$$\frac{\partial l}{\partial k_{nm}} = \sum_i^{\text{size\_out}} \sum_j \frac{\partial l}{\partial c_{ij}} \cdot \frac{\partial c_{ij}}{\partial k_{nm}} = \sum_i^{\text{size\_out}} \sum_j \delta(i,j) \cdot \frac{\partial c_{ij}}{\partial k_{nm}}$$

$$c_{ij}^{l} = \sum_n^{N_{l+1}} \sum_m^{M_{l+1}} a^{l+1}(i+n, j+m)\, k(n,m)$$

$$= \sum_i^{\text{size\_out}} \sum_j \delta(i,j) \frac{\partial \sum_{n'} \sum_{m'} a^{l+1}(i+n', j+m')\, k(n',m')}{\partial k_{nm}}$$

$k_{nm} = k(-n,-m)$

$$= \sum_i^{\text{size\_out}} \sum_j \underbrace{\delta(i,j)}_{(3,3)} \cdot \underbrace{a^{l+1}(i+n, j+m)}_{(1,1)}$$

(only when $n'=n$, $m'=m$)

$(I-N+1, J-M+1)$

$(5,5)$

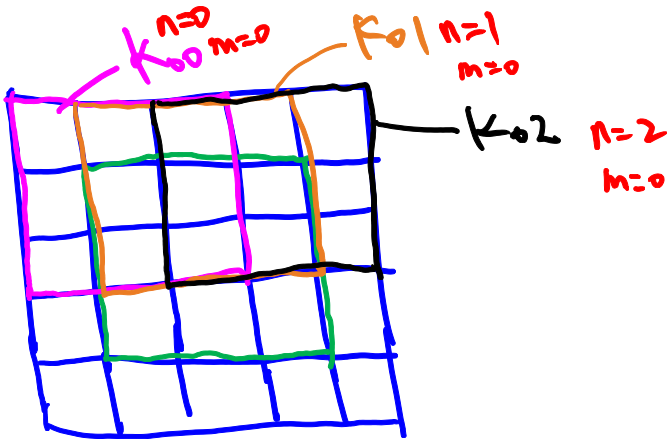
interpretation: sliding $N \times M$ times to generate update coefficients

$I \times J$ is the conv output size

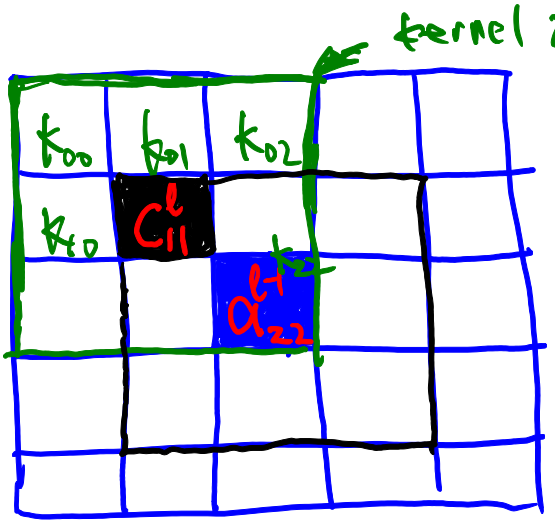
the size of $\delta(i,j)$ is less than that of $a^{l+1}(i',j')$ which is the input.

size_in $= I \times J$

size_out $= (I-N+1) \times (J-M+1)$

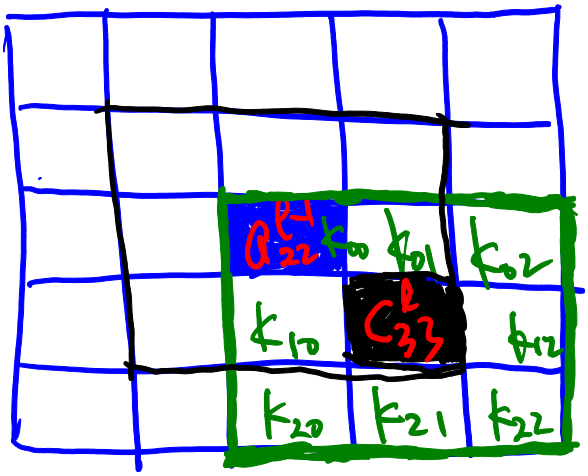


K(-n, -m) is the default kernel layout we store in memory!



$$\frac{\partial l}{\partial a_{22}^{l-1}} = k_{22} * \frac{\partial l}{\partial c_{11}^{l}}$$

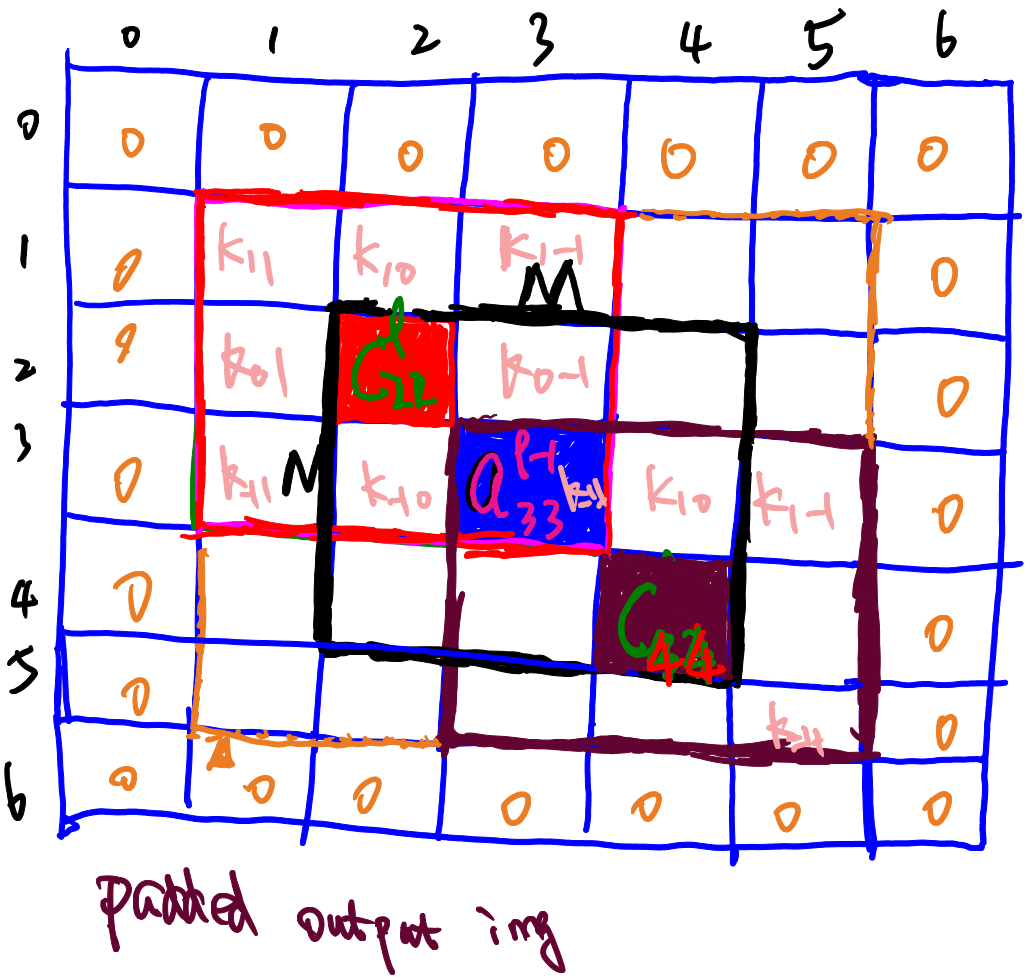
$$\frac{\partial l}{\partial a_{22}^{l-1}} = k_{21} * \frac{\partial l}{\partial c_{12}^{l}}$$

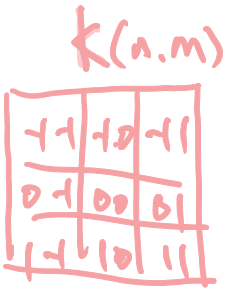


$$\frac{\partial l}{\partial a_{22}^{l-1}} = k_{00} * \frac{\partial l}{\partial c_{33}^{l}}$$

$$\frac{\partial l}{\partial a_{22}^{l-1}} = k_{22} * \frac{\partial l}{\partial c_{11}^{l}} + k_{21} * \frac{\partial l}{\partial c_{12}^{l}} + \dots + k_{00} * \frac{\partial l}{\partial c_{33}^{l}}$$

$$= k_{22} * \delta^{l}(1,1) + k_{21} * \delta^{l}(1,2) + \dots + k_{00} * \delta^{l}(3,3)$$


0 1 2 3 4 5 6
0 1 2 3 4 5 6
k_{11} k_{10} k_{1-1}
k_{01} k_{0-1}
k_{-11} k_{-10} k_{1-1}
C_{22}
a_{33}
C_{44}
padded output img

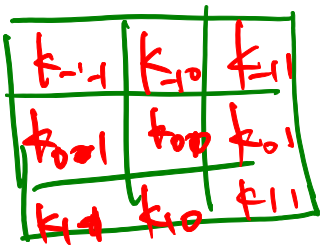

k(n.m)

$$\frac{\partial \ell}{\partial a_{ij}^{\ell-1}} = \sum_{n=-1}^{1} \sum_{m=-1}^{1} \delta^{\ell}(i+n, j+m)\, k(1-n, 1-m)$$

$$= \sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}} \sum_{m=-\frac{M-1}{2}}^{\frac{M-1}{2}} \delta^{\ell}(i+n, j+m)\, k\left(\frac{N-1}{2}-n, \frac{M-1}{2}-m\right)$$

move center of kernel to middle. yield:

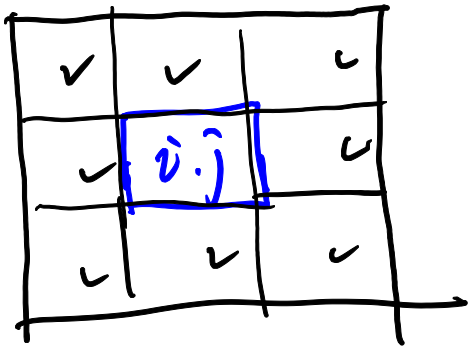
$$= \sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}} \sum_{m=-\frac{M-1}{2}}^{\frac{M-1}{2}} \delta^{\ell}(i+n, j+m)\, k(-n, -m)$$



$$C_{ij}^{\ell} = \sum_{n=-\frac{N-1}{2}}^{\frac{N-1}{2}} \sum_{m=-\frac{M-1}{2}}^{\frac{M-1}{2}} a^{\ell}(i+n, j+m)\, k(-n, -m)$$

$$\frac{\partial \ell}{\partial a_{ij}^{l-1}} = \sum_{i'} \sum_{j'} \frac{\partial \ell}{\partial C_{i'j'}^{l}} \cdot \frac{\partial C_{i'j'}^{l}}{\partial a_{ij}^{l-1}}$$

a_{ij} is only contributing to output pixels in a $M \times N$ kernel box with it center pixel being (i,j), which is same as a_{ij}

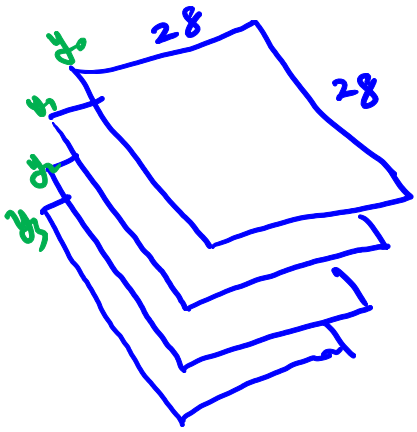


$$= \sum_{i'=i-\frac{M-1}{2}}^{i'=i+\frac{M-1}{2}} \sum_{j'=j-\frac{N-1}{2}}^{j'=j+\frac{N-1}{2}} \delta(i',j') \frac{\partial C_{i'j'}^{l}}{\partial a_{ij}^{l-1}}$$

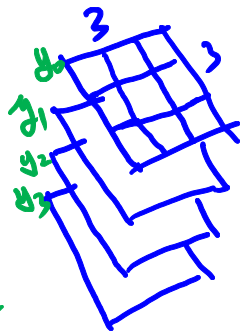
Check ydo DOC notes for further rigorous derivation.

$a^{l-1}_{ij} = 4 \times 28 \times 28$

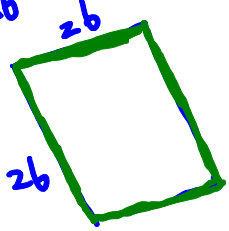
$k^{l}_{(n,m)} = 4 \times 3 \times 3$



$\otimes$

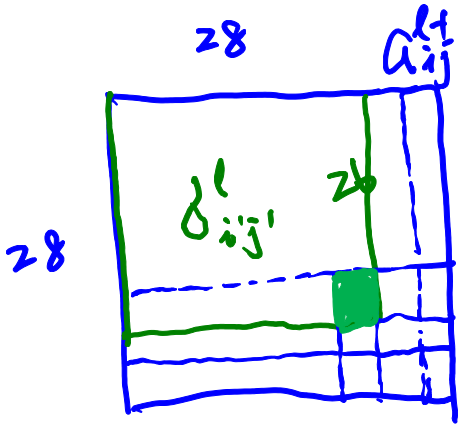


$C^{l}_{y\,i'j'} : 26 \times 26$

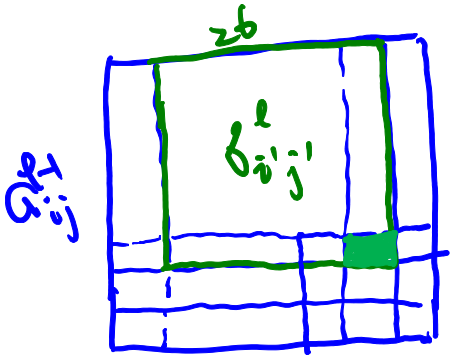


$$\frac{\partial l}{\partial k_{ynn}} = \sum_{i'=0}^{25} \sum_{j'=0}^{25} a^{l-1}_{y}(i'+n, j'+m)\, \delta^{l}_{(i',j')}$$

y is the channel index in input channel or filter channel



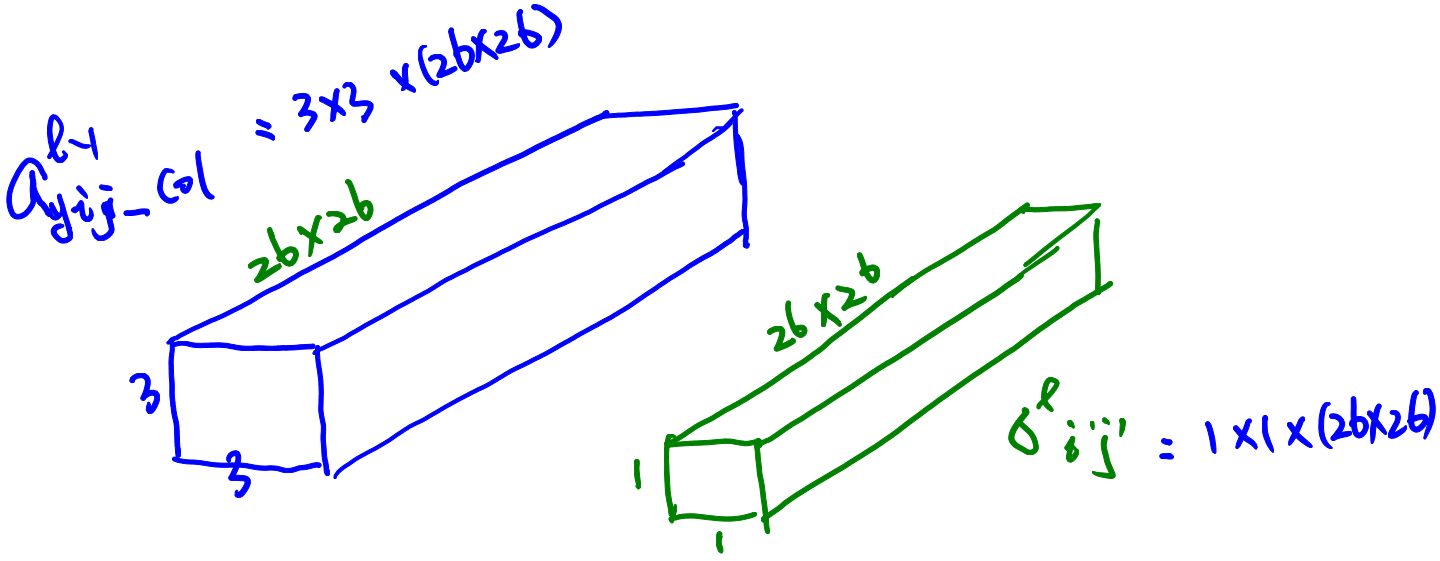
$= k(0,0)$



$= k(0,1)$

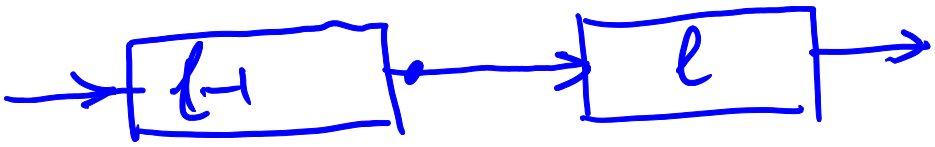
$\vdots$

im2Col for $\frac{\partial l}{\partial k_{ynm}}$

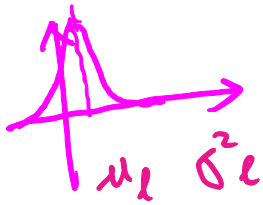


This is for each channel of filter, if filter size is N x 3 x 3. then we have N such Col volumn.

On batch normalization



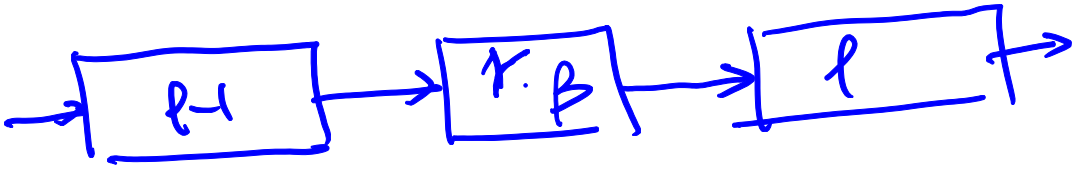
provided from previous layer



input characteristic required

So there is a mismatch between given and requirement which will degrade the system performance

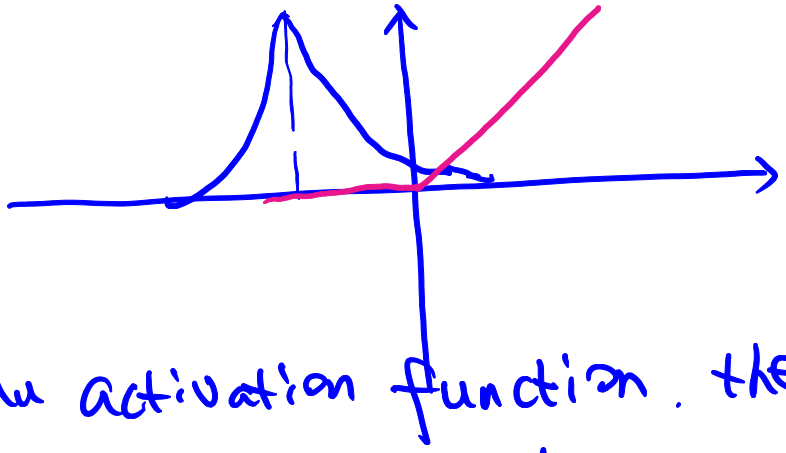
A solution to this is Adding scale & shifting factor to it.



distribution transformation.

The second benifit from batch normalization is we can speed up training!

The reason for this is that. we can avoid output distribution like belove



For relu activation function. the output will be very sparse (too many zero layer with neurons carries less feature information, this will tend to cause all zero output in the layers that follows!

In training without BN we often set the learing rate to be small to avoid such sparse situation.

With BN. we tend to move the centroid of the data distribution around zero, which avoids the sparse condition in some degree. so that we can make the learning rate a little bigger.

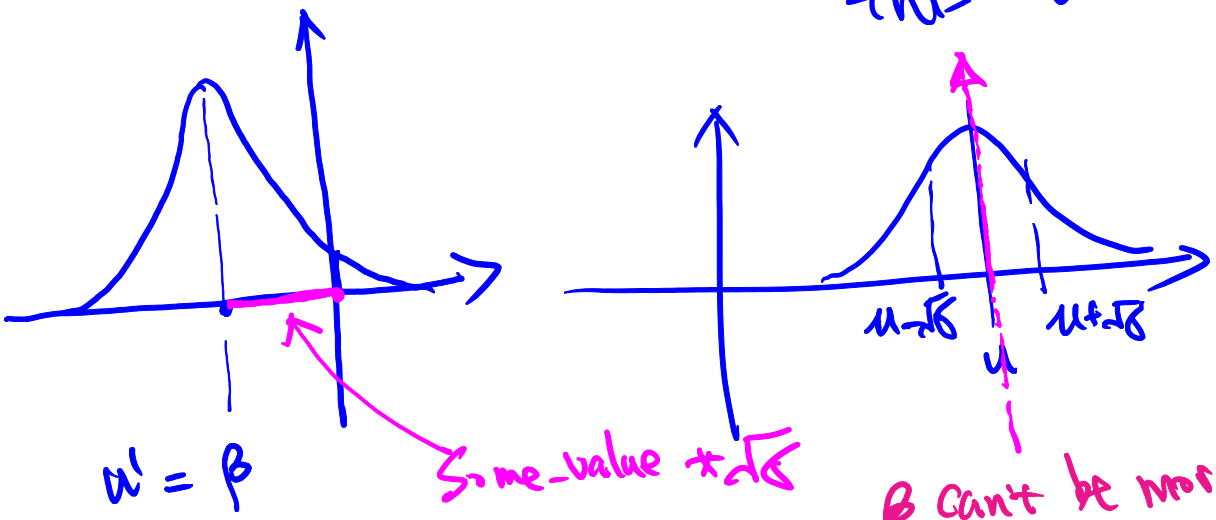
A suggestion to make the learing rate even bigger

Adding constrain to β. since we know the negative β tend to move the distribution to the lefte. we can constrain it so it does not move too much to the left.

since we know the current output distribution we can limit it to

$$\beta + \underbrace{\text{some\_value}} * \sqrt{\sigma} > 0$$

try to find out this value



so in update process:

$$\beta = \beta + d\beta$$

if $\beta <$ - some_value $* \sqrt{\sigma}$

$\beta =$ some_value $* \sqrt{\sigma}$