nodeflux

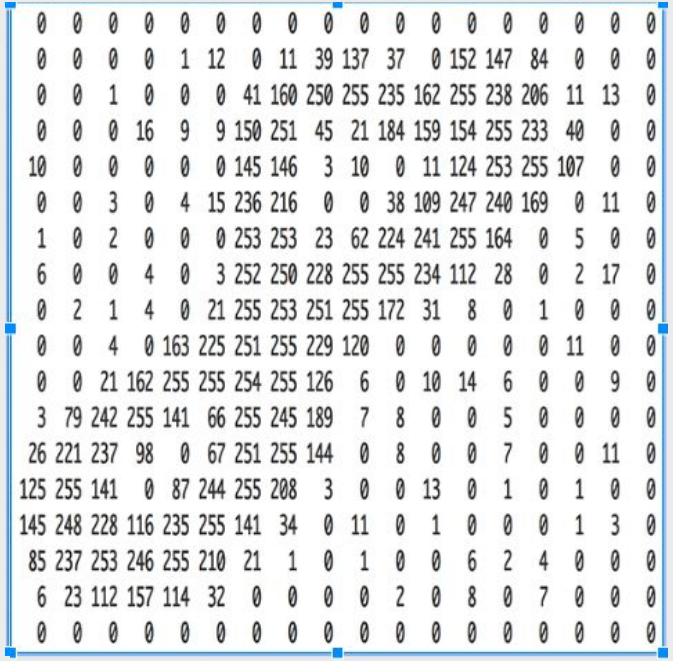
Extending Vision Beyond Imagination

Convolutional Neural Network In-depth

Outline

Introduction	**Intro to ConvNet** How computers perceive image, overview of artificial neural network. Why we need ConvNet and intuition behind it
Forward Prop	**Forward propagation** Layers of CNN, what we compute in each layer and how (the intuition)
Back Prop	**Back propagation** How we train the network
Applications	**ConvNet Applications** Where else we use ConvNet, why is it still impactful

How computers perceive images ?



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	12	0	11	39	137	37	0	152	147	84	0	0	0
0	0	1	0	0	0	41	160	250	255	235	162	255	238	206	11	13	0
0	0	0	16	9	9	150	251	45	21	184	159	154	255	233	40	0	0
10	0	0	0	0	0	145	146	3	10	0	11	124	253	255	107	0	0
0	0	3	0	4	15	236	216	0	0	38	109	247	240	169	0	11	0
1	0	2	0	0	0	253	253	23	62	224	241	255	164	0	5	0	0
6	0	0	4	0	3	252	250	228	255	255	234	112	28	0	2	17	0
0	2	1	4	0	21	255	253	251	255	172	31	8	0	1	0	0	0
0	0	4	0	163	225	251	255	229	120	0	0	0	0	0	11	0	0
0	0	21	162	255	255	254	255	126	6	0	10	14	6	0	0	9	0
3	79	242	255	141	66	255	245	189	7	8	0	0	5	0	0	0	0
26	221	237	98	0	67	251	255	144	0	8	0	0	7	0	0	11	0
125	255	141	0	87	244	255	208	3	0	0	13	0	1	0	1	0	0
145	248	228	116	235	255	141	34	0	11	0	1	0	0	0	1	3	0
85	237	253	246	255	210	21	1	0	1	0	0	6	2	4	0	0	0
6	23	112	157	114	32	0	0	0	0	2	0	8	0	7	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Image by: Adam Geitgey via Medium

Image as a giant grid of pixels

When we as a human could easily tell that's a number, which is "8"

That's really just a bunch of number for computers ...

Black-White images have 1 matrix, while Colour images have 3, each for the colour Red, Green, and Blue

What we have learned

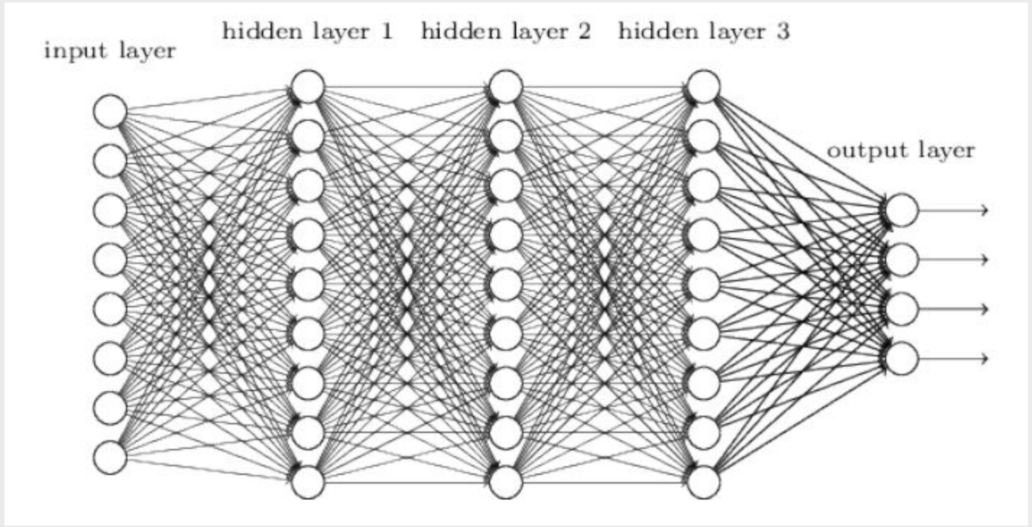


Image by: Michael Nielsen via *Neural Network and Deep Learning*

ANN unroll each channel, WxH image, into a vector.

They **ignore the spatial structure** of the input images,

which makes the **network incapable of recognising object at the different part of the image.**

Weight or **number of parameters** may be enormous

CNN Key Features

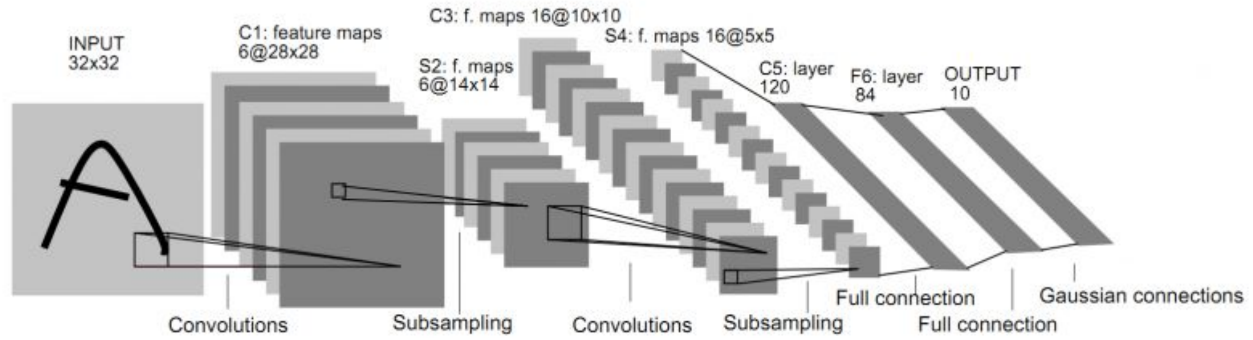


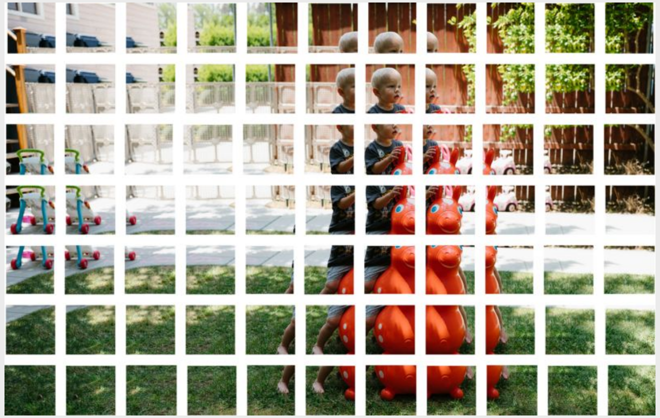
Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

- "We need to give our neural network understanding of ***translation invariance***—an "8" is an "8" no matter where in the picture it shows up."
- "***Local receptive fields,*** in which hidden units are connected to local patches of the layer below"
- ***Weight sharing***
- ***Pooling***, which condenses information from previous layer

CNN - Intuition

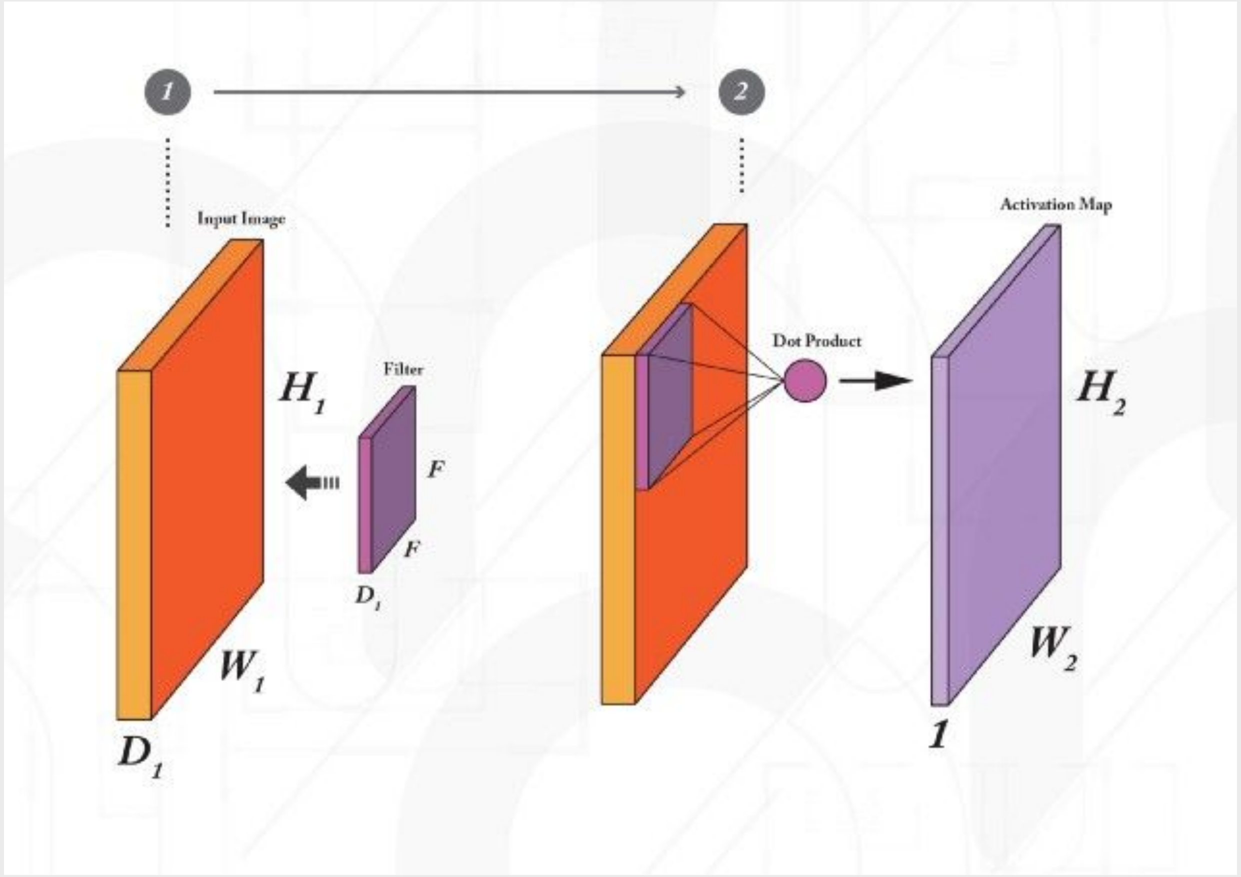
Image via: PyImage Search

Image by: Adam Geitgey via Medium

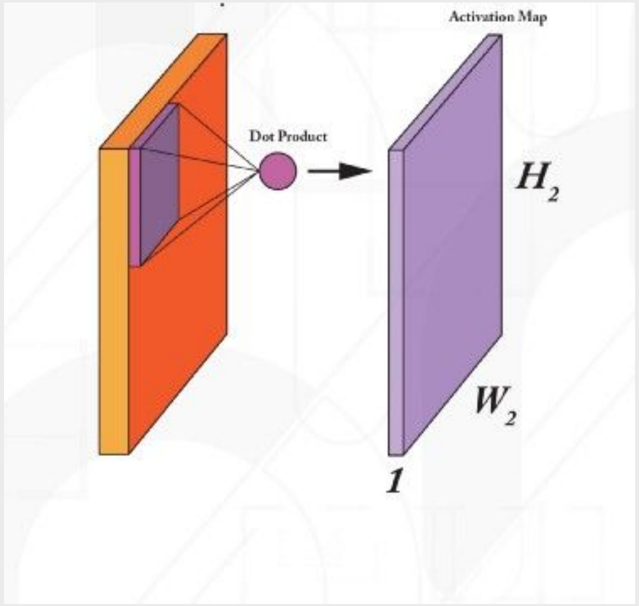


Forward Prop

Convolutional Layer - Convolution Process



Convolutional Layer - Activation map



Say we have 32 x 32 x 3 image with 5 x 5 x 3 filter.

Dot product result of the filter with a specific part of the image (5 x 5 x 3 = 75 dimensional dot product).

Resulting in 28 x 28 x 1 sheet of neuron outputs called ***activation map***.

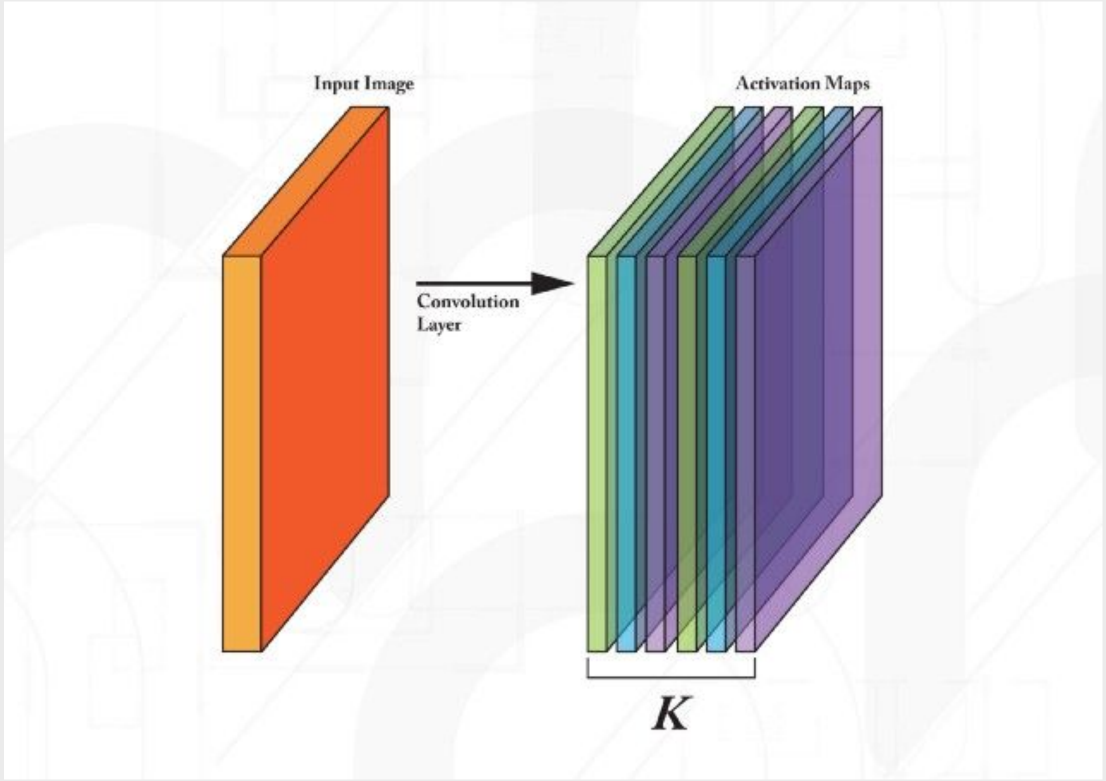
- Each is connected to small region in the input
- All of them share parameters.

As we slide the filter through with these weights, we use **the same weights** throughout because it's the same filter.

"All neurons have same weights ***w*** but looking at slightly different part of the image"

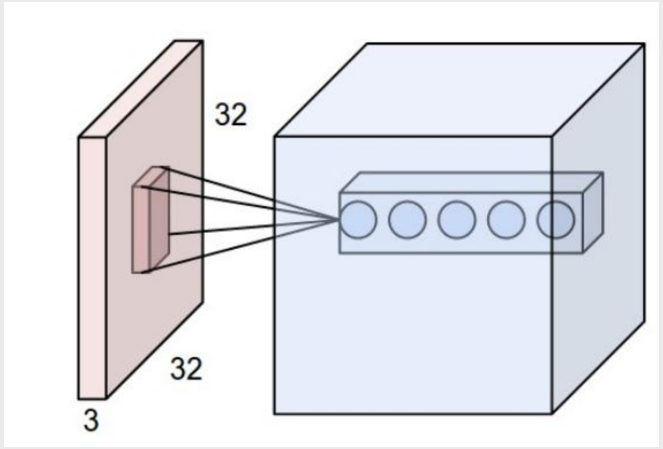
In ConvNet, Filters are trained, not initialise to something specific like edge filter.

Convolutional Layer



If we had 6 different 5x5 filters, we’ll get 6 separate activation maps.

Convolutional Layer



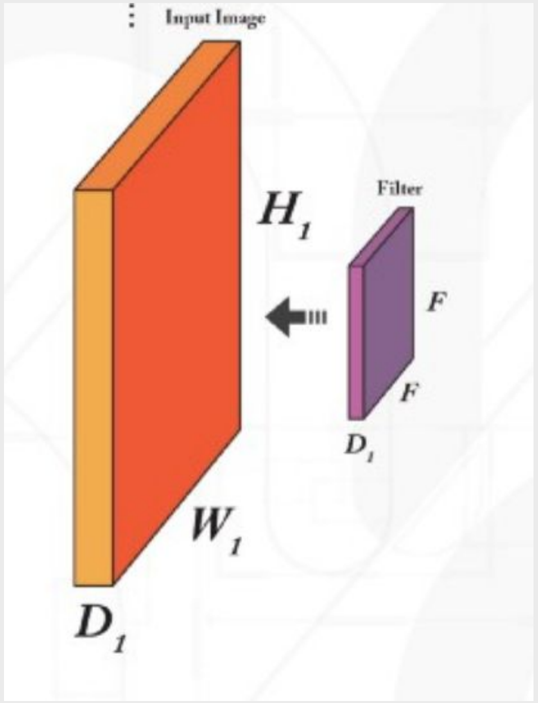
with 5 filters, 5 different neurons **looking at the same region** in the input volume.

Across space: sharing weight (parameter sharing)

Across depth: sharing input patch (local connectivity) ➜ not fully global, way too many parameters

Image by: Andrej Karpathy via *http://cs231n.github.io/convolutional-networks/*

Convolutional Layer - Output size



Since we are dealing with **square** images (non rectangular image process is still unknown), H = W, or let's call it N.

The output size of a Convolutional Layer can then be decide by a formula, as follows:

$$[\ (N + 2P - F) / S\ ] + 1$$

where P is the zero-padding size and S is the stride.

For example,

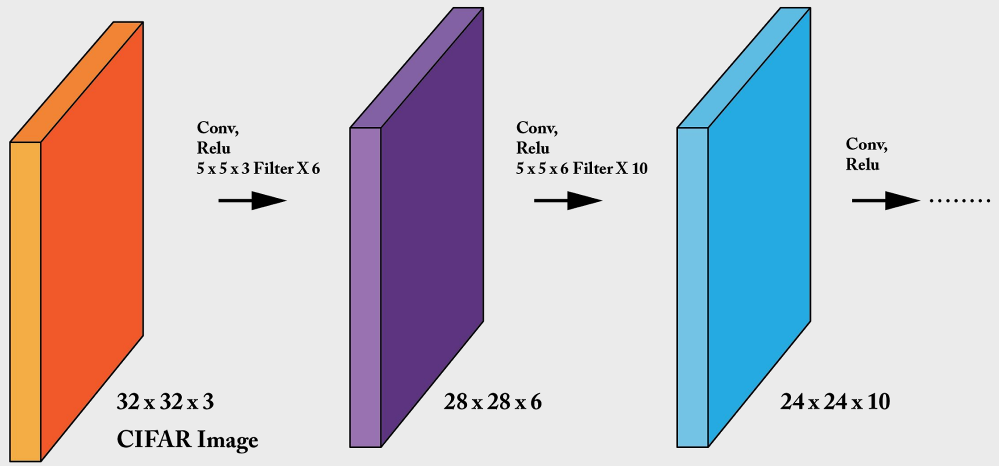
Stride 1 → (7 - 3) / 1 +1 = 5

Stride 2 → (7 - 3) / 2 +1 = 3

Stride 3 → (7 - 3) / 3 +1 = 2.33 (x) not possible

Stride 3 → (7 +2 - 3) / 3 +1 = 3 with pad it is possible

Convolutional Net



ConvNet is a sequence of ***convolutional layer***, interspersed with activation function.

Typical architecture of ConvNet:

conv → relu → conv → relu → pool → conv → relu → conv → relu → pool → conv → relu → conv → relu → pool

There are no specific rule of how many convolutional layer do we need, or where is the best place to put a pooling. To produce “the best” features representation, how many filters are we supposed to used? Which pooling method, which activation function is the best to use? *It’s all fully experimental ! I mean hey, that’s what makes machine learning is fun, innit? :)*

Pooling Layer

Convolutional Layer won't shrink spatial size, preserved.

Pooling Layer ➔ squeeze input spatially by downsampling operation. Operates over each activation map independently.

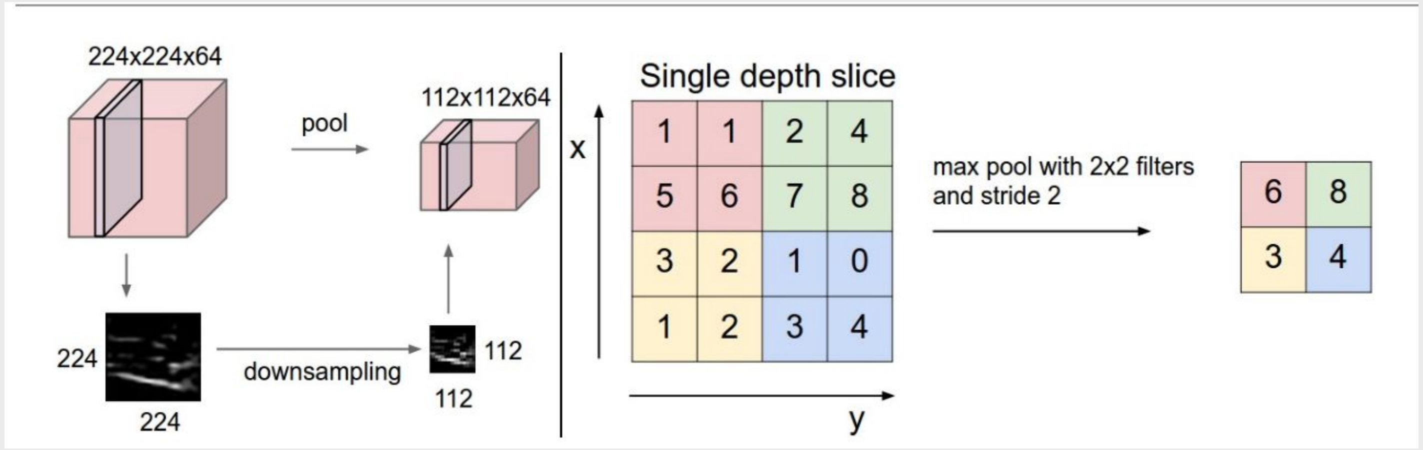


Image by: Andrej Karpathy via *http://cs231n.github.io/convolutional-networks/*

Pooling Layer

Pooling operation only changes the spatial size and not the depth, so:

$W1 \times H1 \times D1 \rightarrow W2 \times H2 \times D2$
$W2 = (W1 - F) / S + 1 = H2$
$D2 = D1$

where F is the pooling filter size.

The most popular pooling method is the max pooling, although there are several other method such as *average pooling* or *L2-norm pooling.*

"**Getting rid of pooling**. Many people dislike the pooling operation and think that we can get away without it. For example, Striving for Simplicity: The All Convolutional Net proposes to discard the pooling layer in favor of architecture that only consists of repeated CONV layers. To reduce the size of the representation they suggest using larger stride in CONV layer once in a while. Discarding pooling layers has also been found to be important in training good generative models, such as variational autoencoders (VAEs) or generative adversarial networks (GANs). It seems likely that future architectures will feature very few to no pooling layers."

Summary

- Accepts volume of size:
  $W1 \times H1 \times D1 \rightarrow W2 \times H2 \times D2$
  $W2 = (W1 - F + 2P) / S + 1 = H2$
  $D2 = K$

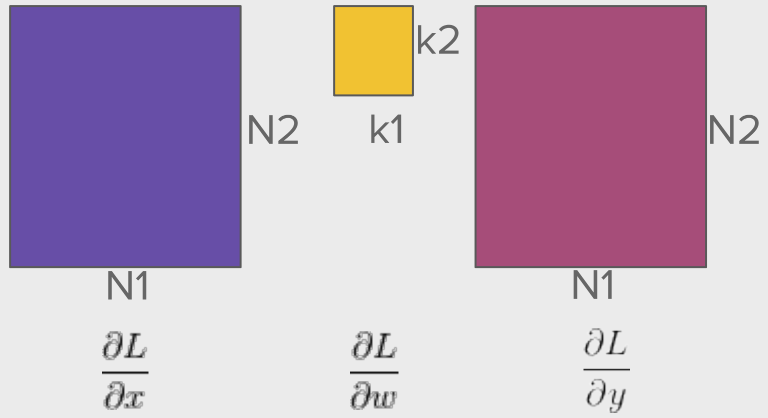
K : number of filters
F : filter spatial extent
S : stride
P : padding zero

- Introduce F x F x D1 weights per filter so there are F x F x D1 x K weights + K biases (number of parameters to compute)
- Common settings:
  - K powers of 2, computational reason. Some libraries have some special subroutine which are very efficient whenever they see power of 2 in terms of kernel/dimension.
  - F is usually odd, 1,3,5,7,11. For convenience purposes, giving a nice representation as a filter (left and right balance). Filter size 1 is make sense because of the depth.
  - P is zero, probably in order to make the pad not contributed to the filter.
  - Input always squares for images, non rectangular image process still unknown.
  - Different F in a same layer is not common, for computational convenience
  - No pad → shrink input (spatial) → bad idea!
  - Only the 1st conv layer has access to original image. 2nd has access to 1st output and so on.

Back Prop

nodeflux
Extending Vision Beyond Imagination

Back Propagation of Error in Convolutional Layer



N2
k2
k1
N2
N1
N1
$\frac{\partial L}{\partial x}$
$\frac{\partial L}{\partial w}$
$\frac{\partial L}{\partial y}$

L is the ***loss function***. It is common for classification tasks to use ***cross-entropy error function*** as its loss function.

$$E^n = -(t^n \ln y^n + (1 - t^n) \ln(1 - y^n)) \ .$$

We already have the gradient for loss w.r.t the output, now we need to compute the gradient w.r.t the weights and the input

Gradient w.r.t Weights

The forward prop (output) can be described as:

$$y[r,c] = \sum_{a=0}^{k_1} \sum_{b=0}^{k_2} x(r+a, c+b) w(a,b) \quad \text{... (1)}$$

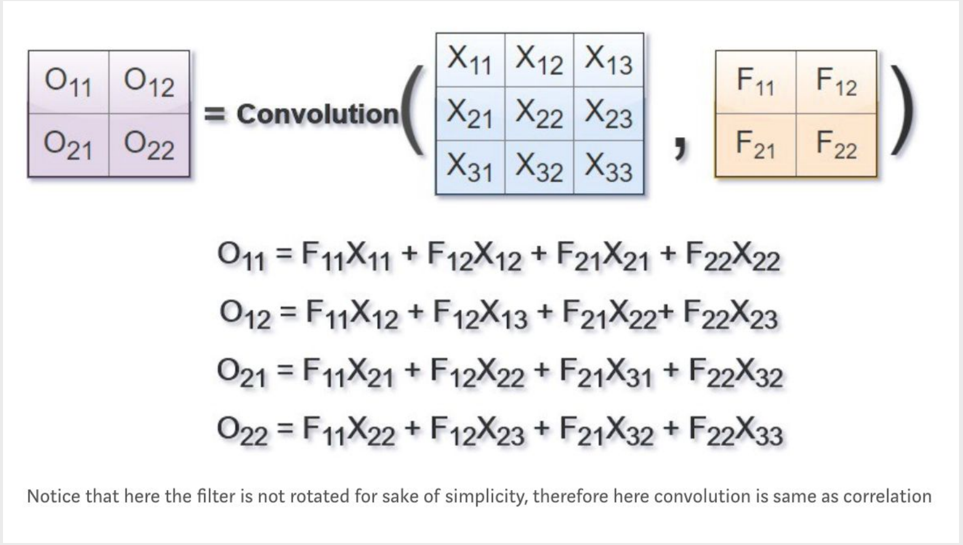
Since the weights are shared to all spatial space of the image, we need to compute each of the gradient w.r.t each part/component of the weight, as in:

$$\frac{\partial L}{\partial w[a,b]} = \sum_{r=0}^{N_1} \sum_{c=0}^{N_2} \frac{\partial L}{\partial y[r,c]} \frac{\partial y[r,c]}{\partial w[a,b]} \quad \text{... (2)}$$

We already know the first term (gradient w.r.t output), and the second term can be derived from Eq. (1) into:

$$\frac{\partial L}{\partial w[a,b]} = \sum_{r=0}^{N_1} \sum_{c=0}^{N_2} \frac{\partial L}{\partial y[r,c]} x(r+a, c+b)$$

Forward prop



$$\begin{bmatrix} O_{11} & O_{12} \\ O_{21} & O_{22} \end{bmatrix} = \textbf{Convolution}\left(\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix}, \begin{bmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{bmatrix} \right)$$

$$O_{11} = F_{11}X_{11} + F_{12}X_{12} + F_{21}X_{21} + F_{22}X_{22}$$

$$O_{12} = F_{11}X_{12} + F_{12}X_{13} + F_{21}X_{22} + F_{22}X_{23}$$

$$O_{21} = F_{11}X_{21} + F_{12}X_{22} + F_{21}X_{31} + F_{22}X_{32}$$

$$O_{22} = F_{11}X_{22} + F_{12}X_{23} + F_{21}X_{32} + F_{22}X_{33}$$

Notice that here the filter is not rotated for sake of simplicity, therefore here convolution is same as correlation

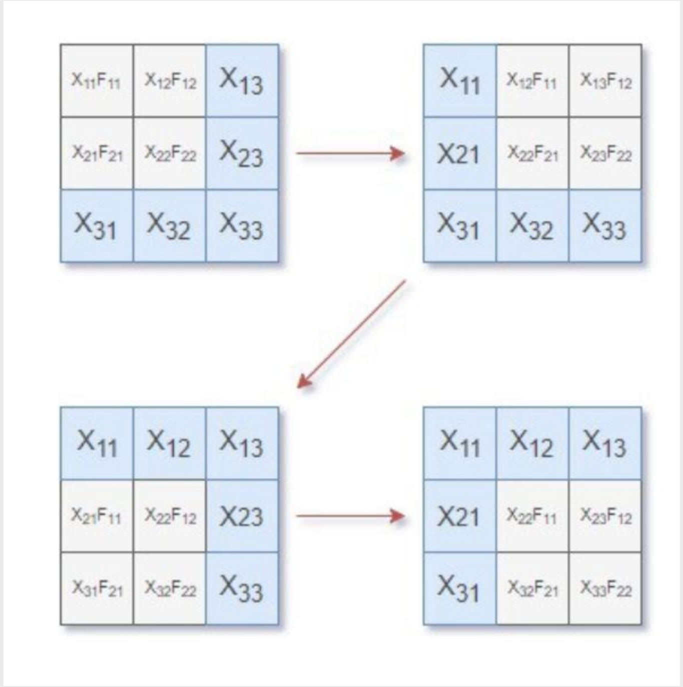


Image by: Sujit Rai via *https://medium.com/@2017csm1006/forward-and-backpropagation-in-convolutional-neural-network-4dfa96d7b37e*

Gradient w.r.t Weights

$$\frac{\partial E}{\partial F_{11}} = \frac{\partial E}{\partial O_{11}}\frac{\partial O_{11}}{\partial F_{11}} + \frac{\partial E}{\partial O_{12}}\frac{\partial O_{12}}{\partial F_{11}} + \frac{\partial E}{\partial O_{21}}\frac{\partial O_{21}}{\partial F_{11}} + \frac{\partial E}{\partial O_{22}}\frac{\partial O_{22}}{\partial F_{11}}$$

$$\frac{\partial E}{\partial F_{12}} = \frac{\partial E}{\partial O_{11}}\frac{\partial O_{11}}{\partial F_{12}} + \frac{\partial E}{\partial O_{12}}\frac{\partial O_{12}}{\partial F_{12}} + \frac{\partial E}{\partial O_{21}}\frac{\partial O_{21}}{\partial F_{12}} + \frac{\partial E}{\partial O_{22}}\frac{\partial O_{22}}{\partial F_{12}}$$

$$\frac{\partial E}{\partial F_{21}} = \frac{\partial E}{\partial O_{11}}\frac{\partial O_{11}}{\partial F_{21}} + \frac{\partial E}{\partial O_{12}}\frac{\partial O_{12}}{\partial F_{21}} + \frac{\partial E}{\partial O_{21}}\frac{\partial O_{21}}{\partial F_{21}} + \frac{\partial E}{\partial O_{22}}\frac{\partial O_{22}}{\partial F_{21}}$$

$$\frac{\partial E}{\partial F_{22}} = \frac{\partial E}{\partial O_{11}}\frac{\partial O_{11}}{\partial F_{22}} + \frac{\partial E}{\partial O_{12}}\frac{\partial O_{12}}{\partial F_{22}} + \frac{\partial E}{\partial O_{21}}\frac{\partial O_{21}}{\partial F_{22}} + \frac{\partial E}{\partial O_{22}}\frac{\partial O_{22}}{\partial F_{22}}$$

$\Rightarrow$

$$\frac{\partial E}{\partial F_{11}} = \frac{\partial E}{\partial O_{11}}X_{11} + \frac{\partial E}{\partial O_{12}}X_{12} + \frac{\partial E}{\partial O_{21}}X_{21} + \frac{\partial E}{\partial O_{22}}X_{22}$$

$$\frac{\partial E}{\partial F_{12}} = \frac{\partial E}{\partial O_{11}}X_{12} + \frac{\partial E}{\partial O_{12}}X_{13} + \frac{\partial E}{\partial O_{21}}X_{22} + \frac{\partial E}{\partial O_{22}}X_{23}$$

$$\frac{\partial E}{\partial F_{21}} = \frac{\partial E}{\partial O_{11}}X_{21} + \frac{\partial E}{\partial O_{12}}X_{22} + \frac{\partial E}{\partial O_{21}}X_{31} + \frac{\partial E}{\partial O_{22}}X_{32}$$

$$\frac{\partial E}{\partial F_{22}} = \frac{\partial E}{\partial O_{11}}X_{22} + \frac{\partial E}{\partial O_{12}}X_{23} + \frac{\partial E}{\partial O_{21}}X_{32} + \frac{\partial E}{\partial O_{22}}X_{33}$$

$$\begin{bmatrix} \partial E/\partial F_{11} & \partial E/\partial F_{12} \\ \partial E/\partial F_{21} & \partial E/\partial F_{22} \end{bmatrix} = \textbf{Convolution}\left(\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix}, \begin{bmatrix} \partial E/\partial O_{11} & \partial E/\partial O_{12} \\ \partial E/\partial O_{21} & \partial E/\partial O_{22} \end{bmatrix} \right)$$

Image by: Sujit Rai via *https://medium.com/@2017csm1006/forward-and-backpropagation-in-convolutional-neural-network-4dfa96d7b37e*

Gradient w.r.t Input

$$\frac{\partial E}{\partial X_{11}} = \frac{\partial E}{\partial O_{11}}F_{11} + \frac{\partial E}{\partial O_{12}}0 + \frac{\partial E}{\partial O_{21}}0 + \frac{\partial E}{\partial O_{22}}0$$

$$\frac{\partial E}{\partial X_{12}} = \frac{\partial E}{\partial O_{11}}F_{12} + \frac{\partial E}{\partial O_{12}}F_{11} + \frac{\partial E}{\partial O_{21}}0 + \frac{\partial E}{\partial O_{22}}0$$

$$\frac{\partial E}{\partial X_{13}} = \frac{\partial E}{\partial O_{11}}0 + \frac{\partial E}{\partial O_{12}}F_{12} + \frac{\partial E}{\partial O_{21}}0 + \frac{\partial E}{\partial O_{22}}0$$

$$\frac{\partial E}{\partial X_{21}} = \frac{\partial E}{\partial O_{11}}F_{21} + \frac{\partial E}{\partial O_{12}}0 + \frac{\partial E}{\partial O_{21}}F_{11} + \frac{\partial E}{\partial O_{22}}0$$

$$\frac{\partial E}{\partial X_{22}} = \frac{\partial E}{\partial O_{11}}F_{22} + \frac{\partial E}{\partial O_{12}}F_{21} + \frac{\partial E}{\partial O_{21}}f_{12} + \frac{\partial E}{\partial O_{22}}F_{11}$$

$$\frac{\partial E}{\partial X_{23}} = \frac{\partial E}{\partial O_{11}}0 + \frac{\partial E}{\partial O_{12}}F_{22} + \frac{\partial E}{\partial O_{21}}0 + \frac{\partial E}{\partial O_{22}}F_{11}$$

$$\frac{\partial E}{\partial X_{31}} = \frac{\partial E}{\partial O_{11}}0 + \frac{\partial E}{\partial O_{12}}0 + \frac{\partial E}{\partial O_{21}}F_{21} + \frac{\partial E}{\partial O_{22}}0$$

$$\frac{\partial E}{\partial X_{32}} = \frac{\partial E}{\partial O_{11}}0 + \frac{\partial E}{\partial O_{12}}0 + \frac{\partial E}{\partial O_{21}}F_{22} + \frac{\partial E}{\partial O_{22}}F_{21}$$

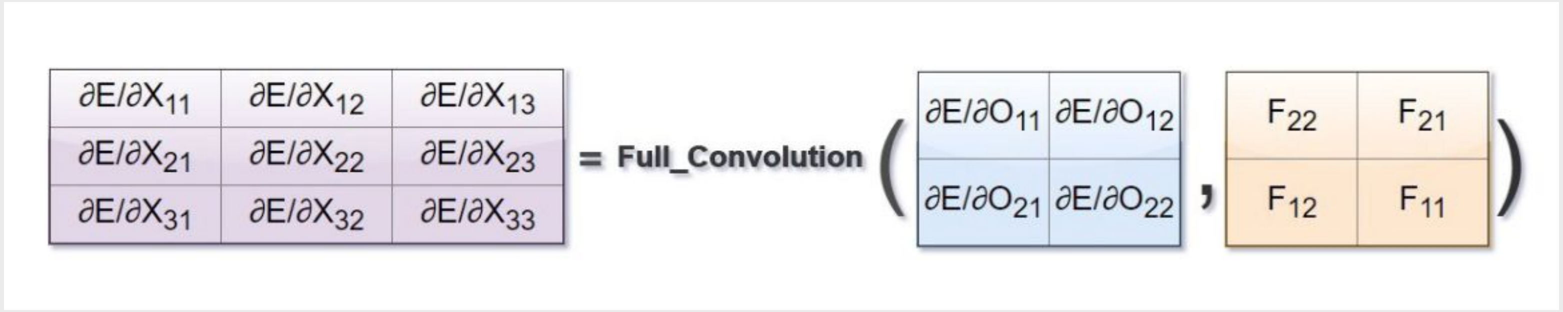
$$\frac{\partial E}{\partial X_{33}} = \frac{\partial E}{\partial O_{11}}0 + \frac{\partial E}{\partial O_{12}}0 + \frac{\partial E}{\partial O_{21}}0 + \frac{\partial E}{\partial O_{22}}F_{22}$$

Similarly, we can do it for the gradient w.r.t the input.
But as you can see on the left, there are some several ‘zeros’ on the equation.

That is because unlike the weight which each point contribute to all of the output, each point of the input contribute to only some point of the output (as the filter convolve around).

Image by: Sujit Rai via *https://medium.com/@2017csm1006/forward-and-backpropagation-in-convolutional-neural-network-4dfa96d7b37e*

Gradient w.r.t Input



$$\begin{bmatrix} \partial E/\partial X_{11} & \partial E/\partial X_{12} & \partial E/\partial X_{13} \\ \partial E/\partial X_{21} & \partial E/\partial X_{22} & \partial E/\partial X_{23} \\ \partial E/\partial X_{31} & \partial E/\partial X_{32} & \partial E/\partial X_{33} \end{bmatrix} = \text{Full\_Convolution}\left(\begin{bmatrix} \partial E/\partial O_{11} & \partial E/\partial O_{12} \\ \partial E/\partial O_{21} & \partial E/\partial O_{22} \end{bmatrix}, \begin{bmatrix} F_{22} & F_{21} \\ F_{12} & F_{11} \end{bmatrix} \right)$$

It also can be seen as a convolution process.

“In order to obtain the gradients of the input matrix we need to rotate the filter by 180 degree and calculate the full convolution of the rotated filter by the gradients of the output with respect to error”

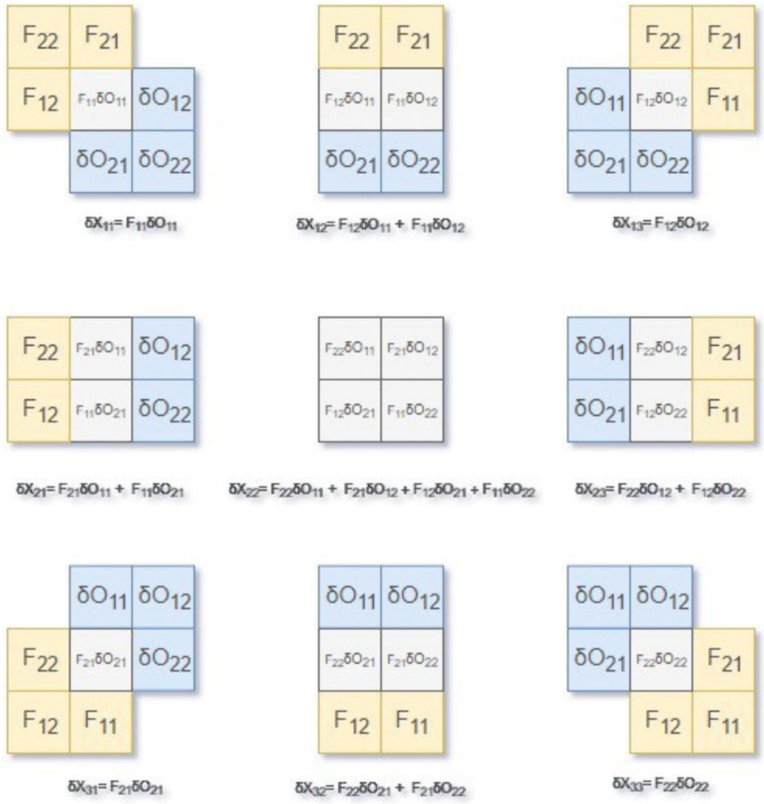
Image by: Sujit Rai via *https://medium.com/@2017csm1006/forward-and-backpropagation-in-convolutional-neural-network-4dfa96d7b37e*

Gradient w.r.t Input

Just like the forward convolution, with the rotated filter, and some padding.

This is due to each point of the input only contribute to some point of the output. In the forward propagation, each hidden unit is connected to a
region of input units (the receptive field)

For example,
The top-left input unit (1,1) is connected to just one hidden unit
Input unit (2,2) is in the receptive fields of 2 ×2 = 4 hidden units
(3,3) is in the receptive fields of 3 × 3 = 9 hidden units



$\delta X_{11} = F_{11}\delta O_{11}$
$\delta X_{12} = F_{12}\delta O_{11} + F_{11}\delta O_{12}$
$\delta X_{13} = F_{12}\delta O_{12}$
$\delta X_{21} = F_{21}\delta O_{11} + F_{11}\delta O_{21}$
$\delta X_{22} = F_{22}\delta O_{11} + F_{21}\delta O_{12} + F_{12}\delta O_{21} + F_{11}\delta O_{22}$
$\delta X_{23} = F_{22}\delta O_{12} + F_{12}\delta O_{22}$
$\delta X_{31} = F_{21}\delta O_{21}$
$\delta X_{32} = F_{22}\delta O_{21} + F_{21}\delta O_{22}$
$\delta X_{33} = F_{22}\delta O_{22}$

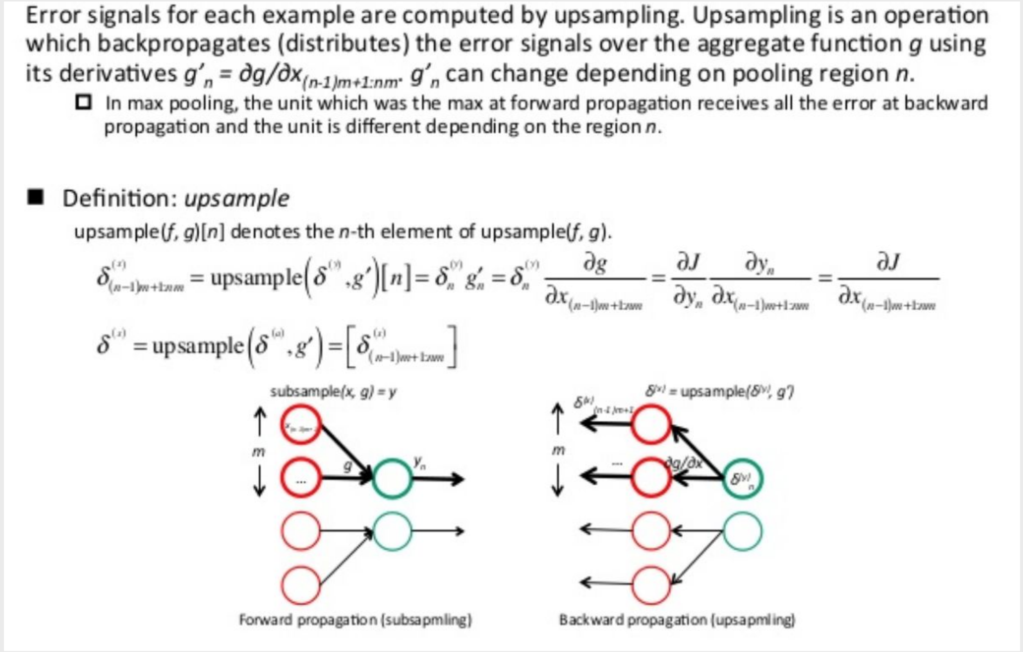
Here 'δX' represents the gradients of error with respect to X

Image by: Sujit Rai via *https://medium.com/@2017csm1006/forward-and-backpropagation-in-convolutional-neural-network-4dfa96d7b37e*

Backpropagation in Pooling Layer

For calculating the gradients of the pooling and Relu layers the gradients can be calculated by following the same procedure of using chain rule of derivatives.

“There is no gradient with respect to non maximum values, since changing them slightly does not affect the output. … Thus, the gradient from the next layer is passed back to only that neuron which achieved the max. All other neurons get zero gradient.”
Reference:
https://datascience.stackexchange.com/questions/11699/backprop-through-max-pooling-layers



Error signals for each example are computed by upsampling. Upsampling is an operation which backpropagates (distributes) the error signals over the aggregate function g using its derivatives $g'_n = \partial g / \partial x_{(n-1)m+1:nm}$. g'_n can change depending on pooling region n.

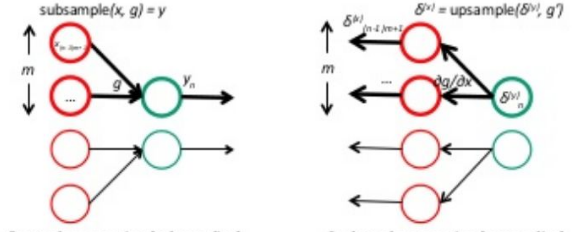
- In max pooling, the unit which was the max at forward propagation receives all the error at backward propagation and the unit is different depending on the region n.

- Definition: *upsample*

upsample(f, g)[n] denotes the n-th element of upsample(f, g).

$$\delta^{(x)}_{(n-1)m+1:nm} = \text{upsample}\left(\delta^{(y)}, g'\right)[n] = \delta^{(y)}_n g'_n = \delta^{(y)}_n \frac{\partial g}{\partial x_{(n-1)m+1:nm}} = \frac{\partial J}{\partial y_n}\frac{\partial y_n}{\partial x_{(n-1)m+1:nm}} = \frac{\partial J}{\partial x_{(n-1)m+1:nm}}$$

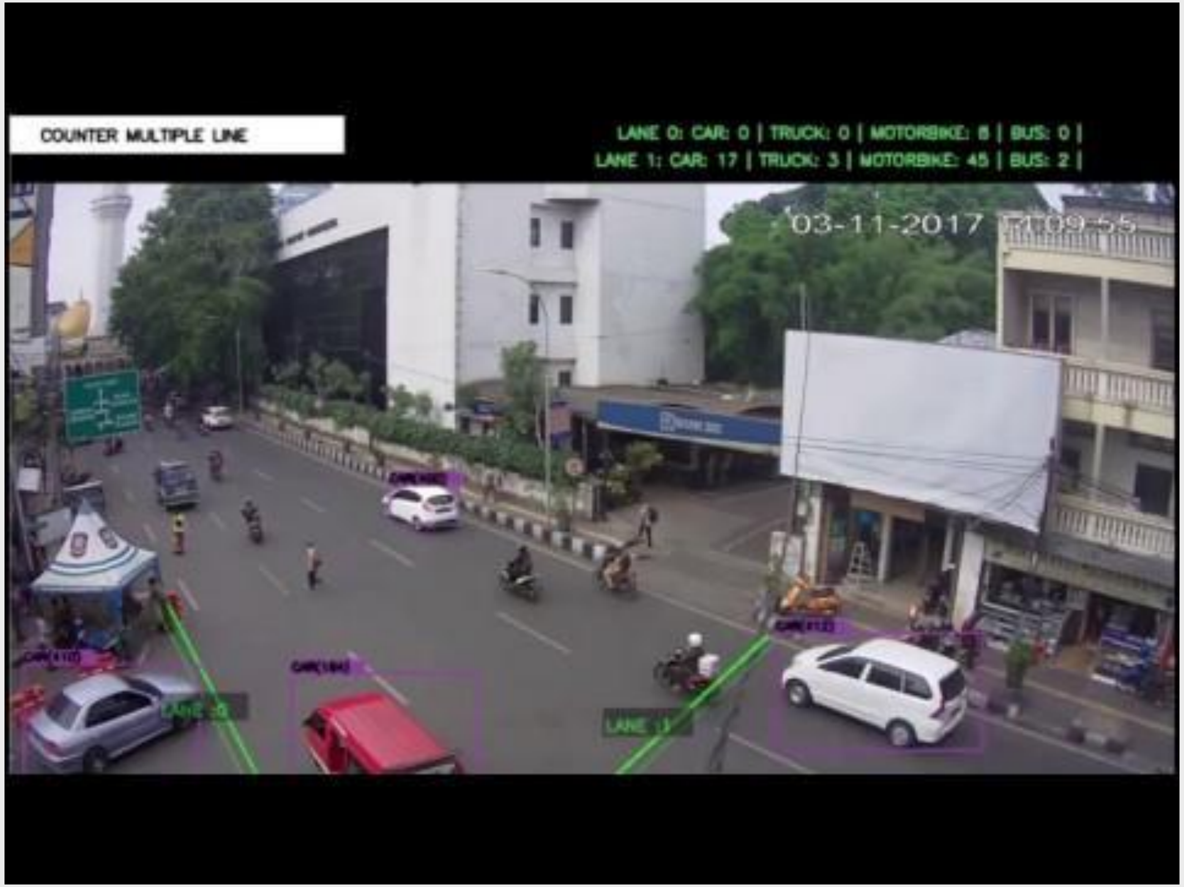
$$\delta^{(x)} = \text{upsample}\left(\delta^{(a)}, g'\right) = \left[\delta^{(x)}_{(n-1)m+1:nm}\right]$$


subsample(x, g) = y
m
g
…
y_n
$\delta^{(x)} = \text{upsample}(\delta^{(y)}, g')$
$\delta^{(x)}_{(n-1)m+1}$
m
…
∂g/∂x
$\delta^{(y)}_n$

Forward propagation (subsapmling) Backward propagation (upsapmling)

Image by: https://www.slideshare.net/kuwajima/cnnbp

Beyond Image Classification ...


COUNTER MULTIPLE LINE
LANE 0: CAR: 0 | TRUCK: 0 | MOTORBIKE: 8 | BUS: 0 |
LANE 1: CAR: 17 | TRUCK: 3 | MOTORBIKE: 45 | BUS: 2 |
03-11-2017
LANE 0
LANE 1

richard@nodeflux.io