Convolution

Deep Learning

Inductive bias

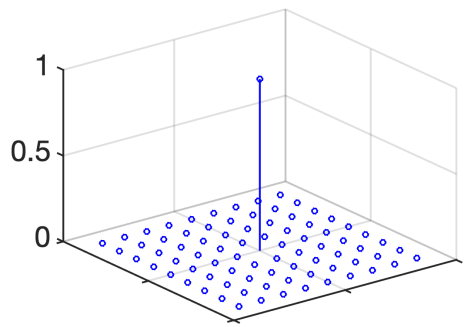
- For image processing the weight matrix in the first layer can become very large.
- This can lead to overfitting.
- By restricting the connections between layers, we can reduce the number of parameters.
- The restrictions need to be appropriate to the problem.
- Restricting the form of a network is an example of inductive bias.
- Inductive bias embeds prior knowledge about how to solve a particular problem into the neural network.

Convolution as inductive bias

- With images, it seems reasonable to consider operations that are position invariant.
- Also, we consider linear transformations, since multilayer networks use a linear transformation and a nonlinear activation.
- Any linear, position invariant transformation can be represented by a convolution.

Impulse Function

$$\delta_{i,j} = \begin{cases} 1 & i = j = 0 \\ 0 & \text{else} \end{cases}$$



Impulse Response

$$w_{i,j} = \mathcal{W}(\delta_{i,j})$$

Image as Impulses

$$p_{m,n} = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} p_{i,j} \delta_{m-i,n-j}$$

Tranform the Image with Linear and Position Invariant $\mathcal{W}$

$$z_{m,n} = \mathcal{W}(\mathbf{P})_{m,n} = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} p_{i,j} \mathcal{W}(\delta_{m-i,n-j})$$

$$= \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} p_{i,j} w_{m-i,n-j} = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} w_{i,j} p_{m-i,n-j}$$

$$\mathbf{Z} = \mathbf{P} * \mathbf{W} = \mathbf{W} * \mathbf{P}$$

Convolution with Local Receptive Field

$$z_{m,n} = \sum_{i=0}^{r-1} \sum_{j=0}^{c-1} w_{i,j} p_{m-i,n-j}$$

Correlation

$$z_{m,n} = \sum_{i=0}^{r-1} \sum_{j=0}^{c-1} w_{i,j} p_{m+i,n+j}$$

Abbreviated Notation

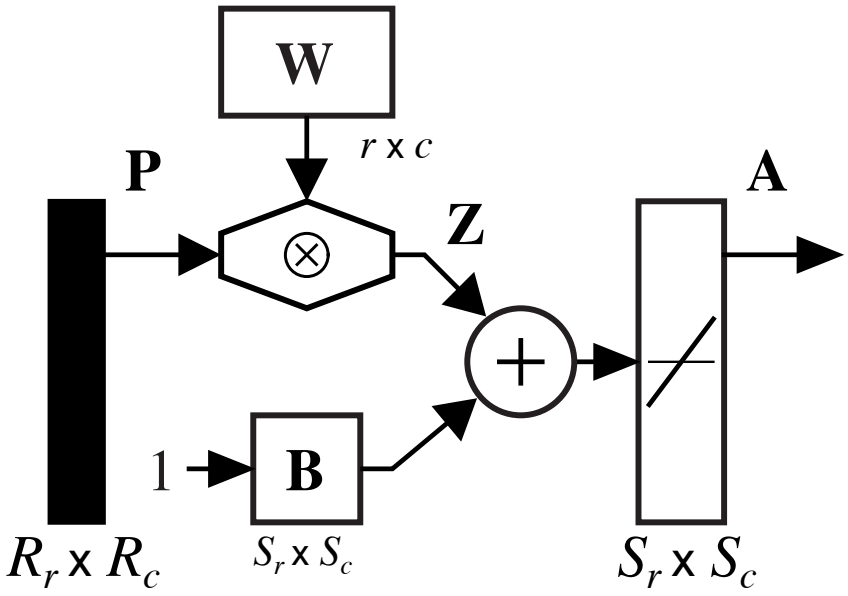
$$\mathbf{Z} = \mathbf{W} \otimes \mathbf{P}$$

Weight sharing

$$\begin{bmatrix} z_{1,1} & z_{1,2} \\ z_{2,1} & z_{2,2} \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{bmatrix} \otimes \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} \\ p_{2,1} & p_{2,2} & p_{2,3} \\ p_{3,1} & p_{3,2} & p_{3,3} \end{bmatrix}$$

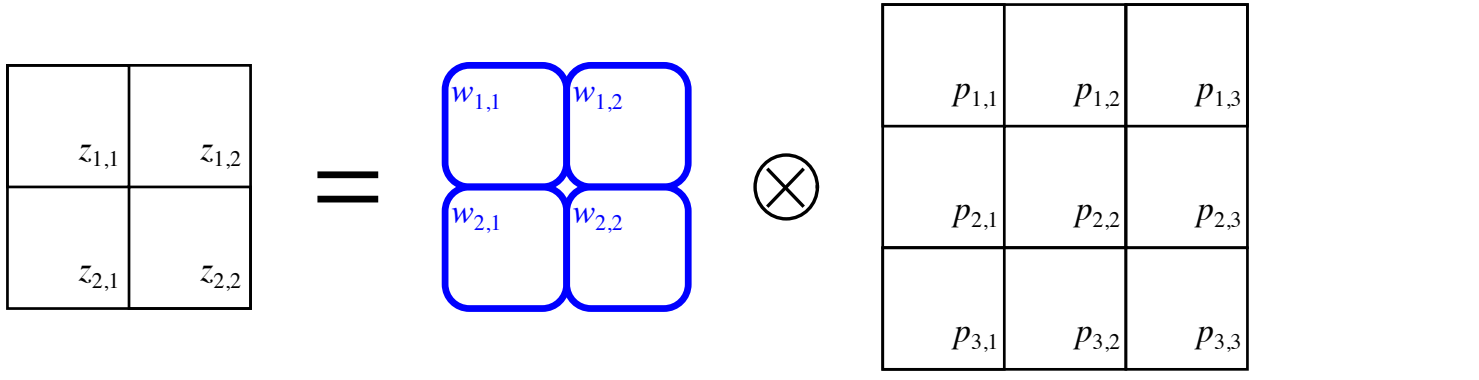
$$\begin{bmatrix} z_{1,1} \\ z_{2,1} \\ z_{1,2} \\ z_{2,2} \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{2,1} & 0 & w_{1,2} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & w_{1,1} & w_{2,1} & 0 & w_{1,2} & w_{2,2} & 0 & 0 & 0 \\ 0 & 0 & 0 & w_{1,1} & w_{2,1} & 0 & w_{1,2} & w_{2,2} & 0 \\ 0 & 0 & 0 & 0 & w_{1,1} & w_{2,1} & 0 & w_{1,2} & w_{2,2} \end{bmatrix} \begin{bmatrix} p_{1,1} \\ p_{2,1} \\ p_{3,1} \\ p_{1,2} \\ p_{2,2} \\ p_{3,2} \\ p_{1,3} \\ p_{2,3} \\ p_{3,3} \end{bmatrix}$$

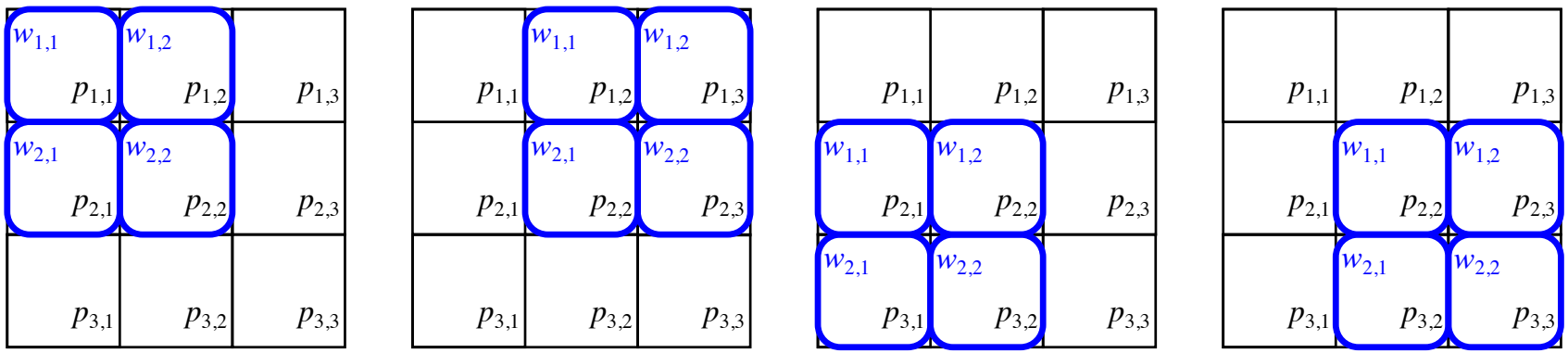
Basic convolution layer



- The weight is sometimes called a kernel.
- This layer has an untied bias (not tied to the kernel).
- A tied bias would only have one element.

Convolution operation





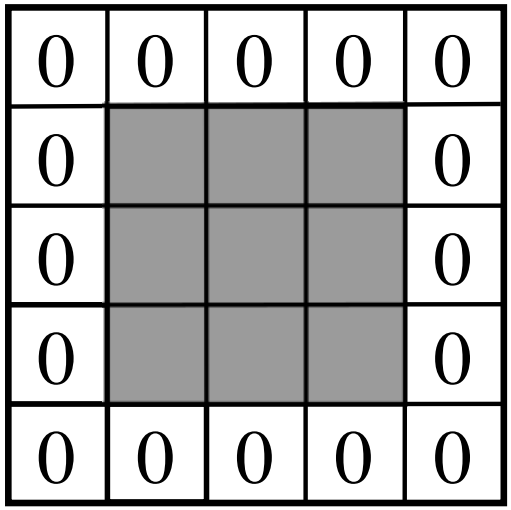
$$z_{m,n} = \sum_{i=1}^{r} \sum_{j=1}^{c} w_{i,j} p_{m+i-1,n+j-1}$$

$$S_r = R_r - r + 1$$

$$S_c = R_c - c + 1$$

Padding

- The output image will be smaller than the input image in basic convolution.
- We can correct for this by artificially increasing the input image size.
- Padding adds pixels of value 0 around the edge.

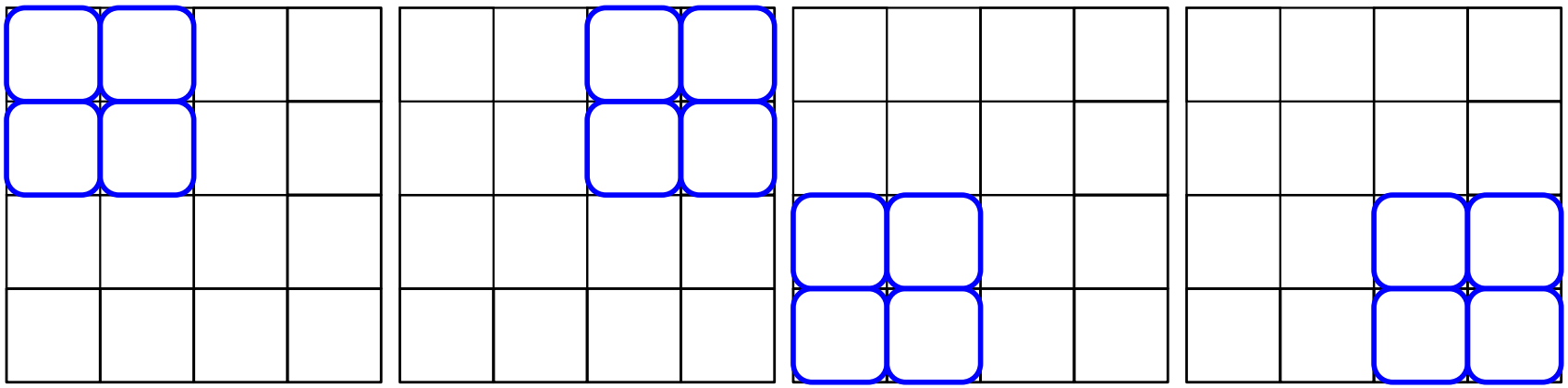


$$S_r = R_r + 2P^d - r + 1$$

$$S_c = R_c + 2P^d - c + 1$$

Stride

- Normally, the kernel is moved one step at a time.
- We can decrease the output image size by increasing the stride.
- This example has a stride of 2.

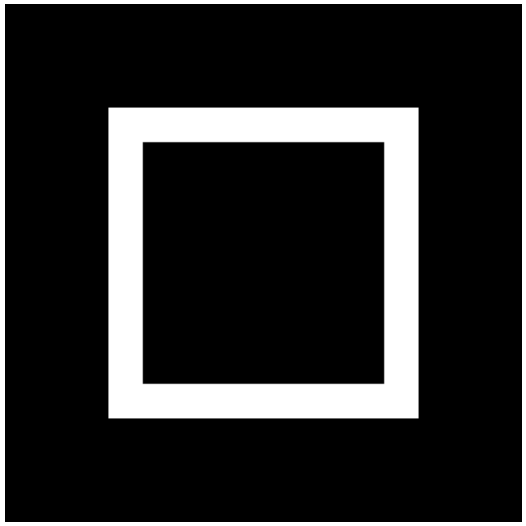


Determining output image size

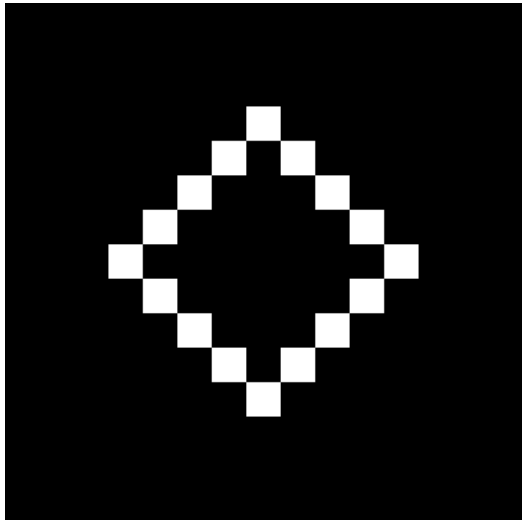
If we use both stride and padding, the output image size is computed as follows:

$$S_r = (R_r + 2P^d - r)/S^t + 1$$
$$S_c = (R_c + 2P^d - c)/S^t + 1$$

Template matching – matched filter



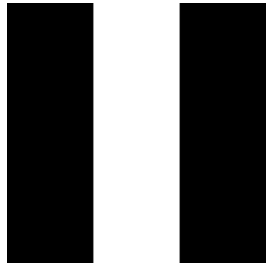
Square



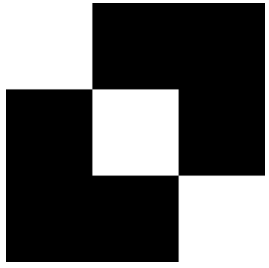
Diamond

Filter kernels

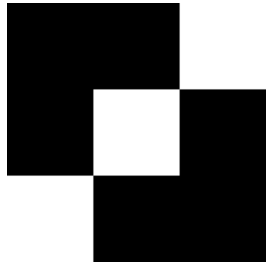
Horizontal kernel



Vertical kernel

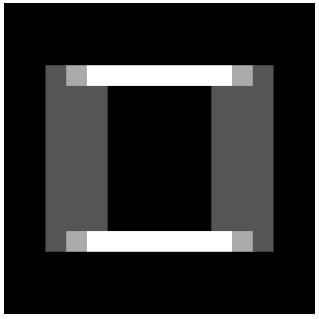


Slash kernel

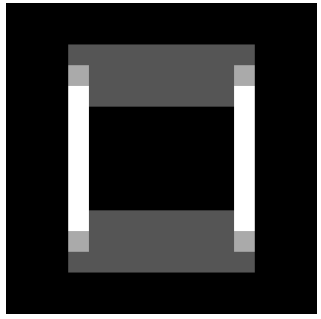


Backslash kernel

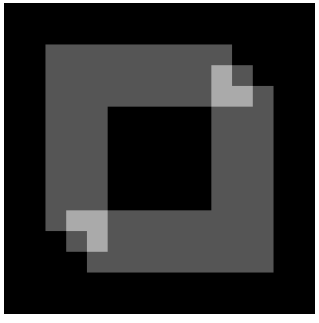
Filtered square



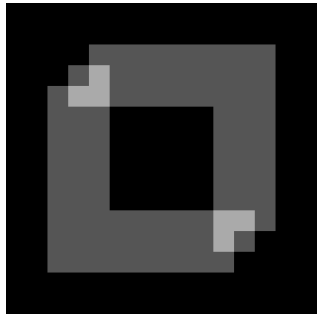
Horizontal filtered square



Vertical filtered square

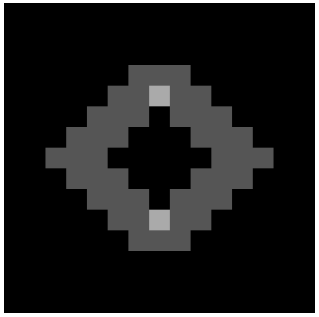


Slash filtered square

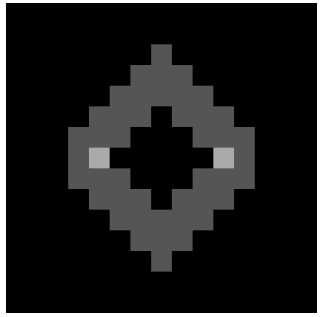


Backslash filtered square

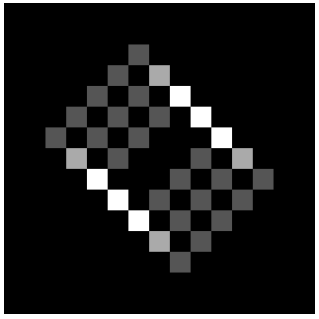
Filtered diamond



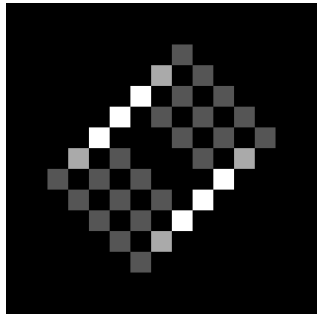
Horizontal filtered diamond



Vertical filtered diamond



Slash filtered diamond



Backslash filtered diamond

Hierachical template matching

- Stacking convolution layers in a hierarchical architecture was proposed by Fukushima in 1980.
- Inspired by the hierarchical model of the cat visual cortex developed by Hubel and Wiesel in 1962.
- Neurons in the early visual cortex are organized in a hierarchical fashion.
- The first cells are responsible for detecting simple patterns like edges and bars.
- Later layers respond to more complex patterns by combining earlier neuron outputs.
- Yann LeCun, in the late 1980s, refined the Fukushima architecture.
- He demonstrated how all network weights could be trained simultaneously using gradient descent.

Multiple kernels (feature maps)

$$z_{i,j}^{m,n,h} = \sum_{l=1}^{H^n} \sum_{u=1}^{r^m} \sum_{v=1}^{c^m} w_{u,v}^{m,n,h,l} \, a_{i+u-1,j+v-1}^{n,l}$$

$$\mathbf{Z}^{m,n,h} = \sum_{l=1}^{H^n} \mathbf{W}^{m,n,h,l} \otimes \mathbf{A}^{n,l}$$

- m represents the current layer number, with $m = 0$ for the input.
- H^m is the number of FMs in Layer m.
- n represents the previous layer number.
- h, l represent the Feature Map (FM) number

- The purpose is to reduce the spatial size of the feature map.
- This reduces the number of parameters in the network.

Average Pooling

$$z_{i,j} = \left\{ \sum_{k=1}^{r} \sum_{l=1}^{c} v_{r(i-1)+k,c(j-1)+l} \right\} w$$

$$\mathbf{Z} = w \boxplus_{r,c}^{ave} \mathbf{V}$$

Max Pooling

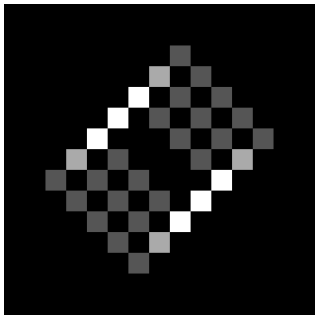
$$z_{i,j} = max \left\{ v_{r(i-1)+k,c(j-1)+l} | k = 1, ..., r; l = 1, ..., c \right\}$$

$$\mathbf{Z} = \boxplus_{r,c}^{max} \mathbf{V}$$

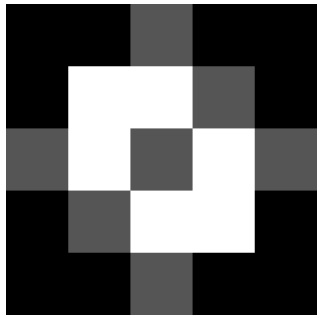
Pooling: choice of stride and size

- As in convolution operations, various strides can be used in pooling operations.
- Normally, the consolidation window is square, and the stride is equal to the window size, so there is no overlap in the consolidations.
- The most common choice is $r = 2$, $c = 2$ and $S^t = 2$.

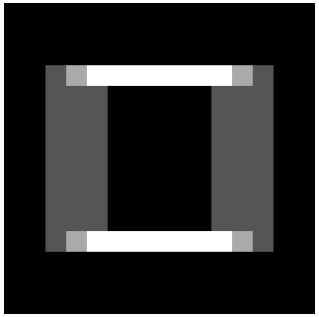
Max pooling (3x3) examples



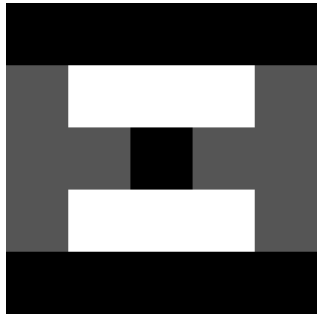
Backslash filtered diamond



Max pooled filtered diamond



Horizontal filtered square



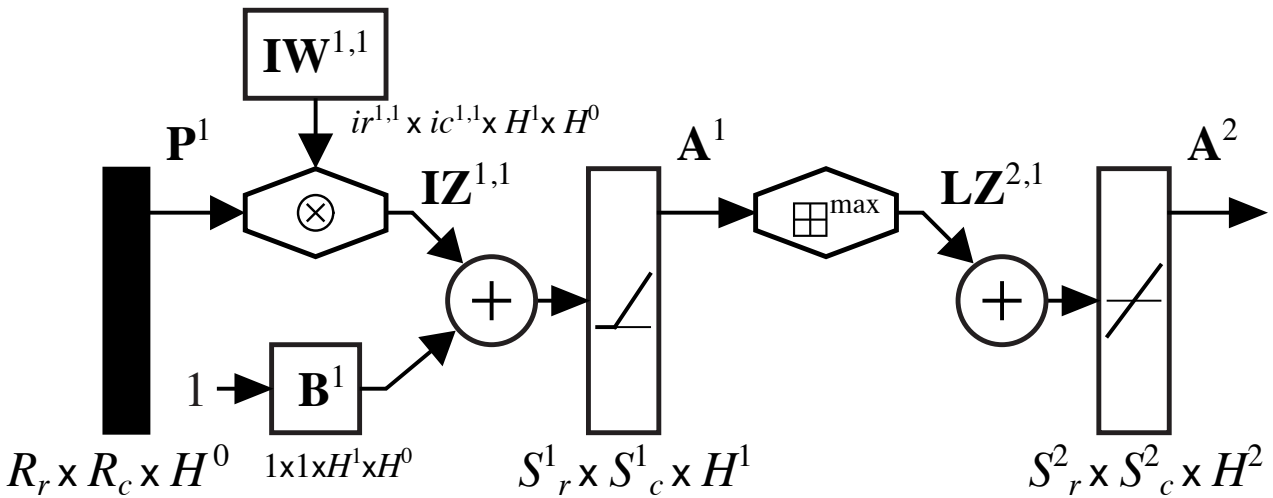
Max pooled filtered square

Feature maps

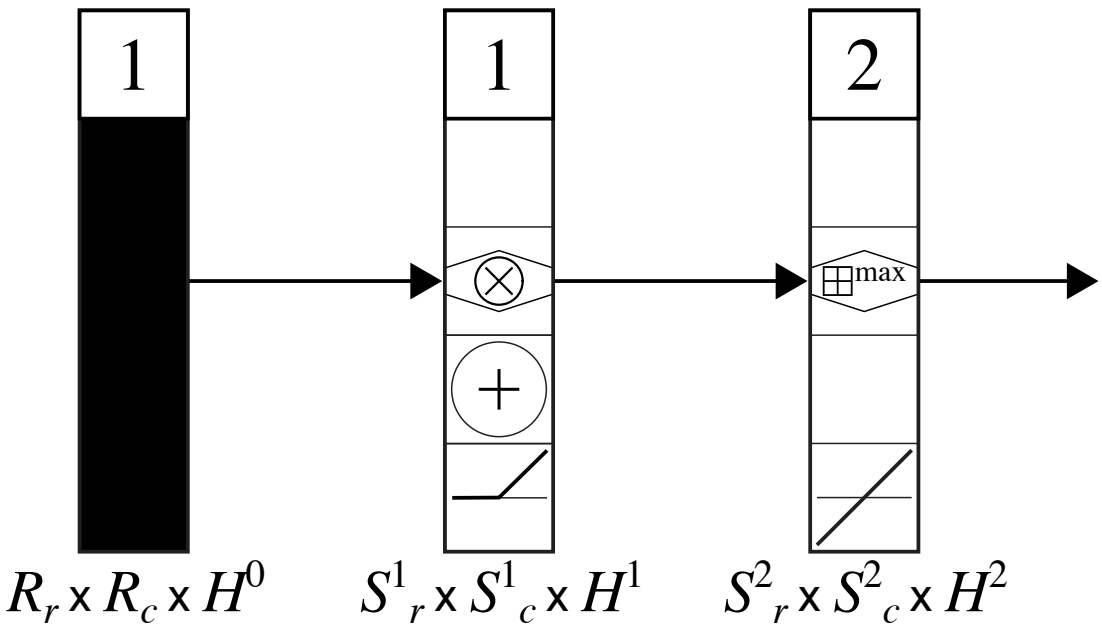
- Each convolution kernel (weight) can identify one elemental feature in the input.
- Complex patterns will consist of combinations of many elemental features.
- Multiple kernels can be included in a single convolution layer to extract multiple features.
- LeCun, in his original developement, called the outputs of one kernel operation a feature map (FM).
- This idea can be extended to the input image. For example, a color image consists of red, green and blue planes.
- A convolution layer then takes a set of feature maps as an input, and produces another set of feature maps as an output.

Network diagram

- A tied bias, with one scalar bias for each kernel.
- There are $H^1 \times H^0$ kernels – H^0 planes in the input, H^1 FMs in the first layer.
- The dimensions of the terms coming into the summation do not have the same dimension, but we assume broadcasting.
- For pooling, the # of input FMs equals the # of output FMs ($H^1 = H^2$).



When many layers are involved, it is helpful to have a simplified notation to represent each layer.



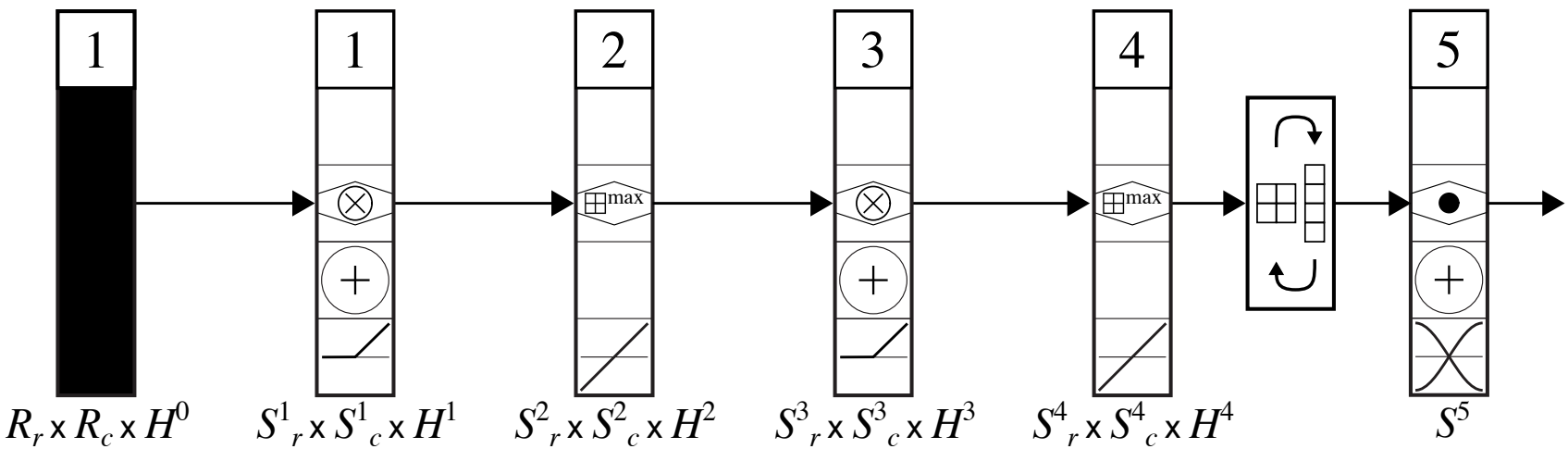
Conversion from matrix to vector

- An additional block can appear between layers of a convolution network.
- It converts the output of a layer from a set of FMs to a single vector.
- It stacks the columns of the FMs on top of each other, starting from the first FM to the last. ($\mathbf{a}_i$ indicates the i^{th} column of matrix $\mathbf{A}$.)

$$vec(\mathbf{A}^m) = \left[\left(\mathbf{a}_1^{m,1}\right)^T \left(\mathbf{a}_2^{m,1}\right)^T \cdots \left(\mathbf{a}_{S_c^m}^{m,1}\right)^T \left(\mathbf{a}_1^{m,2}\right)^T \cdots \left(\mathbf{a}_{S_c^m}^{m,H^m}\right)^T \right]^T$$

Use of matrix to vector conversion

The matrix to vector conversion is normally used before the final layer of a convolution network, which is a standard dot product (matrix multiplication) layer. (The same conversion block can indicate vector to matrix conversion, if needed.)



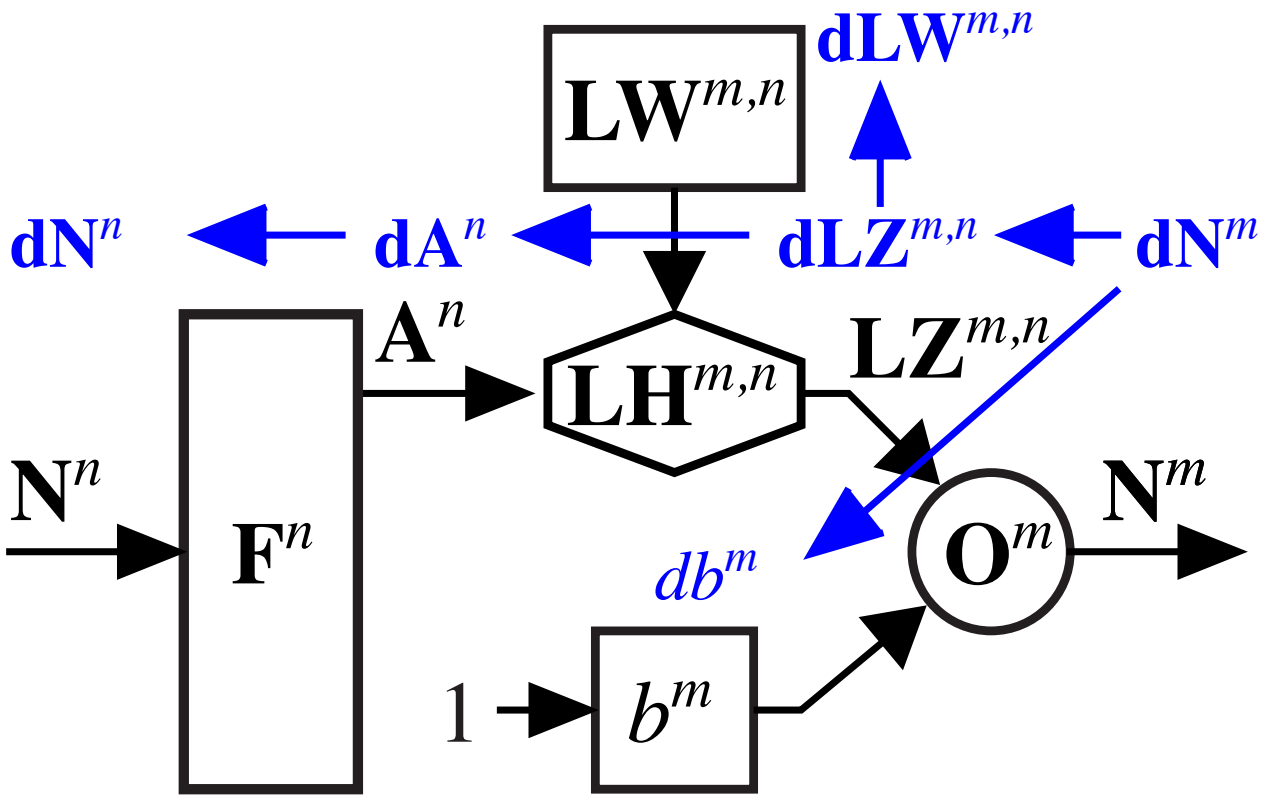
In order to compute the gradient of the performance with respect to the weights and biases, we need to perform a backpropagation operation. Because the net input is a matrix, we will use a different notation for sensitivity – $\mathbf{dN}$. Other derivatives will be defined as:

$$\mathbf{dN} \equiv \frac{\partial F}{\partial \mathbf{N}}, \mathbf{dA} \equiv \frac{\partial F}{\partial \mathbf{A}}, \mathbf{dZ} \equiv \frac{\partial F}{\partial \mathbf{Z}}, \mathbf{dW} \equiv \frac{\partial F}{\partial \mathbf{W}}, \mathbf{db} \equiv \frac{\partial F}{\partial \mathbf{B}}$$

As we backpropagate across Layer m, we will be performing the following operations.

$$\mathbf{dN}^m \rightarrow \mathbf{dLZ}^{m,n} \rightarrow \mathbf{dA}^n \rightarrow \mathbf{dN}^n$$

Backpropagating across a layer



Backpropagating across the summation net input function

Scalar form (dz)

$$dlz_{i,j}^{m,n} = dn_{i,j}^{m}$$

Matrix form ($\mathbf{dZ}$)

$$\mathbf{dLZ}^{m,n} = \mathbf{dN}^{m}$$

Scalar form (db)

$$db^{m} \equiv \frac{\partial F}{\partial b^{m}} = \sum_{i=1}^{S_r^m} \sum_{j=1}^{S_c^m} \frac{\partial F}{\partial n_{i,j}^{m}} \times \frac{\partial n_{i,j}^{m}}{\partial b^{m}} = \sum_{i=1}^{S_r^m} \sum_{j=1}^{S_c^m} dn_{i,j}^{m}$$

Matrix form (db)

$$db^{m} = \boxplus_{S_r^m, S_c^m}^{ave} \mathbf{dN}^{m}$$

Backpropagating across the convolution weight function

Scalar form (da)

$$da^{n,l}_{u',v'} \equiv \frac{\partial F}{\partial a^{n,l}_{u',v'}} = \sum_{h=1}^{H^n} \sum_{i=1}^{S^m_r} \sum_{j=1}^{S^m_c} \frac{\partial F}{\partial lz^{m,n,h}_{i,j}} \times \frac{\partial lz^{m,n,h}_{i,j}}{\partial a^{n,l}_{u',v'}}$$

$$da^{n,l}_{u',v'} = \sum_{h=1}^{H^n} \sum_{i'=1}^{r^m} \sum_{j'=1}^{c^m} lw^{m,n,h,l}_{r^m-i'+1,c^m-j'+1} dlz^{m,n,h}_{i'+u'-r^m,j'+v'-c^m}$$

Matrix form ($\mathbf{dA}$)

$$\mathbf{dA}^{n,l} = \sum_{l=1}^{H^n} ((rot180)\mathbf{LW}^{m,n,h,l}) \otimes \mathbf{dLZ}^{m,n,h}$$

Here rot180 means matrix is rotated by 180 degrees.

Backpropagating to convolution weight

Scalar form (da)

$$dlw_{u,v}^{m,n,h,l} \equiv \frac{\partial F}{\partial lw_{u,v}^{m,n,h,l}} = \sum_{i=1}^{S_r^m} \sum_{j=1}^{S_c^m} \frac{\partial F}{\partial lz_{i,j}^{m,n,h}} \times \frac{\partial lz_{i,j}^{m,n,h}}{\partial lw_{u,v}^{m,n,h,l}}$$

$$dlw_{u,v}^{m,n,h,l} = \sum_{i=1}^{S_r^m} \sum_{j=1}^{S_c^m} dlz_{i,j}^{m,n,h} \times a_{i+u-1,j+v-1}^{n,l}$$

Matrix form ($\mathbf{dLW}$)

$$\mathbf{dLW}^{m,n,h,l} = \mathbf{dLZ}^{m,n,h} \otimes \mathbf{A}^{n,l}$$

Backpropagating across the activation function

$$dn_{i,j}^{n,h} \equiv \frac{\partial \hat{F}}{\partial n_{i,j}^{n,h}} = \frac{\partial a_{i,j}^{n,h}}{\partial n_{i,j}^{n,h}} \times \frac{\partial \hat{F}}{\partial a_{i,j}^{n,h}}$$

$$\frac{\partial \hat{F}}{\partial a_{i,j}^{n,h}} \equiv da_{i,j}^{n,h}$$

$$dn_{i,j}^{n,h} = \dot{f}^{n,h}(n_{i,j}^{n,h}) \times da_{i,j}^{n,h}$$

Matrix form ($\mathbf{dN}$)

$$\mathbf{dN}^{n,h} = \dot{\mathbf{F}}^{n,h} \circ \mathbf{dA}^{n,h}$$

Here $\circ$ is the Hadamard product, which is an element by element matrix multiplication.