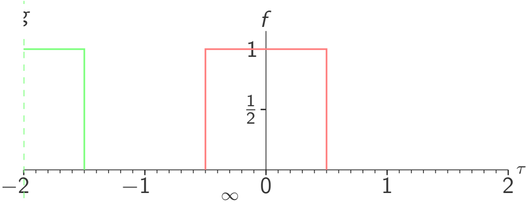
CS-568 Deep Learning

Nazar Khan

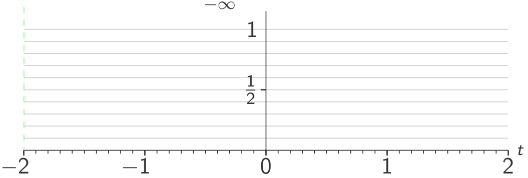
PUCIT

Convolutional Neural Networks

Convolution

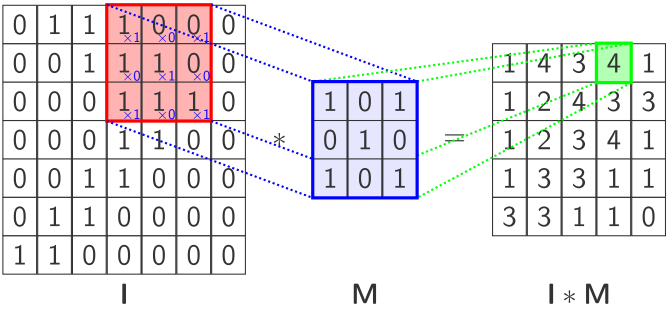


$$f * g = \int_{-\infty}^{\infty} f(\tau) g(t - \tau)\, d\tau$$



Source: `http://www.texample.net/tikz/examples/convolution-of-two-functions/`

2D Convolution
Example



Modified from `https://github.com/PetarV-/TikZ/tree/master/2D%20Convolution`

M is usually called a *mask* or *kernel* or *filter*.

Dealing with boundaries

- What about edge and corner pixels where the mask goes outside the image boundaries?
  - Expand image I with virtual pixels. Options are:
    1. Fill with a particular value, *e.g.* zeros.
    2. Replicating boundaries: fill with nearest pixel value.
    3. Reflecting boundaries: mirror the boundary
  - Fatalism: just ignore them. Not recommended since size of $I * M$ will shrink.

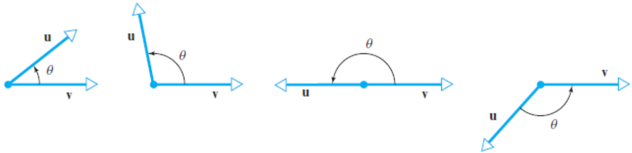
Dealing with boundaries
Expand by zeros

For a 5×5 image and 5×5 mask

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & a & b & c & d & e & 0 & 0 \\ 0 & 0 & f & g & h & i & j & 0 & 0 \\ 0 & 0 & k & l & m & n & o & 0 & 0 \\ 0 & 0 & p & q & r & s & t & 0 & 0 \\ 0 & 0 & u & v & w & x & y & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

A Neuron as a Detector

- A neuron can be viewed as a detector.
- When it fires, the input must have been similar to its weights.
  - Firing $\implies$ $\mathbf{w}^T\mathbf{x}$ was high $\implies$ $\mathbf{w}$ was similar to $\mathbf{x}$
- So neuron firing indicates detection of something similar to its weights.



$$\mathbf{u}^T\mathbf{v} = \|\mathbf{u}\|\|\mathbf{v}\| \cos\theta$$

- Since $-1 \leq \cos\theta \leq 1$, $\mathbf{u}^T\mathbf{v}$ is highest when $\cos\theta = 1$
- That happens when $\theta = 0$
- That happens when vectors $\mathbf{u}$ and $\mathbf{v}$ point in the same direction.

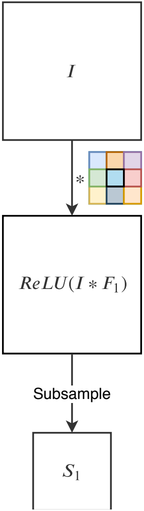
Convolutional Neural Networks

- Now we will look at networks that produce neuronal output via convolution.
- Known as Convolutional Neural Networks (CNNs).
- Most frequently used network architecture.
- Exploits local correlation of inputs.

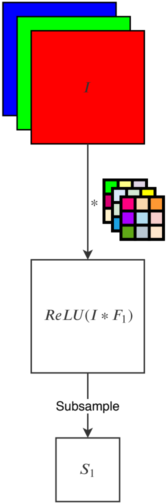
Building Blocks of CNNs

Viewing convolution as neurons

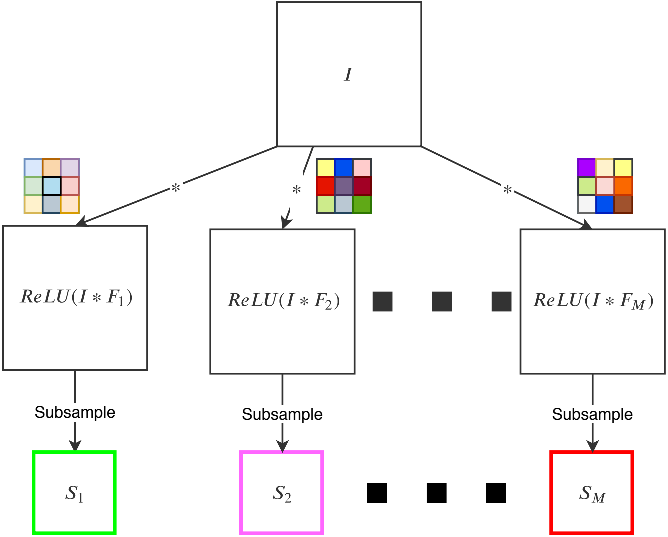
Single channel input



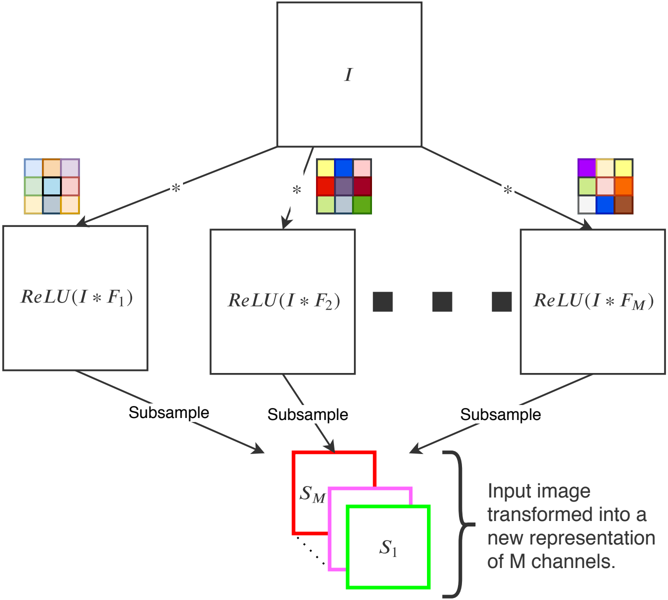
Multichannel input



Building blocks of CNNs

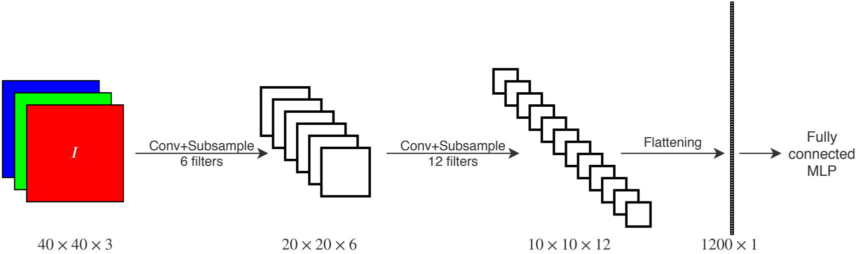


Building blocks of CNNs



CNN

- Convolution by M filters produces M channels.
- They represent an M-channel transformation of the input image I.
- This M-channel image can now be transformed further via additional convolution filters.
- Convolution-subsampling block can be repeated multiple times.
- $I \rightarrow M_1$ channels $\rightarrow M_2$ channels $\rightarrow \cdots \rightarrow M_b$ channels $\rightarrow$ flattening $\rightarrow$ MLP.



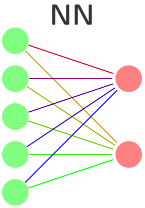
Convolutional Neural Networks

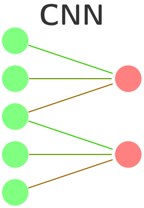
- For recognition of hand-written digits, we have seen that inputs are images and outputs are posterior probabilities $p(C_k|\mathbf{x})$ for $k = 1, \ldots, 10$.
- The digits true identity is invariant under
  - translation, scaling, (small) rotation, and
  - small elastic deformations (multiple writings of the same digit by the same person will have subtle differences).
- The output of the neural network should also be invariant to such changes.
- A traditional fully connected neural network can, in principle, learn these invariances using lots of examples.

Convolutional Neural Networks

- However, it totally ignores the *local correlation* property of images.
  - Nearby pixels are more strongly correlated than pixels that are far apart.
- Modern computer vision exploits local correlation by extracting features from local patches and combines this information to detect higher-order features.
  - Example: Gradients $\longrightarrow$ Edges $\longrightarrow$ Lines $\longrightarrow \ldots$.
- Local features useful in one sub-region can be useful in other sub-regions.
  - Example: same object appearing at different locations.
- This weakness of standard neural nets is overcome by CNNs.

NN vs. CNN





- Global receptive fields due to being fully connected.
- Separate weights for each neuron.

- *Local receptive fields* due to being sparsely connected.
- *Shared weights* among different neurons.
- *Subsampling* of each layer's outputs.

- Receptive field of a neuron consists of previous layer neurons that it is connected to (or looking at).

Convolutional layer

- Consists of multiple arrays of neurons. Each such array is called a *slice* or more accurately *feature map*.
- Each neuron in a feature map
  - is connected to only few neurons in the previous layer, but
  - uses the same weight values as all other neurons in that feature map.
- So within a feature map, we have both
  - local receptive fields, and
  - shared weights.

Convolutional layer

- Example: A feature map may have
  - 100 neurons placed in a 10×10 array, with
  - each neuron getting input from a 5×5 patch of neurons in the previous layer (receptive field), and
  - the same $26 (= 5 \times 5 + 1)$ weights shared between these 100 neurons.
- **Viewed as detectors, all 100 neurons detect the same 5×5 pattern but at different locations of the previous layer.**
- Different feature maps will learn[1] to detect different kinds of patterns.
  - For example, one feature map might learn to detect horizontal edges while others might learn to detect vertical or diagonal edges and so on.

[1] based on their learned weights

Convolutional layer

- To compute activations of the 100 neurons, a dot-product is computed between the same shared weights and different 5×5 patches of previous layer neurons.
- This is equivalent to **sliding a window of weights over the previous layer and computing the dot-product at each location of the window**.
- Therefore, activations of the feature map neurons are computed via *convolution* of the previous layer with a *kernel* comprising the shared weights. Hence the name of this layer.

Subsampling layer

- Reduces the spatial dimensions of the previous layer by downsampling. Also called *pooling* layer.
- Example: downsampling previous layer of $n \times n$ neurons by factor 2 yields a pooled layer of $\frac{n}{2} \times \frac{n}{2}$ neurons.
- No adjustable weights. Just a fixed downsampling procedure.
- Reduces computations in subsequent layers.
- Reduces number of weights in subsequent layers. This reduces overfitting.

Subsampling

- Options: From non-overlapping 2×2 patches
  - pick top-left (standard downsampling by factor 2)
  - pick average (*mean-pooling*)
  - pick maximum (*max-pooling*)
  - pick randomly (*stochastic-pooling*)
- Fractional max-pooling: pick pooling region randomly.

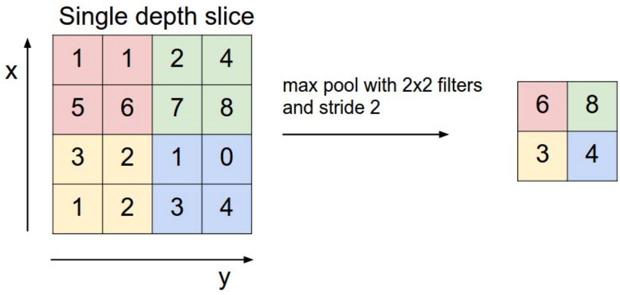


Figure: Max-pooling with 2×2 receptive fields, and stride of 2 neurons. Source: `http://cs231n.github.io/convolutional-networks/`

Subsampling

- The options in the last slide discard 75% of the data.
- They correspond to
  - neurons with 2×2 receptive fields, and
  - *stride* of 2 neurons.
- This is the most commonly used configuration. Other options exist but note that pooling with larger receptive fields discards too much data.
- Subsampling layer can be skipped if convolution layers uses stride>1 since that also produces a subsampled output.

Subsampling

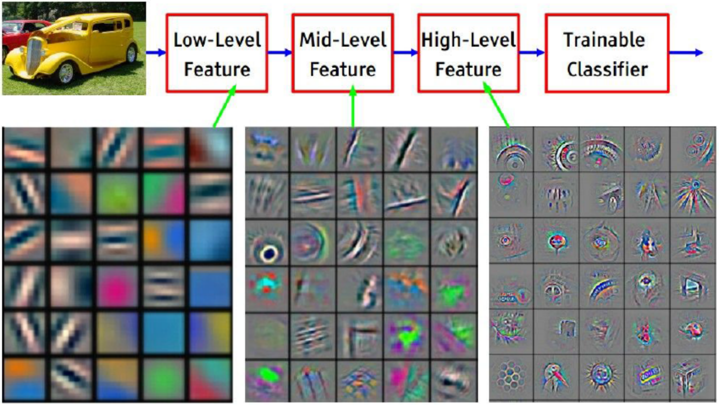
A pooling layer

- with $F \times F$ receptive field and stride S,
- "looking at" a $W_1 \times H_1 \times D_1$ input volume,
- produces a $W_2 \times H_2 \times D_2$ output volume, where
  - $W_2 = \frac{W_1 - F}{S} + 1$
  - $H_2 = \frac{H_1 - F}{S} + 1$
  - $D_2 = D_1$.

Fully Connected Layers

- After flattening, a fully connected MLP can be used.
- The last layer has
  - neurons equal to the desired output size, and
  - activation functions based on the problem to be solved.
- The flattened layer can therefore be viewed as a transformation $\phi(\mathbf{x})$ that is fed into an MLP.
- Similarly, outputs of earlier layers are *intermediate representations* of the input.

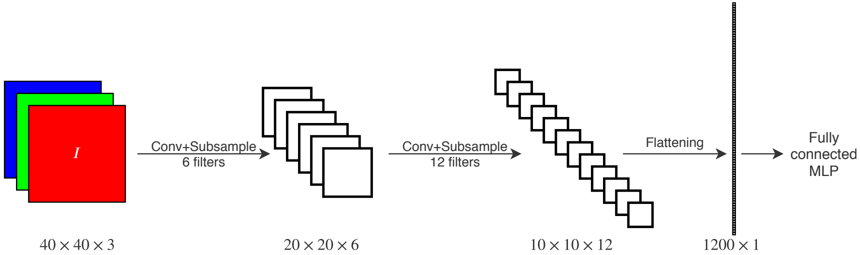
Intermediate Representations



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Intermediate feature representations. Early layers form simple, low-level representations of the input. They are used to incrementally form more complex, high-level representations. Source: `http://cs231n.stanford.edu/slides/winter1516_lecture7.pdf`

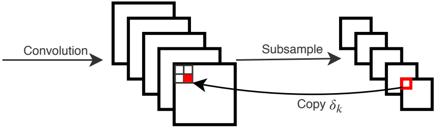
Backpropagation in CNNs



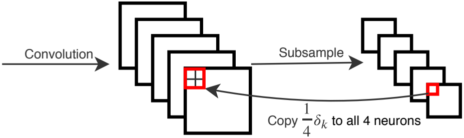
1. Compute $\delta_k = \frac{\partial L}{\partial a_k}$ for each neuron in flattened layer using standard MLP backpropagation.
2. *Directly copy* these δ_ks at corresponding locations of previous subsampling layer.

Backpropagation from subsampling to convolution layer

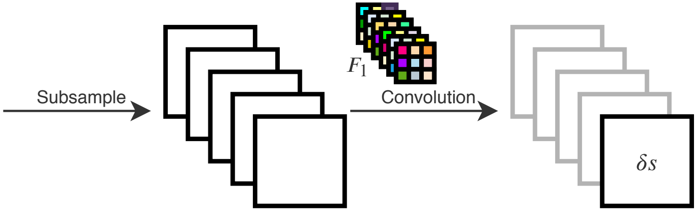
- Record index of pooled neuron during forward pass.
- Backpropagate δ only to this pooled neuron.



- Mean-pooling is different.
  - All neurons are picked with uniform weight in forward pass.
  - So backpropagate δ to each neuron with uniform weight.



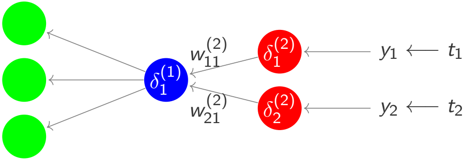
Backpropagation in a convolutional layer



Backpropagation Equation

Recall the backpropagation equation for a traditional neuron.

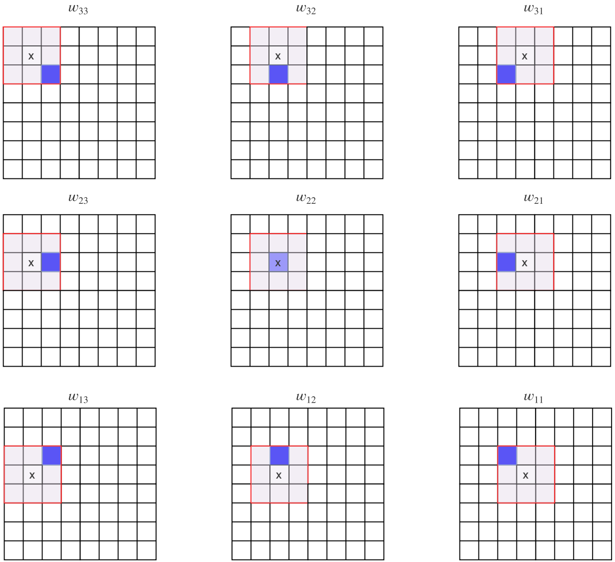
$$\delta_j^{(1)} = h'(a_j) \sum_{k=1}^{K} \delta_k^{(2)} w_{kj}$$



1. Take all neurons affected by neuron j.
2. Compute dot-product between their δ values and connecting weights.
3. Multiply result by derivative of activation function of neuron j.

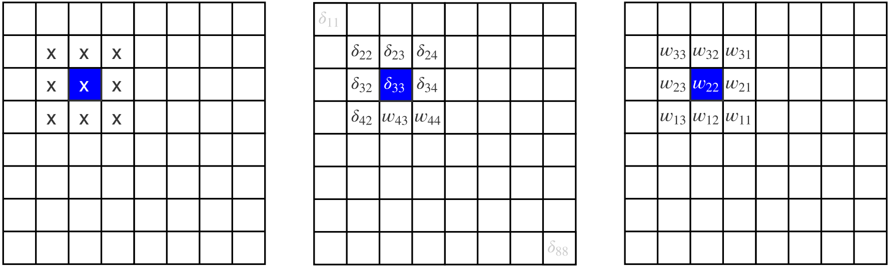
Backpropagation in a convolutional layer

- Now consider a neuron in a convolutional layer.
- In the forward pass, the blue neuron affects all neurons marked by x in the next layer.
- *Notice the flipped role of weights*.



Backpropagation in a convolutional layer

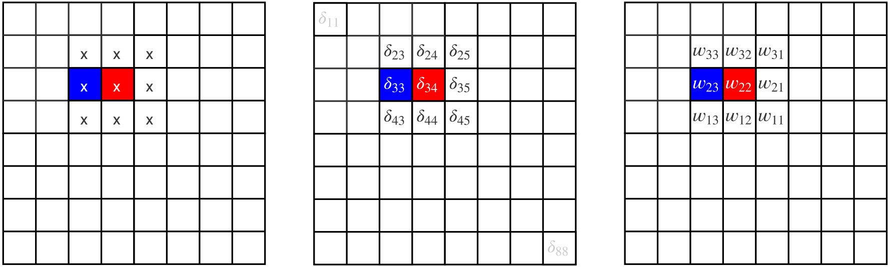
- In the backward pass, the blue neuron computes the dot-product between δ values at the x-locations and connecting weights.



- The connecting weights are a horizontally and vertically flipped version of the weights used in the forward convolution pass.

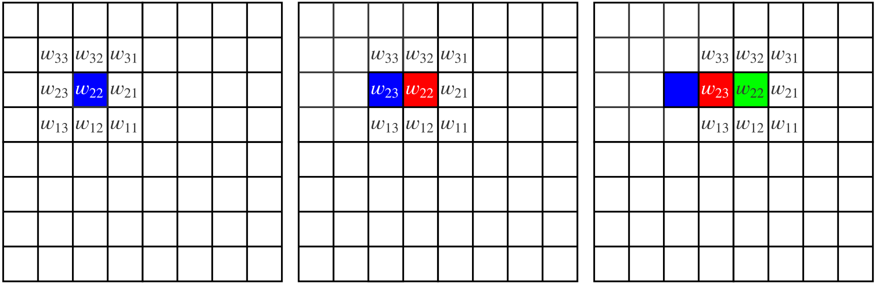
Backpropagation in a convolutional layer

- The adjacent red neuron affects a new but overlapping set of x-locations *using the same connecting weights*.



- *Since the weights are shared*, the only difference is between the x-locations.

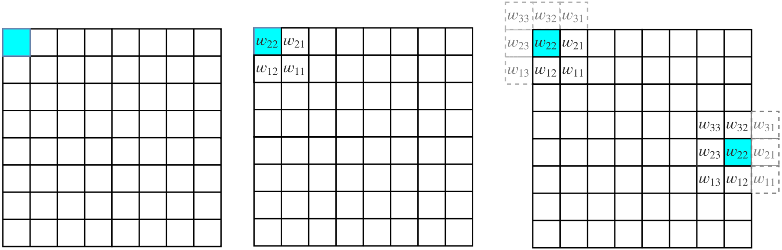
Backpropagation in a convolutional layer



- Equivalent to convolving the δ-map by flipped weights.
- Therefore, backpropagation of δ values from a convolution layer is
  1. *just a convolution of the δ-map using flipped weights*,
  2. followed by multiplication with derivatives of activation functions.

Backpropagation in a convolutional layer

- What about boundary neurons? Who did they affect?



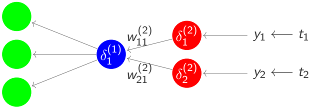
- Equivalent to convolving the δ-map by flipped weights *using zero-padding*.
- Therefore, backpropagation of δ values from a convolution layer is
  1. *just a convolution of the δ-map using flipped weights with zero-padding*,
  2. followed by multiplication with derivatives of activation functions.

Computing gradients in convolutional layer

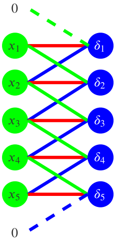
- Consider a *valid* convolution of an $n \times n$ array with another $n \times n$ array.
- What will be the *size* of the result?
- Now consider a *valid* convolution of an $n+1 \times n+1$ array with an $n \times n$ array.
- What will be the *size* of the result?

Computing gradients in convolutional layer
1D case

- Backpropagation computes the per-neuron δ-maps only.
- Per-weight derivatives are computed as the product of a *traditional* neuron's δ value and its input.



- Consider 1D convolutional layer with 3×1 filter.



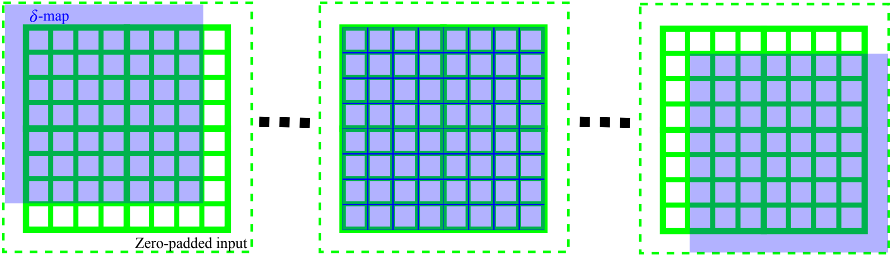
$$\left.\begin{aligned}\frac{\partial L}{\partial w_1} &= \delta_1 0 + \delta_2 x_1 + \delta_3 x_2 + \delta_4 x_3 + \delta_5 x_4 \\ \frac{\partial L}{\partial w_2} &= \delta_1 x_1 + \delta_2 x_2 + \delta_3 x_3 + \delta_4 x_4 + \delta_5 x_5 \\ \frac{\partial L}{\partial w_3} &= \delta_1 x_2 + \delta_2 x_3 + \delta_3 x_4 + \delta_4 x_5 + \delta_5 0\end{aligned}\right\} \Longrightarrow \begin{array}{c}\begin{bmatrix}\delta_1 & \delta_2 & \delta_3 & \delta_4 & \delta_5\end{bmatrix} \\ \star \text{ (valid)} \\ \begin{bmatrix}0 & x_1 & x_2 & x_3 & x_4 & x_5 & 0\end{bmatrix}\end{array}$$

- Verify that $\frac{\partial L}{\partial b} = \sum \delta_i$.

Computing gradients in convolutional layer

2D case

1. Zero-pad the input array with $\lfloor \frac{K}{2} \rfloor$ zeros on each side[2].
2. Perform *valid* convolution of the *zero-padded input array* by the δ-map of the next layer to obtain a $K \times K$ array of derivatives of the convolution weights.



3. Derivative of bias is just the sum of the δ-map.

[2] Assuming square $K \times K$ convolution filter where K is odd

CNN Variations

- There are *lots* of variations.
  - Fully convolutional networks. No pooling and no fully connected layer.
  - 1×1 convolutions to reduce computations.
  - Inception modules to combine multiple filter sizes.
  - Residual blocks to avoid vanishing gradients.
  - Depthwise separable convolutions to reduce parameters and computations.
  - Lightweight and fast models (SqueezeNet, MobileNet, ...) for edge computing.
  - Fast search over hyperparameters (EfficientNet).