}{m}\left[(\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + (\hat{y}_3 - \hat{y}_3)^2\right]$$

→ Dot Product of two vectors give us similarity between vectors.

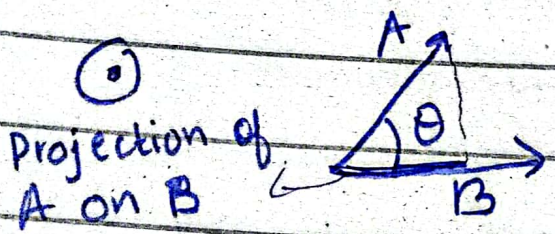
$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}_{3\times 1} \qquad W = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}_{3\times 1}$$

$$\underset{1\times 1}{Z} = \underset{1\times 3}{W^T}\ \underset{3\times 1}{X} = w_1x_1 + w_2x_2 + w_3x_3$$

: If multiplication of two vectors in a matrix give us big value then these vectors are same. (Similar)

: If $Z >>>$ then big value

: If $Z <<<$ then small value.



$$||B|| = 1$$

$$\vec{A}\cdot\vec{B} = ||A||\,||B||\cos\theta \qquad \text{: Inner Product}$$

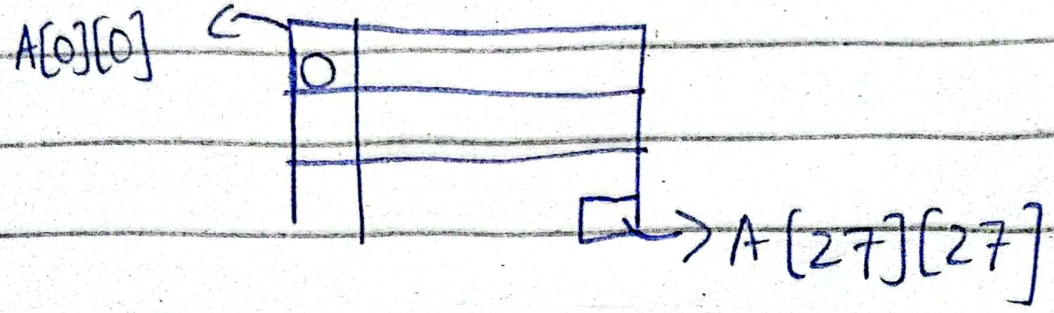
$$\vec{A}\cdot\vec{B} = AB\cos\theta$$

→ Outer Product has many applications like transformer use in Chatgpt.

⊙ Pixel:

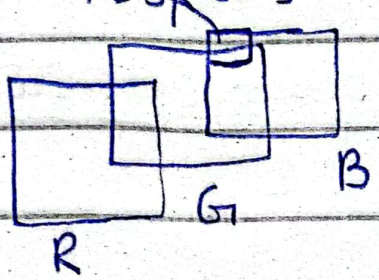
→ 2D Array:

⊙ Index (Position), values (Brightness/Intensity)



→ RGB:

⊙ Three Channels (Red, Green, Blue)



→ Scaling:

$$S \times \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

→ Hadamard:

(Element wise)

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \odot \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a \times e & b \times f \\ c \times g & d \times h \end{bmatrix}$$

→ Determinant:

$$|A| = \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a \begin{vmatrix} e & f \\ h & i \end{vmatrix} + b \begin{vmatrix} d & f \\ g & i \end{vmatrix}$$

⊙ Trace:

→ sum of diagonals elements

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = a_{11}+a_{22}+a_{33}$$

⊙ Inverse:

$$5 \times \frac{1}{5} = 1 \quad \text{(Identity element)}$$

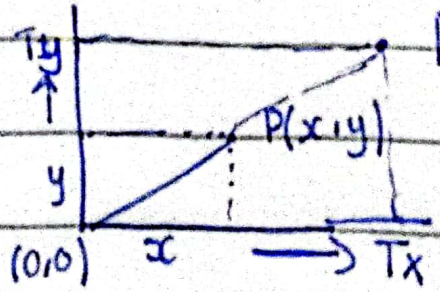
$$[\,] \times [\,]^{-1} = \text{Identity Matrix}$$

⊙ 2-D Transformation:

Rigid (No change in size)
- 1) Translation (object move/shift from one place to other)
- 2) Rotation (orientation change)

Non Rigid
- 3) Scaling (Make Big or small)

① → E.g.:



$$X' = X + Tx$$
$$Y' = Y + Ty$$
$$\Rightarrow \begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} X \\ Y \end{bmatrix} + \begin{bmatrix} Tx \\ Ty \end{bmatrix}$$

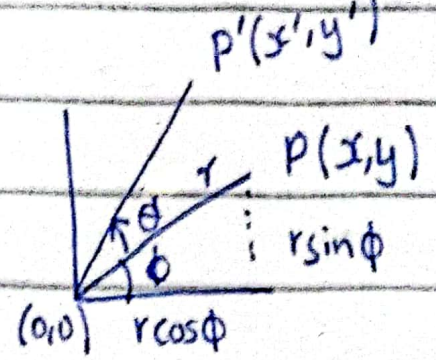
→

$A=(2,7)$, $B=(7,10)$, $C=(10,2)$

$T=(3,4)$

$3 \rightarrow T_x$, $4 \rightarrow T_y$

② Eg:



$x = r\cos\phi$

$y = r\sin\phi$

$x' = r\cos(\phi+\theta) = r\cos\phi\cos\theta - r\sin\phi\sin\theta$

$y' = r\sin(\phi+\theta) = r\sin\phi\cos\theta + r\sin\theta\cos\phi$

$x' = X\cos\theta - Y\sin\theta$

$y' = X\sin\theta + Y\cos\theta$

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

③ → Eg:

δ_x (Delta x)

$x' = \delta_x * x \qquad \delta_x > 1$

$y' = \delta_y * y \qquad \delta_y > 1$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \delta x & 0 \\ 0 & \delta y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

⊙ $\delta x = 2$

$\delta y = 3$

$A = (2, 5)$

$B = (7, 10)$

$C = (10, 2)$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \delta x & 0 \\ 0 & \delta y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \delta x & 0 & 0 \\ 0 & \delta y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

1 ↙ Dummy 1 added

(*) Homework:

→ HW #1

(0,0), (3,2), (4,3) △

A B C

(i) Rotate 45° counter clockwise about (-2,-2)

(Before we translate then we rotate

$X' = T R T (X)$

(45°) (-2,-2)

→ Rotation 3×3

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

→ Translation 3×3

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

→ HW #2:

Modify Triangle (0,0), (1,1), (5,2) to twice its size, while keep (5,2) fix.

11/2/25

⊛ Image Recognition:

-> 3D images changes into 2D in camera and digitize it.

① → Input Image —Image Processing / Quality→ Output Image

→ Input Image —Image classification→ Classify object

→ Image Detection (Position of object

② Hiss Equalization Algorithm
(Enhance image using Histogram).

⊙ Digitization:

→ Two terms : (i) Sampling (Finite) (ii) Quantization

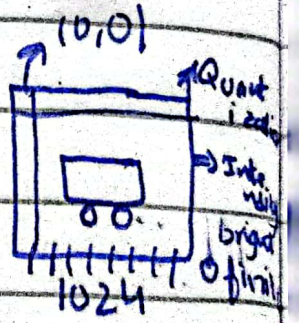
⊙ Sampling:

→ Finite → Discrete

⇒ Size

→ Resolution

→ Digitization of size is called sampling



⊙ Quantization:
(Values)

→ Digitization of intensity value is called quantization.

0: black
1: white

→ 1 bit = $2^1 = 2$

2 bit = $2^2 = 4$

8 bit = $2^8 = 256$

Function :

$$y = f(x)$$

→ x : Domain

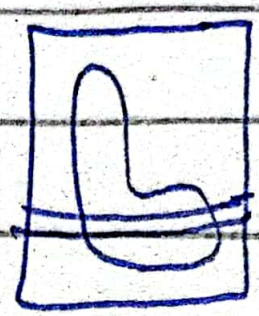
→ $y = f(x)$: Range

→ Sampling (Discretization of Domain)

Quantization (Discretization of Range)

Grayscale ($f(10,10)$)

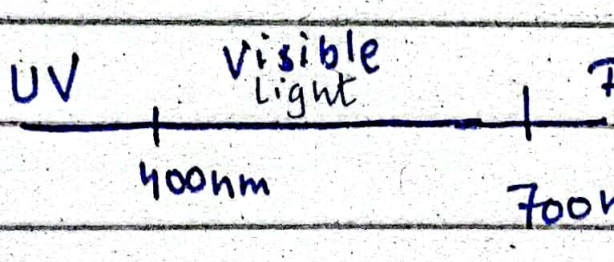
RGB ($f(10_x, 10_y, 3_z)$)



→ PCA → SVD (Important / Dominant Data Extracting)

→ Types of Image: (i) Binary (1 bit) (ii) Grayscale (8 bit) (iii) Color (3, 8 bits)

Color :



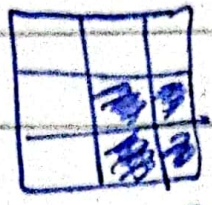
13/2/25

* Data Augmentation. (Make the dataset size big) (Crop and Resize)

* Image Processing (Make Quality Better)

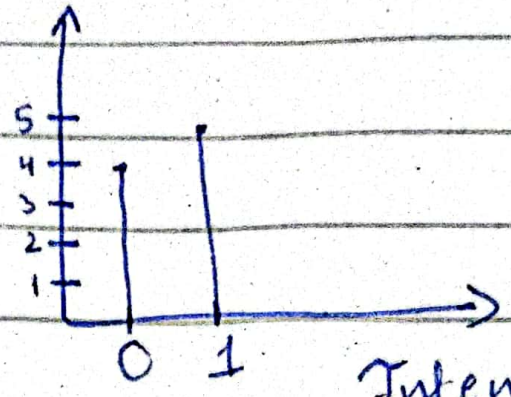
$$f'(x,y) = g[f(x,y)]$$

⊙ Histograms:



Black & White

3x3



Histogram

→ More spread = More Quality.

⊙ Contrast of the image:

→ Quality better by making contrast better.

⊙ Histogram Equalization

(i) Calculate the frequency against each intensity value.

(ii) Compute the probability of each frequency (normalize: 0-1)

(iii) Compute cumulative Probability

(iv) Multiply required intesity level

→ python code:

```
# importing libraries
import cv2 as cv
# OpenCV function
cv.equalizeHist(image)
```

: Equalize (spread more)

: Only one channel histogram is possible.

* Homework:

→ Dark Image of oursel Equalize.

→ Read and convert RGB image into Grayscale.

* Constant values in Tupple. (Array like values in it)

18/2/25

* Slicing:

→ X = [0, 1, 2, 3, 4]

print(X[1:3]) : 1, 2

print(X[:-1]) : 0, 1, 2, 3

: X[:0] (Do it)

→ for i, a in enumerate(list1)

Index

value

: (return two values)

→ x ** 2 (x²) / Power

→ ① [x ** 2 ③ for x ② in nums]

④ print (squares);

: ① [x**2 ④ for x ② in nums ③ if x % 2 == 0]

⑤ (Append ——)

→ Dictionary:

⊙ key value pair

⊙ d = {"cat": "cute", "dog": "Furry"}

⊙ d.get("Monkey", "N/A")

(if present then display otherwise N/A

→ Sets:

: Distinct elements

animals = {"cat", "dog"}
print("cat" in animals)

→ Tupple:

⊙ Const value will be assigned to it (Address, Name etc....)

→ Function:

```
def Hello(name, loud=False):
```

→ Class:

```
class Greeter(object):

    def __init__(self, name):
        self.name = name
```

:_ "private"
:__ "protected"

→ Numpy.

→ import numpy as np

```
a = np.array([1,2,3])
```
: Rank: 1

a.shape(): (3,)

```
a = np.array([[1,2,3]])
```
: Rank: 2

```
a.shape()
```
: (1,3)

→
```
a = np.array([1,3,2])
```
: (3,)

```
b = np.array([3,2,1])
```
: (3,)

```
c = np.dot(a, b.T)
d = np.dot(a.T, b)
```
: No change in column or row on transpose

Single Value : To make 3x3 we use Rank 2

→ Slicing for Matrix:

row col

$a\left[:2, 1:3\right]$

0→1 1→2

→ Bool - Index:

$$a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

b = a > 2 : print(a[b>2]

$$\begin{bmatrix} F & F \\ T & T \\ T & T \end{bmatrix}$$

→ Element-wise & Matrix - Multiplicatio

(x * y) (np.Multiply(x,y),

→ axis=0 (6)

$$\begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

axis=1 (5) (7) (9)

np.sum(a);

→ Broadcasting:

$$\odot \quad [1\ 2\ 3] + 5$$

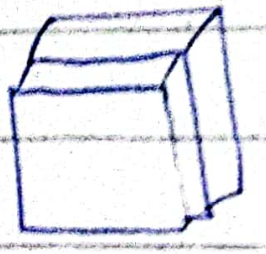
$$[1\ 2\ 3] + [5\ 5\ 5]$$

(1,3) (1,3)

:image dtype (uint8)
(unsigned)

⑥ $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 6 & 7 \end{bmatrix}_{(3,2)} + \begin{bmatrix} 1 & 1 \end{bmatrix}_{(1,2)}$ → $\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$

$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 6 & 7 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$ → $\begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix}$

⑦  * $[1, 0.95, 0.9]$
(ch1, ch2, ch3)

→ Matplot Lib:

plt.subplot(1, 2, 1)
(row) (col) (index)

25/02/25

⊙ Image filtering :

① Ehance Image Quality (Denoise, Blur)

② Find edges

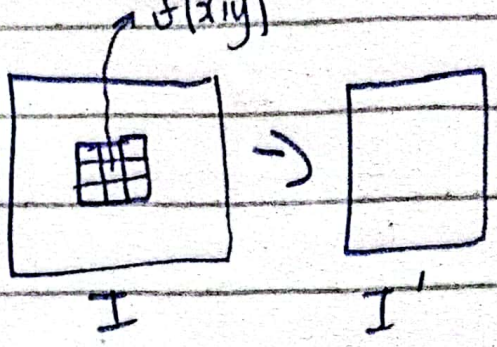
③ fixed Pattern

④ Sharp Image

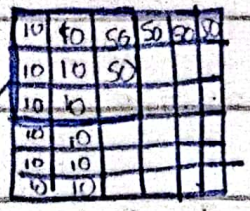
→ Point Operation

→ Neighborhood operation.

Point Operation



: Check left, right, top, bottom and diagonal values for neighbor operation



6×6

*

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

1×1

(Kernel, filter, weight matrix, mask, sliding window)

(Generally order is odd)
(3×3, 5×5, 7×7)

(1,1) first mask

$= 10\times1 + 10\times0 + 50\times-1 +$
$10\times1 + 10\times0 + 50\times-1 +$
$10\times1 + 10\times0 + 50\times-1$

$= -120$

(Convolutional Operation)

(Then slide window to next (1,2)) And then to (2,1)

-120	-120	0	0
-120	-120	0	0
-120	-120	0	0
-120	-120	0	0

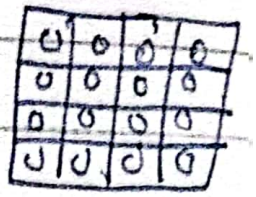
→ Now filter Dimension changes:

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

(Horizontal)

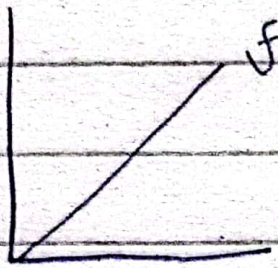
$$= 10\times1 + 10\times1 + 50\times1 +$$
$$10\times0 + 10\times0 + 50\times0$$
$$10\times-1 + 10\times-1 + 50\times-1$$

$$= 0$$



IY

⊙ Discrete Derivative:



$X = 2$

$X = 2.0001$

$\Delta X = 0.0001$

$\Delta f = 0.0003$

$$\frac{\Delta f}{\Delta x} = \frac{0.03}{0.01} = 3 = \text{Slope}$$

$$f'(x) = 3$$

$$\text{Derivative} = \frac{f(x) - (fx - \Delta x)}{\Delta x}$$

$\therefore \Delta x = 1$

$$\text{Slope} = \frac{f(x) - (fx) - 1)}{1}$$

→ (i) Backward difference: → (ii) Foward difference

$f(x) - (fx - 1)$

(iii) Central difference

$f(x)$ = | 10 15 10 25 20 20 20 |

$f'(x)$ = | -5 -5 15 -5 0 0 |

(Backward difference)

We can also apply Kernel on it
X [-1, 1] (multiply) and we get same value. (Dot Product)

⊙ Derivative in 2-D:

$f(x,y)$

→ Gradient vector.

⊙ For every pixel we need vertical and horizontal change

$$\nabla f(x,y) = \begin{vmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{vmatrix} = \begin{bmatrix} f_x \\ f_y \end{bmatrix}$$

→ Gradient Magnitude:

$$\Delta f = \sqrt{f_x^2 + f_y^2}$$

: Laplacian (Used to find 2nd derivative)

→ Input Image and Kernel (weights) are same then output is big / Large value and vice versa.

(Noise Remove Kernel)

1	1	1
1	1	1
1	1	1

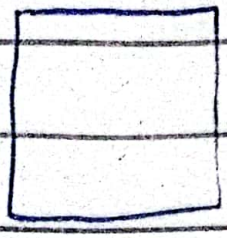
→

255	0	50
255	0	50
255	0	50

May be unwanted info because the image is not in that format.

4/3/25

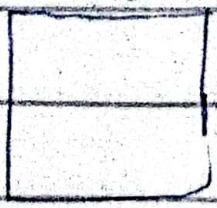
Image

→  * $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

(Box filter) Kernel

Image

→  * $\frac{1}{16}$

1	2	1
2	4	2
1	2	1

(Guassian filter) : Focus on central Pixel

→ (Edge find)

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

(Horizontal edge) (Privot filter)

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

(Vertical edge) (Sobel filter)

→

0	0	0
0	1	0
0	0	0

(No change filter)

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

(Horizontal edge)

→

0	0	0
0	0	1
0	0	0

(Left shift filter)

0	0	0
1	0	0
0	0	0

(Right shift filter)

0	0	1
0	0	0
0	0	0

*(?. Find it)

→ Original Image − Smooth Images = Edges

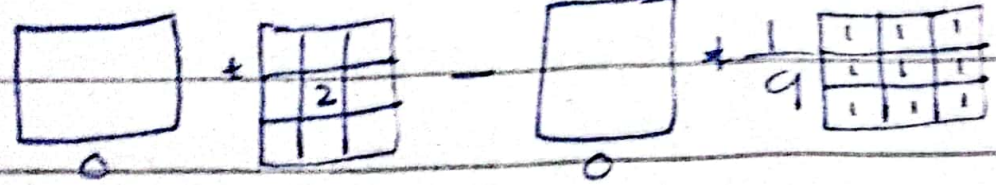
→ If no variation in portion of images, then value of center remains same by applying box filter. And after it we subtract orignal and smooth image, we get edges (New method)

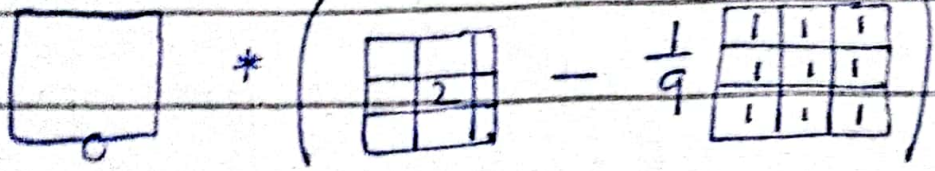
→ Original Images + Edges = Sharp Edge Image

↓

Short method : Original Image * [0 0 0 / 0 2 0 / 0 0 0] (2 times Images) − 1/9 [1 1 1 / 1 1 1 / 1 1 1] = Sharp Edge Image

(2 times image − Box filter = Sharp Image)

Another method → 

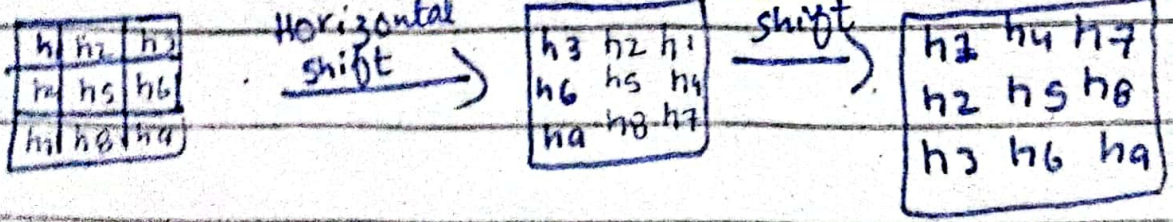
(Computation less) 

→ Correlation (**) & Convolutional Operation

Correlation Operation

(shift it horizontally and vertically) (Then apply correlation)

Kernel



: Correlation and Convolutional Operation becomes equal when Kernel is same (Box filter)

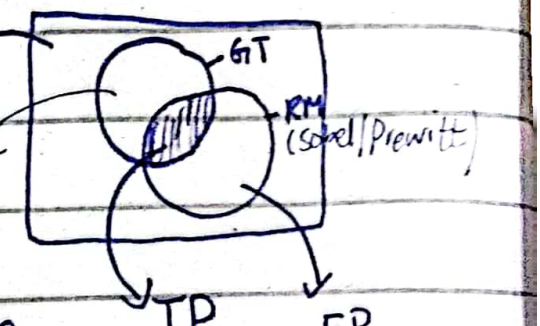
(*) 1-D Gaussian filter has relation with 2-D Gaussian filter. (Multiply by 16 for 3×3)

11/3/25

⊙ Edge Detection:

→ An edge with thickness of one pixel is true edge.

→ Precision
Recall
F1-Score



GT ∩ RM = True Positive (TP)
(Ground Truth) (Result Method)

⊙ Precision:

→ Predicted edges are correct or not

→ $\text{Precision} = \frac{GT \cap RM}{RM} = \frac{TP}{TP+FP}$

e.g: TP = 10, FP = 0 ∴ $\frac{10}{10+0} = 1 = 100\%$.

⊙ **Recall:**

→ whether you predict all edges

→ Recall $= \frac{GT \cap RM}{GT} = \frac{TP}{TP+FN}$

E.g: $\frac{10}{10+990} \times 100 = 1\%$

⊙ **F1-Score:**

$$= \frac{P * R}{\frac{P+R}{2}} = \frac{2PR}{P+R}$$

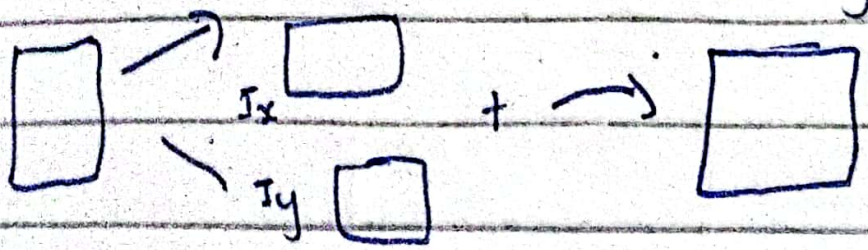
$$= \frac{2*1\times0.01}{1+0.01} = \frac{0.02}{1.01} = 0.02$$

→ Area more then, better algorithm.

⊙ **Gaussian - Laplacian Edge Detected:**

→ predict well localized edge
(Thickness more of edge)

→ Sobel (i) Gray Scale (ii) Vertical & Horizontal edges
(iii) Gradient magnitude



$\sqrt{Ix^2+Iy^2}$

→ Gradient threshold will be set

(*) Sobel (1st derivative)

(*) Laplacian (2nd derivative)

(*)

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

(Horizontal & Vertical & Diagonal & both edges)

(*) $[-1 \quad 1]$: First order filter

$= [-1 \quad 1 \quad 0] * [1 \quad -1 \quad 0]$: To make it order

: we apply padding

$= [0 \quad -1 \quad 1 \quad 0 \quad 0] * [1 \quad -1 \quad 0]$

$= [1 \quad -2 \quad 1]$

$= \begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix}$: To make it first vertical and horizontal

$= \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

:

$$= [1 \ -2 \ 1] * [10 \ 10 \ 10 \ 20 \ 20 \ 20 \ 10 \ 10]$$

$$= [-10 \ 0 \ 10 \ -10 \ 0 \ -10 \ 10 \ 0 \ 0]$$

E E E

④ : Zero Crossing:

(i) value negative & value positive

(Edge detection) (ii) positive & negative

(iii) positive, zero, negative

(iv) Negative, zero, positive

Cases:

(i) -ve (E) +ve

(ii) +ve -ve (E)

(iii) +ve 0 -ve (E)

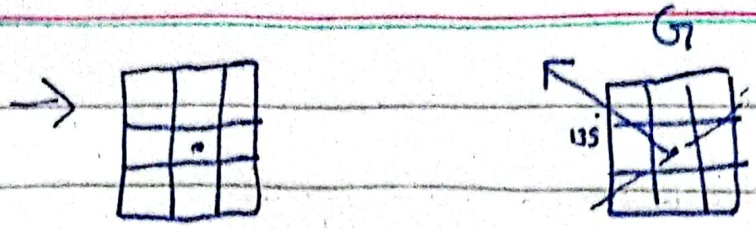
(iv) -ve (E) 0 +ve

⑤ If we convolve both laplacian and gussian filter then it is called guassian of laplacian filter.

⑥ **Canny Edge Detector:**

→ Smooth Image with Guassian filter

→ Compute derivative of filter image



: $G_1x = 6$, $G_1y = -6$

$$G_1 = \sqrt{62 + -62} = 8.4$$

Direction : $\theta = \tan^{-1}\left(\frac{G_1y}{G_1x}\right) = 135^\circ$

⊙ Non-Maximum Suppression (To detect real edge)
(Detect whether that pixel is edge or the neighbor ore pixel)
: If Gradient is greater than other neighbor then value retain, otherwise value become zero. (Thin edge detect)

⊙ Value Quantization:
0, 45°, 90°, 135°, 180°
: if 10°, then 45°
" 50°, then 45°

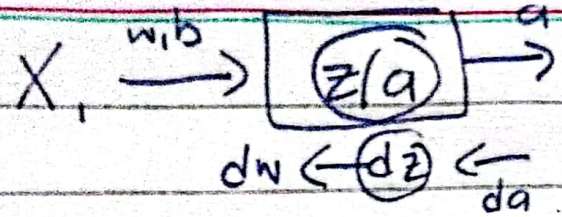
⊙ Strong Edges : Gray-next
Weak Edges White: Stay
Low Edges Zero: Low

18/3/25

* Binary Classification :

⇒ Logistic Regression

⇒ NN (shallow NN)



① $z = w^T x + b$

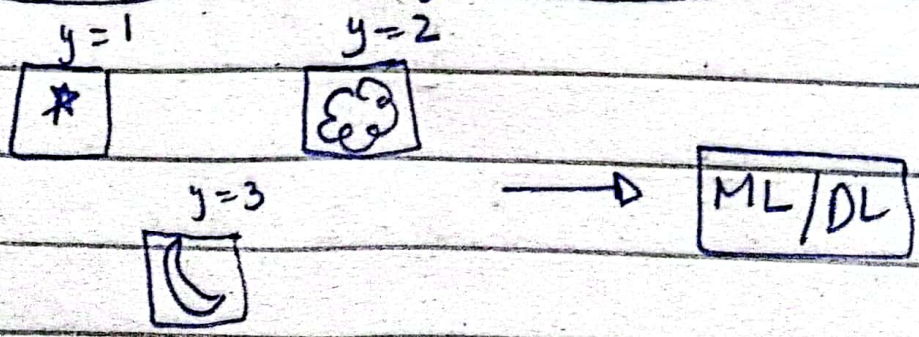
② $\hat{y} = a = \frac{1}{1+e^{-z}}$

③ $\mathcal{L}(\hat{y}, y) = -[y \log \hat{y} + (1-\hat{y}) \log(1-\hat{y})$.

④ Gradient compute

⑤ $w = w - \alpha\, dw$

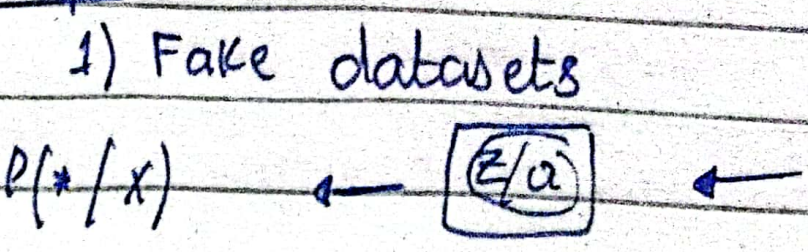
⊛ Multi-class classification :



⇒ We can identify multi-class by binary class
(YES, ⇒ One Vs ALL)

⇒ Steps:

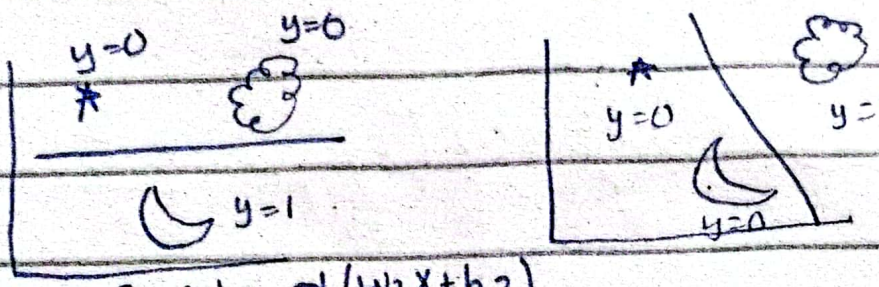
1) Fake datasets



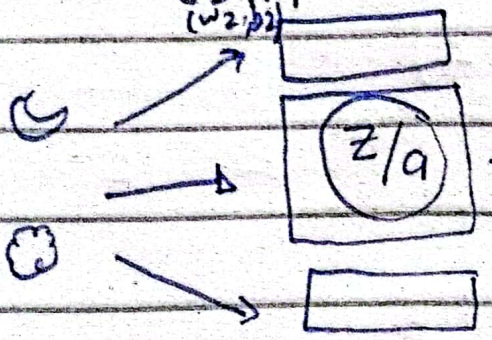
2) Train Binary classifier (L.R / NN)

$f_{(w_1,b_1)}(x) = a = \sigma(w_1 x + b_1)$

⇒ Now obtain 3 classifiers:



$f_{(w_2,b_2)}(x) = \sigma(w_2 x + b_2)$



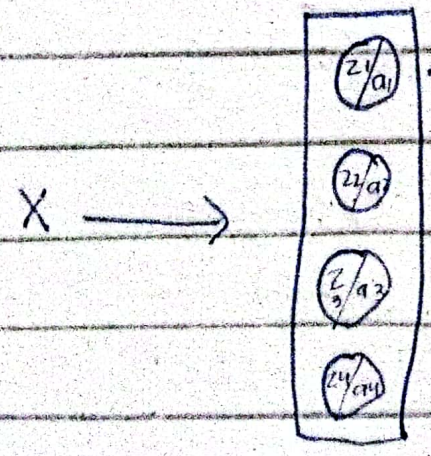
: Cloud Moon Star

: we compare Probabilities

$f_{(w_3,b_3)}(x) = \sigma(w_3 x + b_3)$

: only (0,1) answer so we use

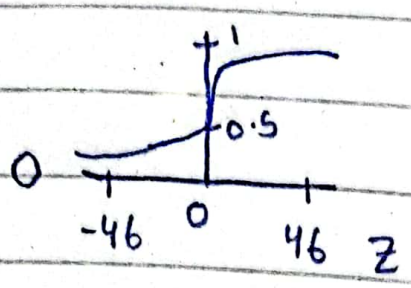
⇒ To make it easy, we use Softmax:



$$a = \frac{1}{1+e^{-z}} = \frac{1}{1+\frac{1}{e^z}} = \frac{1}{\frac{e^z+1}{e^z}} = \boxed{\frac{e^z}{e^z+1}}$$

(Generalized form of Softmax)

$$a = \frac{e^z}{e^z + e^0}$$



$$\Rightarrow \text{Softmax}(z_1) = \frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}+e^{z_4}}$$

$$\text{Softmax}(z) = \frac{e^z}{\sum_{j=1}^{n} e^{z_j}}$$

$$\hat{y} = \text{Softmax}\left(\begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}\right) = ?$$

(i) $\text{Softmax}(5) = \frac{e^5}{e^5+e^2+e^{-1}+e^3} = 0.842$

$\text{Softmax}(2) = \frac{e^2}{e^5+e^2+e^{-1}+e^3} = 0.041$

" $(-1) = 0.002$

" $(3) = 0.11$

$$= \begin{bmatrix} 0.84 \\ 0.04 \\ 0.002 \\ 0.11 \end{bmatrix} \Rightarrow X = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix}_{(786,1)}$$

$\therefore z_1 = z \quad , z_2 = 0$

$$\frac{e^z}{e^{z_1} + e^{z_2}} = \frac{e^z}{e^z + e^0} = \frac{e^z}{e^z + 1}$$

(Sigmoid)

$\Rightarrow$ z/a $\Rightarrow$ $a = P(y=1/x)$

: For two neurons

: $1 - a$

* $\mathcal{L}(\hat{y}, y) = -\left(\sum_{i=1}^{4} y_i \log \hat{y}_j\right)$

$= -[y_1 \log \hat{y}_1 + y_2 \log \hat{y}_2 + y_3 \log \hat{y}_3 + y_4 \log \hat{y}_4]$

* **Label:**

star, cloud, moon, other

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

(For Star) (for cloud) (for Moon)

* **Loss for cloud:**

$$\mathcal{L}\left(\begin{bmatrix} 0.84 \\ 0.04 \\ 0.002 \\ 0.11 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}\right) = -\log(0.002)$$

: **For star:**

$$\mathcal{L}(\hat{y}, y) = -\log(0.84)$$

⇒ For 1000 example:

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(3)} \\ | & | & & | \end{bmatrix}$$

$(784, 1)$

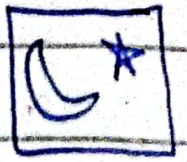
$$Y = \begin{bmatrix} 1 & 0 & & 0 \\ 0 & 0 & & 1 \\ 0 & 1 & & 0 \\ 0 & 0 & & 0 \end{bmatrix}$$

$$\Rightarrow J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \left[\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) \right]$$

$$= \frac{1}{m} \sum_{i=1}^{m} \left[- \sum_{j=1}^{c} y_j^{(i)} \log \hat{y}_j^{(i)} \right]$$

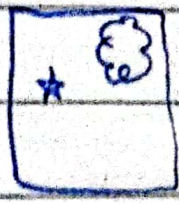
$$= \frac{\partial J(w,b)}{\partial a}$$

⊙ Multi-Label Classification:

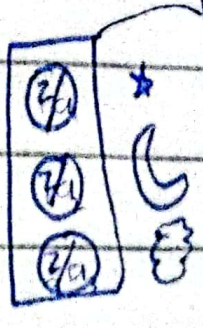


⇓ Classes = C = 3

$$\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$



$$\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$



$a_1 = \frac{1}{1+e^{-z_1}}$

: Here we use sigmoid

: Here Every class in independent of others

$$\Rightarrow \mathcal{L}(\hat{y}, y) = \sum_{j=1}^{C} \left[y_j \log \hat{y}_j + (1-y)_j \log (1-\hat{y}_j)_j \right]$$

$$y = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \quad \hat{y} = \begin{bmatrix} 0.6 \\ 0.7 \\ 0.4 \end{bmatrix}$$

$$\mathcal{L}(\hat{y}, y) = -1.14$$

* Next Lecture (PyTorch)

25/03/25

Deep Learning Libraries:

: Torch, PyTorch (Jump for Exam), TensorFlow (Mature language)

Theano, Keras (Easy)

: model.fit (X, Y, 200)

Sum = Sum + dw

→ PyTorch Tensor:

: Anaconda Environment and install PyTorch in it

* specialized version of numpy

→ CPU (4-16) # cores (Parallel Processing)

→ GPU (1000's) (Computational processors more required)

: Multiple Processors for convolutional operations.

→ TPU (Tensor Processing Unit)
(Dedicated for Deep Learning)
(Millions Cores)

✱ Implementation:

⊙ Torch: import Torch

x1 = torch.rand(2, 3)

x2 = torch.tensor([[1,2,3],[4,5,6]])

x2.size();

y = torch.randn(4, 4)
(Normal distribution)

x[:, 1]
(all rows) (first column)

x.flatten || x.view(16)

x.view(8, -1)
(4x4/8 = 2)

→ x = torch.randn(1, 4, 32, 24)

y = x.view(8, 2, -1, 3, 8)

x.size();

: $\frac{1 \times 4 \times 32 \times 24}{8 \times 2 \times 3 \times 8} = 8$

→ Matrix Multiplication:

res = torch.mm(mat1, mat2)

: Search by yourself:

? mat1 @ mat2
mat1.dot(mat2)
torch.dot(mat1, mat2)

→ Batch Matrix Multiplication:

b1 = [□ 3×4, □ 3×4, □ 3×4, ...] (10, 3, 4)

b2 = [□ 4×5, □ 4×5, □ 4×5, ...] (10, 4, 5)

torch.bmm(b1, b2)

↓

(Batch Matrix Multiplication)

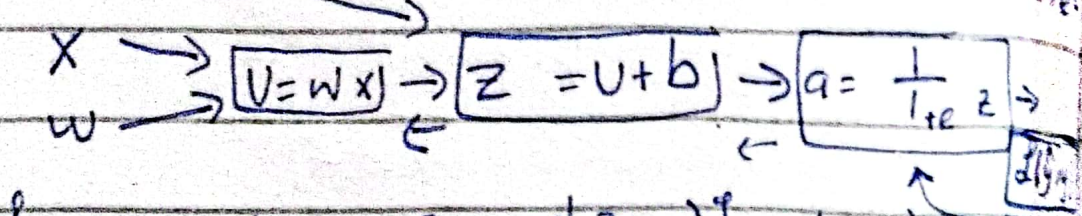
⊙ Computational Graph:

→ Backward Propagation computation efficient

: $X \longrightarrow (z/a) \rightarrow \hat{y}$: $z = w^T x + b$

$a = \sigma(z) = \frac{1}{1+e^{-z}}$

: Forward Pass



$\frac{\partial J}{\partial w} = dw$ $X = \begin{bmatrix} y^{(1)} \\ y^{(2)} \end{bmatrix}$ $dz = \frac{\partial J}{\partial z}$ $da = \frac{\partial J}{\partial a}$

$X = \begin{bmatrix} x^{(1)} & x^{(2)} \\ \vdots & \vdots \end{bmatrix}$ $w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$

: Tensor & Numpy Difference

w = torch.randn(2, 1, req_Grad = true

(only performed in torch)

: p = torch.sigmoid(torch.mm(x, w)+b

:

p.torch.backward().

⊙ Training Procedure:

→ Build Neural Networks using PyTorch

→ Forward (nn.Module (parent will be inherited from it))

Implement ② functions

① Constructor Write ② Forward Function.

: Anaconda (PyTorch Installation

26/3/25

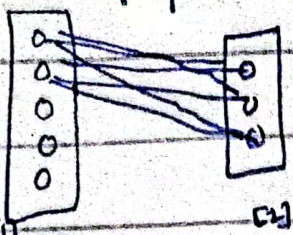
$$z^{[l]} = w^{[l]} A^{[l-1]} + b$$: Assignment

$$z = (np.dot(w, A) + b)$$

⊙ Python API to Interact with PYTORCH FrameWork

- Torch
  - nn (module)
    - functional
  - optim (Gradient Desc.)
  - util.data
    - Data Loader
    - Dataset (__getitem__)
- TorchVision (more info about CV)
  - datasets
    - MNIST (Digits)
  - Models
  - Transforms
    - ① ② ③
    - E T L
    - (Extract) (Transform) (Load)
    - (Download)

FC / Linear / Dense



Suppose [1] n=50 [2] n=10

→ Transform. to tensor ⇒ [0 — 1]

Transform number

[-1 1]

μ

σ

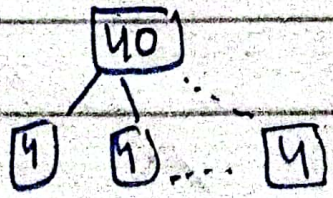
→ $X = \frac{X-\mu}{\sigma} = \frac{0-0.5}{0.5} = -1$

$= \frac{1-0.5}{0.5} = 1$

: On zero centric data transformation fast

→ Dataset and Model will be on same device (CPU or GPU)

→



10 iteration = 1 Epochs

→ Total training samples = 80000

Batch size = 50

Each iterations take 30 secs

Hours for 3 epochs of training

50 ← 30 sec

1 — $\frac{30}{50}$

80000 — $\frac{30}{50} \times 80000$ (into mins)

$\downarrow$

3 x epochs

: 40 Hours