WORKSHEET-1

(8) Input image :-

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

$n = 6 \times 6$

$*$

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

$f = 3 \times 3$ → ? (4×4)

Assume $s = 1$ (stride) ; $p = 0$ (padding)

o/p size $= ((n - f + 2p)/s) + 1$

$= [(6 - 3 + 2 \times 0)/1] + 1$

$= 4 \times 4$

Convolution operations —

1) $10\times1 + 10\times0 + 10\times-1 + 10\times1 + 10\times0 + 10\times-1 + 10\times1 + 10\times0 + 10\times-1 = 0$

10×1	10×0	10×-1
10×1	10×0	10×-1
10×1	10×0	10×-1

2)

10×1	10×0	0×-1
10×1	10×0	0×-1
10×1	10×0	0×-1

$\Rightarrow 10+0+0+10+0+0+10+0+0$

$\Rightarrow 30$

3)

10×1	0×0	0×-1
10×1	0×0	0×-1
10×1	0×0	0×-1

$= 10+0+0+10+0+0+10+0+0 = 30$

4)

0×1	0×0	0×-1
0×1	0×0	0×-1
0×1	0×0	0×-1

$= 0$

5)

10×1	10×0	10×-1
10×1	10×0	10×-1
10×1	10×0	10×1

$= 0$

6) $\begin{array}{|c|c|c|}\hline 1\times1 & 0\times0 & 0\times-1 \\ \hline 1\times1 & 0\times0 & 0\times-1 \\ \hline 1\times1 & 0\times0 & 0\times-1 \\ \hline \end{array} = 30$

7) $\begin{array}{|c|c|c|}\hline 1\times1 & 0\times0 & 0\times-1 \\ \hline 1\times1 & 0\times0 & 0\times-1 \\ \hline 1\times1 & 0\times0 & 0\times-1 \\ \hline \end{array} = 30$

8) $\begin{array}{|c|c|c|}\hline 0\times1 & 0\times0 & 0\times-1 \\ \hline 0\times1 & 0\times0 & 0\times-1 \\ \hline 0\times1 & 0\times0 & 0\times-1 \\ \hline \end{array} = 0$

So far o/p =

0	30	30	0
0	30	30	0

9) $\begin{array}{|c|c|c|}\hline 1\times1 & 0\times0 & 0\times-1 \\ \hline 1\times1 & 0\times0 & 0\times-1 \\ \hline 1\times1 & 0\times0 & 0\times-1 \\ \hline \end{array} = 0$

10) $\begin{array}{|c|c|c|}\hline 1\times1 & 0\times0 & 0\times-1 \\ \hline 1\times1 & 0\times0 & 0\times-1 \\ \hline 1\times1 & 0\times0 & 0\times-1 \\ \hline \end{array} = 30$

11) $\begin{array}{|c|c|c|}\hline 1\times1 & 0\times0 & 0\times-1 \\ \hline 1\times1 & 0\times0 & 0\times-1 \\ \hline 1\times1 & 0\times0 & 0\times-1 \\ \hline \end{array} = 30$

12) $\begin{array}{|c|c|c|}\hline 0\times1 & 0\times0 & 0\times-1 \\ \hline 0\times1 & 0\times0 & 0\times-1 \\ \hline 0\times1 & 0\times0 & 0\times-1 \\ \hline \end{array} = 0$

o/p =

0	30	30	0
0	30	30	0
0	30	30	0

13) $\begin{array}{|c|c|c|}\hline 1\times1 & 0\times0 & 0\times-1 \\ \hline 1\times1 & 0\times0 & 0\times-1 \\ \hline 1\times1 & 0\times0 & 0\times-1 \\ \hline \end{array} = 0$

14) $\begin{array}{|c|c|c|}\hline 1\times1 & 0\times0 & 0\times-1 \\ \hline 1\times1 & 0\times0 & 0\times-1 \\ \hline 1\times1 & 0\times0 & 0\times-1 \\ \hline \end{array} = 30$

15) $\begin{array}{|c|c|c|}\hline 1\times1 & 0\times0 & 0\times-1 \\ \hline 1\times1 & 0\times0 & 0\times-1 \\ \hline 1\times1 & 0\times0 & 0\times-1 \\ \hline \end{array} = 30$

16) $\begin{array}{|c|c|c|}\hline 0\times1 & 0\times0 & 0\times-1 \\ \hline 0\times1 & 0\times0 & 0\times-1 \\ \hline 0\times1 & 0\times0 & 0\times-1 \\ \hline \end{array} = 0$

Final o/p —

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

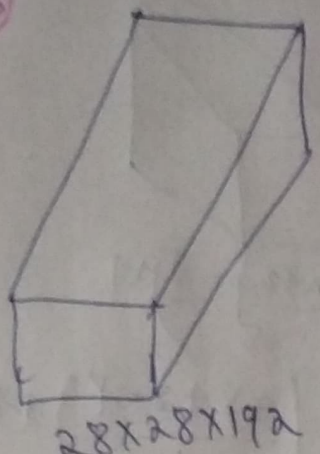
→ This filter is a vertical edge detector.

→ It detects vertical transitions from left to right.

→ Produces strong positive ~~response~~ responses (30) at patches that straddle the edge from 1 to 0 & zero elsewhere.

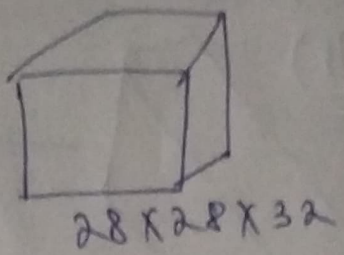
... the no. of convolutional

(13)



CONV

$f = 28 \times 28 \times 192$
$\# f = 32$
same conv
$(p = 1)$
$s = 1$



$$o/p = \left(\frac{n + 2p - f}{s}\right) + 1$$

$$o/p = \left(\frac{28 + 2 - 28}{1}\right) + 1 = 3 \times 3$$

→ which is not we're seeing in o/p dimension.

→ so, probably the given 'f' dimensions are wrong.

Assume $f = 3 \times 3 \times 192$;

$$o/p = \left(\frac{n + 2p - f}{s}\right) + 1 = \left(\frac{28 + 2 - 3}{1}\right) + 1 = 28 \times 28 \times 32$$

Computational cost / no. of multiplications —

o/p = $28 \times 28 \times 32$ → 32 channels

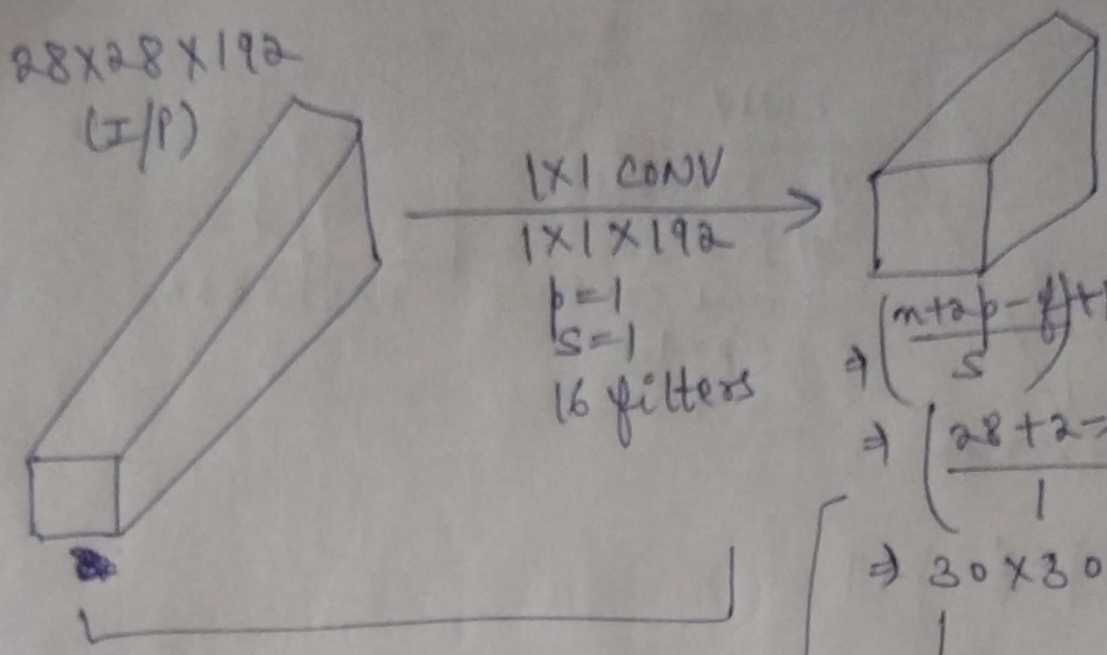
Multiplications per o/p channel = $3 \times 3 \times 192$

so, total multiplications = $28 \times 28 \times 32 \times 3 \times 3 \times 192$

= 43,352,064

1X1 conv —

↳ Reduces the computational complexity.



Cost = 30×30×16×1×1×192
= 2,764,800

Cost = 28×28×32×
~~30×30×16~~
5×5×16

= 10,035,200

Total cost = 2,764,800 + 10,035,200
= 12,800,000

Earlier cost was 143,352,064 ⟶ reduced to 12,800,000. using 1×1 CONV.

① last FC layer $z_{(i)}$ = [2.0, 1.0, 0.1]
Prob. for 3 classes?
Apply softmax function ⇒ $P_i = \frac{e^{z_i}}{\sum_{i=1}^{3} e^{z_i}}$

$e^{2.0} = 7.389$
$e^{1.0} = 2.718$
$e^{0.1} = 1.105$

Class-1 ⇒ $P_1 = \frac{e^{2.0}}{11.212\ (\text{total})} = \frac{7.389}{11.212} = 0.659$

Class-2 ⇒ $P_2 = \frac{2.718}{11.212} = 0.242$

Class-3 ⇒ $P_3 = \frac{1.105}{11.212} = 0.099$

(9) Impact of increasing the no. of convolutional layers in CNN?

↳ More hierarchical feature extraction

- early layers ⇓ learn simple features (edges, corners, colours)
- middle layers ⇓ learn combinations of these (shapes, textures, patterns)
- deeper layers ⇓ very abstract features (face, etc.)

↳ More layers improve performance on complex tasks (like in case of ImageNet classification).

↳ Risk of overfitting due to too many parameters; can memorize rather than generalizing; so to avoid this, regularization techniques such as — dropout, etc. is needed.

↳ Vanishing or exploding gradients problem may arise; so to avoid this, ReLU is used or batch normalization is done.

↳ High computation and memory cost.

(10) (a) Weight sharing in CNN —

* In CNN, the same set of weights (Kernel or filter) is applied across different spatial locations of the input.

* Advantages
  - faster computation & easy to parallelize
  - huge reduction in no. of parameters
  - translation invariance (i.e. network learns features that are position independent).

(b) Sparsity of connections —

* In CNN, each neuron (or filter) connects only to a small local region of the input i/p a "receptive field".
* Advantages → Reduced no. of parameters.
  → Network learns spatial hierarchies (edges → shapes → objects)
  → Reduce overfitting, better generalization.

(7) Keep probability (dropout) —

* Dropout prevents overfitting by setting a fraction of neuron's activations to zero.
* Keep prob. (p) is the probability that a neuron remains active (not dropped).

$$p = 1 - \text{dropout rate (DR)}$$

e.g.:- DR = 0.2
$p = 0.8$ (i.e, each neuron has 80% chance to stay active).

* Let, h = vector of activations before dropout.
  u = random binary mask ($1 \to p$, and $0 \to 1-p$).

Then, o/p after dropout —

$$h_{drop} = u \odot h$$

$$\mathbb{E}[h_{drop}] = \mathbb{E}[u \odot h] = p \cdot h$$

Means that on avg. o/p activations become smaller by a factor of p during training. So, when dropout is disabled at test time, activations are suddenly larger by about $1/p$ times.

* Solution — Scale activations by $1/p$ during training $\Rightarrow h_{drop} = \frac{u \odot h}{p}$

$$\mathbb{E}[h_{drop}] = \mathbb{E}\left[\frac{u \odot h}{p}\right] = \frac{p \cdot h}{p} = h$$

Network behaves consistent train & test.

∴ Expected activation stays same as in non-dropout case.

(4) I/P H.L-1 O/P
3 neurons 4 neurons 2 neurons

Total no. of parameters = ?

Ans:- For layer-1 :-
wts. = 3×4 = 12
bias = 4
16

For layer-2 :-
wts. = 4×2 = 8
bias = 2
10

Total = 16+10 = 26 parameters.

(6) $X = [2, 4, 6, 8, 10]$
gamma = 2
$\beta = +1$
Batch normalization — O/P = ?

Ans:- $\mu = \frac{2+4+6+8+10}{5} = 6$

$$\sigma^2 = \frac{1}{5}[(2-6)^2 + (4-6)^2 + (6-6)^2 + (8-6)^2 + (10-6)^2]$$

$\sigma^2 = 8$; $\sigma = \sqrt{8} = 2.828$

$$\hat{x} = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \rightarrow \text{ignore it} \quad = \frac{x_i - \mu}{\sigma}$$

$$\hat{x}_1 = \frac{2-6}{2.828} = -1.414$$

$\hat{x}_2 = -0.707$

$\hat{x}_3 = 0$

$\hat{x}_4 = 0.707$

$\hat{x}_5 = 1.414$

$\tilde{x} = \gamma\hat{x} + \beta$ ⇒ Scall & shift

$\tilde{x}_1 = 2 \times (-1.414) + 1 = -1.828$

$\tilde{x}_2 = 2 \times (-0.707) + 1 = -0.414$

$\tilde{x}_3 = 2 \times (0) + 1 = 1$

$\tilde{x}_4 = 2 \times (0.707) + 1 = 2.414$

$\tilde{x}_5 = 2 \times (1.414) + 1 = 3.828$

(14)

I/P $32 \times 32 \times 192$

3×3 CONV $\times 192$
$P = 1$
$S = 1$
$\# 128$

$\left(\frac{n + 2p - f}{s}\right) + 1$

$\Rightarrow \left(\frac{32 + 2 - 3}{1}\right) + 1$

$\Rightarrow 32 \times 32 \times 128$

5×5 CONV $\times 192$
$P = \frac{f-1}{2} = 2$
$S = 1$
$\# 32$

$\Rightarrow \left(\frac{32 + 4 - 5}{1}\right) + 1$

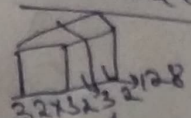
$\Rightarrow 32 \times 32 \times 32$

(a) Multiplications

$\Rightarrow (32 \times 32 \times 128 \times 3 \times 3 \times 192)$
$+$
$(5 \times 5 \times 192 \times 32 \times 32 \times 32)$

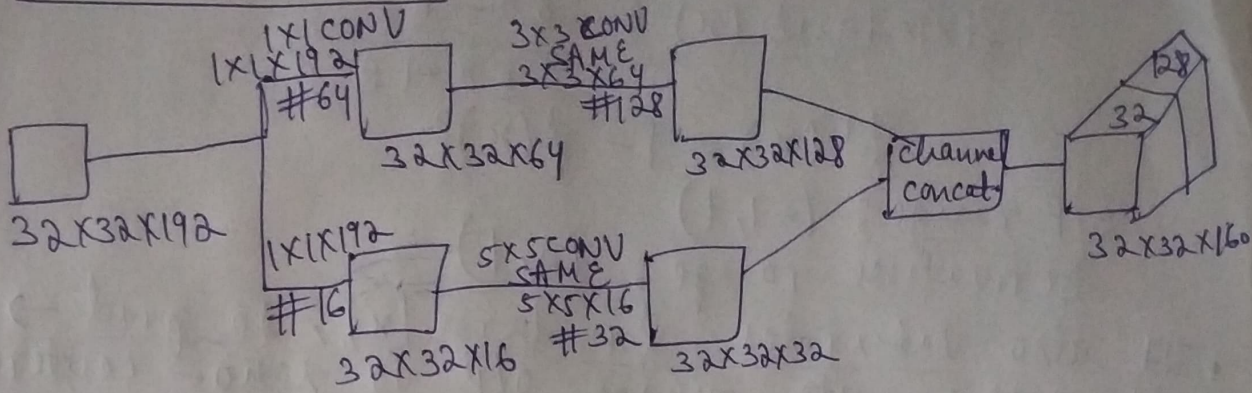
$\Rightarrow 383,778,816 \approx 400$ million

channel concatenation



(b) 1×1 CONV to be applied to ↓ no. of multiplications.

New architecture —



3X3 branch :-

$32 \times 32 \times 64 \times 192 = 12582912$ +

$32 \times 32 \times 128 \times 3 \times 3 \times 64 = 75497472$

88080384

5X5 branch :-

$32 \times 32 \times 16 \times 192 = 3145728$ +

$32 \times 32 \times 32 \times 5 \times 5 \times 16 = 13107200$

16252928

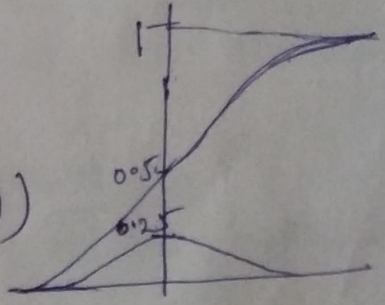
$\therefore$ Total $= 88080384 + 16252928 = 104,333,312$
$\approx$ 100 million

Hence, cost reduced with new architecture of 1X1 CONV, thus reduces the computational cost to 4 folds.

(5) Sigmoid func° —

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

$$f'(x) = f(x)(1-f(x))$$



o/p range: (0, 1)
derivative range: (0, 0.25)

- For large +ve or −ve inputs, sigmoid saturates near 1 or 0. In those regions, $f'(x) \approx 0$.
- During backprop, multiplying by many small derivatives (≈ 0.1 or less), leads to vanishing grad.

Tanh function —

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

$$f'(x) = 1 - f(x)^2$$

O/P range: $(-1, 1)$

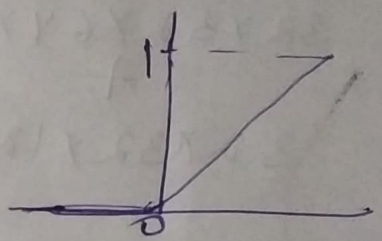
Derivative range: $(0, 1)$

- It also saturates for large $|x|$, so grad. $\to 0$.
- However, its mean O/P is zero-centered, which helps slightly.

ReLU func. —

$$f(x) = \max(0, x)$$

$$f'(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$



O/P range: $[0, \infty)$

Derivative range: 1 for +ve I/P, 0 for -ve I/P

- For +ve I/P, grad. = 1 → grad. neither vanish nor explode.
- For -ve I/P, grad. = 0 → neuron is inactive ("dead ReLU").
- When ~~wts~~ weights are initialized poorly, then explodes gradient.

(15) Option - (a) ~~is better~~

↳ Flattening destroys the 2-D spatial relationships in image.

↳ Nearby px are treated as unrelated features.

↳ weights = 1024 × 5 = 5120 ; or no feature extraction or hierarchical learning, can't learn local patterns

Option - (b) — BETTER

↳ captures local patterns

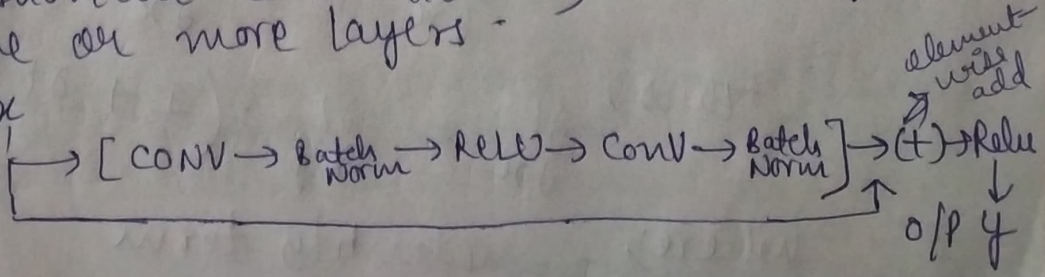
↳ local connectivity ↓es no. of parameters; and wt. sharing also. (Translation invariant pattern learning)

↳ Better scalability & generalization

(11) ResNet —

* Earlier, researchers tried to make network deeper (20 / 50 / 100 layers) to get better accuracy, but it performed worse than shallower one (Training error ↑ed, Vanishing gradient). So, then idea of "ResNet" came in; i.e. if some layers do nothing (act as identity function) then deeper network will act as shallow ones.

* It introduces "skip connections" (K/a "shortcut connections") that bypasses one or more layers.



$y = F(x, W_i) + x$

where;

$F(x, W_i)$ = o/p of few layers (like two CONV → Batch Norm → ReLU layers) parameterized by W_i

The skip connection directly adds x to the output of $F(x, W_i)$.

* To make residual block to behave like an identity func., i.e; $y = x$, then, $F(x, W_i) = 0$

$y = 0 + x = x$ (or)

Residual func. outputs zero & entire block just pass I/p through unchanged.

* Thus, gradient can flow directly via skip connections, avoiding vanishing gradients.

(12) * Problem :- without skip connections, backprop. to

chain rule :- $\frac{\partial L}{\partial x_0} = \frac{\partial L}{\partial x_L} \cdot \frac{\partial x_L}{\partial x_{L-1}} \cdot \frac{\partial x_{L-1}}{\partial x_{L-2}} \cdots \frac{\partial x_1}{\partial x_0}$

If each derivative has values smaller than 1, their products gets smaller & smaller as we move backward (i.e; Vanishes gradient)

* Soln :- Gradient flow via a residual block —

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial x}$$

since, $y = F(x, W_i) + x$, so

$$\frac{\partial y}{\partial x} = \frac{\partial F(x, W)}{\partial x} + I$$

Thus,

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} \cdot \left(I + \frac{\partial F(x, W)}{\partial x} \right)$$

where, I = Identity term

Even if $\frac{\partial F(x,W)}{\partial x}$ becomes very small, 'I' ensures gradients flow directly backward via skip connection. thus avoids the above issue of explode grad.

(16) CNN Visualization methods →

1) Filter (Kernel) Visualization

* Each filter in CNN learns to detect a particular feature or pattern (layer)
  → edges / colour gradients, deeper layers
  → more abstract features).
* Take learned filter weights (from conv layers) after training.

↓

For the 1st conv layer, each filter directly operates on image pixels, so you can visualize it as a small image patch.

↓

Can display these filters as grids of small images.

Thus, can detect if the network has learnt meaningful patterns or noise.

2) Activation (feature map) Visualization

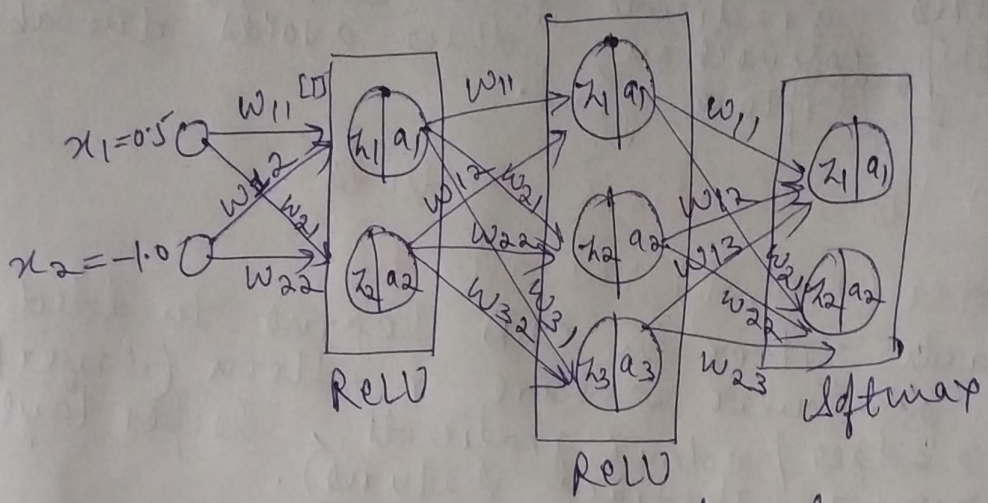
* Input an image to trained CNN.
  Extract o/p of chosen layer (activations)
  Visualise each as a heatmap.

Early layer act. (edges) → Mid-level act. (ears/legs) → Deep layers act. (whole face/body)

3) Maximally activating image patches

* To look at which parts of real images cause strongest activations in neuron.
* Pass many images via CNN → Record the patches that maximise act.
  Display those patches.

(2) Given :- $X = \begin{bmatrix} 0.5 \\ -1.0 \end{bmatrix}$, no bias term



Initialize weights randomly;

W_1 (layer-1) $= \begin{bmatrix} w_{11} & w_{12} \\ 0.2 & -0.1 \\ w_{21} & w_{22} \\ 0.4 & 0.3 \end{bmatrix}_{2\times 2}$, W_2 (layer-2) $= \begin{bmatrix} w_{11} & w_{12} \\ 0.1 & 0.2 \\ w_{21} & w_{22} \\ -0.3 & 0.4 \\ w_{31} & w_{32} \\ 0.2 & -0.2 \end{bmatrix}_{3\times 2}$

, W_3 (o/p layer) $= \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ 0.3 & -0.2 & 0.1 \\ w_{21} & w_{22} & w_{23} \\ -0.1 & 0.2 & 0.4 \end{bmatrix}_{2\times 3}$

Forward pass :-

layer-1 —

$z_1^{[1]} = w_{11}^{[1]} x_1 + w_{12}^{[1]} x_2$ and $w_{21}^{[1]} x_1 + w_{22}^{[1]} x_2$

$z^{[1]} = W_1 X$

$z^{[1]} = \begin{bmatrix} 0.2 & -0.1 \\ 0.4 & 0.3 \end{bmatrix} \begin{bmatrix} 0.5 \\ -1.0 \end{bmatrix} = \begin{bmatrix} 0.2 \\ -0.1 \end{bmatrix} \begin{matrix} \rightarrow z_1^{[1]} \\ \rightarrow z_2^{[1]} \end{matrix}$

$a^{[1]} = \text{ReLU}(z^{[1]}) = \begin{bmatrix} \max(0, 0.2) \\ \max(0, -0.1) \end{bmatrix}$

$a^{[1]} = \begin{bmatrix} 0.2 \\ 0.0 \end{bmatrix} \begin{matrix} \rightarrow a_1^{[1]} \\ \rightarrow a_2^{[1]} \end{matrix}$

layer-2 —

$z^{[2]} = \begin{bmatrix} 0.1 & 0.2 \\ -0.3 & 0.4 \\ 0.2 & -0.2 \end{bmatrix} \begin{bmatrix} 0.2 \\ 0.0 \end{bmatrix} = \begin{bmatrix} 0.02 \\ -0.06 \\ 0.04 \end{bmatrix}$

$a^{[2]} = \begin{bmatrix} 0.02 \\ 0.0 \\ 0.04 \end{bmatrix}$

layer-3 —

$$z^{[3]} = \begin{bmatrix} 0.3 & -0.2 & 0.1 \\ -0.1 & 0.2 & 0.4 \end{bmatrix} \begin{bmatrix} 0.02 \\ 0.0 \\ 0.04 \end{bmatrix}$$

→ 0.006 + 0 + 0.004 = 0.01

→ −0.002 + 0 + 0.016 = 0.014

$$z^{[3]} = \begin{bmatrix} 0.01 \\ 0.014 \end{bmatrix}$$

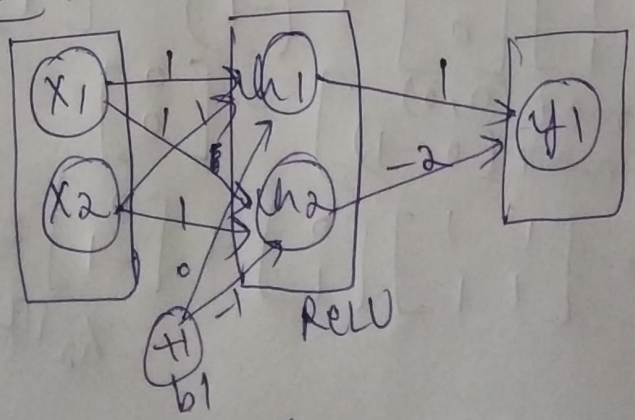
$$a^{[3]} = \begin{bmatrix} \max(0, 0.01) \\ \max(0, 0.014) \end{bmatrix} = \begin{bmatrix} 0.01 \\ 0.014 \end{bmatrix} \Rightarrow \text{raw logits}$$

Softmax applied on raw logits —

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

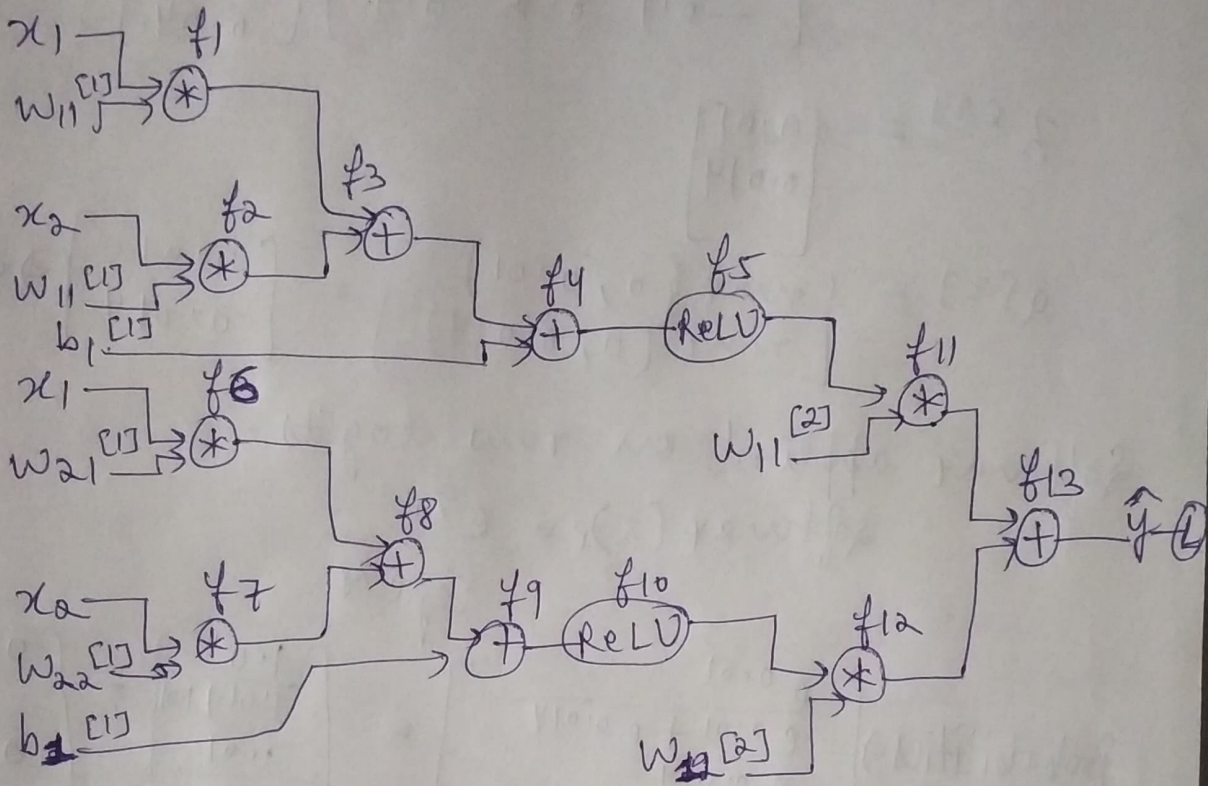
$$\text{Probabilities} \Rightarrow \begin{bmatrix} \frac{e^{0.01}}{e^{0.01} + e^{0.014}} \\ \frac{e^{0.014}}{e^{0.01} + e^{0.014}} \end{bmatrix} = \begin{bmatrix} \frac{1.01}{1.01 + 1.01} \\ \frac{1.01}{1.01 + 1.01} \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \approx \hat{y}$$

(3) Given :-



x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

Computational graph :-



Forward pass :-

i = sample

layer-1 :- $z_i^{[1]} = W_1 x_i + b_1$; i = sample

sample-1 $\Rightarrow z_1^{[1]} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$

$a_1^{[1]} = \begin{bmatrix} \max(0,0) \\ \max(0,-1) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

sample-2 $\Rightarrow z_2^{[1]} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix}$

$z_2^{[1]} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

$a_2^{[1]} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

sample-3 $\Rightarrow z_3^{[1]} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$

$a_3^{[1]} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

sample-4 $\Rightarrow z_4^{[1]} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix}$

$z_4^{[1]} = \begin{bmatrix} 2 \\ 1 \end{bmatrix} = a_4^{[1]}$

O/P layer :-

sample-1 $\Rightarrow z = \begin{bmatrix} 1 \\ -2 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow a = \begin{bmatrix} 0 \\ 0 \end{bmatrix} = 0$

sample-2 $\Rightarrow z = \begin{bmatrix} 1 \\ -2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \Rightarrow a = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 1$

sample-3 $\Rightarrow z = \begin{bmatrix} 1 \\ -2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \Rightarrow a = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 1$

sample-4 $\Rightarrow z = \begin{bmatrix} 1 \\ -2 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ -2 \end{bmatrix} \Rightarrow a = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$

$\Downarrow$

$$O/P \Rightarrow \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \hat{y} = y$$

$$\text{loss} = \frac{1}{2}\sum_{i=1}^{4} (\hat{y} - y)^2 = \frac{1}{2}(0+0+0+0) = 0$$

(L)

Backward pass :-

Since loss (L) is exactly zero, all the gradients of loss w.r.t. x, W and b will also be zero.

Update weights :-

$W := W - \eta \nabla_W L$ with $\nabla_W L = 0$, every parameter (weight) remains unchanged.