CNN Architecture :-

i) Convolution Operation :-

assume stride = 1

eg. image with 2 channels

1	0	0
0	1	0
0	0	1

0	0	1
0	1	0
1	0	0

eg. filter of each channel

1	0
0	1

0	1
1	0

Resultant Matrix →

2	0
0	2

0	2
2	0

Resultant Height (or) width =

((Image Height or, width − Filter Height (or, width)) / stride) + 1

In this eg :-

∴ resultant ht or, wd = ((3−2)/1) + 1 = 2 × 2

⇒ using im2col approach

channel 1 : $\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$ 4×4

filter 1 : $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

img size = (1, 2, 4, 4)

channel 2 : $\begin{bmatrix} 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 \\ 25 & 26 & 27 & 28 \\ 29 & 30 & 31 & 32 \end{bmatrix}$ 4×4

filter 2 : $\begin{bmatrix} 1 & -1 \\ 1 & 0 \end{bmatrix}$

filter size = (2, 2, 2)

im2col ⟹ matrix

im2col

1	2	3	5	6	7	9	10	11
2	3	4	6	7	8	10	11	12
5	6	7	9	10	11	13	14	15
6	7	8	10	11	12	14	15	16
17	18	19	21	22	23	25	26	27
18	19	20	22	23	24	26	27	28
21	22	23	25	26	27	29	30	31
22	23	24	26	27	28	30	31	32

8×9

generally :

⇒ img (batch_size, channels, ht, wd)

⇒ filter (filters, channels, ht, wd)

⇒ im2col size = ((channels * filter(ht) * filter(wd)), (new_ht * new_wd * batch_size))

⇒ convolution operation resultant matrix size = ((Image ht/wd − filter ht/wd)/stride) + 1 = (new_ht, new_wd)

Algorithm to calculate Convolution using im2col method :-

step1 : Convert each stride of the convolutional filter over an image, into a column of a matrix. (X_im2col)

step2 : Flatten convolutional filter across the no of filters. (conv_flatten)

step3 : Calculate C = (conv_flatten) @ (X_im2col) ⇒ @ − matrix multiplication

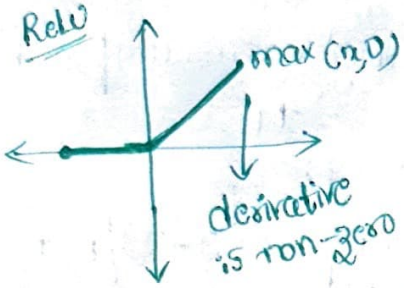
step4 : Reshape C to fit the resulting shape of matrix.

ReLU: → It is a non-linear activation function which filters out negative values.

→ It does not face problems such as vanishing gradients.

→ It result in non-zero gradients for positive values.

ReLU



Max pooling: → It is a process of extract low level features in the image. This is done by picking image chunks of pre-determined sizes, and keeping the largest values from each of these chunks

2	1	-11	0
0	10	-15	-16
23	100	2	3
-23	0	7	20

→ Max pooling →

10	0
100	20

filter htlwd = 2
stride = 2

Resultant size (htlwd) = ((image htlwd − filter htlwd)/stride) + 1

→ ((4−2)/2 + 1) = (2, 2)

Soft max function: → It converts a vector of real values to a vector of values that range b/w 0 and 1. The newly transformed vector adds up to 1; the transformed vector becomes a probability distribution. A large value will be transformed to a value that is close to 1, a small value will be close to 0.

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{i=1}^{N} e^{x_i}}$$

if $x_i = [+3, -2, -100]$, denominator $= \sum_{i=1}^{N} e^{x_i} = e^{3} + e^{-2} + e^{-100} = 20.22$

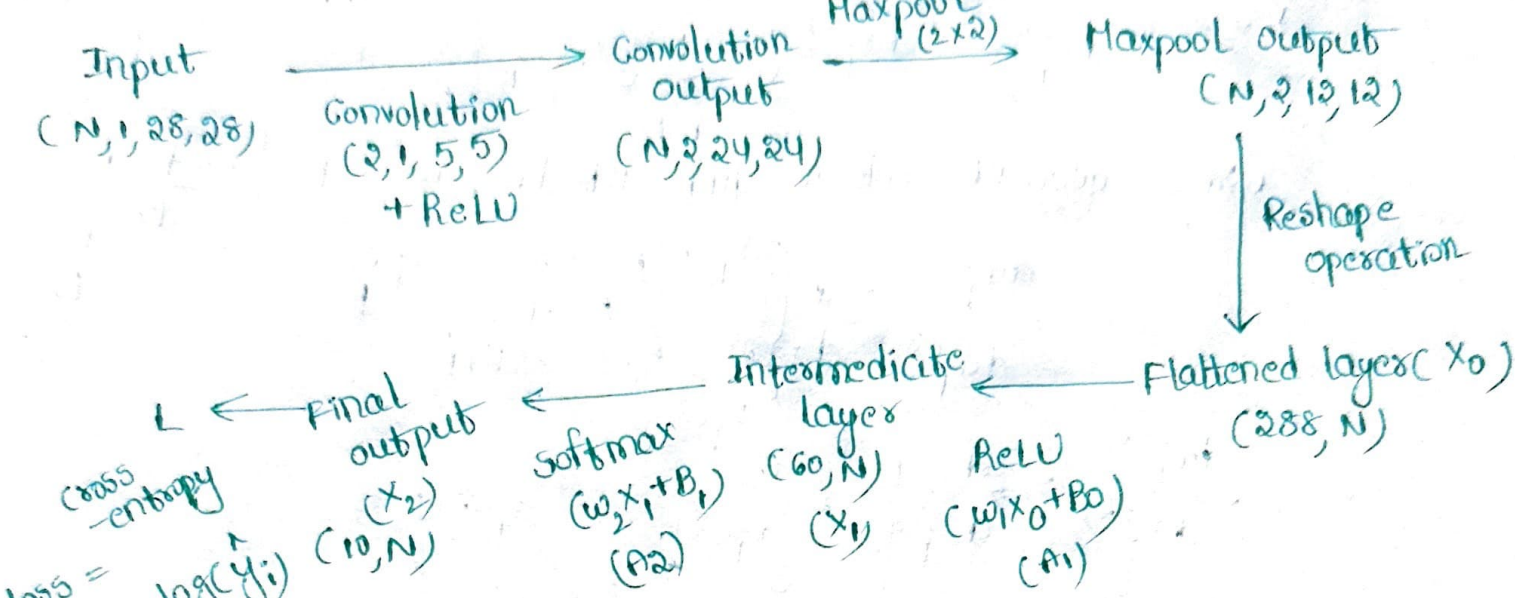
→ $\text{softmax}(3) = \frac{e^{3}}{20.22} = 0.99$

→ $\text{softmax}(-2) = \frac{e^{-2}}{20.22} = 0.0066$

→ $\text{softmax}(-100) = \frac{e^{-100}}{20.22} = 1.83 \times 10^{-45}$ (close to zero)

Back Propagation :

→ consider the below CNN network.



i) Firstly let's find out Gradient descent through the fully connected layers :

Gradient descent, $\theta = \theta - \alpha \frac{dF(\theta)}{d\theta}$

α — learning rate.

1) Calculating derivative of cross entropy with respect of ω_2 and B_1

i.e; $\frac{dL}{d\omega_2}$, $\frac{dL}{dB_1}$:

let $A_2 = \omega_2 X_1 + B_1$

$X_2 = \text{softmax}(A_2)$

$$\frac{dL}{d\omega_2} = \frac{dL}{dX_2} \cdot \frac{dX_2}{dA_2} \cdot \frac{dA_2}{d\omega_2} = \frac{dL}{dX_2} \cdot \frac{dX_2}{dA_2} \cdot \frac{d}{d\omega_2}(\omega_2 X_1 + B_1)$$

$$= \frac{dL}{dX_2} \cdot \frac{dX_2}{dA_2} \cdot (X_1)$$

i) Finding $\frac{dX_2}{dA_2}$:

Let $a_i^{(2)}$ be the softmax output corresponding to $a_i^{(2)}$

$$\Rightarrow \frac{d\,x_i^{(2)}}{da_i^{(2)}} = \frac{d}{da_i^{(2)}} \left[\frac{e^{a_i^{(2)}}}{\sum_{j=1}^{N} e^{a_j^{(2)}}} \right]$$

there are two cases here, when $i=j$, $i \neq j$

when $i=j$, $\frac{da_i^{(2)}}{da_i^{(2)}} = \frac{d}{da_i^{(2)}}\left(\frac{e^{a_i(2)}}{\sum_{i=1}^{K} e^{a_i(2)_K}}\right)$

$\left[\text{softmax}(x) = \frac{e^{a}}{\sum_{i=1}^{K} e^{a_i}}\right]$

using quotient rule of differentiation, we get $\left[\because \frac{d}{dx}\left(\frac{u}{v}\right) = \frac{v\frac{du}{dx} - u\frac{dv}{dx}}{v^2}\right]$

$$\frac{da_i^{(2)}}{da_i^{(2)}} = \frac{d}{da_i^{(2)}}\left[e^{a_i(2)} \Big/ e^{a_i(2)} + \sum_{K \neq i} e^{a_i(2)_K}\right]$$

$$\frac{da_i^{(2)}}{da_i^{(2)}} = \frac{d}{da_i}\left[\frac{e^{a_i(2)}}{e^{a_i(2)} + \sum_{K \neq i} e^{a_K(2)}}\right]$$

$$= \frac{\left(e^{a_i(2)} + \sum_{K \neq i} e^{a_K(2)}\right)\frac{d}{da_i}\left(e^{a_i(2)}\right) - e^{a_i(2)}\frac{d}{da_i}\left(e^{a_i(2)} + \sum_{K \neq i} e^{a_K(2)}\right)}{\left(e^{a_i(2)} + \sum_{K \neq i} e^{a_K(2)}\right)^2}$$

$$\Rightarrow \frac{\left(e^{a_i(2)} + \sum_{K \neq i} e^{a_K(2)}\right)\left(e^{a_i(2)}\right) - e^{a_i(2)} \cdot e^{a_i(2)}}{\left(e^{a_i(2)} + \sum_{K \neq i} e^{a_K(2)}\right)^2}$$

$$\frac{da_i^{(2)}}{da_i^{(2)}} = \frac{e^{a_i(2)} \cdot \sum_{K \neq i} e^{a_K(2)}}{\sum_{K=1}^{N} e^{a_K(2)} \cdot \sum_{K=1}^{N} e^{a_K(2)}} = a_i^{(2)}\left(1 - a_i^{(2)}\right)$$

$\left[a_i^{(2)} = \frac{e^{a_i^{(2)}}}{\sum_{K=1}^{N} e^{a_K^{(2)}}}, \quad 1 - a_i^{(2)} = \frac{\sum_{K \neq i} e^{a_i^{(2)}}}{\sum_{K=1}^{N} e^{a_K^{(2)}}}\right]$

$\therefore \frac{dx_2}{dA_2} \cdot \frac{da_i^{(2)}}{da_i^{(2)}} = a_i^{(2)}\left(1 - a_i^{(2)}\right)$

when $i \neq j$

$$\frac{da_i^{(2)}}{da_j^{(2)}} = \frac{d}{da_j^{(2)}}\left[\frac{e^{a_i^{(2)}}}{\sum_{j=1}^{N} e^{a_j^{(2)}}}\right]$$

$$= \frac{0 - e^{a_i^{(2)}} \cdot e^{a_j^{(2)}}}{\left(\sum_{j=1}^{N} e^{a_j^{(2)}}\right)^2} = \frac{-e^{a_i^{(2)}}}{\sum_{j=1}^{N} e^{a_j^{(2)}}} \cdot \frac{e^{a_j^{(2)}}}{\sum_{j=1}^{N} e^{a_j^{(2)}}} = -x_i^{[2]} \cdot x_j^{[2]}$$

$\therefore$ thus, $\frac{dx_i^{(2)}}{da_i^{(2)}} = \begin{cases} -x_i^{[2]} x_j^{[2]}, & \text{for } i \neq j \\ +x_i^{[2]}(1 - x_i^{[2]}), & \text{for } i = j \end{cases}$

Now,

(i) $\frac{dL}{dx_2}$: let the estimates of the model be $x_i^{[2]}$ and let the actual labels be $z_i^{[2]}$

so, cross entropy $= -\sum_{i=1}^{N} z_i^{[2]} \log(x_i^{[2]})$ — (1)

$$\frac{dL}{dx_2} = -z_i^{[2]} \frac{1}{x_i^{[2]}}$$

$\therefore$ Now, $\frac{dL}{dx_2} \cdot \frac{dx_2}{da_2} = -\sum_{j=i} z_i^{[2]} \frac{1}{x_i^{[2]}} \cdot x_i^{[2]}(1 - x_i^{[2]})$

$$-\sum_{j \neq i} z_j^{[2]} \frac{1}{x_j^{[2]}} (-x_i^{[2]} x_j^{[2]})$$

$$= \sum_j z_j^{[2]} \cdot x_i^{[2]} - \sum_{j=i} z_i^{[2]}(1 - x_i^{[2]})$$

combining these two

$$= \sum_j z_j^{[2]} x_i^{[2]} - \sum_{j=i} z_i^{[2]}$$

$z_j^{[2]}$: one-hot encoded; $z_i^{[2]}$: for single point

$$= x_i^{[2]} - z_i^{[2]}$$

$$\therefore \boxed{\frac{dL}{dw_2} = (x_2 - z_2) * x_1^T}$$

similarly, $dB_1 \rightarrow \frac{dL}{dB_1} = \frac{dL}{dx_2} \cdot \frac{dx_2}{dA_2} \cdot \frac{dA_2}{dB_1}$

$$= (x_2 - t_2) \cdot \frac{d}{dB_1}(w_2 x_1 + B_1) = (x_2 - t_2)$$

$$\therefore dB_1 = (x_2 - t_2)$$

2) Calculating derivative of cross entropy with respect to w_1 and B_0:

i.e; $\frac{dL}{dw_1}, \frac{dL}{dB_0}$: we know that, $A_1 = w_1 x_0 + B_0$

$$x_1 = ReLU(A_1)$$

$$A_2 = w_2 x_1 + B_1$$

$$x_2 = softmax(A_2)$$

$$\therefore \frac{dL}{dw_1} = \frac{dL}{dx_2} \cdot \frac{dx_2}{dA_2} \cdot \frac{dA_2}{dx_1} \cdot \frac{dx_1}{dA_1} \cdot \frac{dA_1}{dw_1}$$

$$= (x_2 - t_2) \cdot w_2^T \cdot (ReLU'(A_1)) \cdot x_0$$

$$\therefore \frac{dL}{dw_1} = \left[w_2^T * (x_2 - t_2)\right] \cdot \left[ReLU'(A_1)\right] * x_0$$

where, '.' → element wise product (dot product)

'*' → matrix multiplication

Similarly, $\frac{dL}{dB_0} = \frac{dL}{dx_2} \cdot \frac{dx_2}{dA_2} \cdot \frac{dA_2}{dx_1} \cdot \frac{dx_1}{dA_1} \cdot \frac{dA_1}{dB_0}$

$$\frac{dL}{dB_0} = (x_2 - t_2) \cdot w_2^T \cdot (ReLU'(A_1)) \cdot (1).$$

$$\therefore \frac{dL}{dB_0} = \left[w_2^T * (x_2 - t_2)\right] \cdot \left[ReLU'(A_1)\right]$$

3) Calculating the derivative of loss function with respect to x_0:

i.e. $\frac{dL}{dx_0}$: $\frac{dL}{dx_0} = \frac{dL}{dx_2} \cdot \frac{dx_2}{dA_2} \cdot \frac{dA_2}{dx_1} \cdot \frac{dx_1}{dA_1} \cdot \frac{dA_1}{dx_0}$

$$\therefore \frac{dL}{dx_0} = \left[w_1^T * \left[w_2^T * (x_2 - t_2)\right]\right] \cdot \left[ReLU'(A_1)\right]$$

(ii) Calculating gradients at Max pooling Layer: we know that the first fully connected layer is a reshaped version of the max pooling layer, we just need to reshape our gradient matrix at the first fully connected layer, back to the shape of the max pooling layer.

$$\therefore \text{delta\_maxpool} = \text{reshape to maxpool o/p shape}\left(\frac{dL}{dx_0}\right)$$

(iii) Calculating gradient at Convolutional layer: To calculate the gradients at the convolutional layer, we need to move each gradient element back to the positions in the convolutional layer, from where the maximum pixel values were extracted.

Forward Pass → let's say x_conv (before pooling) is:

$$X_{conv} = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

If we apply 2×2 max-pooling with a stride of 2, we get

Max pooled output = [4]

Since '4' was the selected max value, the max_indices array stores {max_indices} = [(0,1,1,0,0,0,0,0)]

This means,

(1,1,0) → original location of max value '4' in x_conv.

(0,0,0) → is the corresponding location in delta_maxpool.

Backward Pass: Now, assume the gradient from max pooling (delta_maxpool) is $\delta_{maxpool} = [10]$

Using our max_indices values:

delta_conv[0,1,1,0] = delta_maxpool[0,0,0,0]

This means : $\delta_{conv}(1,1,0) = \delta_{maxpool}(0,0,0) = 10$

∴ The updated delta_conv $= \begin{bmatrix} 0 & 0 \\ 0 & 10 \end{bmatrix}$

→ And finally, we apply desirative of ReLU to delta_conv, to ensure that the gradient only flows through the active neurons (i.e., X_conv > 0).

4) Calculating gradient of convolutional filter : Now that we have found the gradient of the convolutional layer, we need to calculate the gradient of the convolutional filter. This will be used to optimize the filter in each learning step.

Algorithm to calculate gradient of convolutional filter :-

Let 'G' be the gradient matrix for the convolutional layer. This has a dimension of (1, 2, 24, 24). Let 'I' be the input image of shape (1, 2, 28, 28).

i) For a given channel 'C' in 'G', pick an element

ii) Use 5×5 filters with stride = 1 to create 5×5 chunks of image input 'I'.

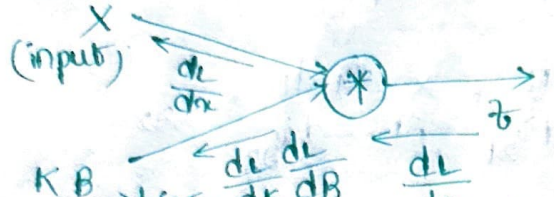
iii) Multiply the chosen gradient element with the consecutive chunks and add them up.

iv) The resultant matrix after the consecutive addition operation is the gradient associated with the channel 'C' in the convolutional filter.

v) Repeat steps 1 to 4 for the rest of the channels in 'G'.

Given that we have to find the gradient with respect to the convolutional filter by pixel, this process will take a lot of time and effort. One way to overcome this is to use the im2col() function.

Back Propagation in Convolution layer :



$$\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} \circledast \begin{bmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{bmatrix} + B = \begin{bmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{bmatrix}$$

here,

$$z_{11} = x_{11}k_{11} + x_{12}k_{12} + x_{21}k_{21} + x_{22}k_{22} + B$$

$$z_{12} = x_{12}k_{11} + x_{13}k_{12} + x_{22}k_{21} + x_{23}k_{22} + B$$

$$z_{21} = x_{21}k_{11} + x_{22}k_{12} + x_{31}k_{22} + x_{32}k_{22} + B$$

$$z_{22} = x_{22}k_{11} + x_{23}k_{12} + x_{32}k_{21} + x_{33}k_{22} + B$$

1) $\Rightarrow \boxed{\frac{dL}{dK}} = \frac{dL}{dz} \cdot \frac{dz}{dK}$

$$\frac{dL}{dK} = \begin{bmatrix} \frac{dL}{dk_{11}} & \frac{dL}{dk_{12}} \\ \frac{dL}{dk_{21}} & \frac{dL}{dk_{22}} \end{bmatrix}$$

$\therefore \frac{dL}{dk_{mn}} = \frac{dL}{dz_{ij}} * \frac{dz_{ij}}{dk_{mn}}$

$z_{ij} \rightarrow z_{11}, z_{12}, z_{21}, z_{22}$

$k_{mn} \rightarrow k_{11}, k_{12}, k_{21}, k_{22}$

From Einstein notation of chain rule multiplication

$$\frac{dL}{dk_{mn}} = \sum \frac{dL}{dz_{ij}} * \frac{dz_{ij}}{dk_{mn}}$$

$$\therefore \frac{dL}{dk_{11}} = \left(\frac{dL}{dz_{11}} \times \frac{dz_{11}}{dk_{11}}\right) + \left(\frac{dL}{dz_{12}} * \frac{dz_{12}}{dk_{11}}\right) + \left(\frac{dL}{dz_{21}} * \frac{dz_{21}}{dk_{11}}\right) + \left(\frac{dL}{dz_{22}} * \frac{dz_{22}}{dk_{11}}\right)$$

$$\frac{dL}{dk_{12}} = \left(\frac{dL}{dz_{11}} * \frac{dz_{11}}{dk_{12}}\right) + \left(\frac{dL}{dz_{12}} * \frac{dz_{12}}{dk_{12}}\right) + \left(\frac{dL}{dz_{21}} * \frac{dz_{21}}{dk_{12}}\right) + \left(\frac{dL}{dz_{22}} * \frac{dz_{22}}{dk_{12}}\right)$$

similarly $\frac{dL}{dk_{21}}$ and $\frac{dL}{dk_{22}}$.

Transforming, $\frac{dL}{dk_{11}}$ by substituting the derivative values,

$$\frac{dL}{dk_{11}} = \frac{dL}{dz_{11}} * x_{11} + \frac{dL}{dz_{12}} * x_{12} + \frac{dL}{dz_{21}} * x_{21} + \frac{dL}{dz_{22}} * x_{22}$$

Similarly, substitute for $\frac{dL}{dk_{12}}, \frac{dL}{dk_{21}}$ and $\frac{dL}{dk_{22}}$.

$$\frac{dL}{dK_{12}} = \frac{dL}{dz_{11}} * X_{12} + \frac{dL}{dz_{12}} * X_{13} + \frac{dL}{dz_{21}} * X_{22} + \frac{dL}{dz_{22}} * X_{23}$$

$$\frac{dL}{dK_{21}} = \frac{dL}{dz_{11}} * X_{21} + \frac{dL}{dz_{12}} * X_{22} + \frac{dL}{dz_{21}} * X_{31} + \frac{dL}{dz_{22}} * X_{32}$$

$$\frac{dL}{dK_{22}} = \frac{dL}{dz_{11}} * X_{22} + \frac{dL}{dz_{12}} * X_{23} + \frac{dL}{dz_{21}} * X_{32} + \frac{dL}{dz_{22}} * X_{33}$$

$$\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix} \circledast \begin{bmatrix} \frac{dL}{dz_{11}} & \frac{dL}{dz_{12}} \\ \frac{dL}{dz_{21}} & \frac{dL}{dz_{22}} \end{bmatrix} = \begin{bmatrix} \frac{dL}{dK_{11}} & \frac{dL}{dK_{21}} \\ \frac{dL}{dK_{21}} & \frac{dL}{dK_{22}} \end{bmatrix} = \frac{dL}{dK}$$

$$X \circledast \frac{dL}{dz} = \frac{dL}{dK}$$

$$\therefore \frac{dL}{dK} = \text{conv}\left(X, \frac{dL}{dz}\right)$$

2) $$\frac{dL}{dB} = \frac{dL}{dz_{ij}} * \frac{dz_{ij}}{dB}$$

$$\therefore \frac{dL}{dB} = \sum \frac{dL}{dz_{ij}} * \frac{dz_{ij}}{dB}$$

$$= \frac{dL}{dz_{11}} * \frac{dz_{11}}{dB} + \frac{dL}{dz_{12}} * \frac{dz_{12}}{dB} + \frac{dL}{dz_{21}} * \frac{dz_{21}}{dB} + \frac{dL}{dz_{22}} * \frac{dz_{22}}{dB}$$

$$\therefore \frac{dL}{dB} = \frac{dL}{dz_{11}} + \frac{dL}{dz_{12}} + \frac{dL}{dz_{21}} + \frac{dL}{dz_{22}}$$

$$\therefore \frac{dL}{dB} = \text{sum}\left(\frac{dL}{dz}\right)$$

3) $$\frac{dL}{dX} = \frac{dL}{dz} * \frac{dz}{dX} = \sum \frac{dL}{dz_{ij}} * \frac{dz_{ij}}{dX_{mn}}$$

$$\frac{dL}{dX_{11}} = \frac{dL}{dz_{11}} * \frac{dz_{11}}{dX_{11}} = \frac{dL}{dz_{11}} * K_{11}$$

$$\frac{dL}{dX_{12}} = \frac{dL}{dz_{11}} * \frac{dz_{11}}{dX_{12}} + \frac{dL}{dz_{12}} * \frac{dz_{12}}{dX_{12}}$$

$$= \left(\frac{dL}{dz_{11}} * K_{12}\right) + \left(\frac{dL}{dz_{12}} * (K_{11})\right)$$

$\left[\frac{dz_{12}}{dX_{11}}, \frac{dz_{21}}{dX_{11}}, \frac{dz_{22}}{dX_{11}} = 0\right.$ as any change in X_{11} does not effect z_{12}, z_{21}, z_{22}]

Therefore,

$$\frac{dL}{dx_{11}} = \frac{dL}{dz_{11}} * K_{11}$$

$$\frac{dL}{dx_{12}} = \frac{dL}{dz_{11}} * K_{12} + \frac{dL}{dz_{12}} * K_{11}$$

$$\frac{dL}{dx_{13}} = \frac{dL}{dz_{12}} * K_{12}$$

$$\frac{dL}{dx_{21}} = \frac{dL}{dz_{11}} * K_{21} + \frac{dL}{dz_{21}} * K_{11}$$

$$\frac{dL}{dx_{22}} = \frac{dL}{dz_{11}} * K_{22} + \frac{dL}{dz_{12}} * K_{21} + \frac{dL}{dz_{21}} * K_{12} + \frac{dL}{dz_{22}} * K_{11}$$

$$\frac{dL}{dx_{23}} = \frac{dL}{dz_{12}} * K_{22} + \frac{dL}{dz_{22}} * K_{12}$$

$$\frac{dL}{dx_{31}} = \frac{dL}{dz_{21}} * K_{21}$$

$$\frac{dL}{dx_{32}} = \frac{dL}{dz_{21}} * K_{22} + \frac{dL}{dz_{22}} * K_{21}$$

$$\frac{dL}{dx_{33}} = \frac{dL}{dz_{22}} * K_{22}$$

By simplifying,

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \frac{dL}{dz_{11}} & \frac{dL}{dz_{12}} & 0 \\ 0 & \frac{dL}{dz_{21}} & \frac{dL}{dz_{22}} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \circledast \begin{bmatrix} K_{22} & K_{21} \\ K_{12} & K_{11} \end{bmatrix} = \begin{bmatrix} \frac{dL}{dx_{11}} & \frac{dL}{dx_{12}} & \frac{dL}{dx_{13}} \\ \frac{dL}{dx_{21}} & \frac{dL}{dx_{22}} & \frac{dL}{dx_{23}} \\ \frac{dL}{dx_{31}} & \frac{dL}{dx_{32}} & \frac{dL}{dx_{33}} \end{bmatrix}$$

(180° inverted 'K' matrix)

$$\therefore \frac{dL}{dx} = \text{conv}\left(\text{padded}\left(\frac{dL}{dz}\right), \text{180° rotated filter } K\right)$$

Therefore,

$$\frac{dL}{dK} = \text{conv}\left(X, \frac{dL}{dz}\right)$$

$$\frac{dL}{dB} = \text{sum}\left(\frac{dL}{dz}\right)$$

$$\frac{dL}{dx} = \text{conv}\left(\text{padded}\left(\frac{dL}{dz}\right), \text{180° rotated filter } K\right)$$

Forward pass

Let's assume, Input image $I = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 \end{bmatrix}$ $(1, 1, 5, 5)$

And 3×3 filter 'F' is:

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

o/p feature map after convolution $= I \circledast F = \begin{bmatrix} -6 & -6 & -6 \\ -6 & -6 & -6 \\ -6 & -6 & -6 \end{bmatrix}$ $(1, 1, 3, 3)$ (O)

Backward Propagation

Finding '$\frac{dL}{dF}$ or $\frac{dL}{dK}$':

Let 'δ' be the gradient from the next layer (same shape as 'O')

$$\delta = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\frac{dL}{dF} = \sum (\delta \times \text{input patch})$$

for example, the top-left value in the gradient for F(0,0) is $\sum_{\text{all positions}} (I_{\text{patch}} \times \delta_{\text{corresponding position}})$ this is done for all filter positions.

⟹ so, for a 3×3 filter & 5×5 input image, we will extract 3×3 = 9 patches, and each patch will be a flattened vector of size 9 (since 3×3 = 9).

$$\therefore \text{img2col} = \begin{bmatrix} 1 & 2 & 3 & 6 & 7 & 8 & 11 & 12 & 13 \\ 2 & 3 & 4 & 7 & 8 & 9 & 12 & 13 & 14 \\ \vdots & & & & & & & & \vdots \\ 13 & 14 & 15 & 18 & 19 & 20 & 23 & 24 & 25 \end{bmatrix}_{9 \times 9}$$

Reshape the gradient matrix (G) or (δ)

$$G = [\,\cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\,]_{1\times 9}$$

Gradient is calculated as

$$\frac{dL}{dF} = \underset{(9\times 9)}{im2col(I)} \times \underset{(9\times 1)}{G^T}$$

$$\therefore \frac{dL}{dF} = (im2col(I) \times G^T).\,reshape(3,3)$$

↳ reshape to filter size.

Algorithm for faster computation of gradient of convolutional filter:

(i) Convert the i/p image to im2col format; im2col matrix is a 2D-matrix where each column is a flattened vector of elements covered in a single stride of filter.

(ii) Reshape the error matrix 'G' to a 2D-matrix; each row is a flattened vector of the error in each channel).

(iii) Multiply these two matrices and reshape the results.

Adam Optimizer: Adam (Adaptive Moment Estimation) is an optimization algorithm that

i) Uses Momentum (Exponential moving average of gradients) to smooth updates

ii) Uses velocity (Exponential moving average of squared Gradients) to adjust the learning rate.

The update rule is:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\, g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\, g_t^2$$

$$\theta = \theta - \frac{\alpha \cdot m_t}{\sqrt{v_t} + \epsilon}$$

where, $m_t \rightarrow$ Momentum (1st moment estimate)
$v_t \rightarrow$ velocity (2nd moment estimate)
$g_t \rightarrow$ gradient of weight
$\beta_1 \rightarrow$ Controls momentum decay (e.g., 0.9)
$\beta_2 \rightarrow$ Controls velocity decay (e.g., 0.995)
$\alpha \rightarrow$ learning rate (e.g., 0.001)
$\epsilon \rightarrow$ small number to avoid division by zero (e.g., 1e-7)

Example

$\Rightarrow$ Consider a small neural network with 2 weights (w_1, w_2), 2 biases (B_0, B_1), 1 convolutional filter (conv1) & let the gradients be dw_1, dw_2, dB_0, dB_1 and conv1_delta.

→ Initial weights $w_1 = 0.5$, $w_2 = 0.3$, $B_0 = 0.2$, $B_1 = -0.1$, conv1 = 0.8

→ Gradients $dw_1 = 0.02$, $dw_2 = 0.04$, $dB_0 = 0.01$, $dB_1 = -0.03$, conv1_delta = 0.05

→ Hyperparameters beta1 = 0.9, beta2 = 0.995, learning_rate = 0.01.

→ Calculating First Moment:

$$\text{momentum\_}w_1 = \text{beta1} * \text{momentum\_}w_1 + (1 - \text{beta1}) * dw_1$$
$$= 0.9 * 0 + (1 - 0.9) * 0.02 = 0.002$$

similarly, momentum_$w_2 = -0.004$, momentum_$b_0 = 0.001$,
momentum_$b_1 = -0.003$, momentum_conv1 = 0.005

→ Calculating second Moment:

$$\text{velocity\_}w_1 = \text{beta2} * \text{velocity\_}w_1 + (1 - \text{beta2}) * (dw_1)^2$$
$$= 0.995 * 0 + (1 - 0.995) * (0.02)^2 = 0.000002$$

similarly, velocity_$w_2 = 0.000008$, velocity_$b_0 = 0.0000005$,
velocity_$b_1 = 0.995 * 0 + (1 - 0.995) * (-0.03)^2 = 0.0000045$,
velocity_conv1 = 0.0000125.

→ Finally update weights using Adam formula:

$$w_1 = w_1 - \text{learning\_rate} * \left(\frac{\text{momentum\_}w_1}{\sqrt{\text{velocity\_}w_1} + \epsilon} \right)$$