



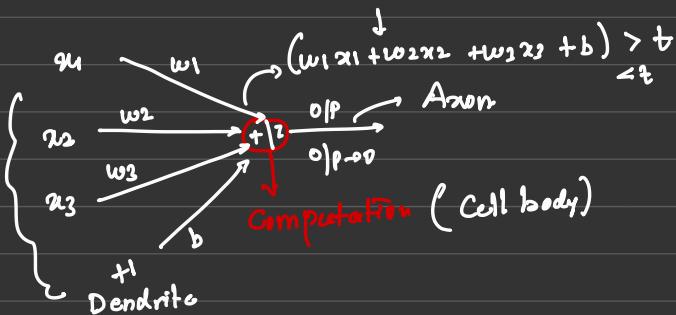
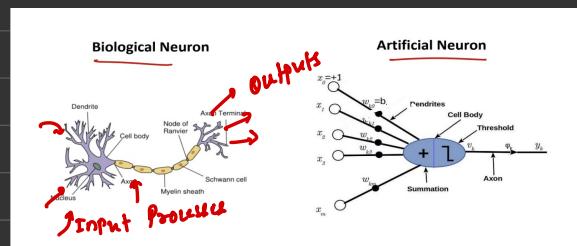

Perceptron

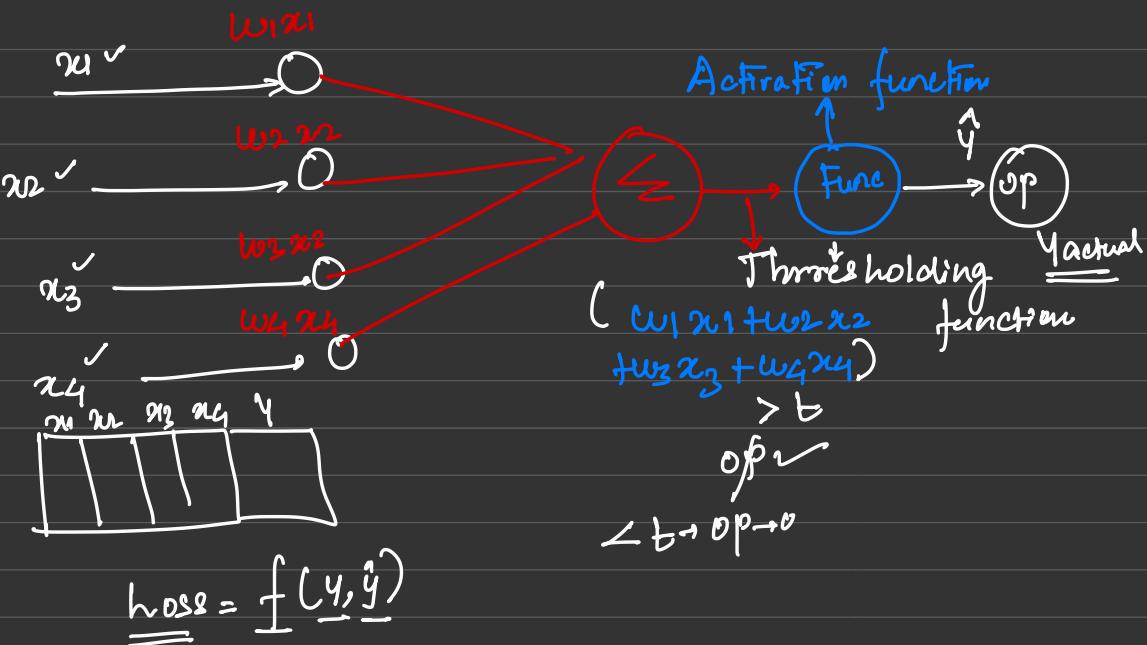
Biological Neuron



axon
myelin
nodes
dendrites

Artificial Neuron





We need to find the best set of weights which bring our \hat{y} closer to y .

find the best set of weights which minimise the loss.

Weight update rule for a neuron

$$\rightarrow \boxed{w_{new} = w_{old} - \alpha \times \frac{\partial L}{\partial w}}$$

when our loss = mLE

$$w_{new} = w_{old} + \alpha \left(\frac{y - \hat{y}}{x} \right)$$

target - predicted
input value

AND GATE

Objective: learn a perceptron
weights such that it can
function like an AND Gate

A	B	Output
0	0	0
1	0	0
0	1	0
1	1	1

$$w_1 = 1.2 \\ w_2 = 0.6$$

$$\text{when} \\ = 1.2 + 0.6(0-1) \times 1 \leq A \\ = 1.2 - 0.6 \\ = 0.6$$

$$(w_1 A + w_2 B) > 1$$



$$w_{1\text{new}} = 0.6 + 0.5(0-1) \times 0 \leq B \\ = 0.6$$

Case 1) $A \rightarrow 0, B \rightarrow 0$

$$1.2 \times 0 + 0.6 \times 0 \rightarrow 0 < 1 \\ \rightarrow 0$$

Case 2) $A \rightarrow 1, B \rightarrow 0$

$$1.2 \times 1 + 0.6 \times 0 = 1.2 > 1 \\ \rightarrow 1$$

Actual $\rightarrow 0$.

Case 3)

$A \rightarrow 0, B \rightarrow 1$

$$1.2 \times 0 + 1 \times 0.6 \\ = 0.6 < 1$$

$$= 0 \checkmark$$

Case 4) $A \rightarrow 1, B \rightarrow 1$

$$1.2 \times 1 + 0.6 \times 1 = 1.8 > 1 \\ \rightarrow 1$$

New weights
 $w_{1\text{new}} = 0.7$
 $w_{2\text{new}} = 0.6$

$$(w_1 \times A + w_2 \times B) > 1$$

Case 1)
 $A \rightarrow 0, B \rightarrow 0$

$$0.7 \times 0 + 0.6 \times 0 = 0 \checkmark$$

Case 2)

$A \rightarrow 1, B \rightarrow 0$

$$0.7 \times 1 + 0.6 \times 0 = 0.7 < 1$$

$$= 0 \checkmark$$

Case 3) $A \rightarrow 0, B \rightarrow 1$

$$0.7 \times 0 + 0.6 \times 1 = 0.6 < 1 \\ = 0 \checkmark$$

Case 4) $A \rightarrow 1, B \rightarrow 1$

$$0.7 \times 1 + 0.6 \times 1 \\ = 1.3 > 1$$

$$\rightarrow 1$$

$$w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

$$\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} | \\ 1 \end{bmatrix} \cdot \begin{bmatrix} w_1 & w_2 w_3 & b \end{bmatrix} \begin{bmatrix} | \\ 1 \end{bmatrix}$$

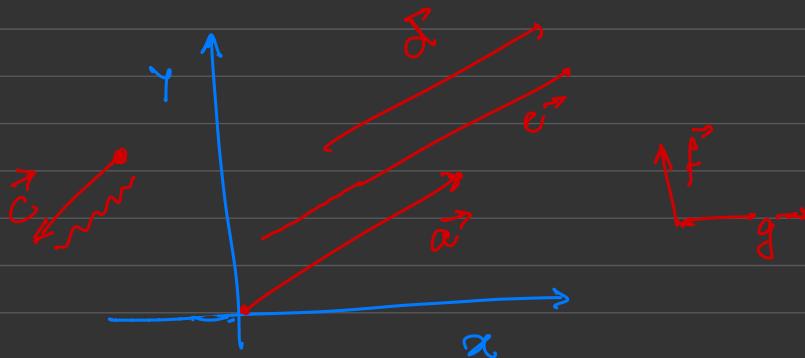
Concept of Mathematics

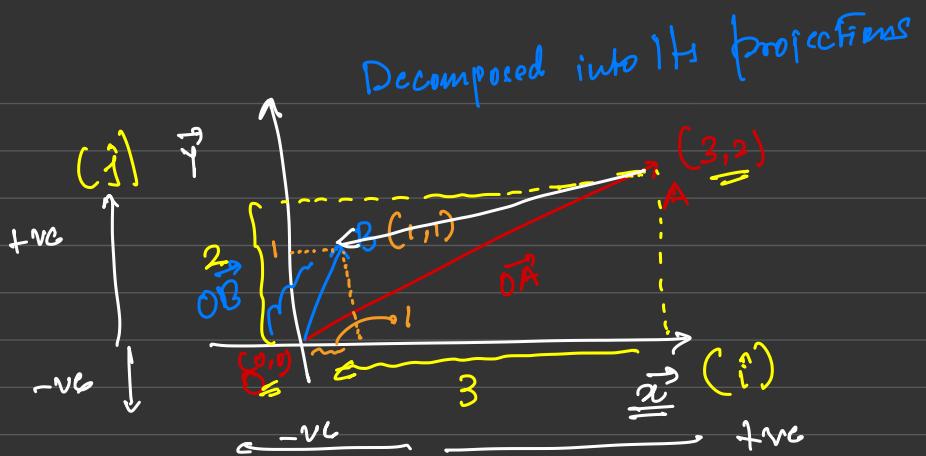
- ① Vectors
- ② Differentiation
- ③ Partial Differentiation
- ④ Gradient of a function
- ⑤ maxima & minima of a function

Vector: it is an object which has a magnitude and a direction

Person → O → North → 5 km/hr (velocity) → VecFor

Person → O → 5 km/hr (Speed) → Scalar





$$\vec{OA} = 3\hat{i} + 2\hat{j} + \hat{k}$$

$$\vec{OB} = \hat{i} + \hat{j} + \hat{k}$$

$$\vec{AB} = \vec{OB} - \vec{OA}$$

$$\vec{AB} = \hat{i} + \hat{j} - (3\hat{i} + 2\hat{j} + \hat{k})$$

$$\vec{AB} = -2\hat{i} - \hat{j} - \hat{k}$$

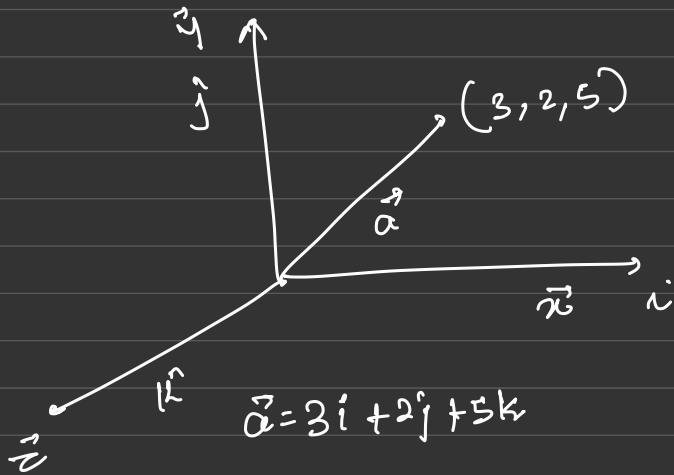
$$|\vec{AB}| = \sqrt{(2)^2 + (-1)^2 + (-1)^2}$$

$$|\vec{AB}| = \sqrt{5}$$

$$|\vec{x}| = \sqrt{a^2 + b^2}$$

$$\vec{x} = a\hat{i} + b\hat{j} + c\hat{k}$$

$$|\vec{x}| = \sqrt{a^2 + b^2 + c^2}$$

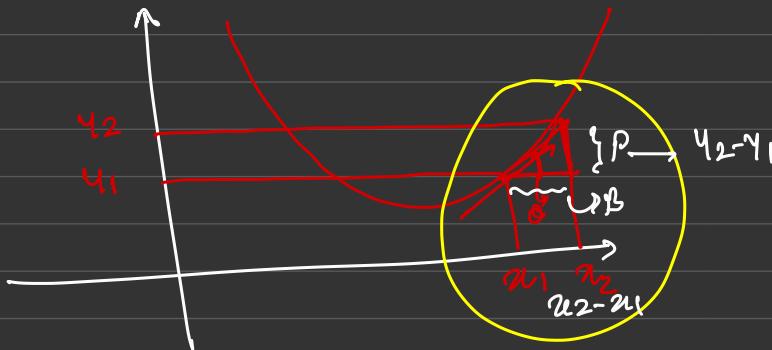


$$\vec{\alpha} = 3\vec{i} + 2\vec{j} + 5\vec{k}$$

Differentiation

$F(x)$

$\frac{dF(x)}{dx} \rightarrow$ rate of change of the function w.r.t to the variable



$$\tan \theta = \frac{P}{B}$$

$$\tan \theta = \frac{(y_2 - y_1)}{(x_2 - x_1)}$$

$$\underset{dx \rightarrow 0}{\underbrace{\frac{dF(x)}{dx}}}_{\Delta x \rightarrow 0} \quad \boxed{\frac{\Delta y}{\Delta x} = dy/dx}$$

$$\lim_{h \rightarrow 0} \frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

$$\lim_{h \rightarrow 0} \frac{f(x_1+h) - f(x_1)}{x_1+h - x_1}$$

Rules of Differentiation

① Power Rule.

$$f(x) = x^n$$

$$f'(x) = dy/dx = nx^{n-1}$$

$$f(x) = 5x^5$$

$$f'(x) = 5 \times \frac{d(x^5)}{dx}$$

$$= 5 \times 5x^4$$
$$= 25x^4$$

② Product Rule

$$h(x) = f(x) \times g(x)$$

$$\frac{d(\sin x)}{dx} = \cos x$$

$$\frac{d(\cos x)}{dx} = -\sin x$$

$$h'(x) = f(x) \times g'(x) + f'(x) \times g(x)$$

$$h(x) = (\overset{1}{x^4} + \overset{2}{2}) \cos x$$

$$h'(x) = -(\overset{1}{x^4} + \overset{2}{2}) \sin x + \cos x (4x^3)$$

$$h'(x) = -\sin x x^4 - 2 \sin x + 4x^3 \cos x$$

Partial Differentiation

$$f(x, y) = x^4 y$$

$$\frac{\partial f(x, y)}{\partial x} \quad / \quad \frac{\partial f(x, y)}{\partial y} \rightarrow \begin{array}{l} \text{differentiate the} \\ \text{function w.r.t } y \\ \text{keeping } x \text{ constant} \end{array}$$

differentiate the function
w.r.t x , keeping y constant

$$\frac{\partial}{\partial y} (\cancel{x^4} \cancel{y}) = x^4$$

$$\boxed{\frac{\partial (x^4 y)}{\partial x} = y \times 4x^3}$$

Gradient

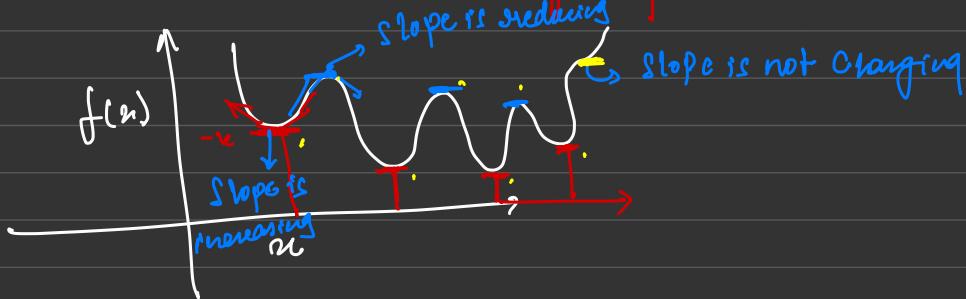
$$f(x, y) \rightarrow \begin{cases} \frac{\partial f(x, y)}{\partial y} \\ \frac{\partial f(x, y)}{\partial x} \end{cases}$$

$$f(x, y) = \cancel{2x^2} + 4y$$

$$\nabla f = \left[\begin{array}{c} \frac{\partial f(x, y)}{\partial x} \\ \frac{\partial f(x, y)}{\partial y} \end{array} \right] = \begin{bmatrix} 4 \\ 4 \end{bmatrix}$$

$$\nabla f = \begin{bmatrix} 4 \\ 4 \end{bmatrix}$$

Maxima & minima of a function



$$f'(x) = 0$$

Critical Points

- Points at which the slope of the function
→ i.e. the slope of the tangent at that point is zero.

Local Points → minima

$$f''(x) > 0$$

Local

$$\frac{d f'(x)}{dx}$$

$$\left\{ \begin{array}{l} f''(x) < 0 \rightarrow \text{maxima} \\ \text{or} \\ \text{Bliss} \end{array} \right.$$

$$f''(x) = 0 \rightarrow \text{inflection points}$$

Calculation of the maxima & minima
of a Bivariate Function.

$f(x, y)$ local maxima and local
minima points

$$P = \boxed{\frac{\partial f(x, y)}{\partial x}}$$

Partial differentiation

$$Q = \boxed{\frac{\partial f(x, y)}{\partial y}}$$

Solve for $P=0$ and $Q=0$
Critical Points.

$$R = \frac{\partial^2 f(x, y)}{\partial x^2}$$

$$S = \frac{\partial^2 f(x, y)}{\partial x \partial y}$$

$$T = \frac{\partial^2 f(x, y)}{\partial y^2}$$

for all the critical points (a, b)

if $(Rt - S^2 > 0)$ and

$R > 0$ (a, b) is a local minima

$R < 0$ (a, b) is a local maxima

$Rt - S^2 = 0$ test fails ∞

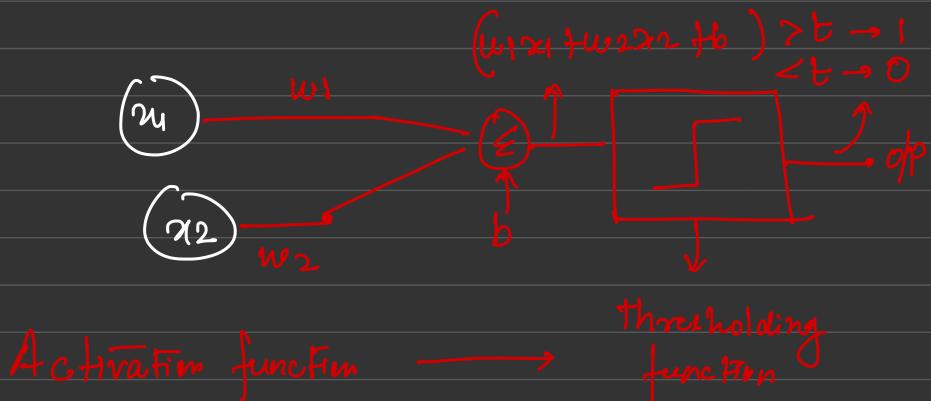
$Rt - S^2 \leq 0$ it's a saddle point

Activation Functions

XOR gate

A	B	O/p
0	0	0
1	0	1
0	1	1
1	1	0

Perceptron couldn't train the logic on the XOR



Activation function \longrightarrow thresholding function

Cannot capture non-linear patterns

Activation Functions:

- ① Sigmoid
- ② ReLU (Rectified Linear Unit)
- ③ Leaky ReLU
- ④ tanh (Hyperbolic tangent)
- ⑤ Softmax
- ⑥ Parametric ReLU

Sigmoid activation: Binary classification Problem

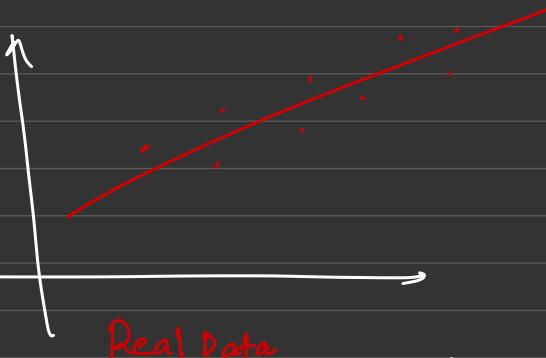
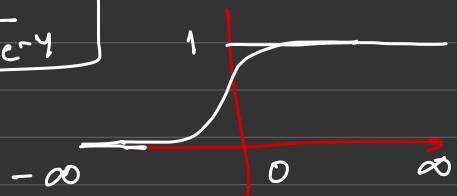
Vanishing function

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

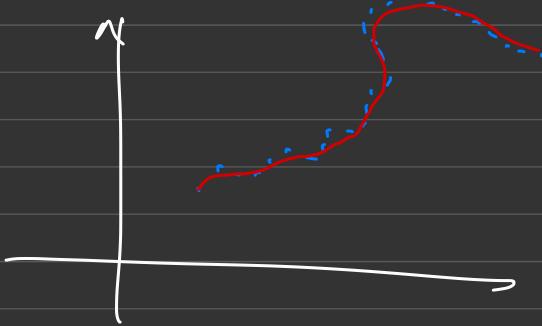
$$-\infty < x < \infty$$
$$0 < \sigma(x) < 1$$

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

$$\delta(y) = \frac{1}{1+e^{-y}}$$
$$(0-1)$$



Real Data



Derivative of the sigmoid

$$\boxed{w_{\text{new}} = w_{\text{old}} - \alpha \frac{\partial J}{\partial w}}$$

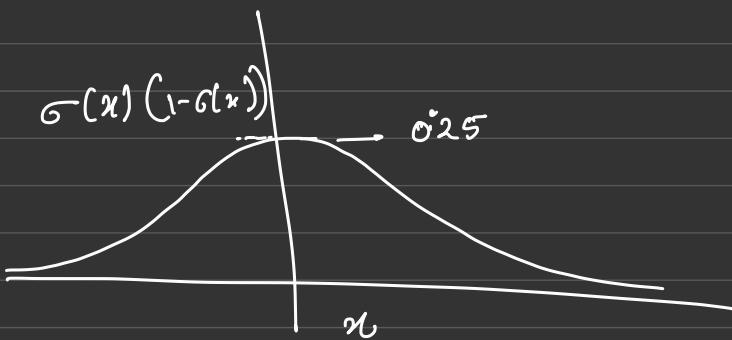
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\sigma'(x) = (1+e^{-x})^{-1}$$

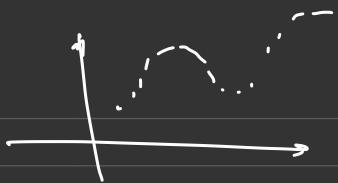
$$\sigma'(x) = (1+e^{-x})^{-2} \times (0+e^{-x})(-1)$$

$$\sigma'(x) = - (1+e^{-x})^{-2} \times e^{-x}$$

$$\sigma'(x) = \sigma(x) \times (1-\sigma(x))$$

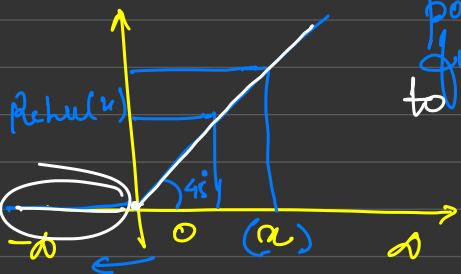


ReLU → Rectified linear Unit



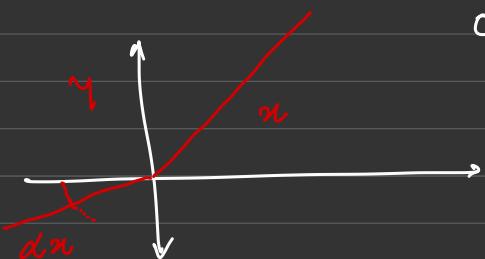
$$\text{ReLU}(x) = \max(x, 0) \quad \left\{ \begin{array}{l} x \in (-\infty, 0) \\ \text{ReLU} \in (0, \infty) \end{array} \right.$$

popularly used as an activation
function in multi-layer perceptrons
to bring in non-linearity.

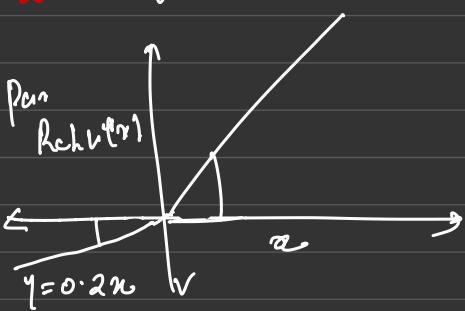


Parametric ReLU:

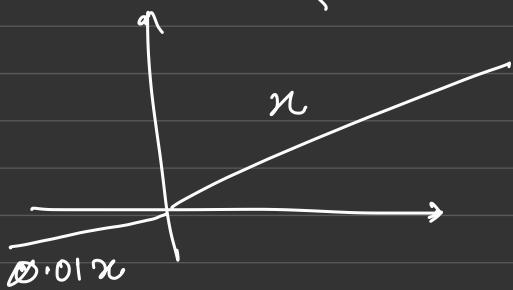
$$\text{ReLU}(x) \rightarrow \begin{cases} u & x > 0 \\ 0 & x = 0 \\ d/x & x < 0 \end{cases}$$



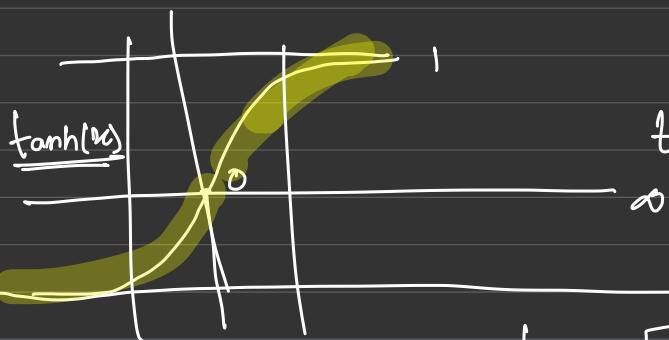
$0 < \alpha < 1$
Learned by the model.



$$\text{heavy ReLU}(x) \left\{ \begin{array}{l} x > 0 \rightarrow x \\ x = 0 \quad 0 \\ x < 0 \quad 0.01x \end{array} \right\} \rightarrow \text{heavy ReLU}$$



Hyperbolic Tangent (Tanh)

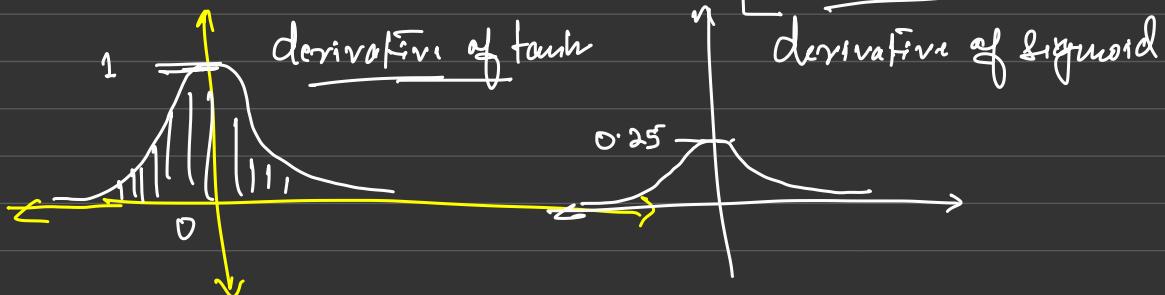


$$x \in (-\infty, \infty)$$

$$\tanh(x) \in (-1, 1)$$

$$\tanh'(x) = 1 - \tanh^2(x)$$

$$\tanh = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

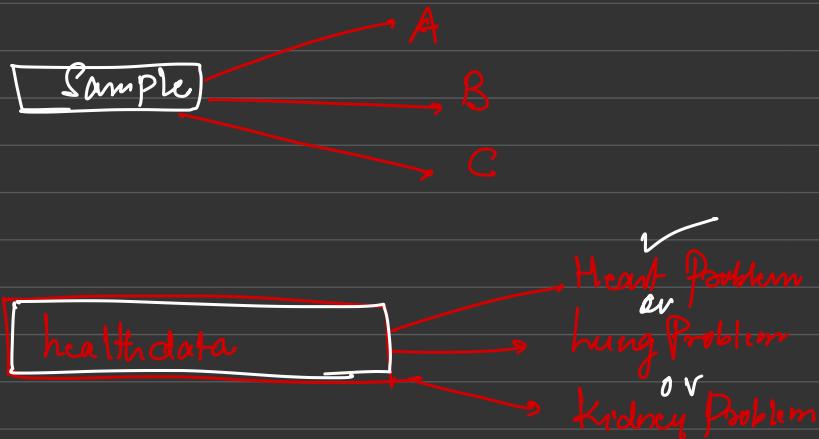


tanh is an activation function which is popularly used in the NN for creating non-linear features

Softmax activation function

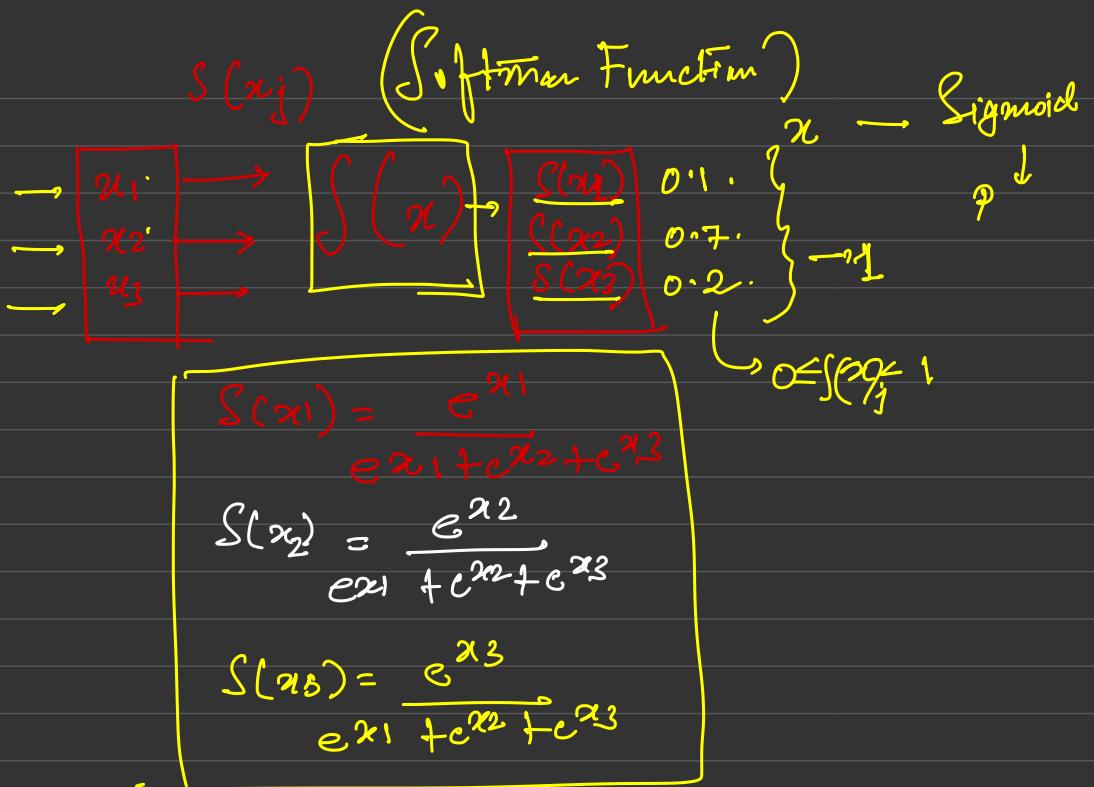
Special case of activation function

Multiclass classification Problem



Softmax function
basically a more generalised
form of sigmoid



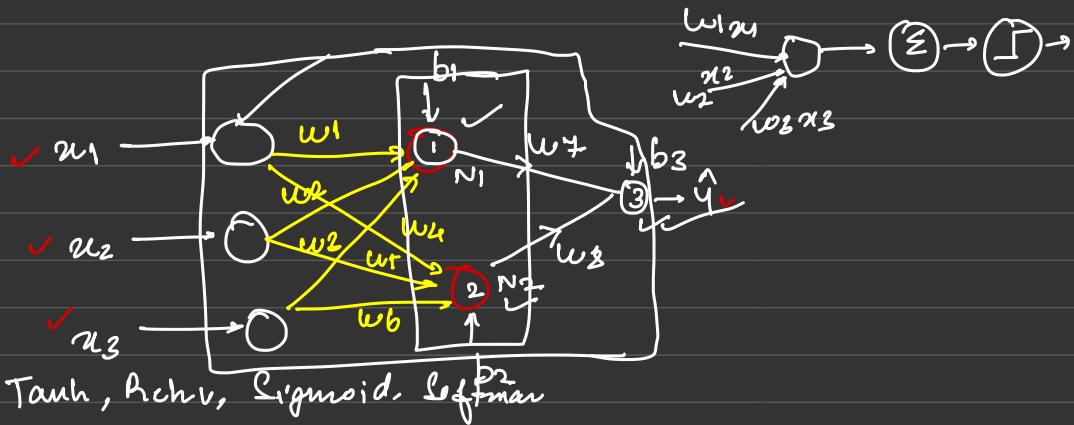


$$S(x_j) = \frac{e^{x_j}}{\sum_{i=1}^p e^{x_i}}$$

Given sample data \rightarrow A
 or
 B
 or
 C.
 one of the n classes
 $= 3$

Output in a multiclass classification problem

Forward and Backward propagation



$$N_1 = (w_1 x_1 + w_2 x_2 + w_3 x_3 + b_1)$$

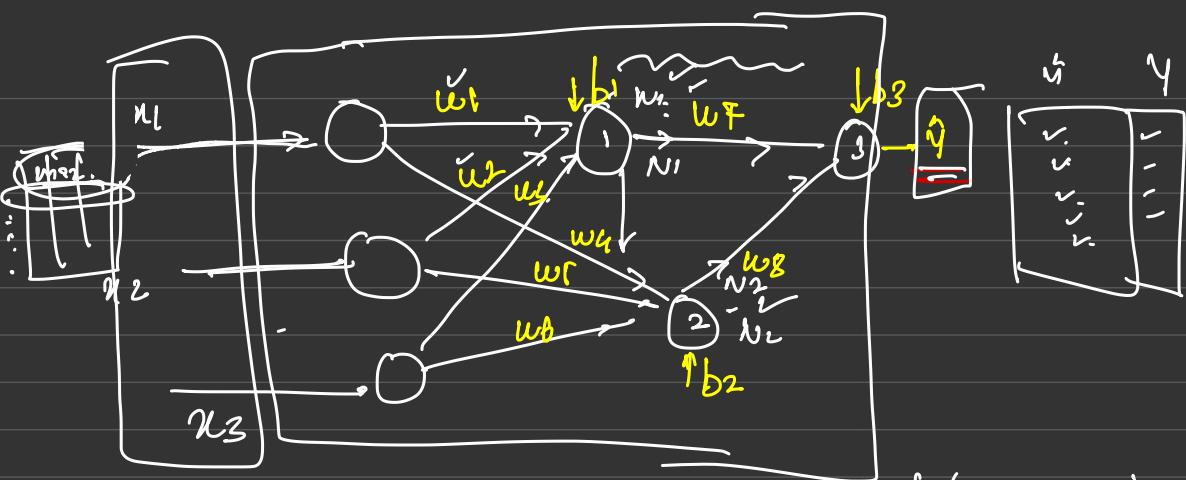
$$N_2 = (w_4 x_1 + w_5 x_2 + w_6 x_3 + b_2)$$

$$\hat{y} = (w_7 N_1 + w_8 N_2 + b_3)$$

x_1	x_2	x_3	y

$$\begin{aligned} y &\leftarrow f(x_1, x_2, \theta_3) \\ \hat{y} &\leftarrow \text{NLP}(\underline{x_1, x_2, x_3}) \end{aligned}$$

\hat{y} will achieve (\hat{y} becomes closer to y) by manipulating the weights and biases



loss of my neural network (narrp)

$$\text{loss} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

$$y \leftarrow f(x_1, x_2, x_3)$$

Objective: To manipulate my weights and biases to reduce the loss!

1) We are going to initialise my weights and biases randomly.

Forward propagation

2) Based on the weights and biases initialised the network will calculate \hat{y} .

$$N_1 = (w_1 x_1 + w_2 x_2 + w_3 x_3 + b_1)$$

$$N_2 = (w_4 x_1 + w_5 x_2 + w_6 x_3 + b_2)$$

$$\hat{y} = (w_7 N_1 + w_8 N_2 + b_3)$$

3) We will calculate the loss of the network

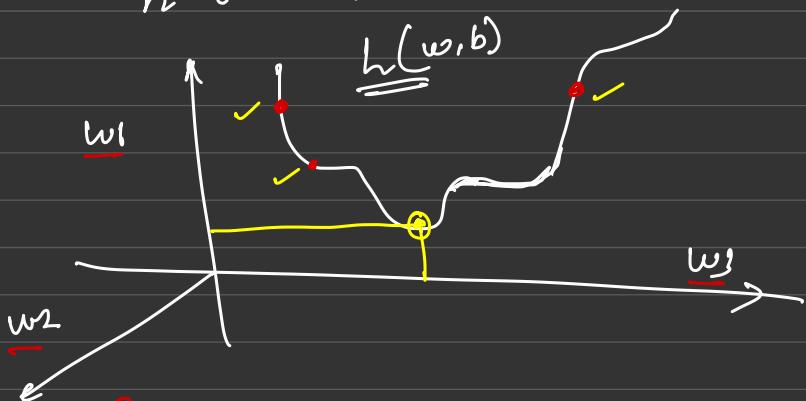
$$\sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Backward Propagation:

- To minimize the loss by manipulating the weights and biases

$$L = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

$$L = \frac{1}{n} \sum_{i=1}^n (y_i - (w_1 x_1 + w_2 x_2 + b))^2$$



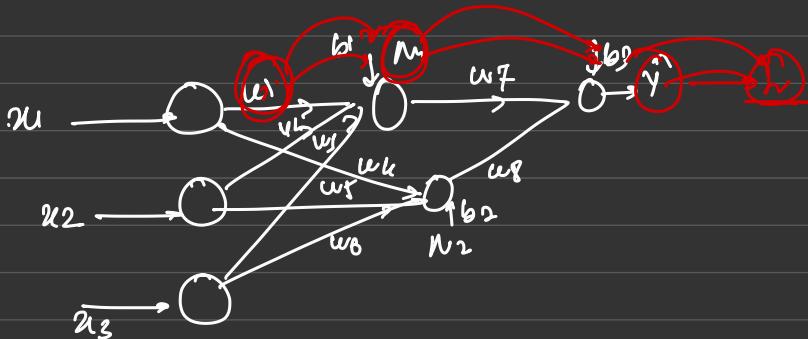
Gradient Descent -



$$w_1(t+1) = w_1(t) - \eta \frac{dL}{dw_1}(w_1)$$

$$\frac{\partial h}{\partial w_1}, \frac{\partial h}{\partial w_2}, \frac{\partial h}{\partial w_3}, \dots, \frac{\partial h}{\partial w_p}$$

$$\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \dots, \frac{\partial L}{\partial w_p}$$



$$h = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

$$h = \frac{1}{n} \sum_{i=1}^n (y_i - (w_7 N_{1i} + w_8 N_{2i} + b_3))^2$$

$$\frac{\partial L}{\partial w_7} = \frac{1}{n} \sum_{i=1}^n -2 (y_i - (w_7 N_{1i} + w_8 N_{2i} + b_3)) N_{1i}$$

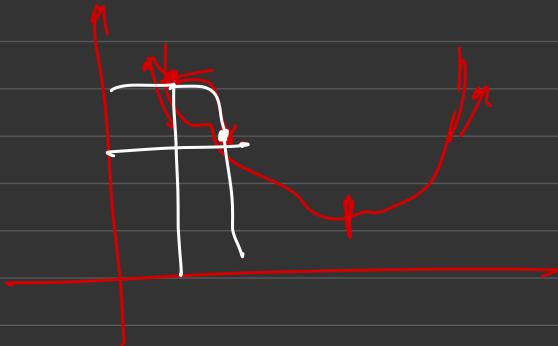
$$\frac{\partial h}{\partial w_8}$$

$$\frac{\partial h}{\partial w_1} = \text{Chain Rule}$$

$$\boxed{\frac{\partial h}{\partial w_1}} = \frac{\partial h}{\partial N_1} \times \frac{\partial N_1}{\partial w_1}$$

$$\boxed{\frac{\partial L}{\partial w_2}} = \boxed{\frac{\partial L}{\partial w_7}} \times \boxed{\frac{\partial w_7}{\partial w_2}}$$

$$\boxed{N_1} = (w_1 x_1 + w_2 x_2 + w_3 x_3 + b_1)$$

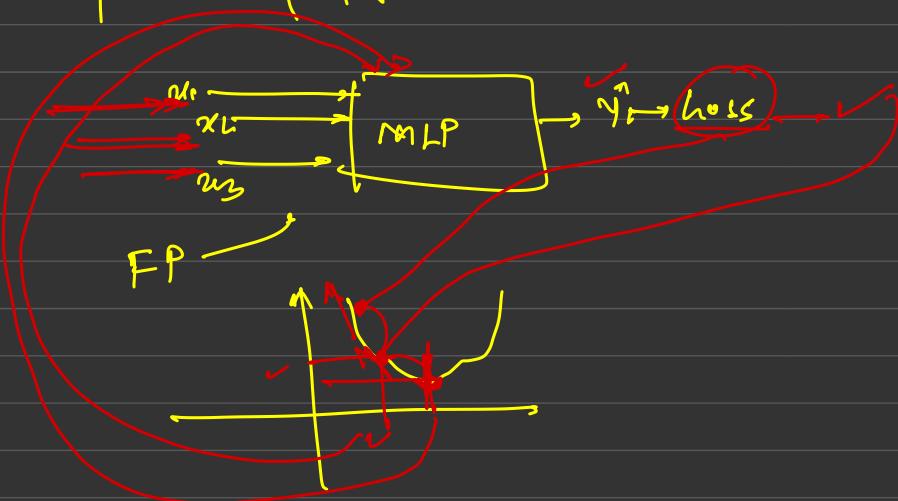


We initialize the weights and biases randomly.

Given these weights and biases we calculate the loss by propagating through network \rightarrow Forward propagation

Given the loss we calculate the derivative w.r.t to all the weights and biases and do the gradient descent to get new set of weights and biases

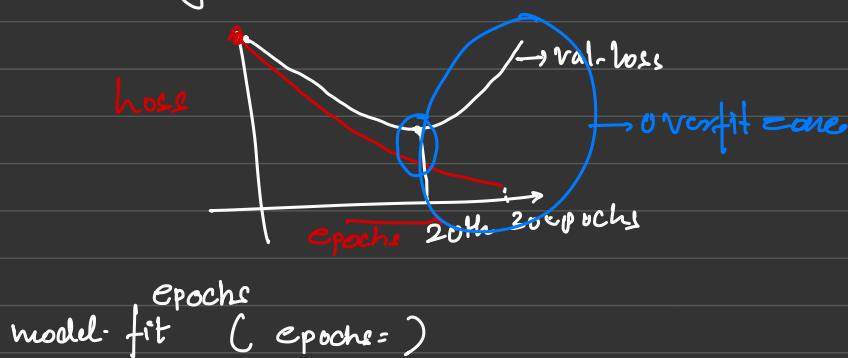
Backward Propagation



Callbacks

- ① Early stopping callback
- ② model checkpoint callback.
- ③ Tensorboard callback

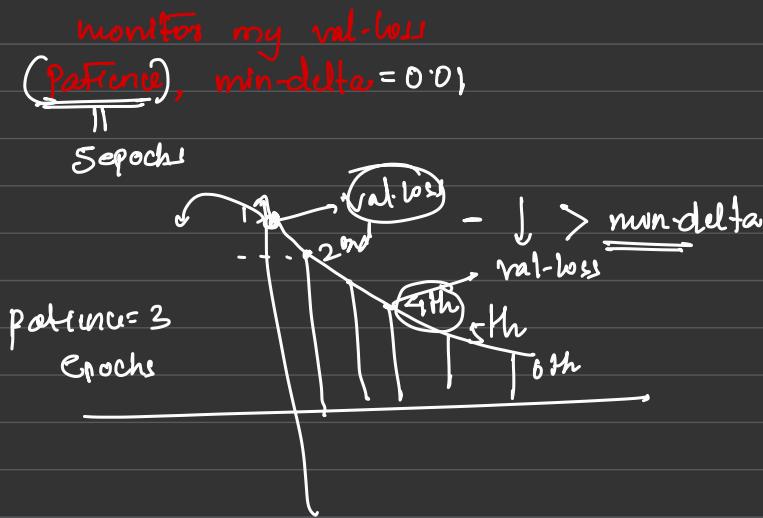
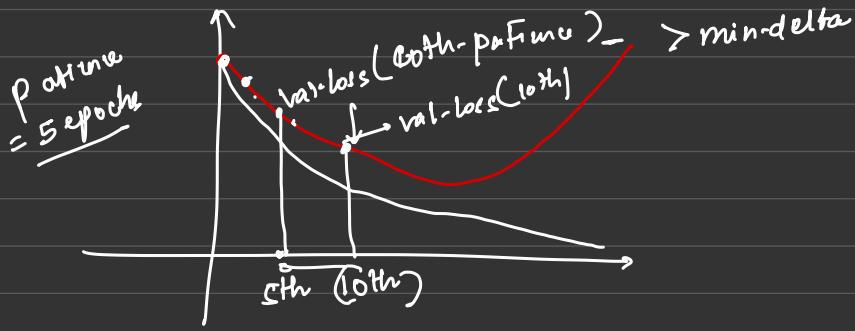
Early Stopping Callback



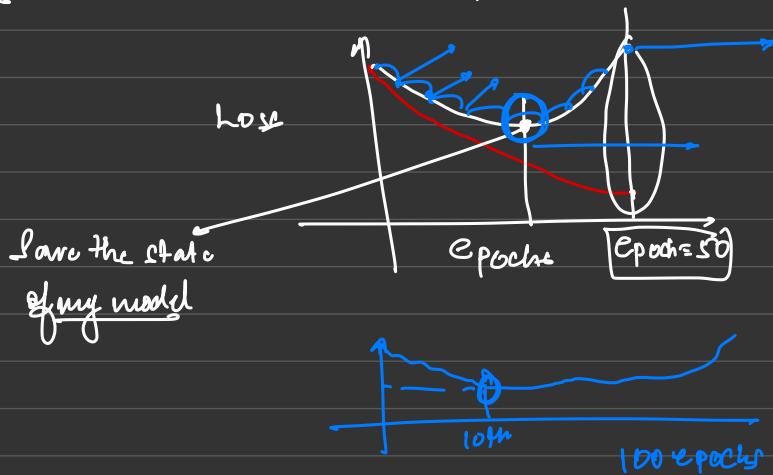
Early Stopping Callback

function that can monitor
the improvement on a metric
can stop the training if it
does not improve beyond
a certain threshold

Early Stopping → min_delta
Patience



(model checkpoint callback.)



Early Stopping and model checkpoint →
not to monitor 100 epochs

Tensorboard call back -

Loss Functions

Regression losses

- Mean Squared Error
- Mean Absolute Error
- Huber loss
- Pseudo Huber loss

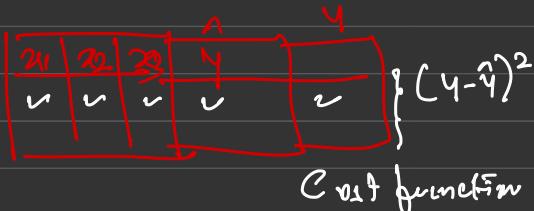
Classification losses

- Binary Cross Entropy
- Categorical Cross Entropy
- Sparse Categorical Cross Entropy
- Hinge loss

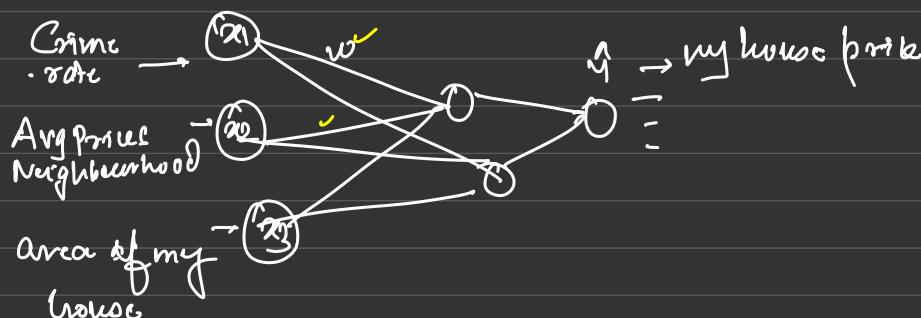
Mean Squared Error Loss.

$$\geq \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Cost vs loss



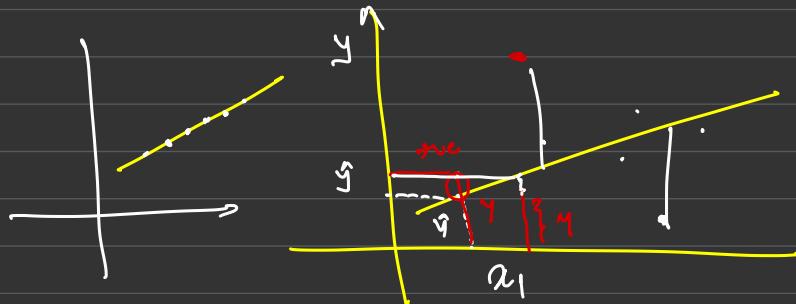
Cost function



	x_1	x_2	x_3	y	\hat{y}	$(y - \hat{y})^2$
v	v	v	v	90	70	400
v	v	v	v	100	120	400
v	v	v	v	110	100	100

$$loss = \sum_{i=1}^n \frac{1}{n} (y - \hat{y})^2$$

$$\frac{900}{3} = 300$$



$$(y - \hat{y})^2$$

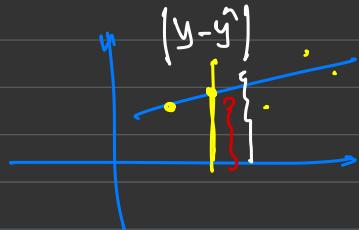
$$\underline{(y - \hat{y})}$$



MSE is susceptible to outliers

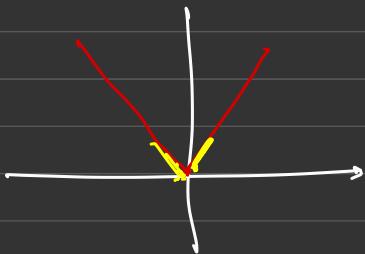
Mean Absolute error

$$\leq \sum_{i=1}^n \frac{1}{n} |(y_i - \hat{y}_i)|$$



x_1	x_2	y	\hat{y}	Loss
✓	✓	20	25	$ 20-25 = 5$
✓	✓	50	72	$ 50-72 = 22$
✓	✓	70	90	$ 70-90 = 20$

$47/3$



Huber loss

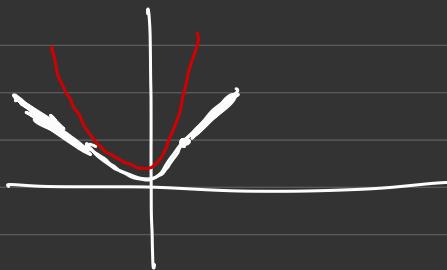
$$= \begin{cases} \frac{1}{2} (y_i - \hat{y}_i)^2 & \text{if } |y_i - \hat{y}_i| \leq \delta \\ \delta |y_i - \hat{y}_i| - \frac{1}{2} \delta^2 & \text{if } |y_i - \hat{y}_i| > \delta \end{cases}$$

$\delta \rightarrow$ threshold parameter that determines the point of transition from quadratic to linear loss

Pseudo Huber loss!

$$\delta^2 \left(\sqrt{1 + \frac{(y_i - \hat{y}_i)^2}{\delta^2}} - 1 \right)$$

make the Huber loss differentiable



Classification losses.

Binary Cross Entropy loss:

x_1	x_2	y	\hat{y}
✓	✓	1	0.7
✓	✗	1	0.6
✗	✓	0	0.1

$$\log_{10} (\text{BCE}) = - [y_i \times \log(\hat{y}_i) + (1-y_i) \times \log(1-\hat{y}_i)]$$

$$= 1 \times \log(0.7) + 0 \times \log(0.3)$$

$$= 0.15 \quad \checkmark$$

$$- [1 \times \log(0.3)] + 0 \times \log(0.7)$$

$$= 0.52 \quad \checkmark$$

$$- 1 \times \log(0.9) + 0 \times \log(0.1)$$

$$= \underline{\underline{0.04}}$$

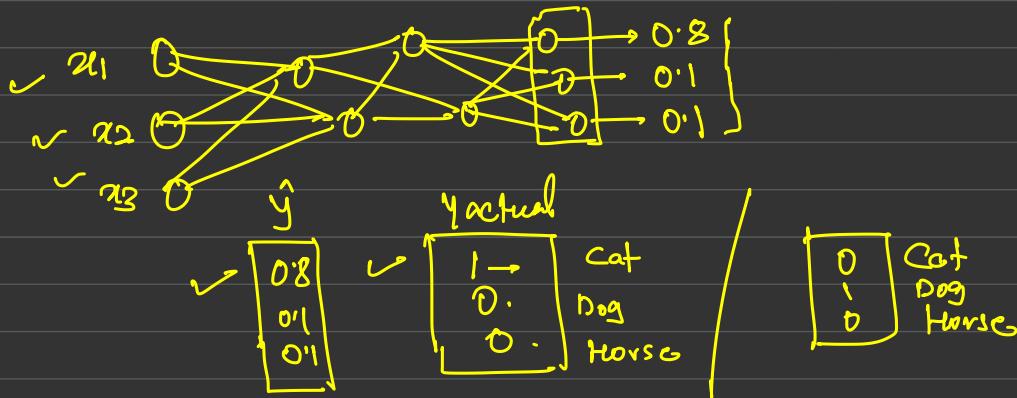
$$- y_i \times \log \hat{y}_i + (1-y_i) \log (1-\hat{y}_i)$$

$$- [0 + 1 \times \log(0.9)]$$

$$= \underline{\underline{0.04}}$$

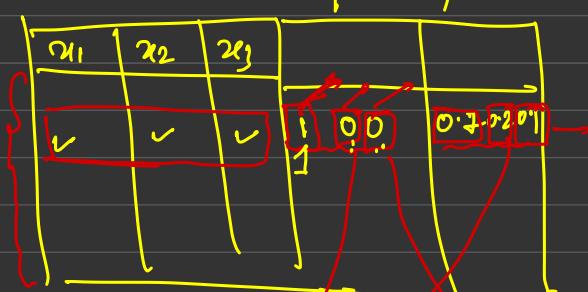
Categorical cross entropy

multi classification problems

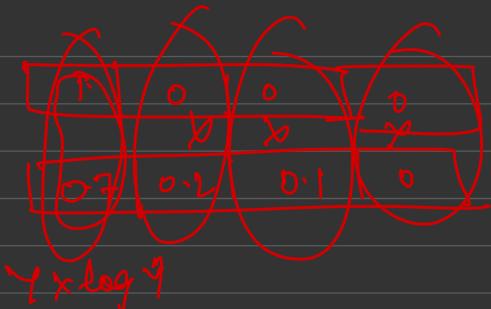


$$\log_s b/w(y, \hat{y})$$

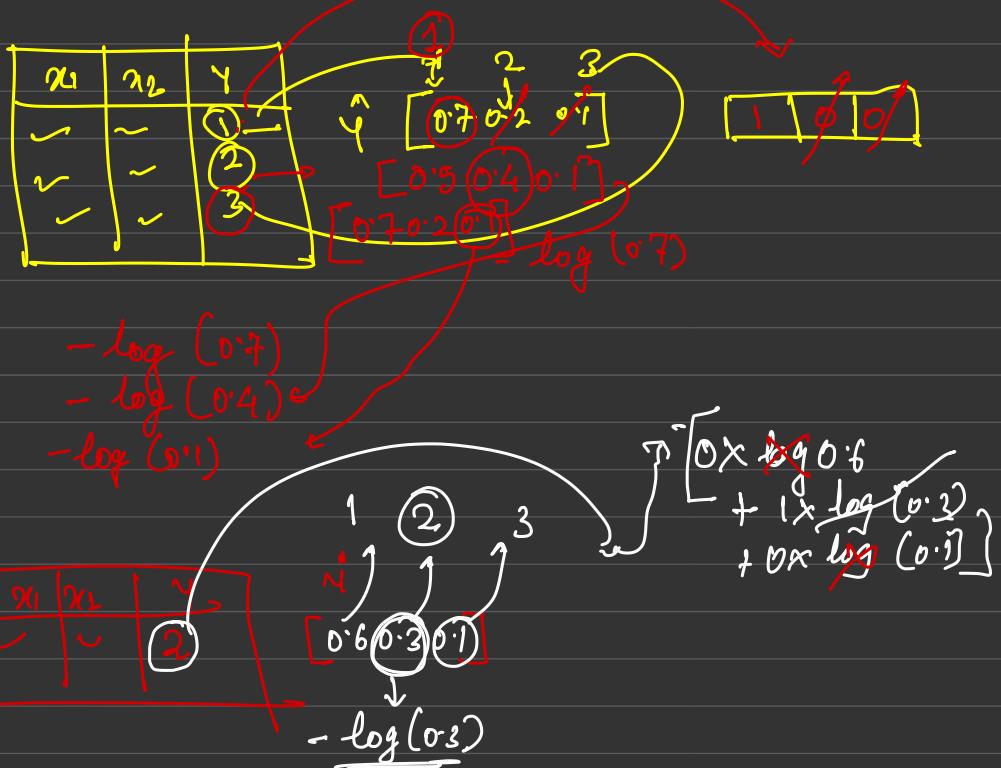
$$-\sum_{i=1}^n \sum_{j=1}^p y_{ij} \log \hat{y}_{ij}$$

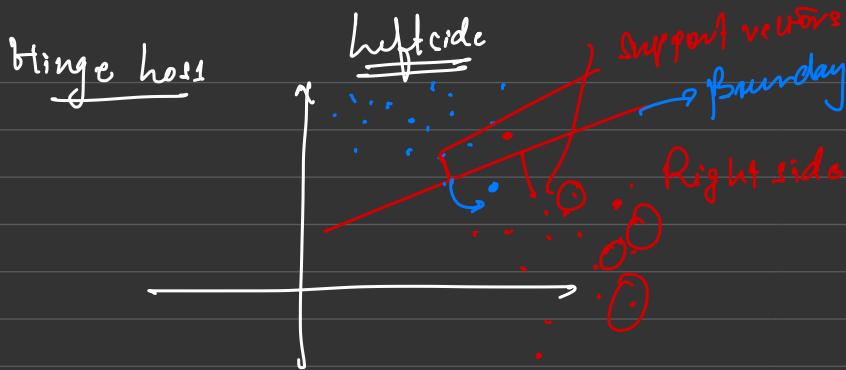


$$-\left[1 \times \log(0.7) + 0 \times \log(0.2) + 0 \times \log(0.1) \right] = 0.223$$



Sparse Categorical CrossEntropy

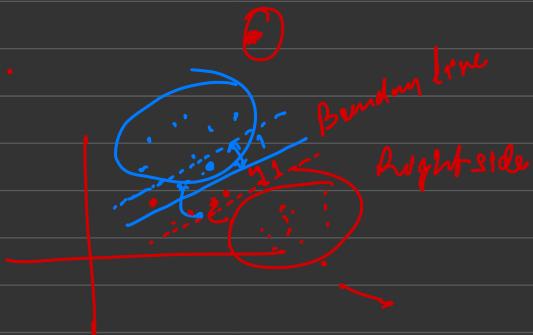
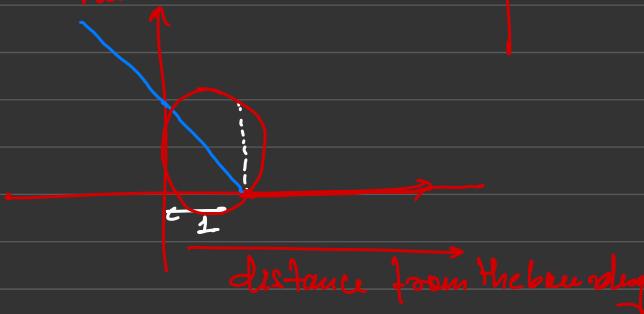




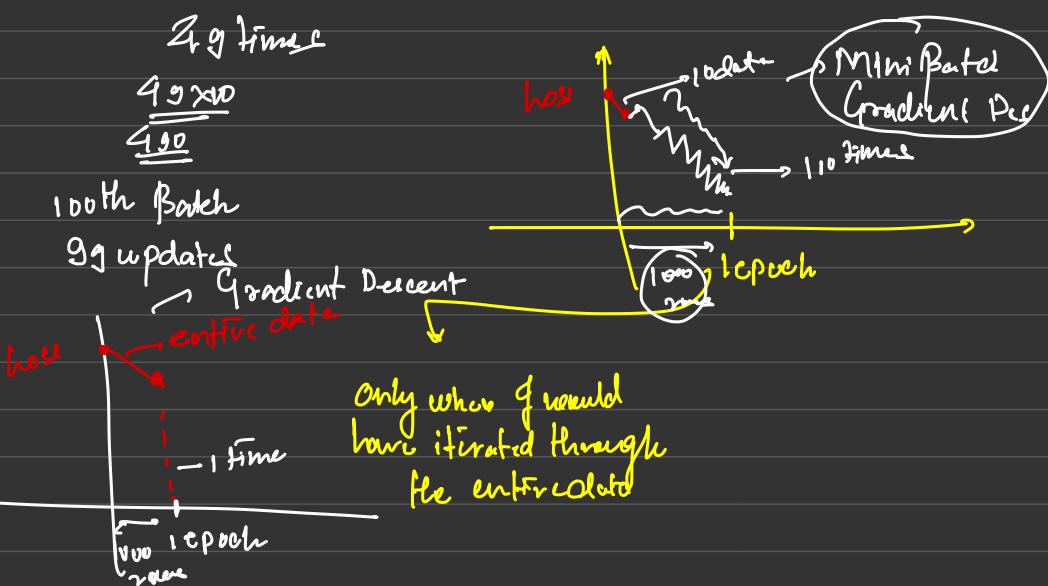
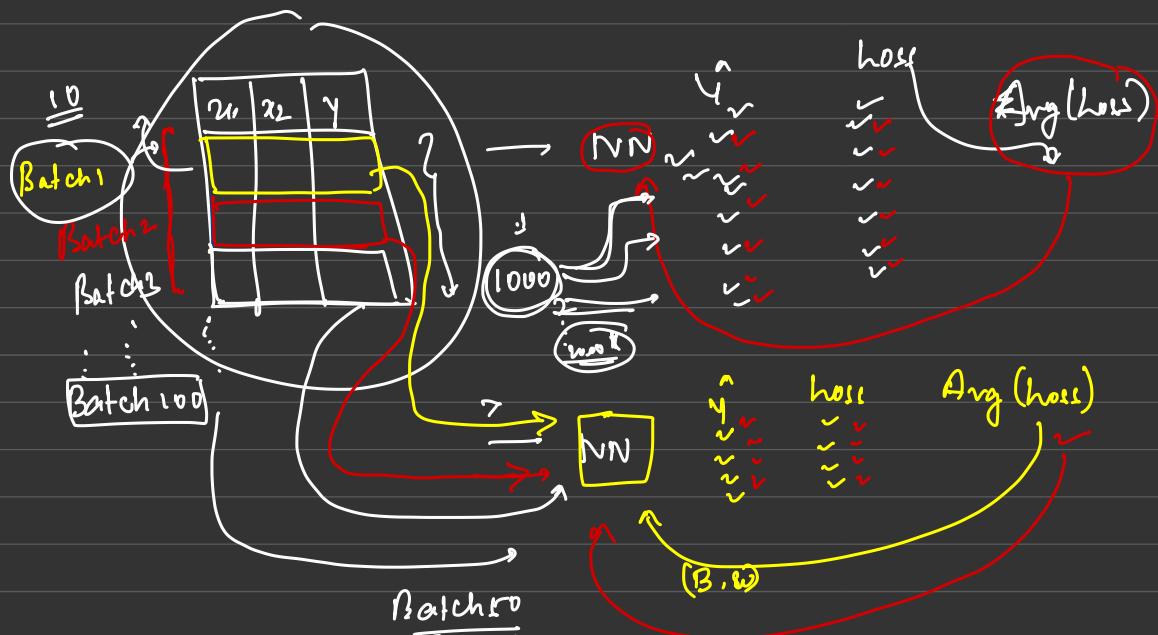
$$\max(0, 1 - y_i \cdot g)$$

$$y \in [-1, 1]$$

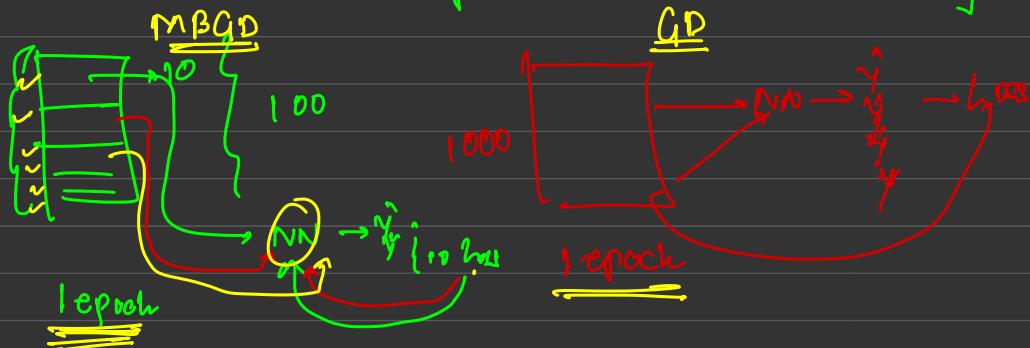
Hinge loss



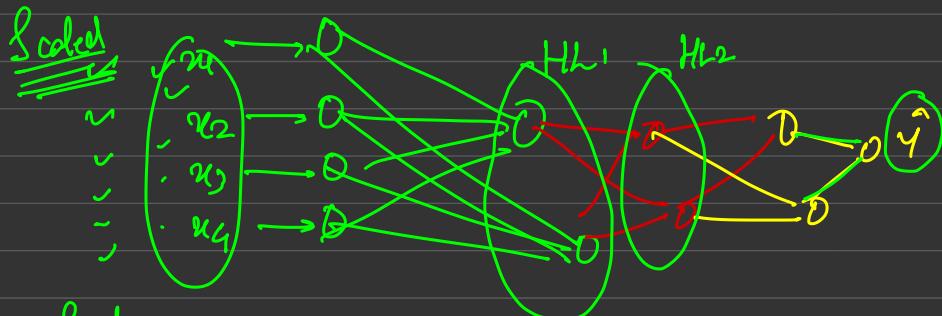
Batch Normalization



In a mini batch gradient descent, we divide the total data set into mini batches → for eg if my total no of samples is lets say 1000, and I choose the batch-size = 10, then I will have 100 batches at my hand.



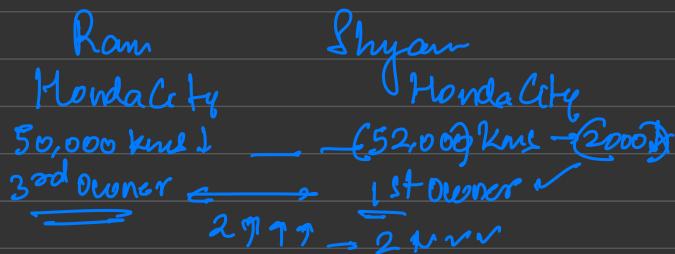
Batch Normalization



Scale our inputs

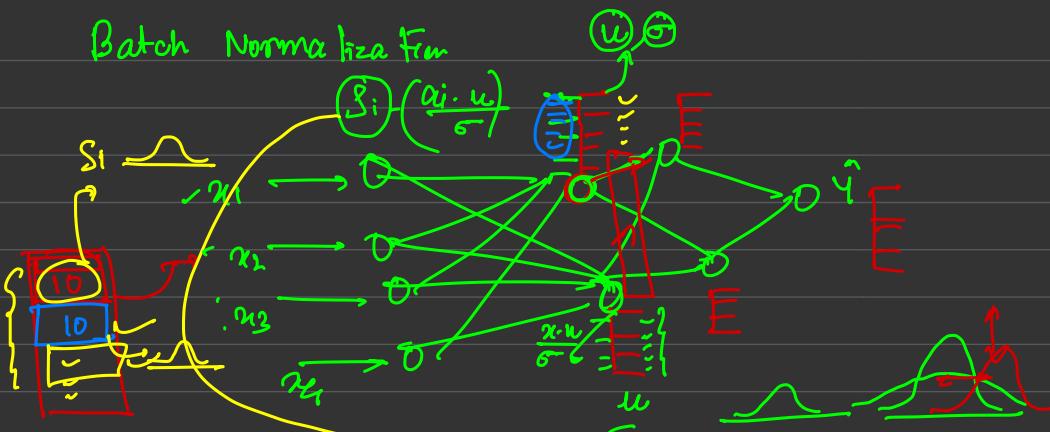
$$y = \beta_0 + (\beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3)$$

β_2



Batch Normalization

Batch Normalization



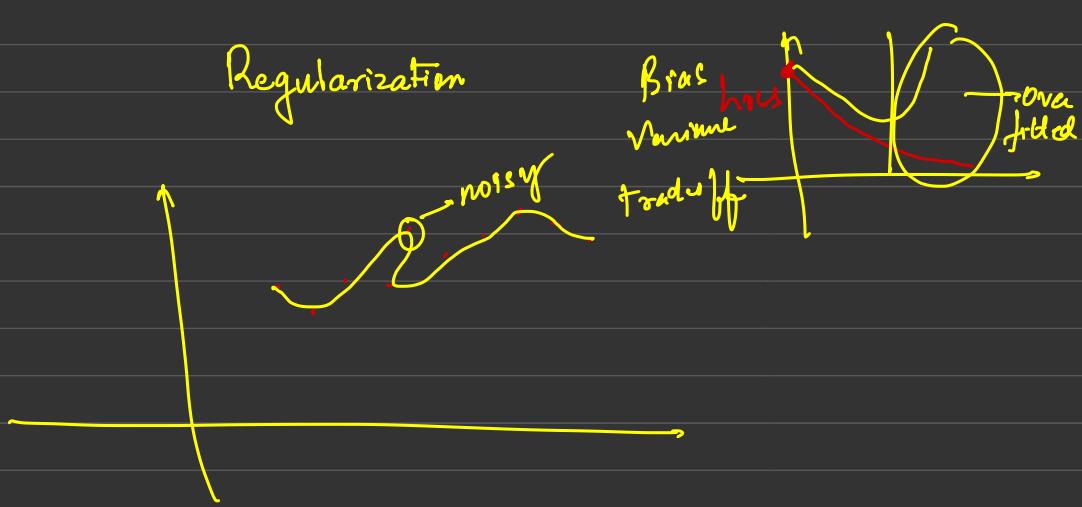
Putting too much of a constant
to my output

$$N_i = \gamma S_i + \beta$$

Learnable parameters

Beta

Gamma



high robust (Variance High)

model changes predictions rapidly
given slight changes in data

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

① $y = [100 + \underbrace{\beta_1 x_1}_{\Delta x_1=1} + \underbrace{\beta_2 x_2}_{\Delta x_2=1}] \rightarrow \text{how variance}$

$\Delta y = 5, \Delta x_2 = 1, \Delta y = 2$

$\underline{\underline{y}} = 100 + \underline{\underline{\beta_1 x_1}} + \underline{\underline{\beta_2 x_2}} \rightarrow \text{High variance}$

if β_1 see a unit change in x_1 ,
 100 units change in y .

" " " " $\rightarrow \beta_1$

$1000 \text{ in } y \sim$

To control / reduce the variance of
 α model, we can say that
 β_1 's should not inflate

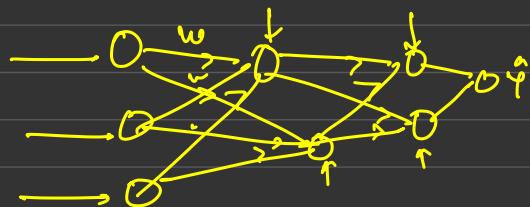
while training \rightarrow the abs $|\beta| \uparrow$

Very high abs $|\beta| \rightarrow$ high variance
 \downarrow weights and bias

Regularization



hasso Progression



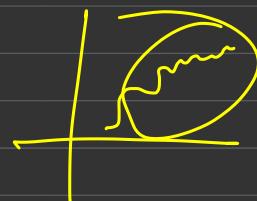
$$\checkmark \text{mse} = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

During your epochs your mse will reduce

$$\text{new w} = \text{mse} + \lambda \sum_{i=1}^p (w_i) \rightarrow \text{hasso}$$

↑ weight modifications

new w = $\text{mse} + \left(\frac{1}{2} \right) \sum_{i=1}^p (w_i^2) \rightarrow \text{Ridge}$



①

$$50 + 50$$

②

$$40 + 40$$

③

$$32 + 32$$

④

$$25 + 25$$

⑤

$$18 + 18$$

Lagrangian Parameter

$$2 = 52$$

$$5 = 45$$

$$7 = 39$$

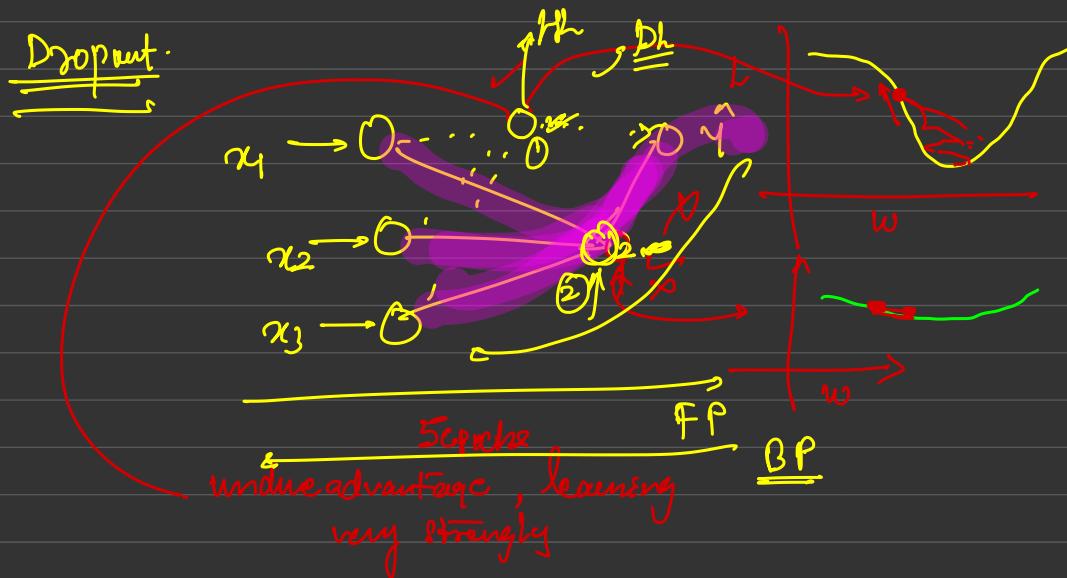
$$11 = 36$$

$$18 = 42$$

Modified
hole Elastic Net

$$= \text{MSE} + \left[\lambda \left(\sum_{j=1}^p |B_j| \right) + \left(\frac{1-\alpha}{2} \sum_{j=1}^p B_j^2 \right) \right]$$

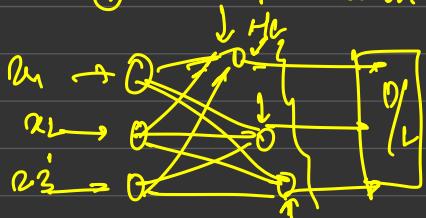
Lasso Ridge.



Randomly we dropout neurons from the training iteration:

H Neuron will stay (σ^2)

① T Neuron will dropout (σ^2)



Weight Initialization

D Define your own architecture, input-dim, $H_1 \rightarrow$ neurons
O/L, activation, loss, metrics

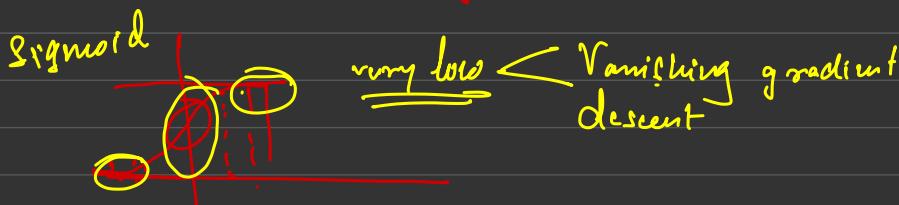
- ② weights, and biases in their \rightarrow randomly initialized
- ③ Training \rightarrow FP & BP, the weights get trained

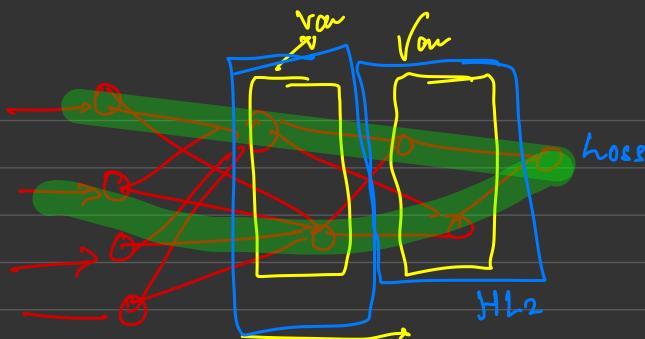
Zero Initialization :

\hookrightarrow the entire network becomes symmetrical

$\Rightarrow \times$ Random Initialization

\hookrightarrow generate weights which are very high
 \hookrightarrow activations high





forward flowing signal

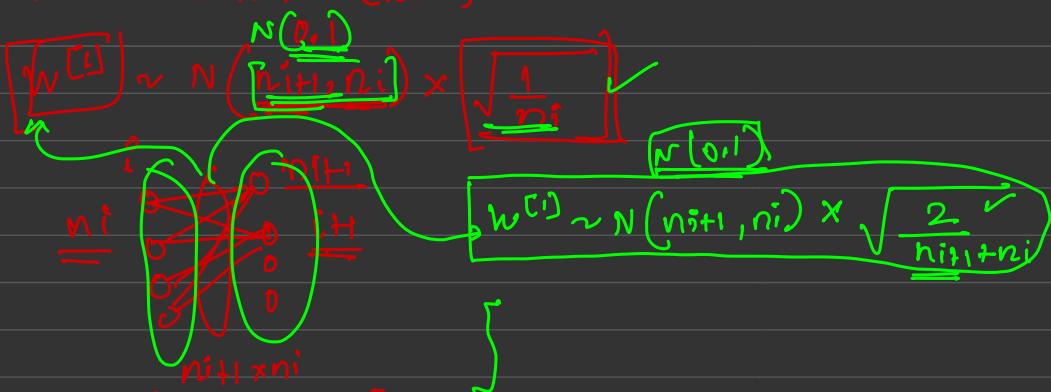
$$\text{Var} \left(\frac{\partial h}{\partial H_{L2}} \right) = \text{Var} \left(\frac{\partial h}{\partial H_{L1}} \right)$$

to maintain that the neural network does not enter into a Vanishing or Exploding gradient problem.

Xavier Initialization

He initialization

Xavier initialization (vanilla)



Normalized Xavier Initialization

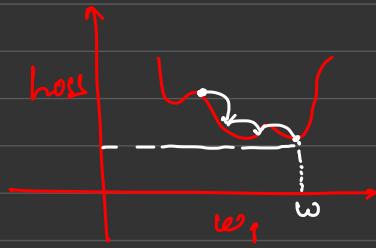
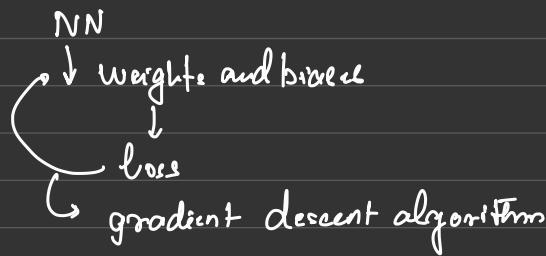
For ReLU activation

He initialization:

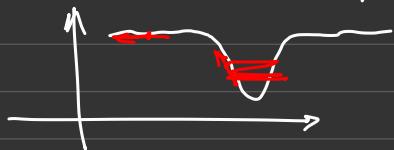
$$W^{(l)} \sim N(\mu, \Sigma)$$

where $\mu = 0$ and $\Sigma = \sqrt{\frac{2}{n_i}} I$

Optimizers



loss curves are extremely complicated
of functions → multiple minimas
↳ be flat at certain regions
be very sharp at certain regions

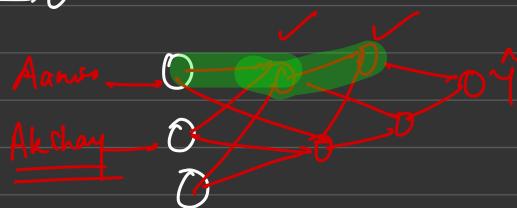


Gradient descent by design has various issues.

- Multiple local minimas.
- The gradients are not consistent
- Sparse features create a problem in (P)

	Akshay	Aamir	IMDB
x_1	✓	0	
x_2	✓	0	
	-	0	
	-	1	
	0	0	
	0	1	

most of the rows of Aamir Khan's feature
 $\rightarrow 0$

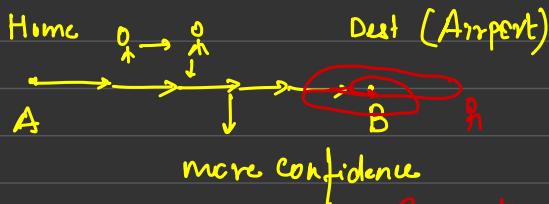


weights related to Aamir Khan

weights related to Akshay Kumar

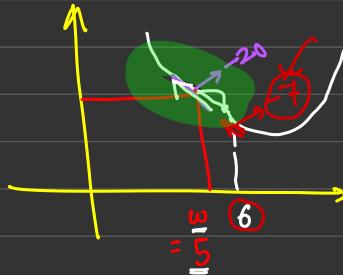
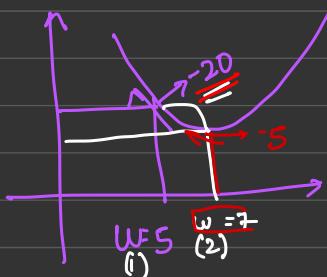
$$\boxed{w_{i+1} = w_i - \alpha \frac{\partial J}{\partial w_i}}$$

Momentum



G_D

G_D with momentum



$$(m(t)) = B m(t-1) + (1-B) \times \frac{dL}{dw t}$$

$$\text{update} \rightarrow [w(2) = w(1) - \alpha(m_t)]$$

$$w(3) = w(2) - 0.1 \times -5$$

$$w(3) = 7 + 0.5$$

$$= 7.5$$

$$m(1) = 0.5 \times 0 + 0.5 \times -20$$

$$m(1) = -10$$

$$w(2) = 5 - 0.1 \times -10$$

$$w(2) = 6.$$

$$m(2) = 0.5 \times -10 - 0.5 \times -7$$

$$m(2) = [5 + 3.5] = -8.5$$

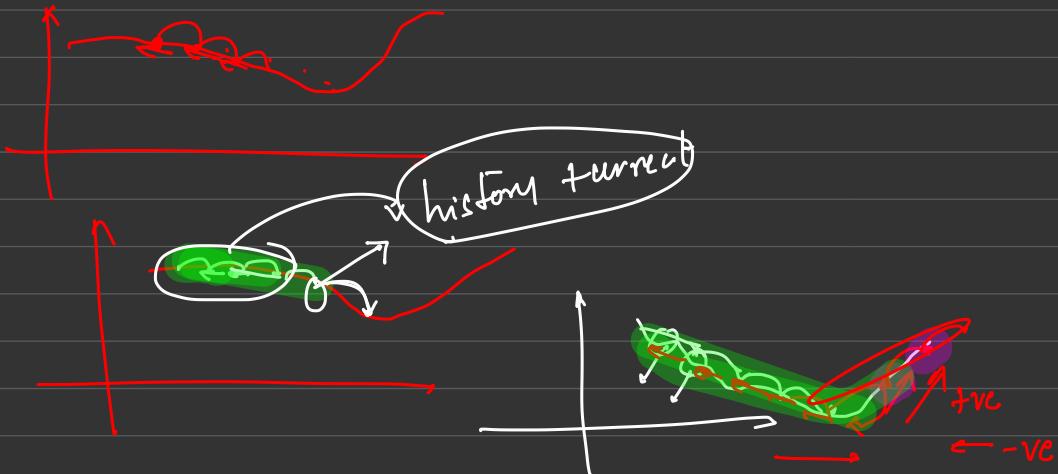
$$w(3) = w(2) - 0.1 \times -8.5$$

$$w(3) = 6 + 0.85$$

$$= 6.85$$

Momentum based gradient update actually takes into account the historical gradient \times weightage + the current gradient \times weightage.

↓
accelerates the process of the update



$$w_{t+1} = w_t - \alpha m_{t+1}$$

$$\boxed{m_{t+1} = \beta \times m_{t+1} + (1-\beta) \times \frac{\partial L}{\partial w_t}}$$

m_{t+1} ↓ history point / current

Draw the graph what mean
it takes longer to converge
it is so fast just
crosses / ignores local
min

Adagrad \rightarrow Adaptive gradient

$$V_t = V_{t-1} + (\nabla w_t)^2$$

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{V_t}} \nabla w_t \rightarrow \frac{\alpha}{\sqrt{V_t}} \nabla w_t$$

Weights which will see big updates \rightarrow will become smaller

Adaptive gradients start punishing weight updates for weights which have already seen big gradient updates
 → don't punish/reward weight updates for weights which have not seen big gradient updates

Rewarding (Amir Khan) $\uparrow \uparrow$ balanced
 Punish (Abhay Kumar) $\downarrow \downarrow$ learning

$$V_t = V_{t-1} + \underline{(\nabla w_t)^2}$$

too high
too fast

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{V_t + \epsilon}} \nabla w_t$$

RMS Prop.

$$v_t = \boxed{B_1 \times v_{t-1}} + \boxed{(1-B_1) \times \sqrt{w_t^2}}$$

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_t + \epsilon}} \times \frac{dh}{dw}$$

ϵ
↓
Epsilon

Adam optimizer:

Adaptive moments

$$\rightarrow m_t = \boxed{B_1 \times m_{t-1} + (1-B_1) \times \nabla w_t}$$

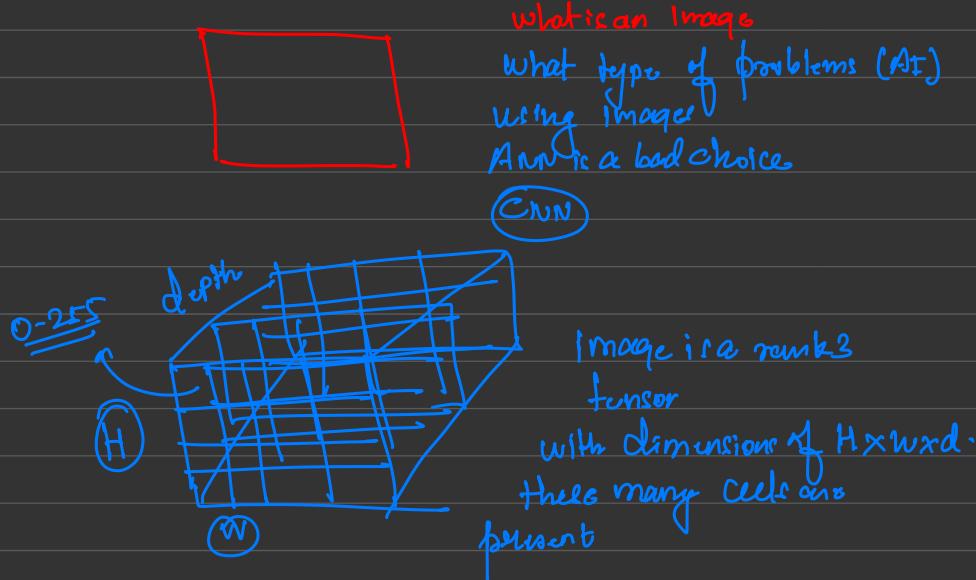
momentum

$$\rightarrow v_t = \boxed{B_2 \times v_{t-1} + (1-B_2) \times \nabla w_t^2} \rightarrow \text{RMS}$$

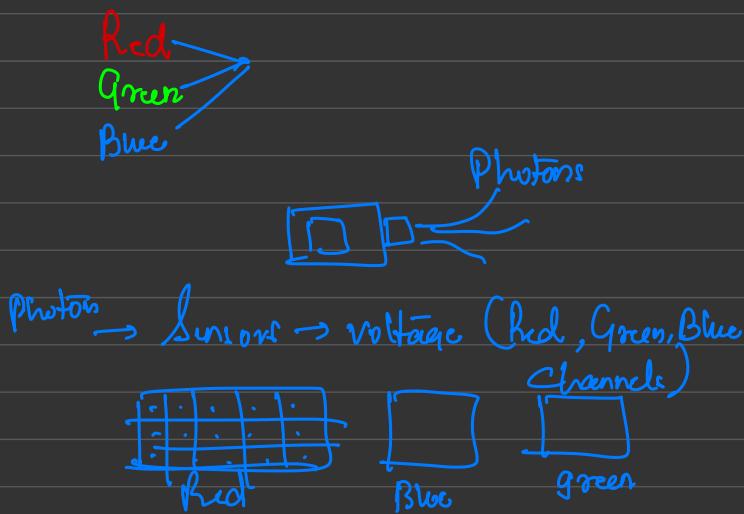
$$w_{t+1} = w_t - \frac{\alpha \times \hat{m}_t}{\sqrt{v_t + \epsilon}}$$

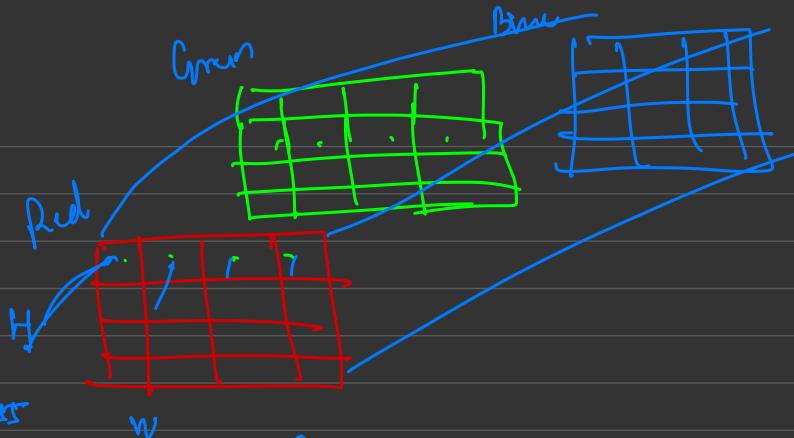
$$\hat{m}_t = \frac{m_t}{1-B_1} \quad \hat{v}_t = \frac{v_t}{1-B_2}$$

Convolutional Neural Network



within these cells we have some numbers (0-255)





as (information content in an image)

→ Variation in the pixel values across the Red, Green, Blue Channel.



detect an object in an image

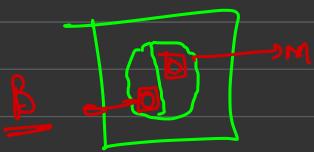
→ image classification



- A) Problems:
- B) Image classification
- C) Object detection and localization



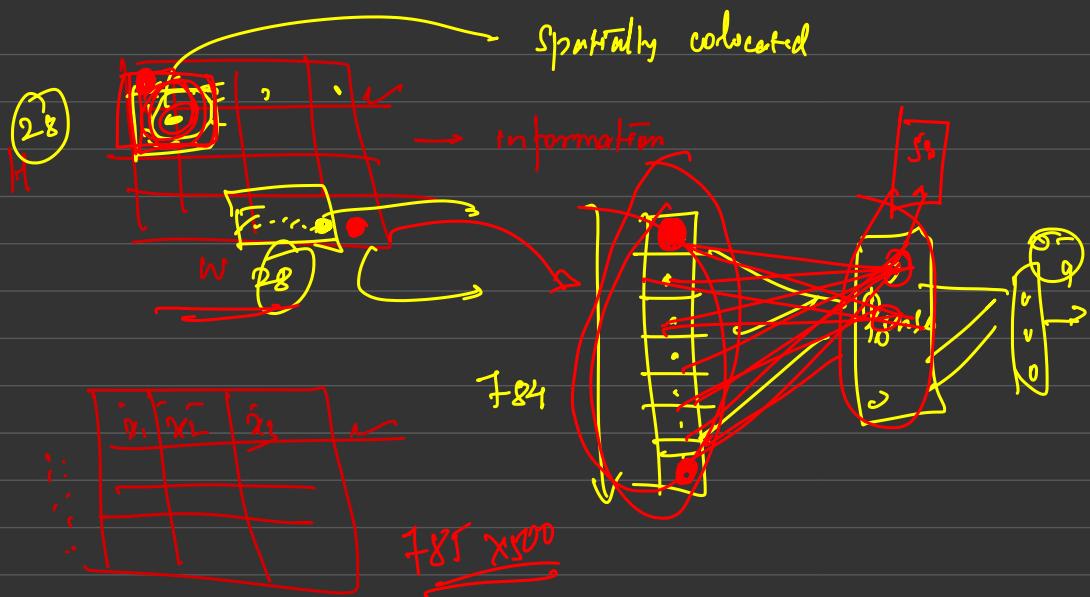
CCTV Camera



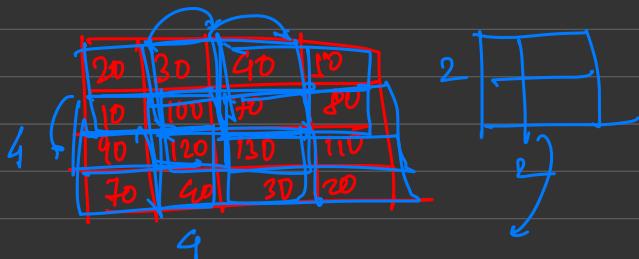
Facial Recognition System (Biometric methods)

Login phone





C_{NN}



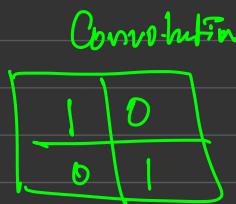
filter will move throughout
the image and extract features

movement of the filter throughout
the image is called Convolution

Image

20	30	40	70
100	100	180	170
60	120	130	140
110	150	170	205

4



4 160

$$20 \times 1 + 30 \times 0 + 100 \times 0 + 140 \times 1$$

$$60 \times 1 + 120 \times 0 + 110 \times 0 + 150 \times 1$$

210

$$40 \times 1 + 70 \times 0 + 180 \times 0 + 170 \times 1$$

285

$$130 \times 1 + 140 \times 0 + 170 \times 0 + 205$$

Feature map

160	210
210	285

2

Extracted features from the image.

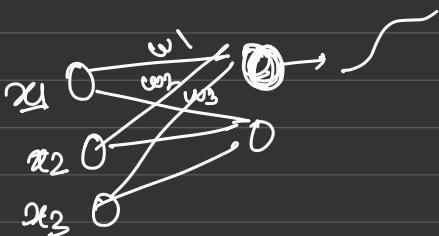
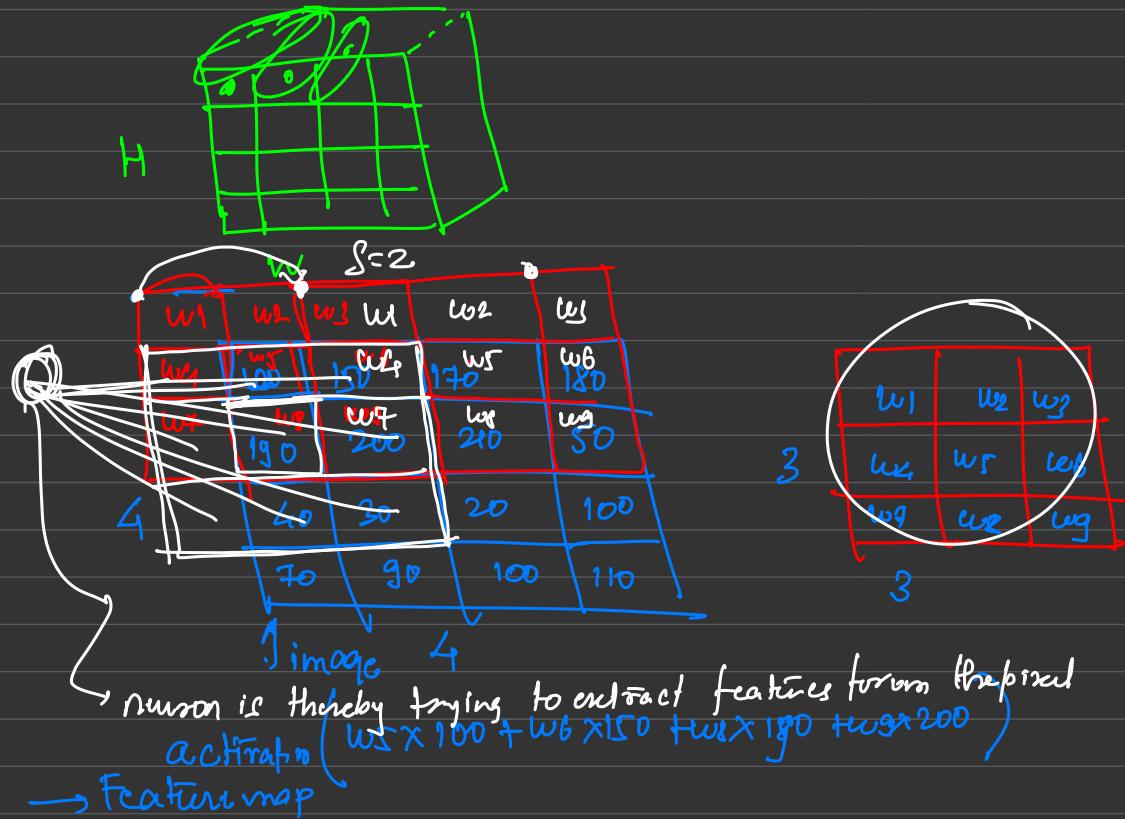
that the filter convolving on an image
is basically a feature extraction process

Edge
Boundary Curved
Complex Boundaries
Color variations

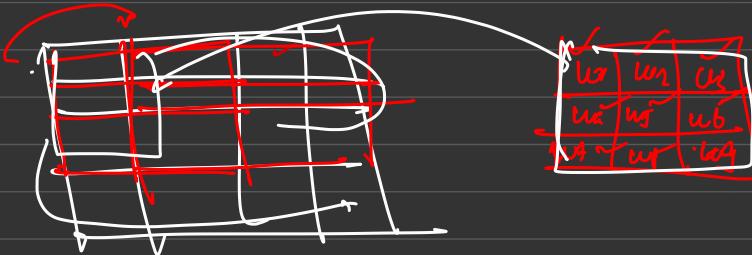


Convolutional neural network

Image classification

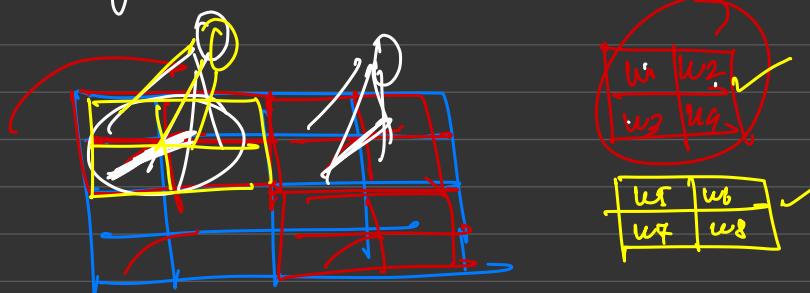


extracting features from an image

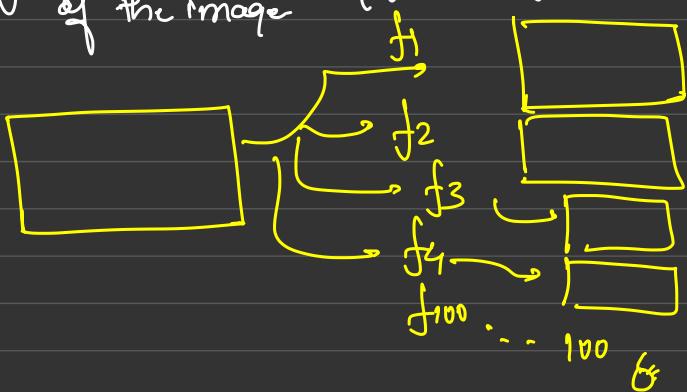


one iteration of the convolution ,

"your filter weights don't change"

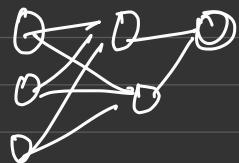


one filter is learning one kind of
feature from the different regions
of the image



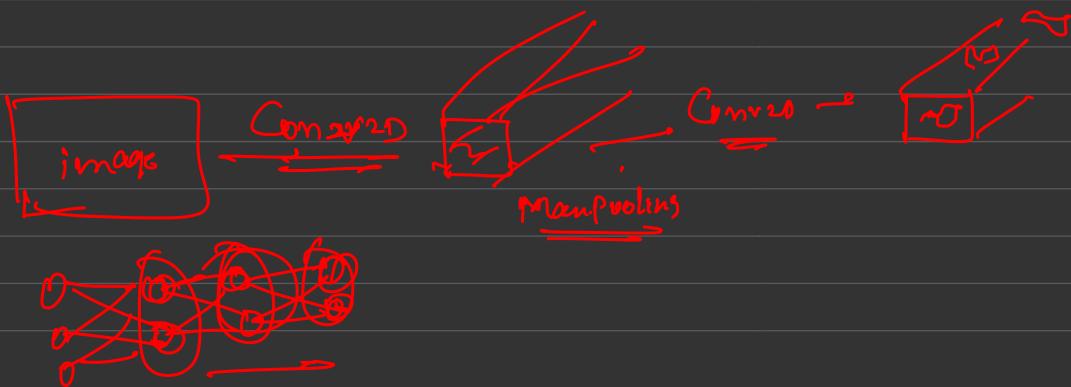
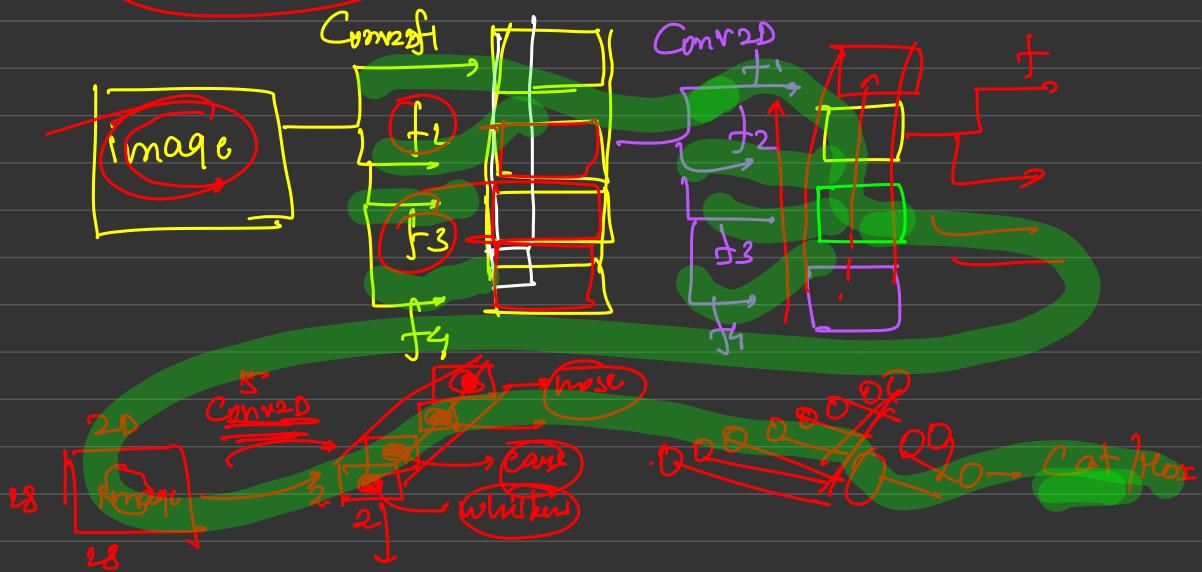
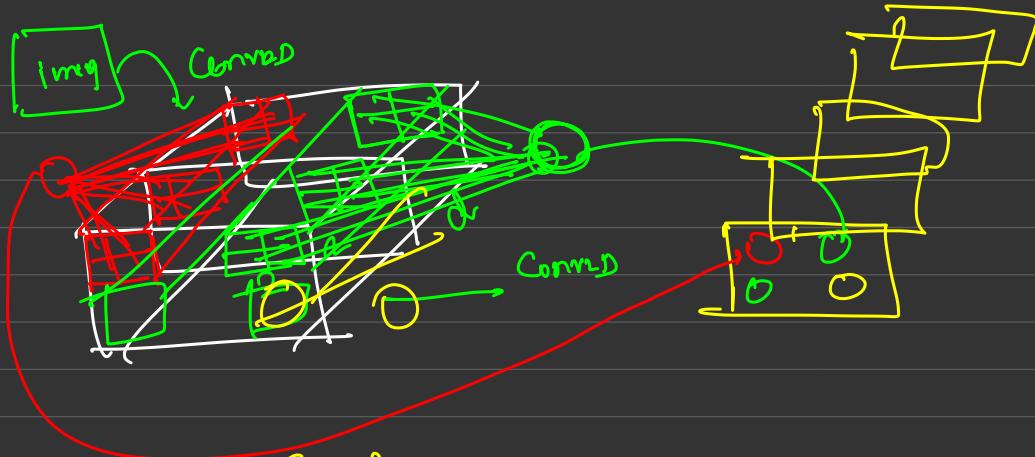
Conv2D

2×2



 **OF**, neuron is going to learn a non-linear combination of the features that were generated of the object placed on the top-left corner of your original image

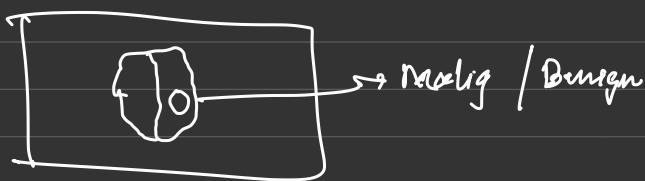
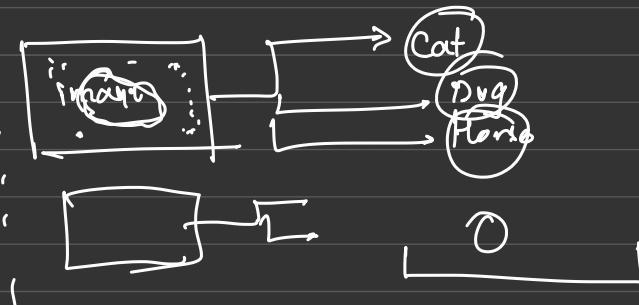
one filter applied on an image
learns one kind of feature at
diff locations
many filters you will learn
many features



Convolutional neural network

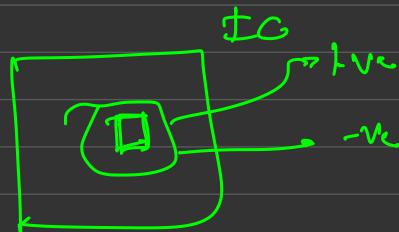
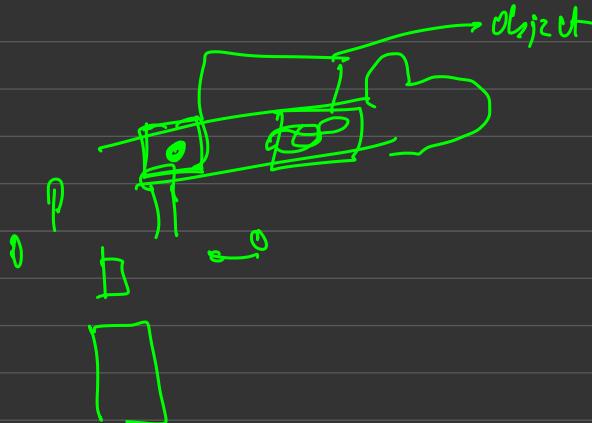
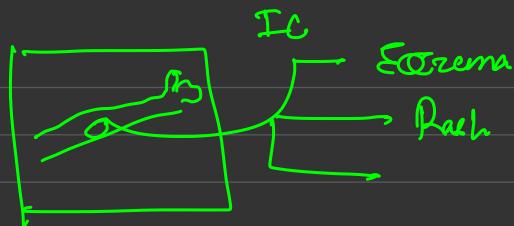
extract information from images

Image Classification



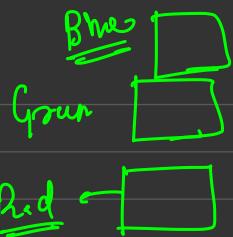
Object detection & localization



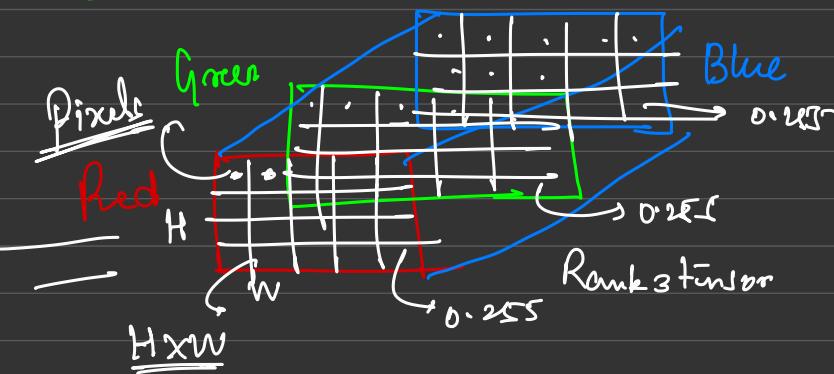


Siamese Network (Facial tracking system)



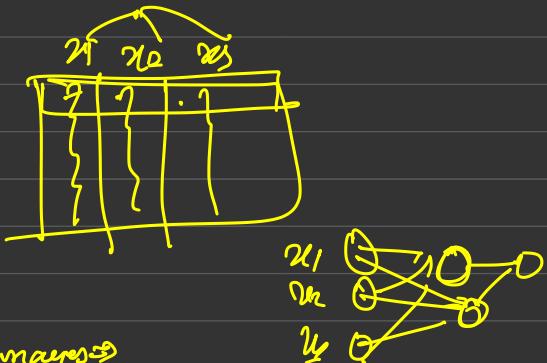
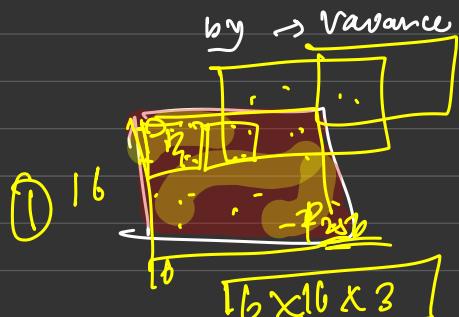


$R + G + B$



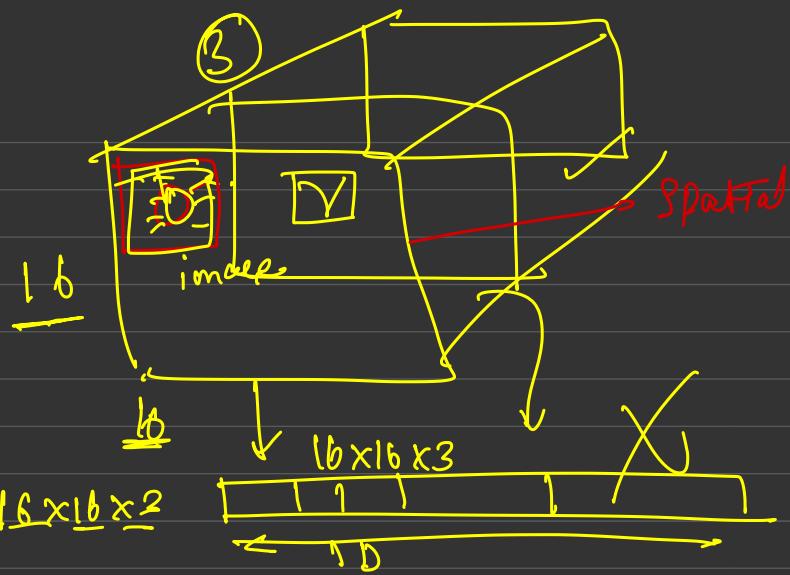
What is information in an image

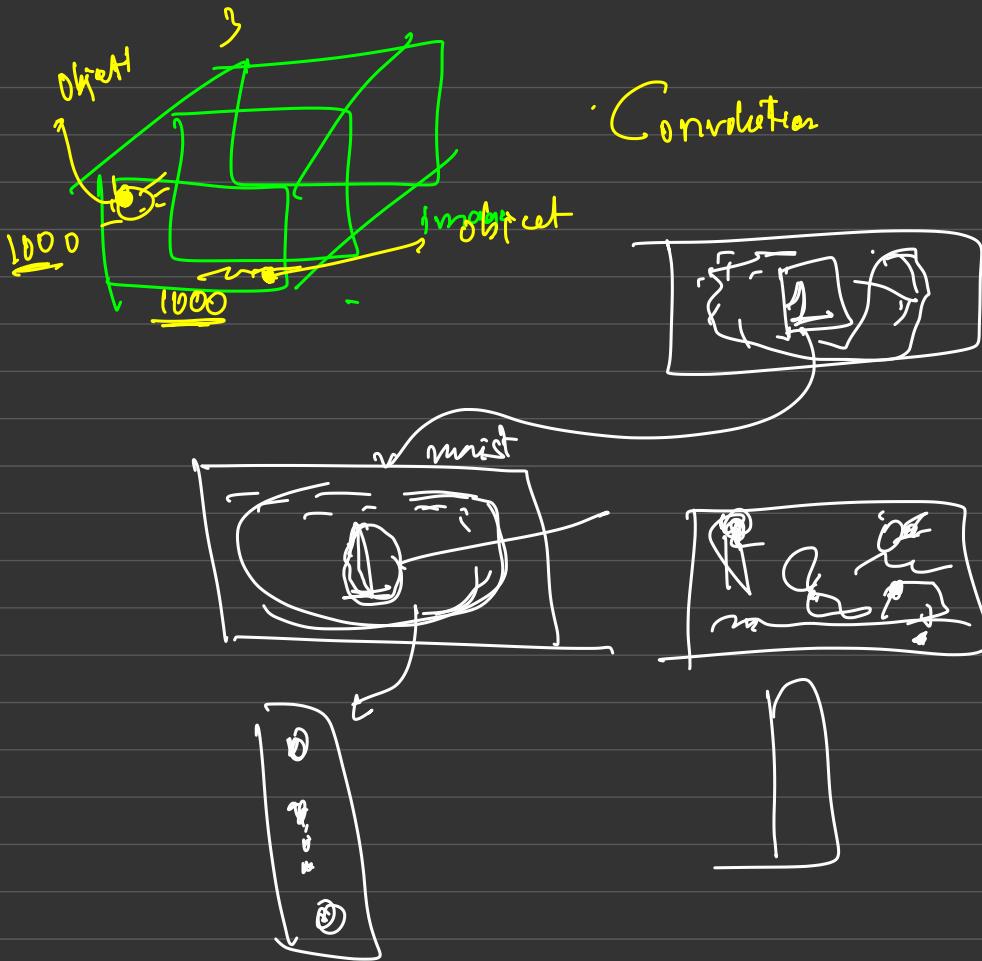
Variance in Pixel values across all the 3 color channels is what we mean

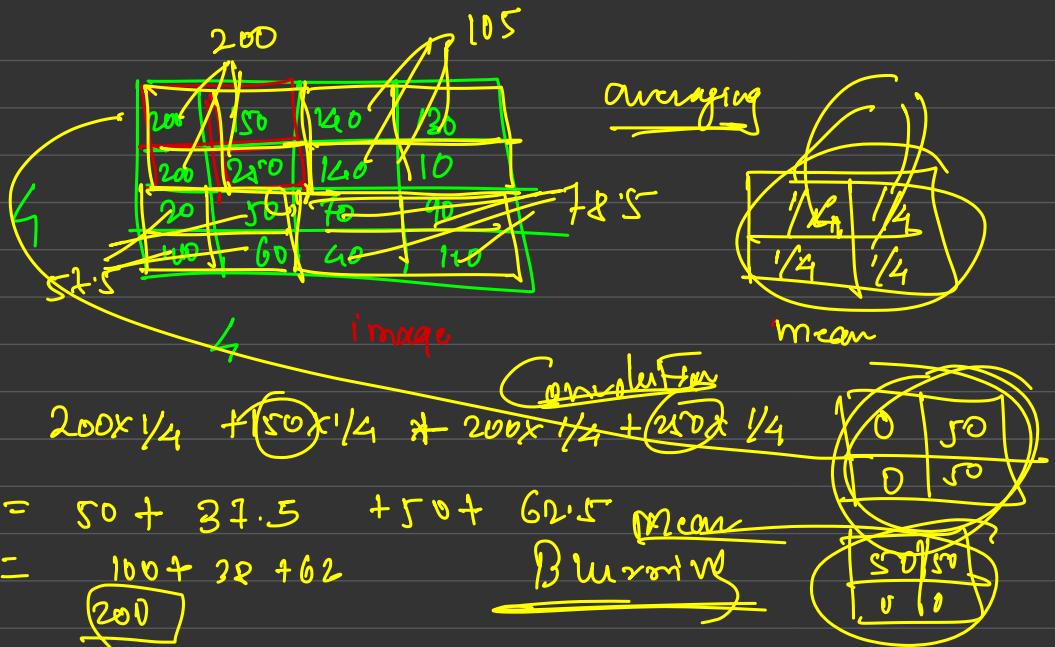


extracting information from images
extracting features / modifying pixel values

$$\text{Image} \sim \begin{pmatrix} p_1 & \dots & p_{256} \\ p_{1-6} & \dots & p_{256} \\ p_{1-12} & \dots & p_{256} \end{pmatrix}$$





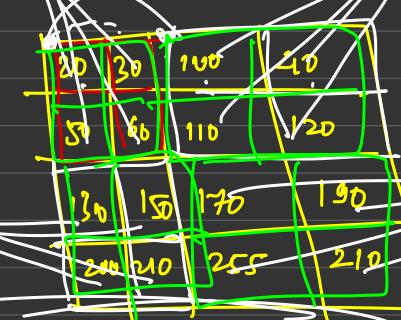


main process of extracting features



$$\begin{array}{|c|c|} \hline w_1 & w_2 \\ \hline u_3 & w_4 \\ \hline \end{array}$$

tanh $(20 \times w_1 + 30 \times w_2 + 50 \times w_3 + 60 \times w_4 + b)$



$$100 \times w_1 + 40 \times w_2 + 110 \times w_3 + 120 \times w_4 + b$$

$$\begin{array}{|c|c|} \hline w_1 & w_2 \\ \hline w_3 & w_4 \\ \hline \end{array}$$

tanh $(170 \times w_1 + 190 \times w_2 + 225 \times w_3 + 210 \times w_4 + b)$

Number $(20 \times w_1 + 30 \times w_2 + 50 \times w_3 + 60 \times w_4 + b)$
entity

ReLU / Tanh

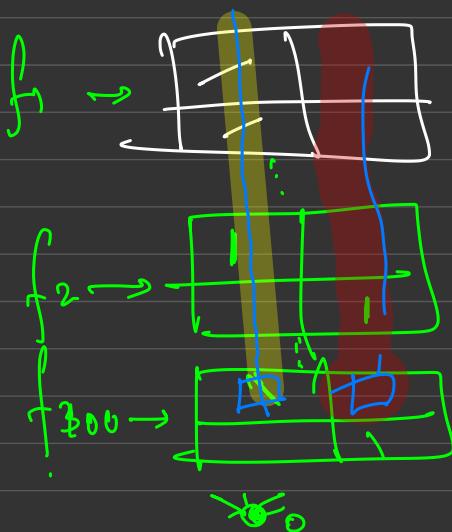
Capturing a feature
non-linear

Using this filter

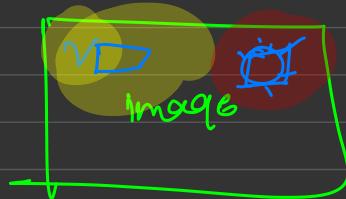
If draw identified 'line' particular pattern across the image

250	-150
60	-100

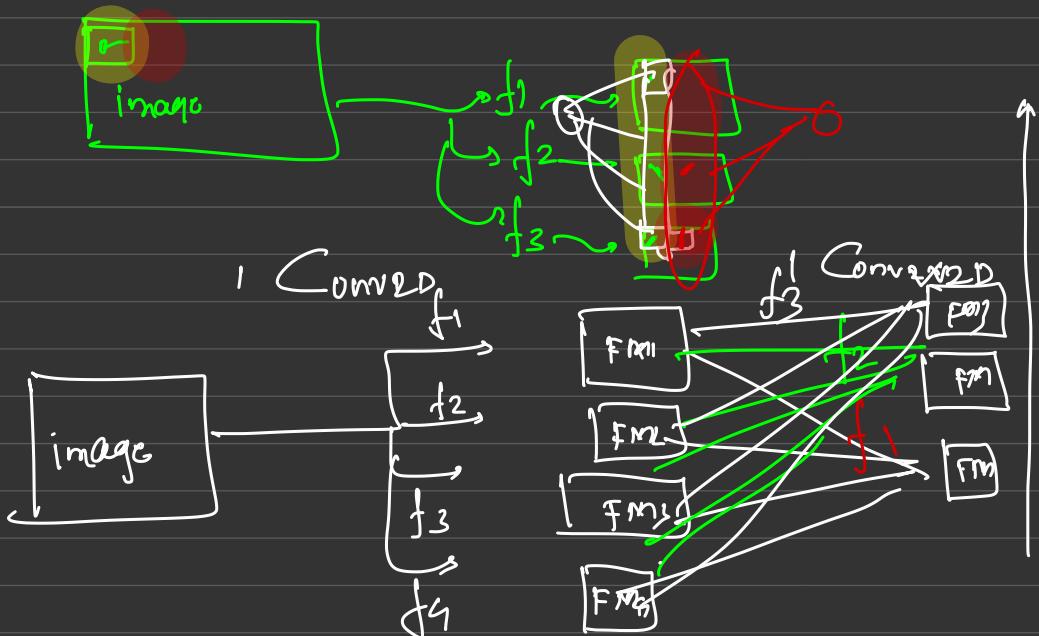
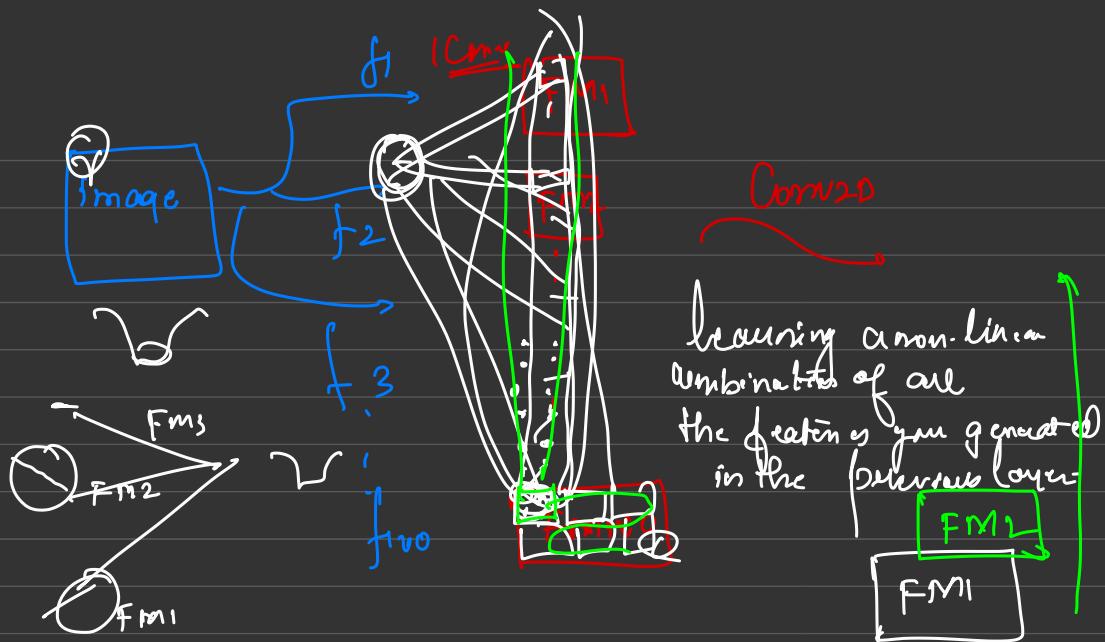
Feature map.

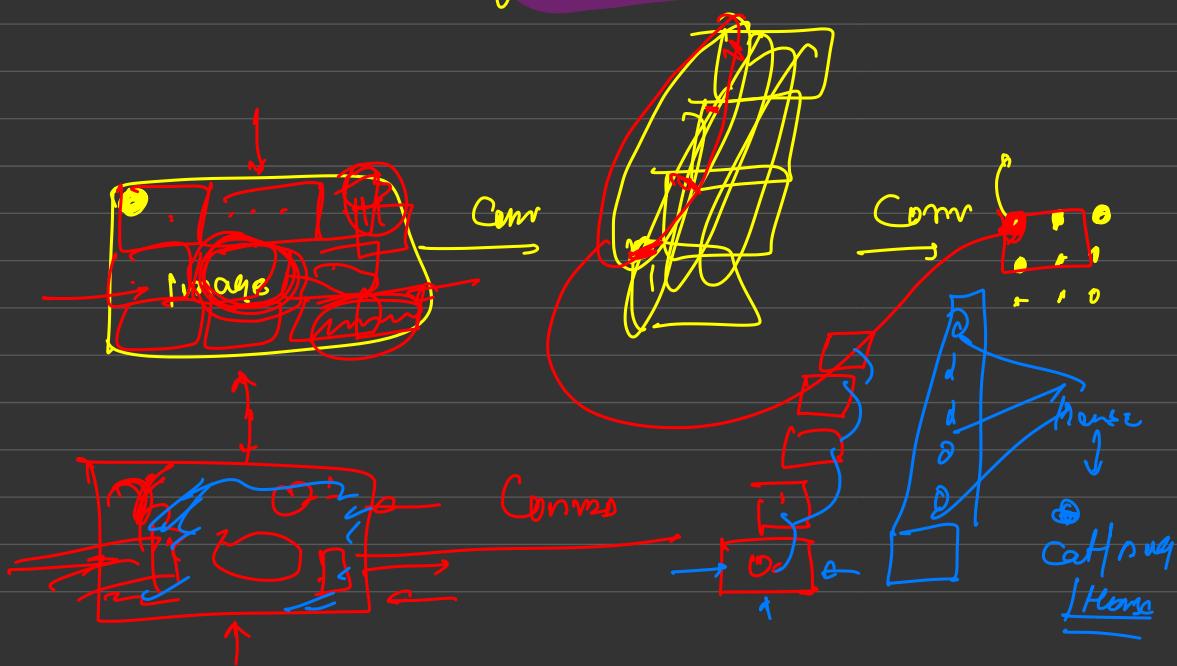
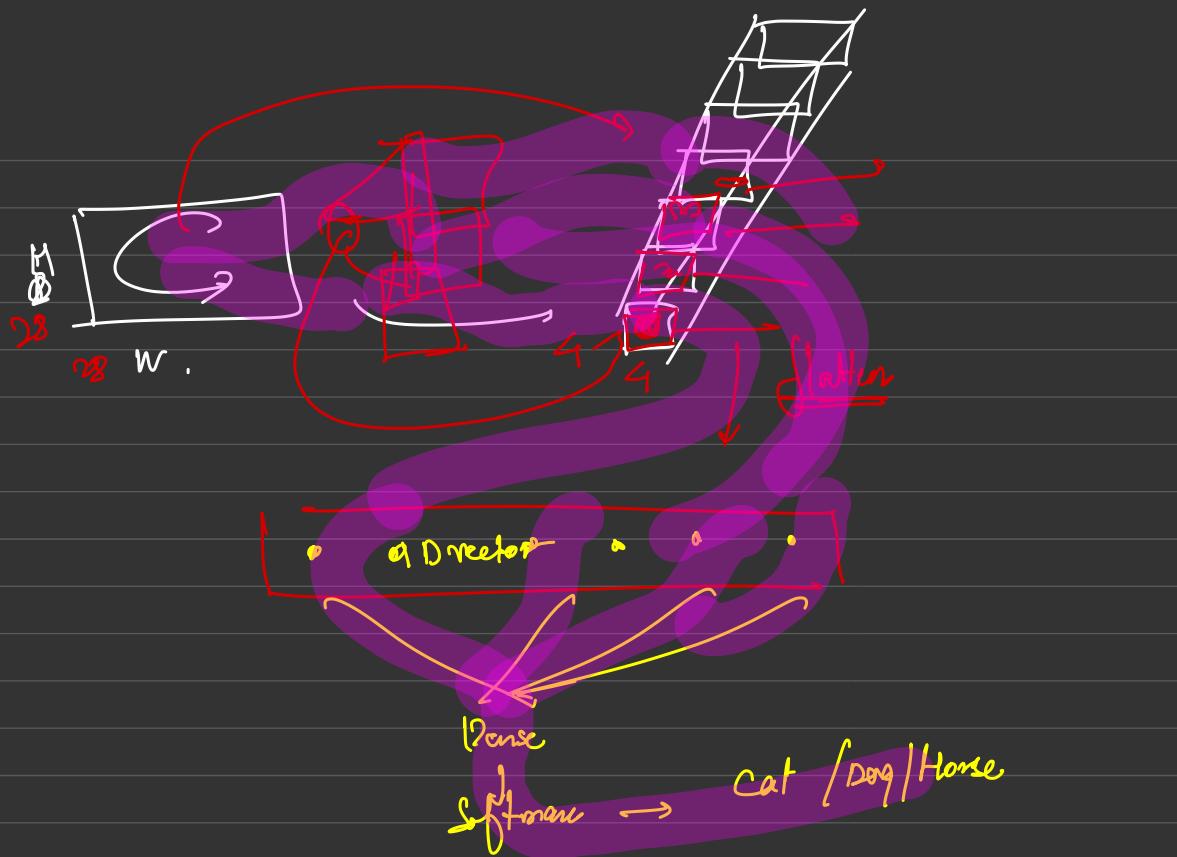


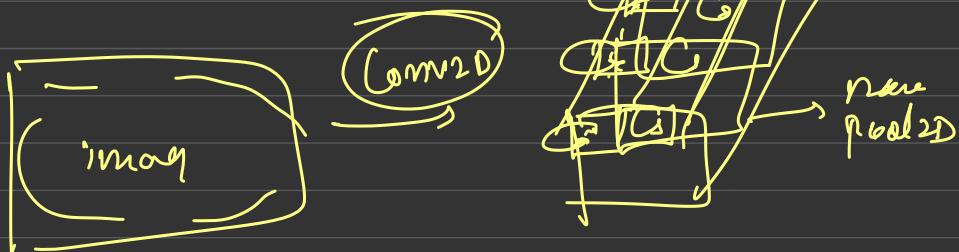
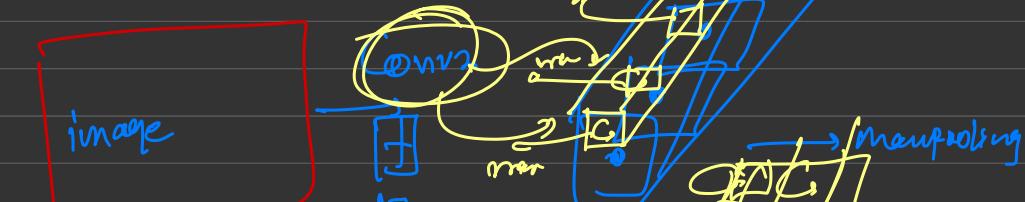
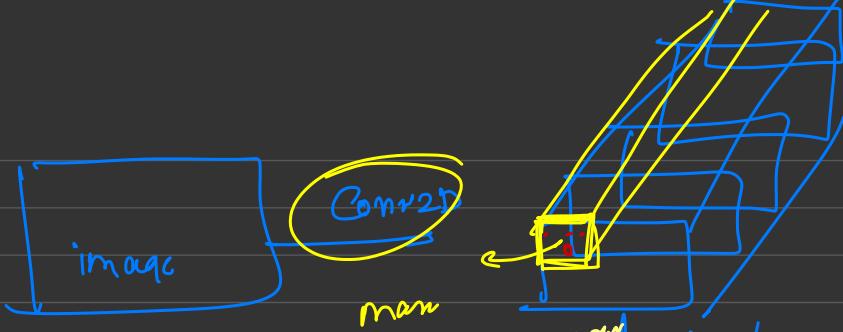
w5	w6
w7	w8

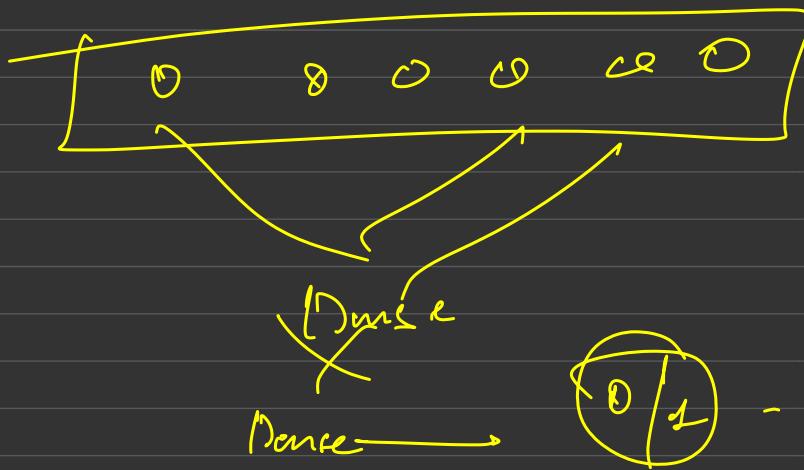


feature maps are
basically the deconstructed
version of your object at
that location

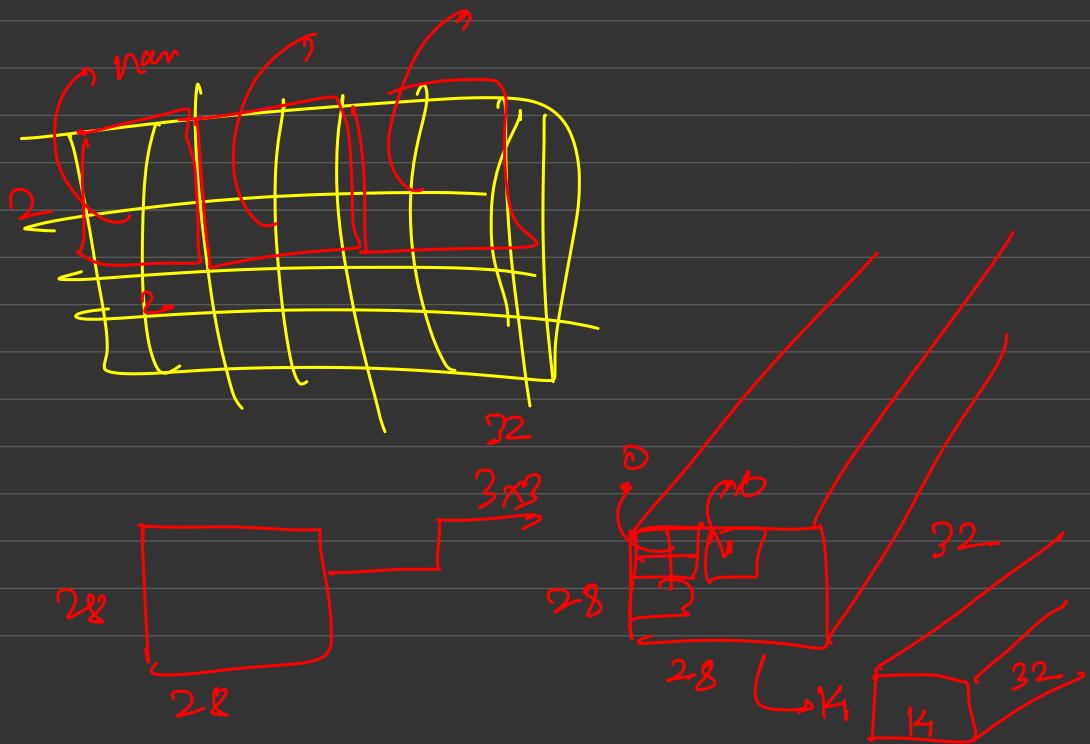
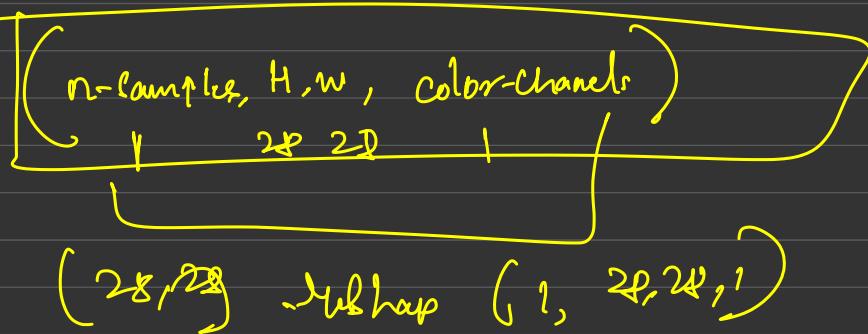


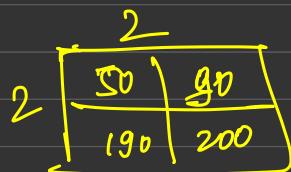
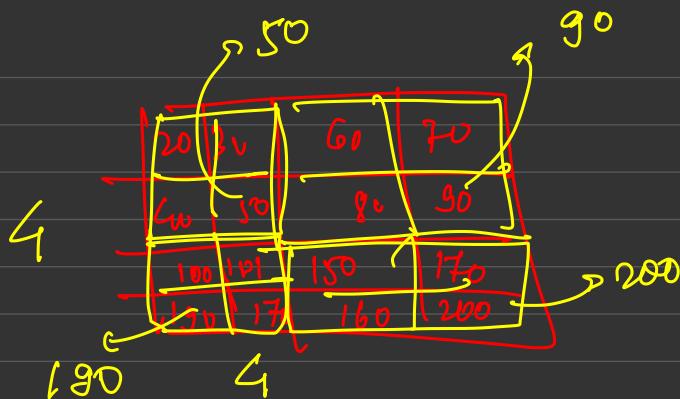




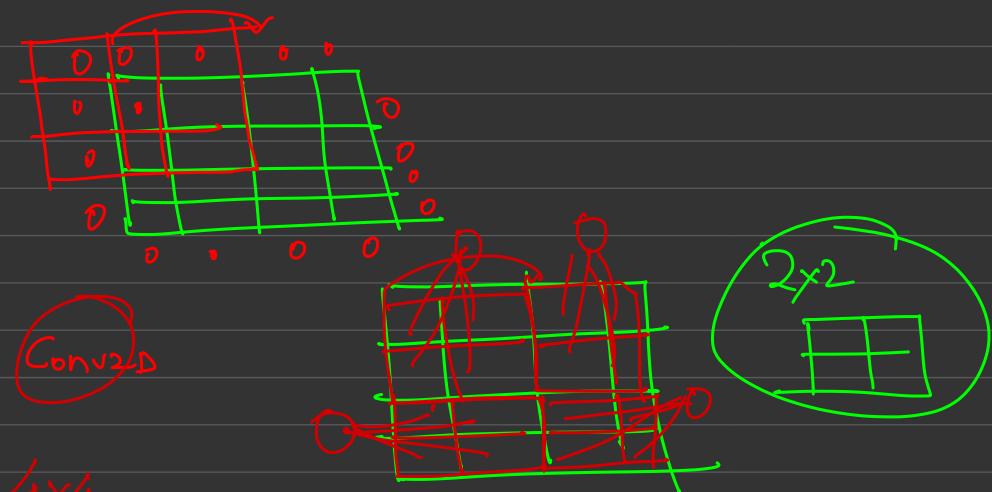
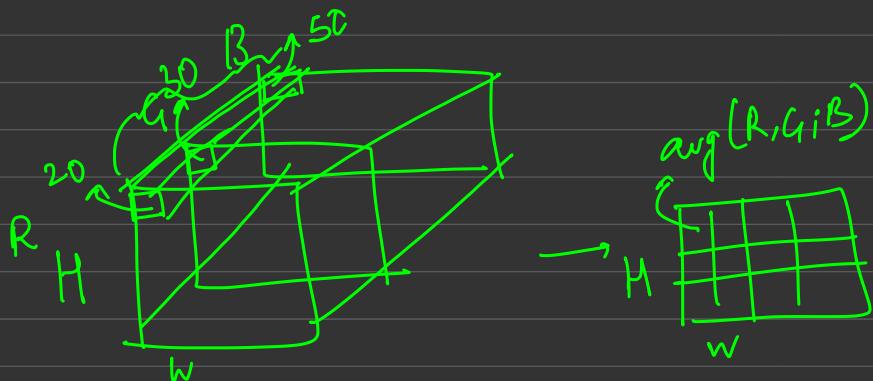


You will always have an image sent to a Conv layer / model as a 4D tensor

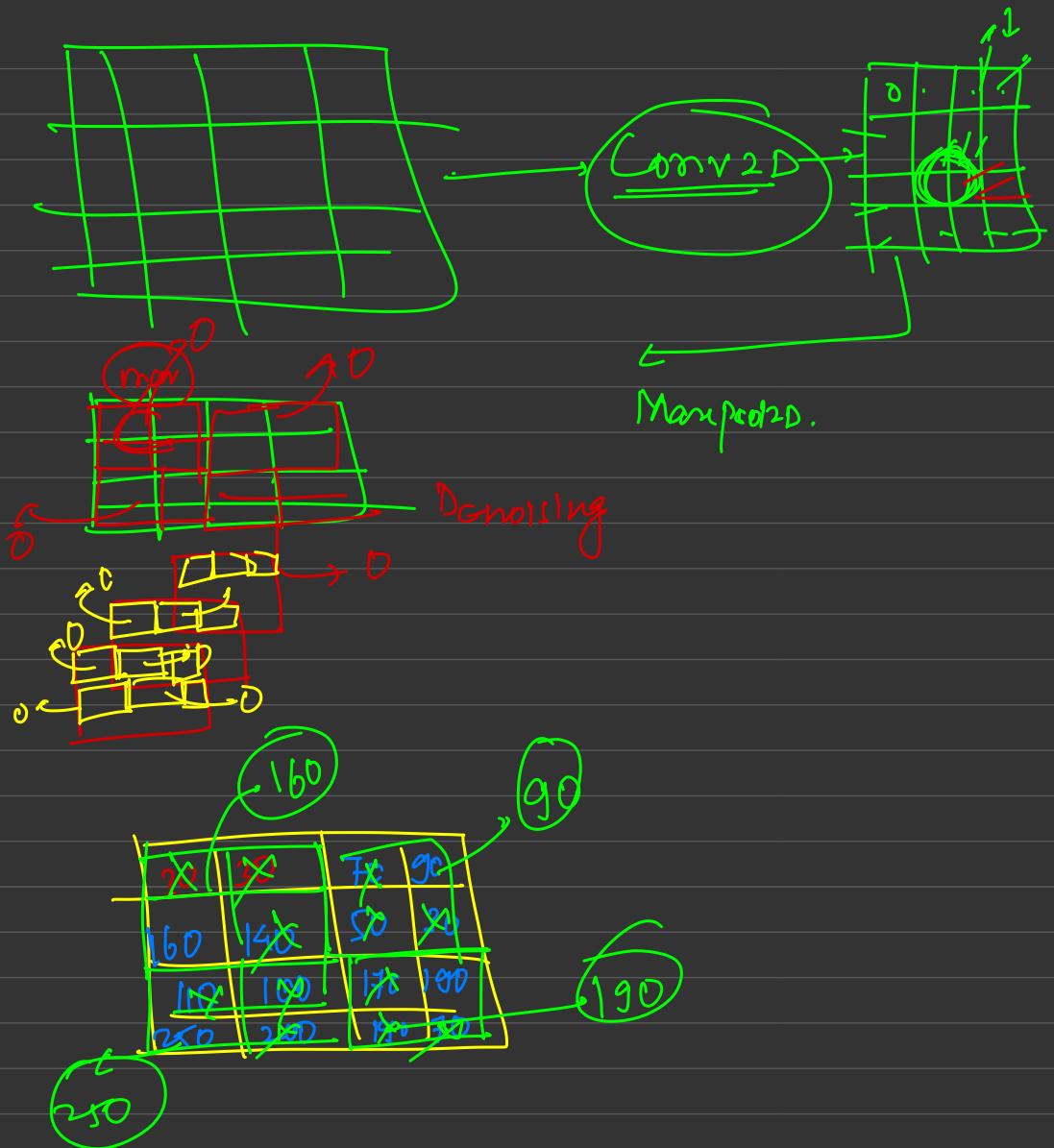


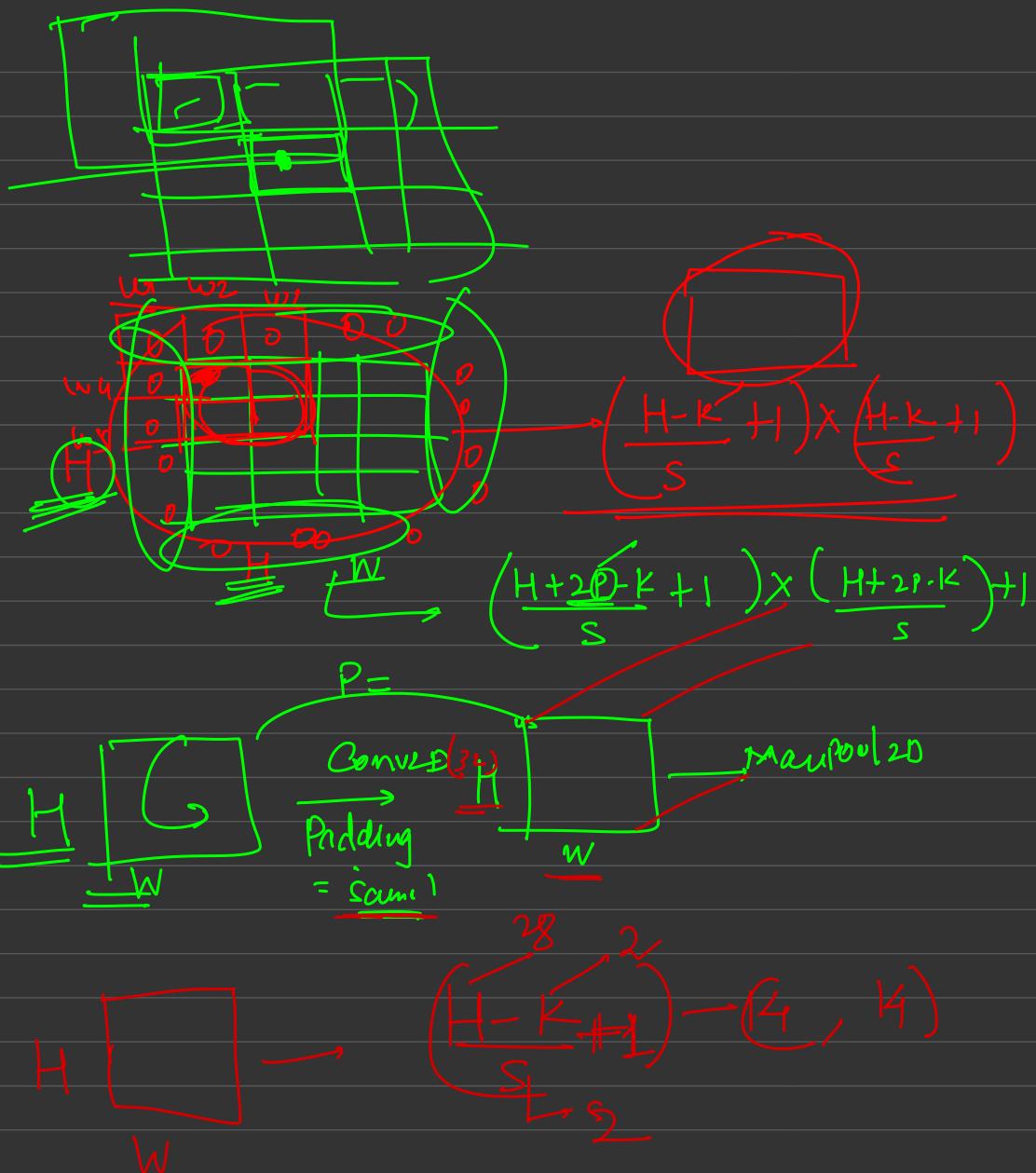


Max Pooling and Padding

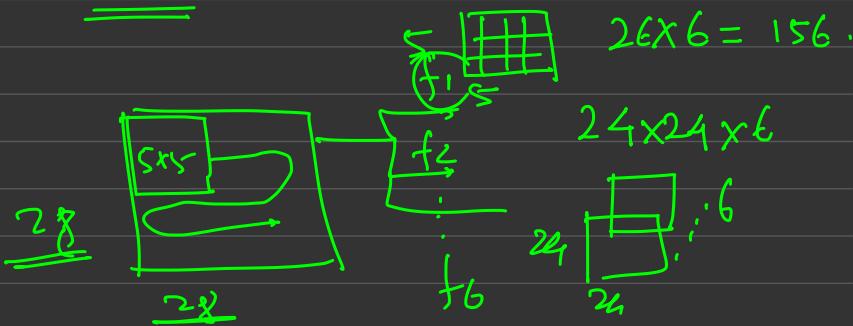


Maxpooling



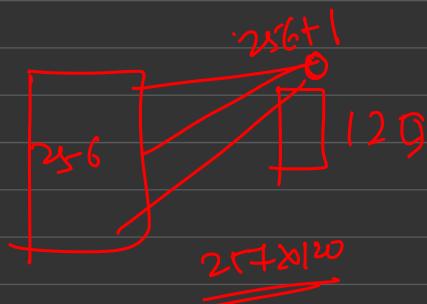
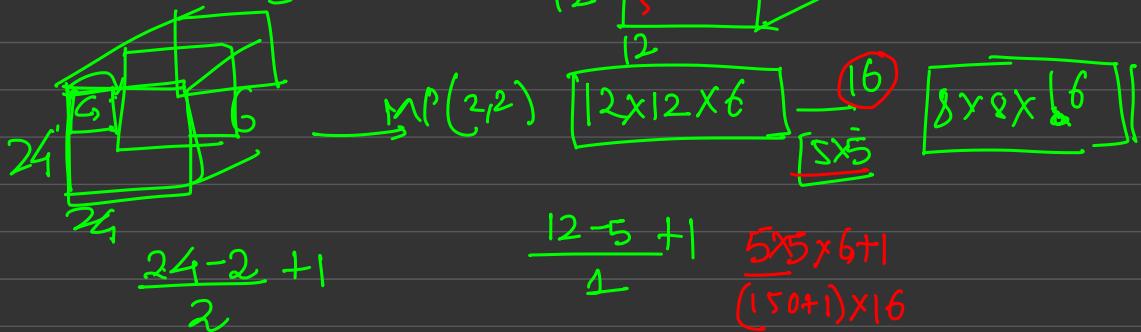


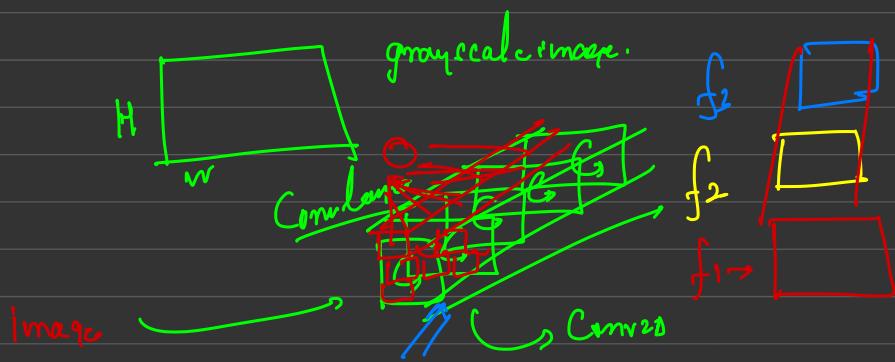
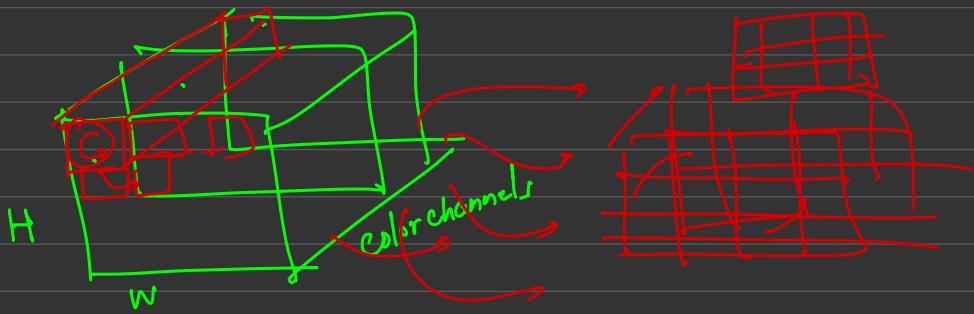
Lenet

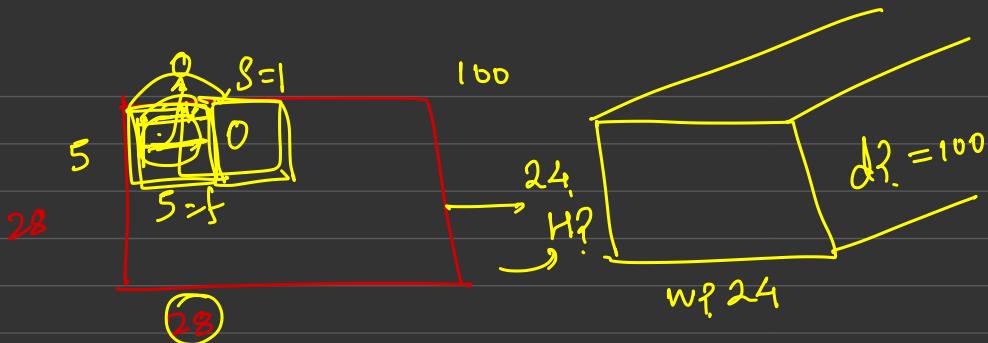


$$H' = \left(\frac{H - k + 1}{s} \right) = 12$$

$$W' = \left(\frac{W - k + 1}{s} \right) = 6$$







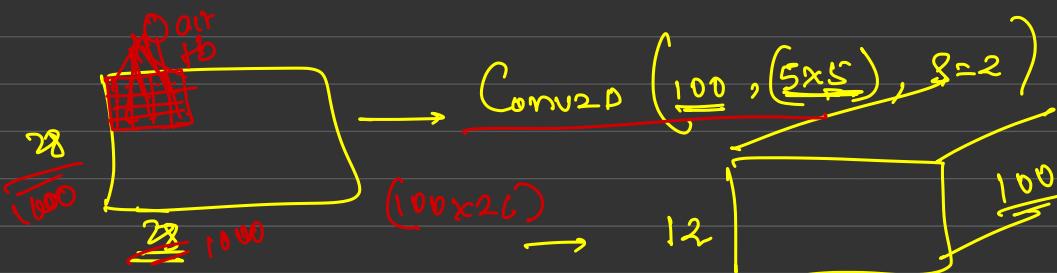
$$H' = \frac{H-f}{s} + 1$$

$$H' = \frac{28-5}{1} + 1$$

$$H' = 24$$

$$W' = \frac{w-f}{s} + 1$$

$$W' = \frac{28-5}{1} + 1 \\ = 24$$



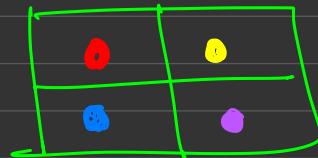
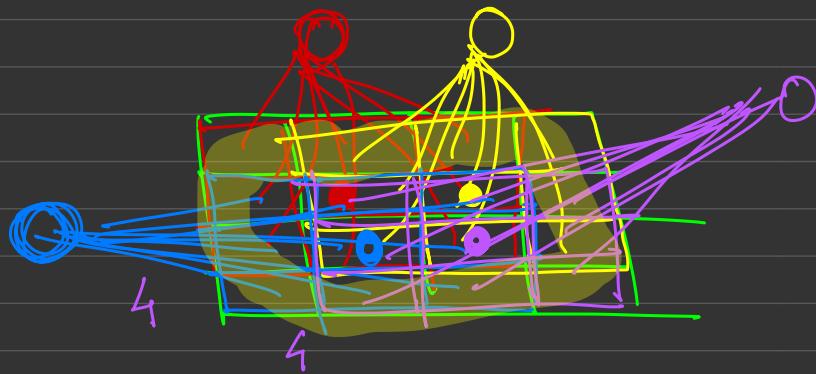
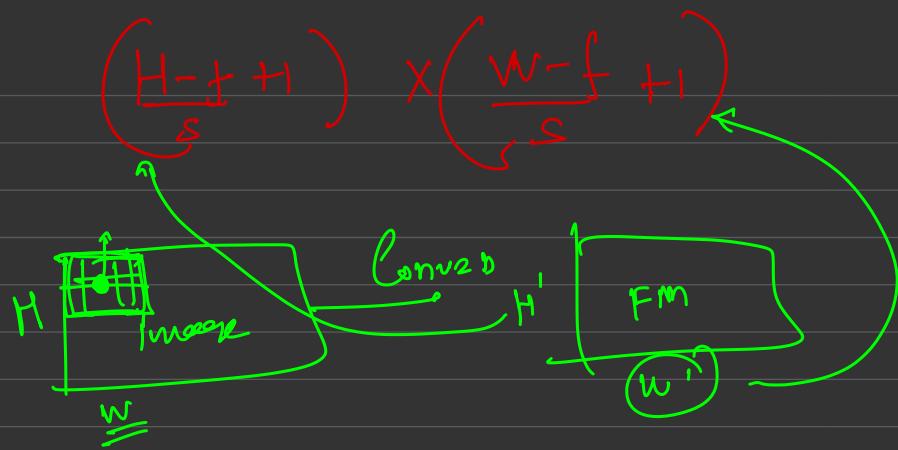
$$\frac{28-5}{2} + 1$$

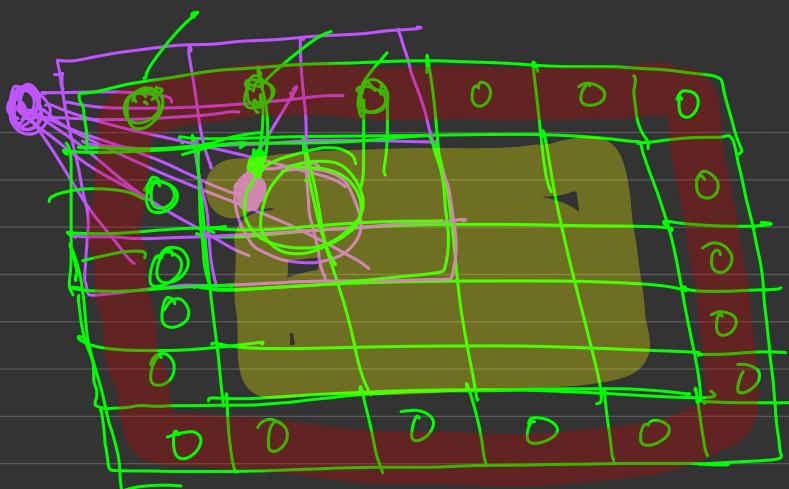
$$11+1 = 12$$



Trainable parameters?

25 weights + 1 bias = 26 parameters





H, W of the image / fm before Conv2D =

$$\frac{H - k + 1}{s} + 1$$

$\frac{(H+2P-k)}{s} + 1$

(H)

$H \times W$ of the image / fm after Conv2D

$$\frac{W - k + 1}{s} + 1$$

$\frac{(W+2P-k)}{s} + 1$

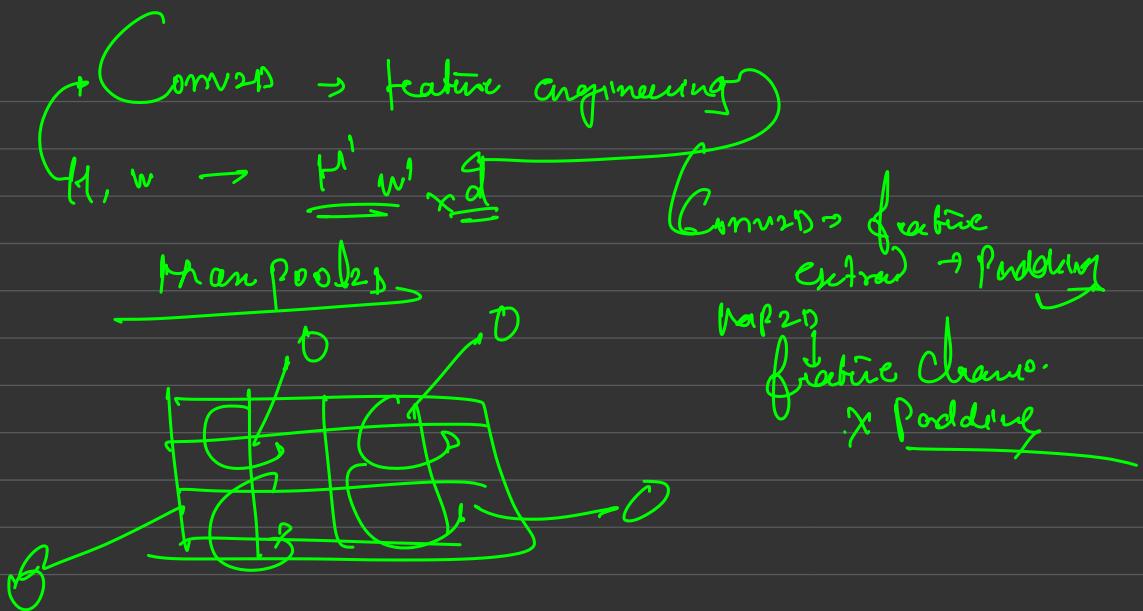
(W)

Image
28x28

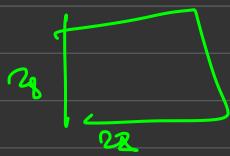


28x28



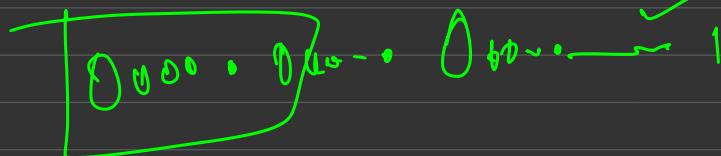
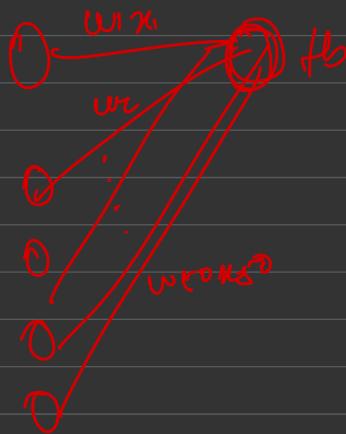
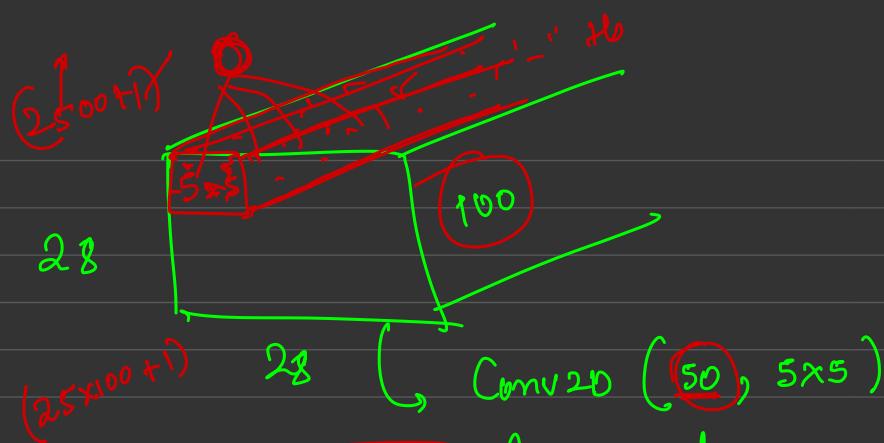


$$\begin{aligned} & H + 2P \\ & \boxed{\frac{H + 2P - f}{S} + 1 = H} \end{aligned}$$



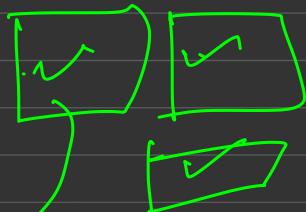
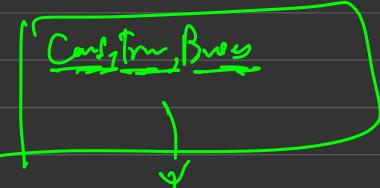
Conv2D (100, 5x5)

→ Learnable parameters



famous architectures

-
-
-



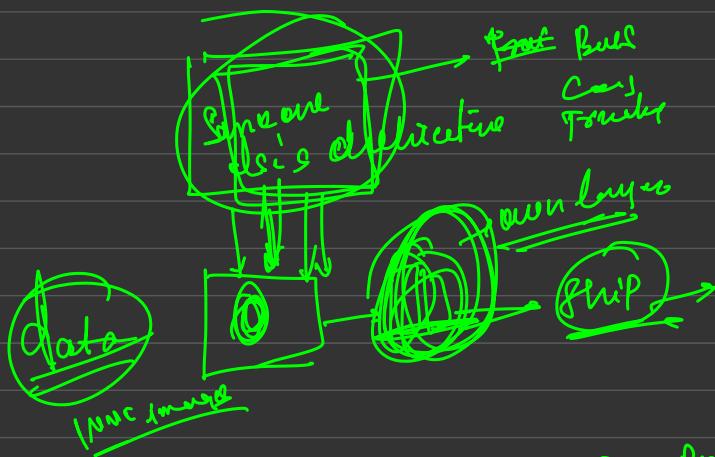
Transfer learning

groups

10,000 images

CNN

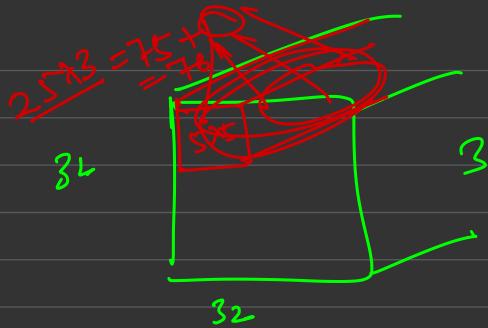
D D G dataset



8:03 : 8:10 pm

hurt

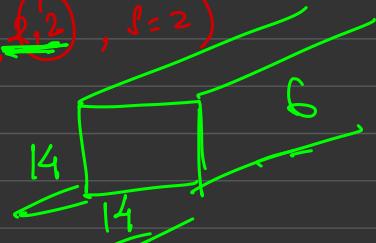
Alexnet



$\xrightarrow{\text{Conv2D } (6, 5 \times 5)}$



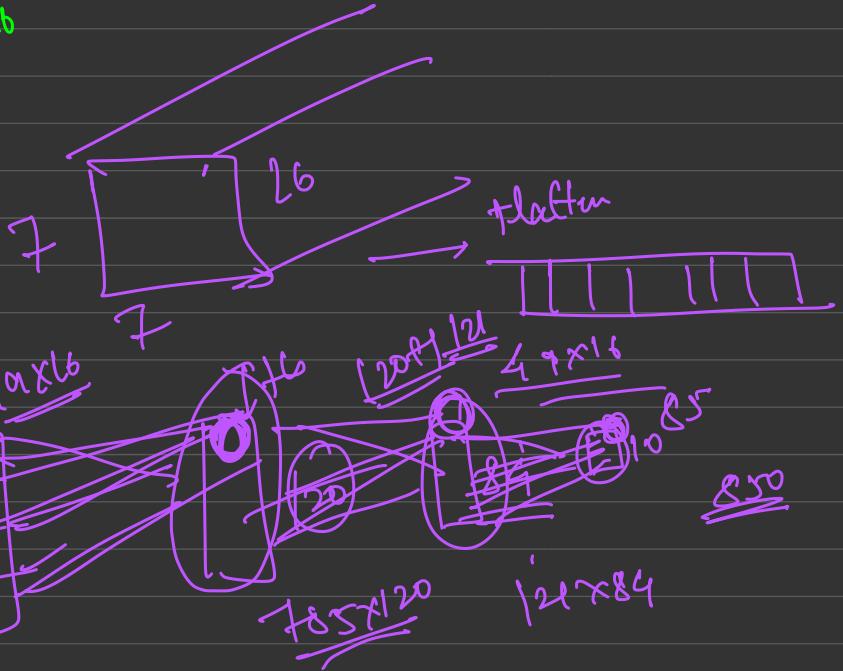
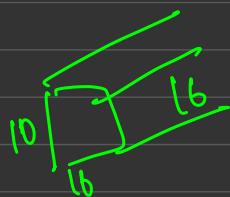
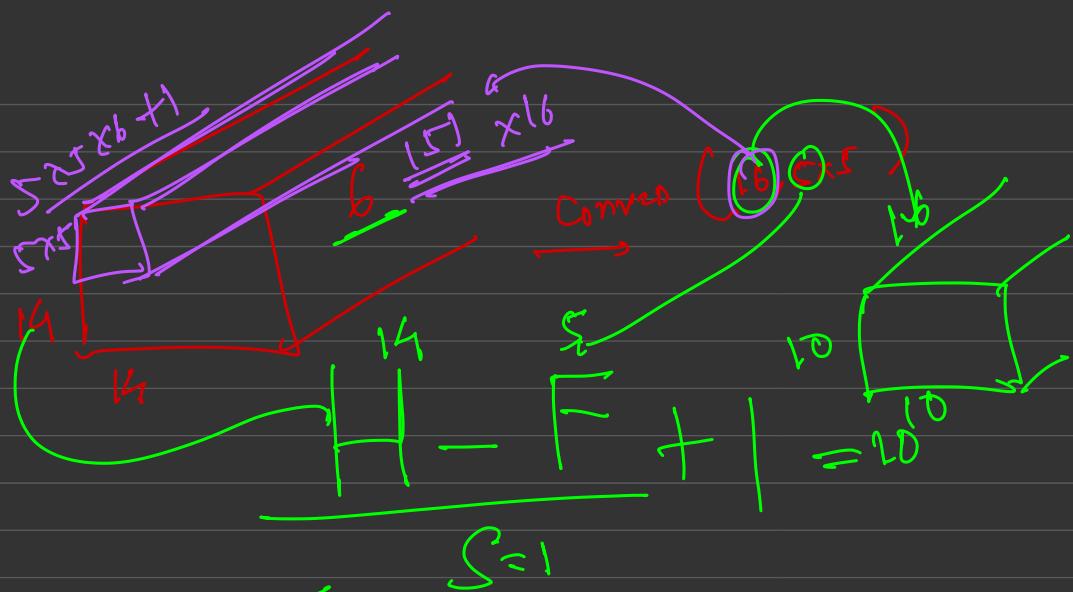
$\xrightarrow{\text{Conv2D } (2, 2)}$

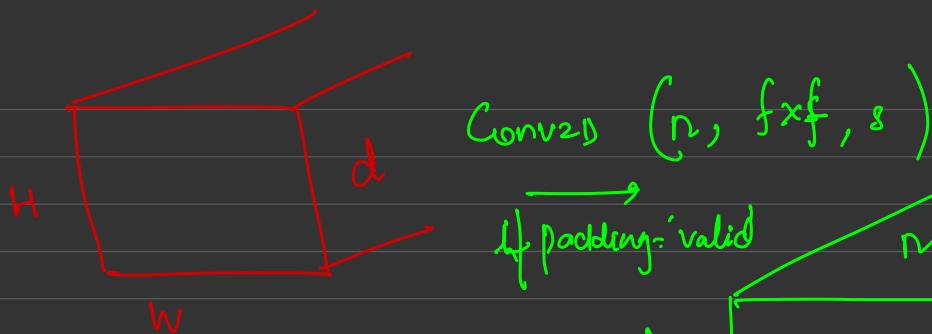


$\xrightarrow{\text{Conv2D } (16, 5 \times 5, \text{valid})}$

$$\#F = \frac{14-5}{1} + 1$$



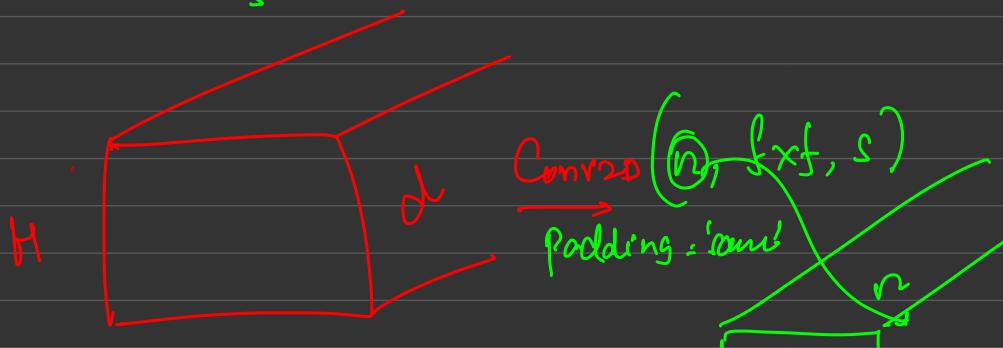




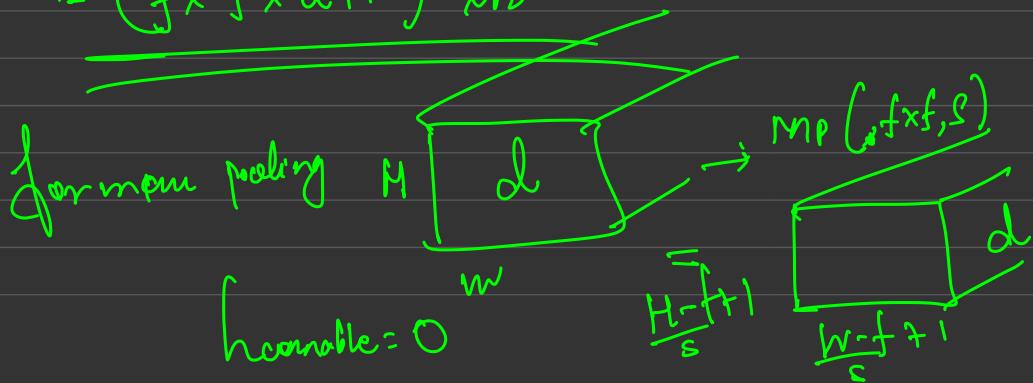
$$H' = \frac{H-f+1}{s} + 1$$

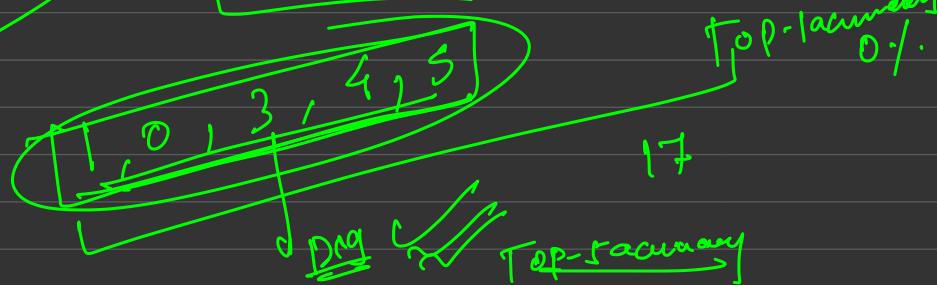
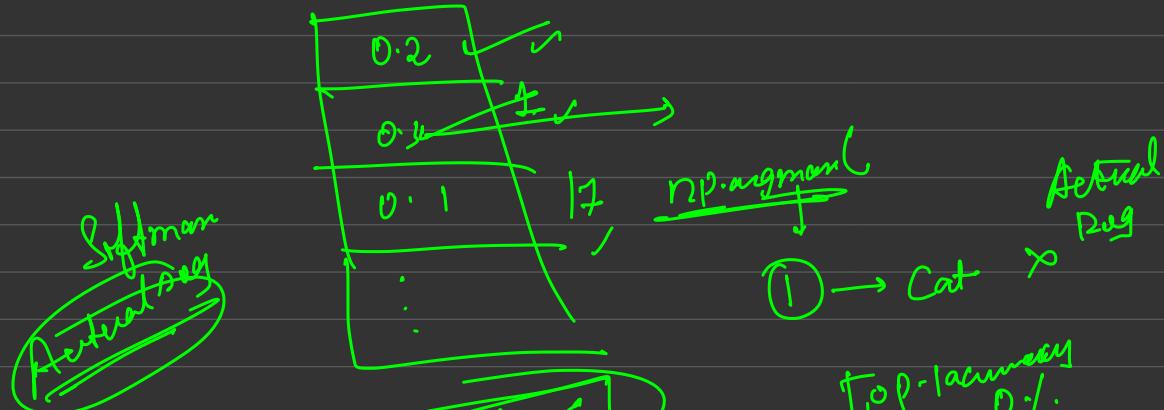
$$W' = \frac{W-f}{s} + 1$$

learnable parameters
 $= (f \times f \times d + 1) \times n$

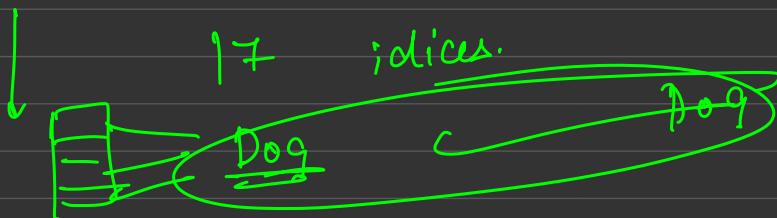


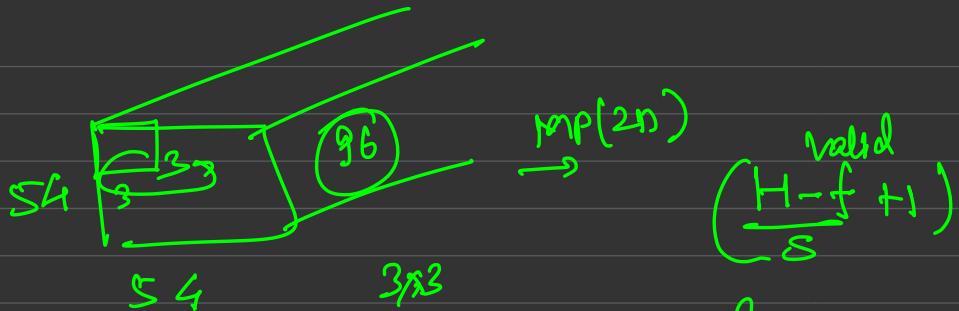
learnable parameters
 $= (f \times f \times d + 1) \times n$





Top-faccuracy



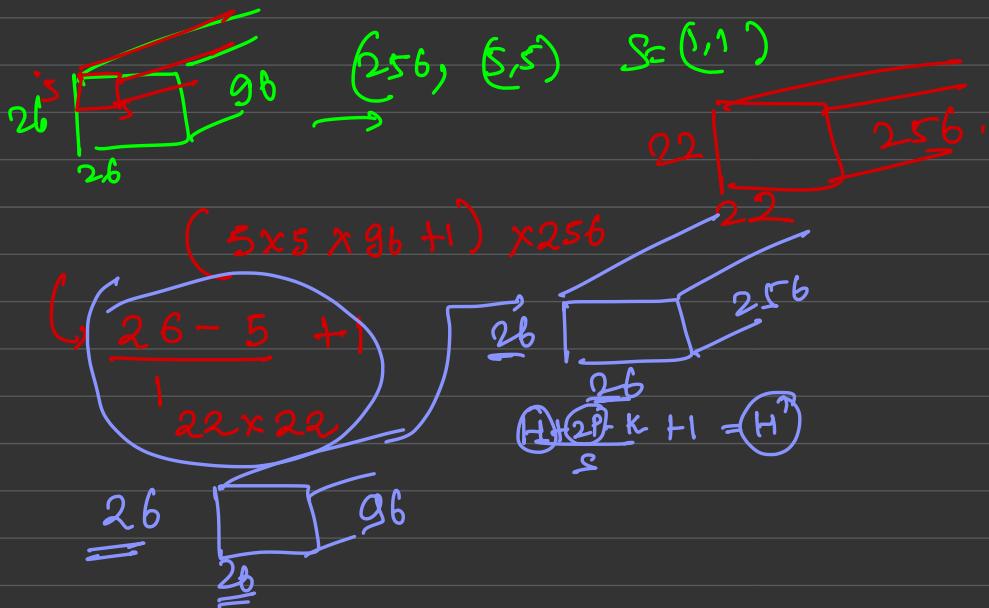


$$\frac{54-3}{2} + 1$$

$$= \frac{H-f+s}{s} + 1$$

$$\frac{26}{2} + 1$$

$$= 26 \times 26 \times 96$$

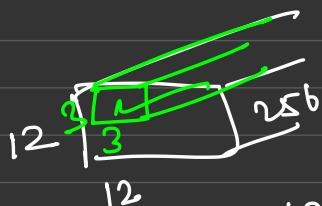




$$\frac{26-3}{2} + 1$$

$$\underbrace{12}_{12}$$

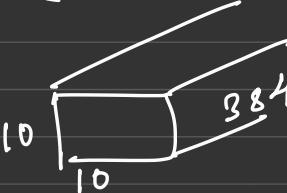
$$256$$

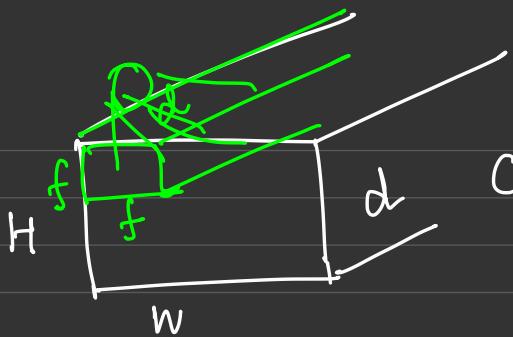


$384, (3,3), (3 \leq 1,1)$

$$(3 \times 3 \times 256 + 1) \frac{12-3}{1} + 1$$

$$\times 384$$





Conv2D ($f \times f, s=s, n$)
no padding

Total learnable parameters
 $= (f \times f \times d + 1) \times n$

padding = 'valid'

$$H'(\text{output}) = \frac{H-f+1}{s}$$

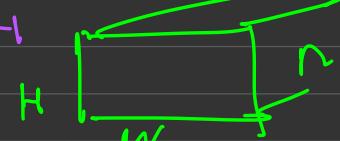


output shape

$$\text{padding} = \text{'Same'} = (H \times w) \times n$$

output shape

$$W'(\text{output}) = \frac{W-f}{s} + 1$$

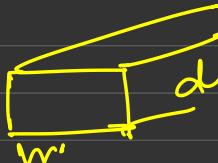


Max Pool (2D)

$(f \times f, s \times s)$



$$H' = \frac{H-f+1}{s}$$



learnable parameters =
 $w' = \left(\frac{W-f}{f} + 1 \right)$



data / image augmentation



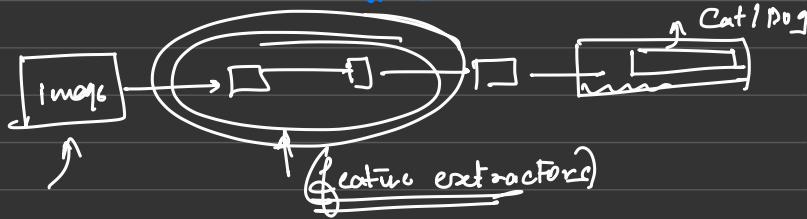
VGG net architecture

Trans for learning.

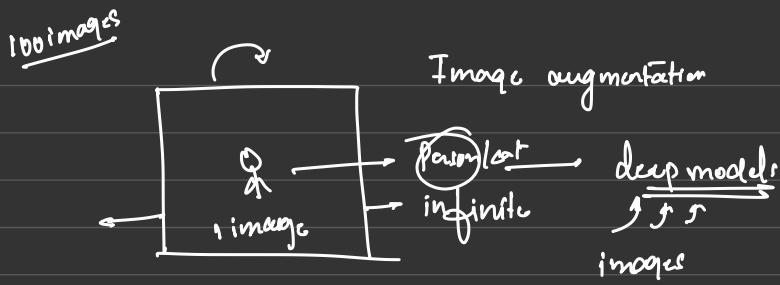
Architectures → don't extremely well on a very big dataset

architecture (VGG)

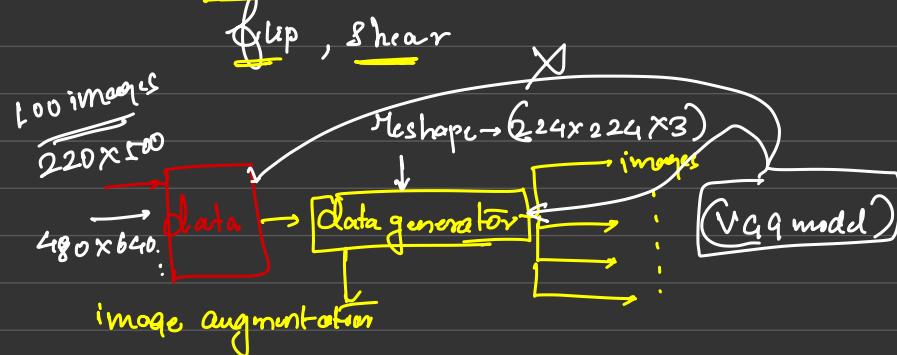
Your data (small) fits into the fact
few(?) layers of
the model.



Transfer learning concept.



relate



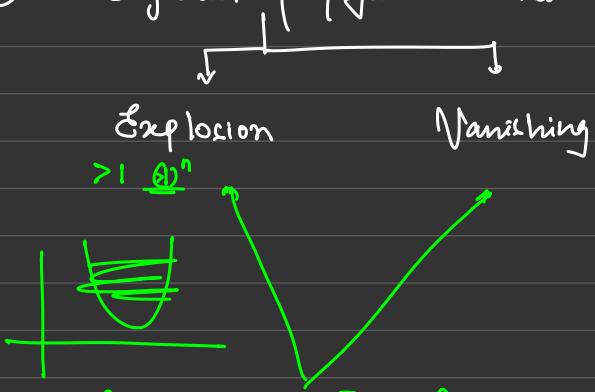
model fit (on data generator)

Resnet Architecture

- ① won first place in ILSVRC 2015
- ① won first place in Imagenet, Imagenet detection
- ① Coco detection, coco segmentation

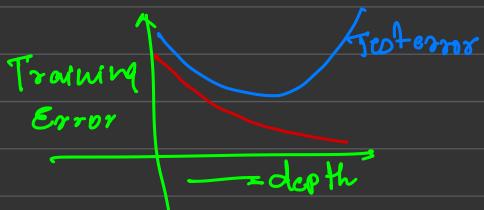
→ depth →

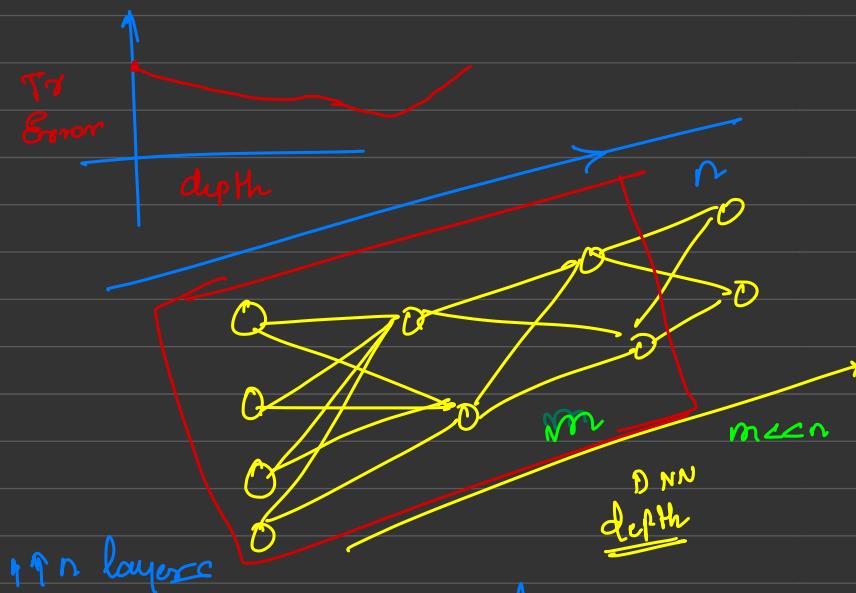
- ① The gradient propagation \rightarrow Chain Rule ($x \times x \times x'$)



- ① Batch Normalization, ReLU, Xavier (init)

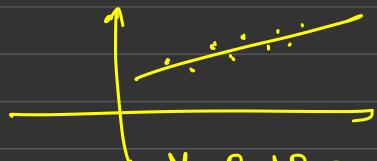
- ① Models deeper





there will be certain (middle) layer would have already learned the most optimal model.

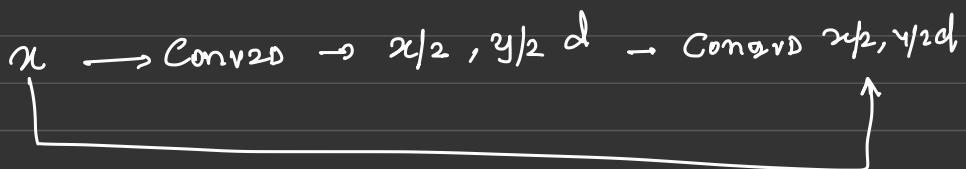
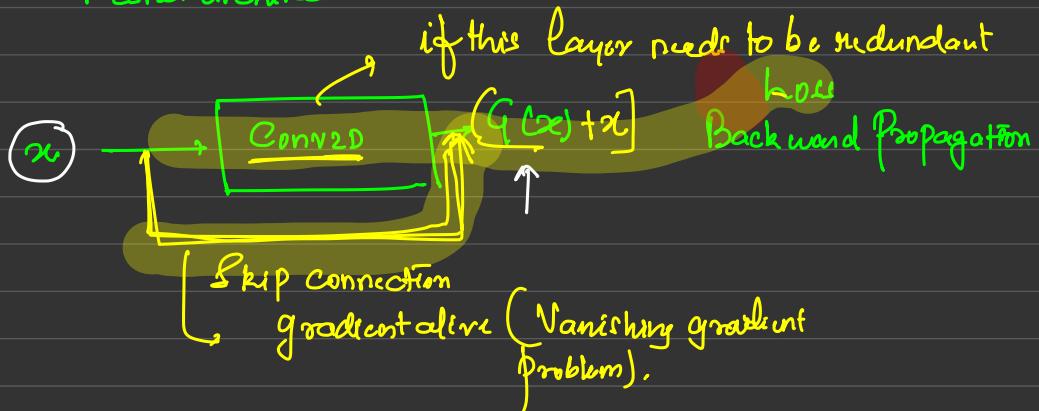
The remaining $n-m$ layers to be an identity map.



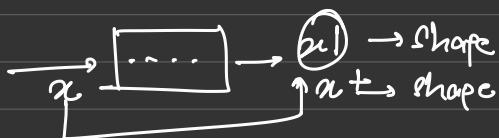
$$Y = B_0 + B_1 x_1 \rightarrow \text{linear equation}$$

$$Y = B_0 + B_1 x_1 + B_2 x_1^2 \rightarrow \text{Quadratic}$$

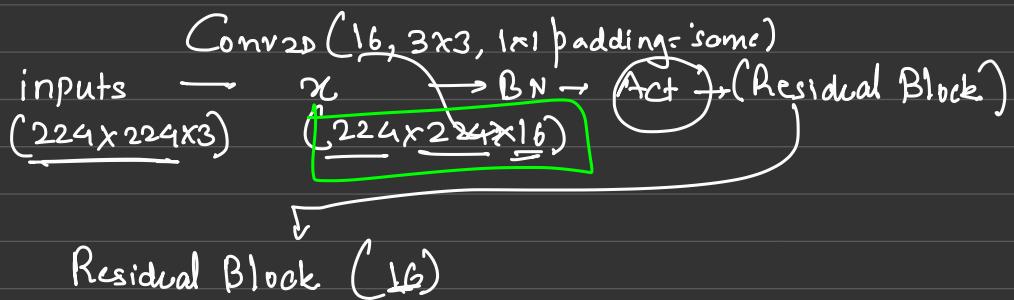
Resnet architecture



Residual Connection (skip connection)



Code understanding



$$y = \text{Conv2D}((16, 3 \times 3), (1 \times 1), \text{padding: 'same'})(x)$$

$$y = \text{BN}(y)$$

$$y = \text{Act}(y)$$

$$\hookrightarrow (224 \times 224 \times 16)$$

$$y = \text{Conv2D}((16, (3 \times 3), (1 \times 1), \text{Same}))(y)$$

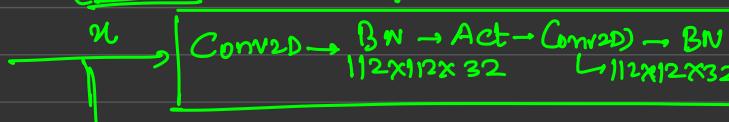
$$y = \text{BN}(y) \rightarrow (224 \times 224 \times 16)$$

$$y = \text{Add}(y, x)$$

downsample = False



$$(224 \times 224 \times 16) \xrightarrow{\text{downsample = True}} (112 \times 112 \times 32)$$



$$112 \times 112 \times 32$$

$$\xrightarrow{\text{Conv2D}((32, (2 \times 2))} 112 \times 112 \times 32$$

Factor of 4 of size reduction