Programming of Neural Networks

Today: Convolution2D Layers

Today: Convolution2D Layers

```
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten

#create model
model = Sequential()

#add model layers
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,1)))
```

➔ We are truly spending an entire lecture for this single line of code!

Why Convolution2D?

- Fully Connected Layers may fail to end up robust against translation of the input

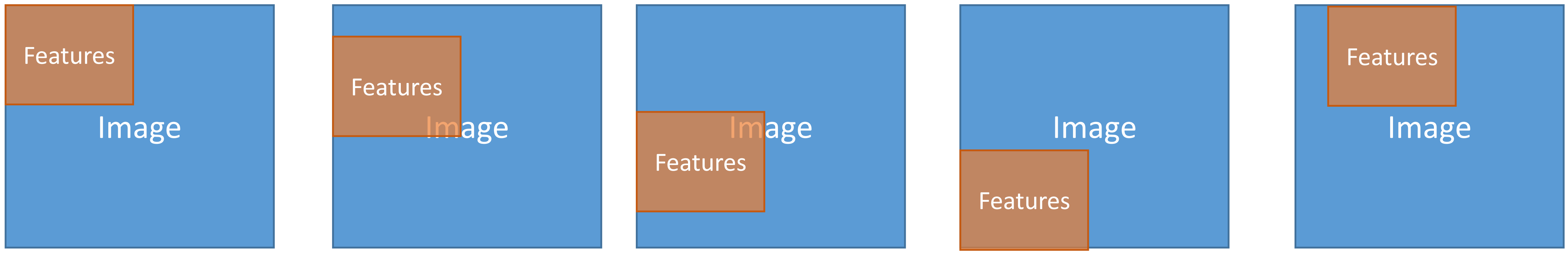
➔

➔We could try to detect the same features at every location in the image

➔ How can this be done?

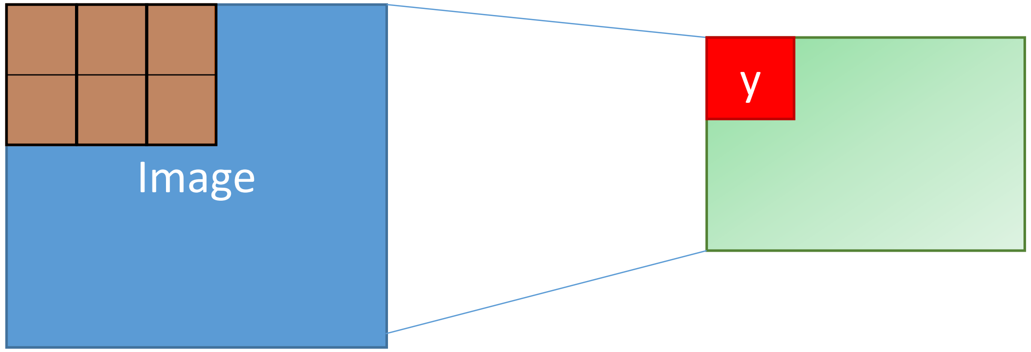
Why Convolution2D?

- A small set of local features is extracted at every (possible) location in the image



Why Convolution2D?

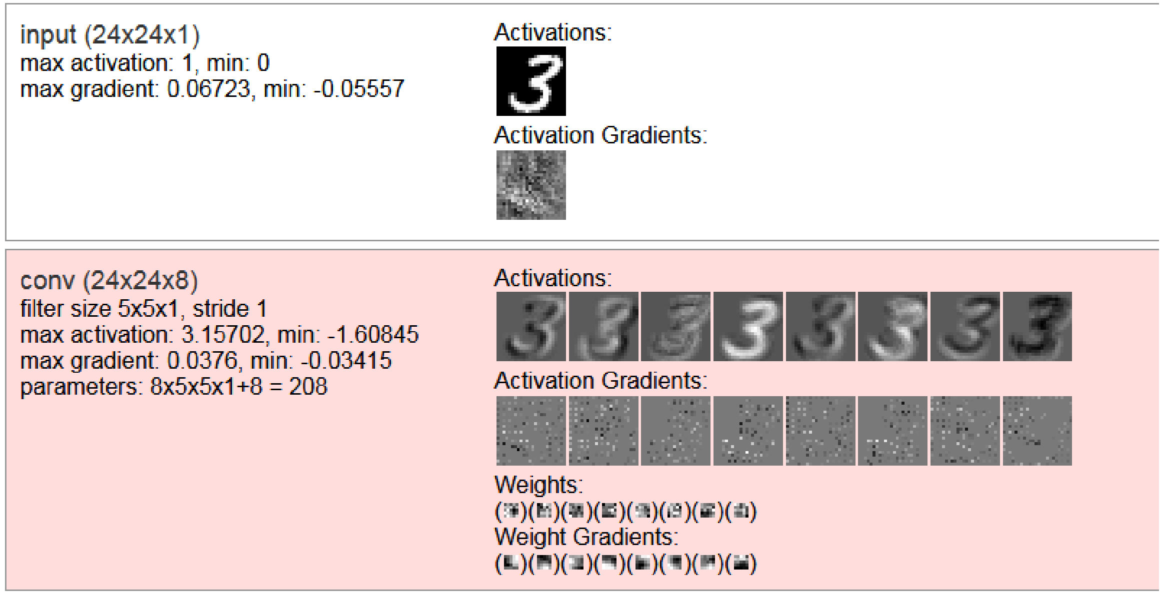
- This extraction is realized as an inner product of the overlapping elements



$$y = i_0 \cdot f_0 + i_1 \cdot f_1 + i_2 \cdot f_2 + \ \ldots$$

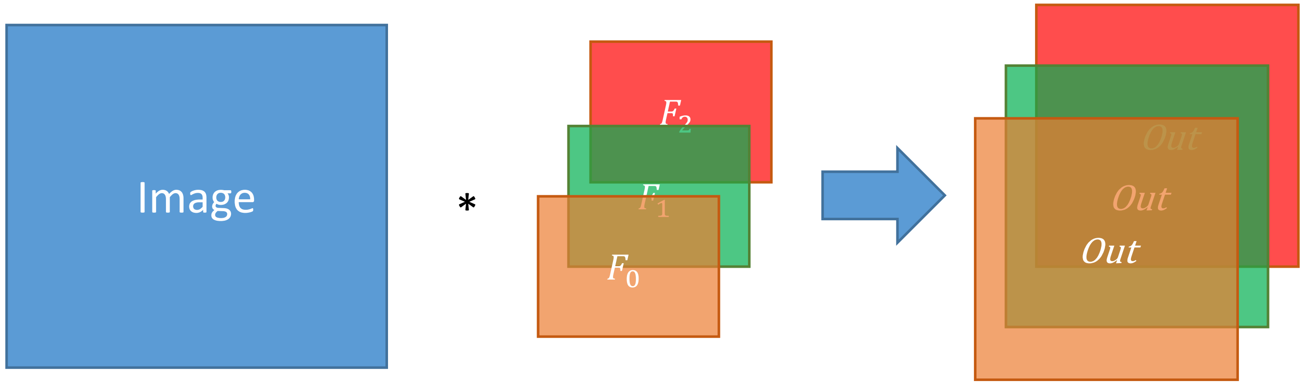
Why Convolution2D?

- One such feature might be related to an edge detection in the image
- But usually we want to extract more features
  - ➔We can simply use more (completely independent) feature masks!
  - ➔We call such a feature mask „kernel“ or „filter“



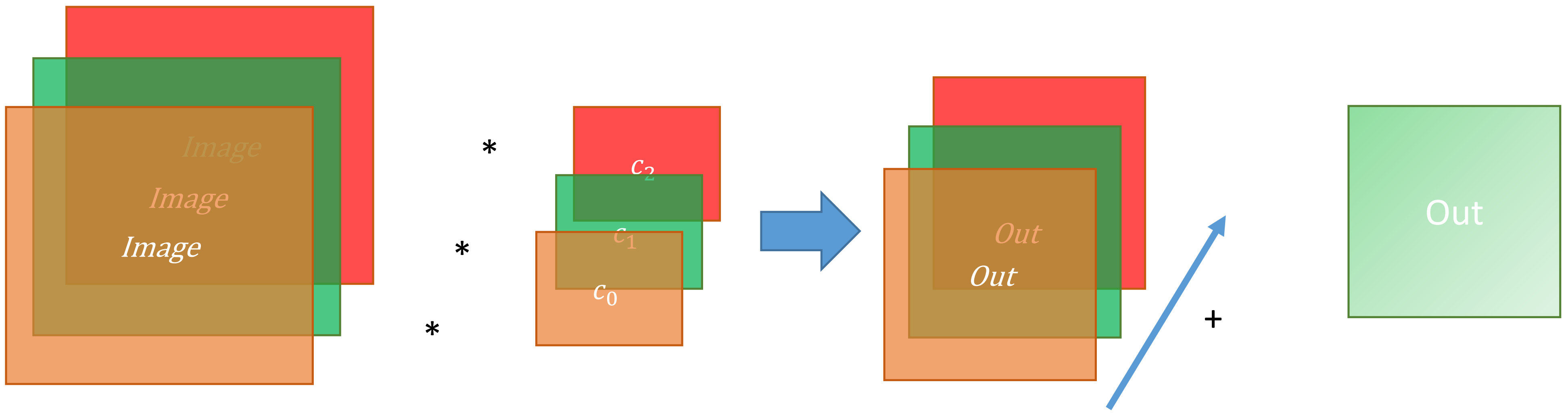
Convolution 2D with multiple filters

- Every application of a filter on the input image creates an output image



Convolution 2D with multiple filters

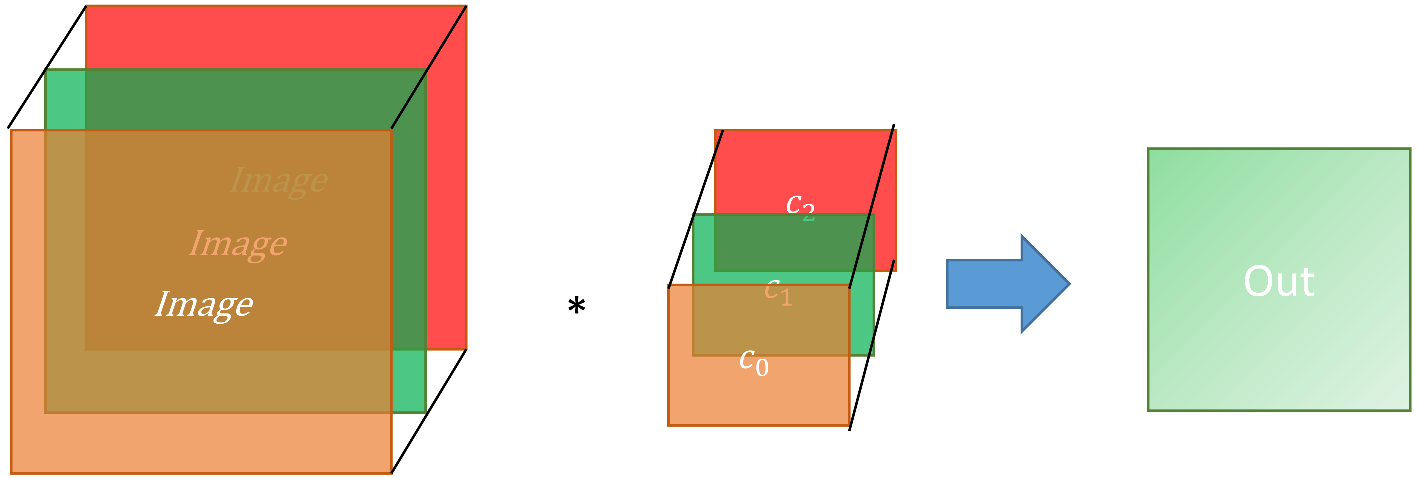
- If the image has a depth (or channel, such as R,G,B), then the filter requires a depth as well



- So one filter produces still just one 2D-Output

Convolution 2D with multiple filters

- Alternative interpretation: Slide a cubic filter in a cubic „image“



- Multiple filters will yet again produce multiple „images“ (or channels)

➔An **out-channel** corresponds to an **in-filter**

Extending the framework

- To extend our previous framework, we will:
  - Create a new implementation of our Layer interface
  - Define its parameters as well as its:
    1. Forward Pass
    2. Backward Pass
    3. Calculation of the Weight Derivatives

Parameters of the Conv2D Layer

- Such a layer contains parameters in the form of a 4-dimensional Tensor, called the kernel-tensor or filter-tensor which forms the weights of this layer

Filter size in x-direction | Filter size in y-direction | Filter depth | Amount filters

Filter [2,2,2,2]

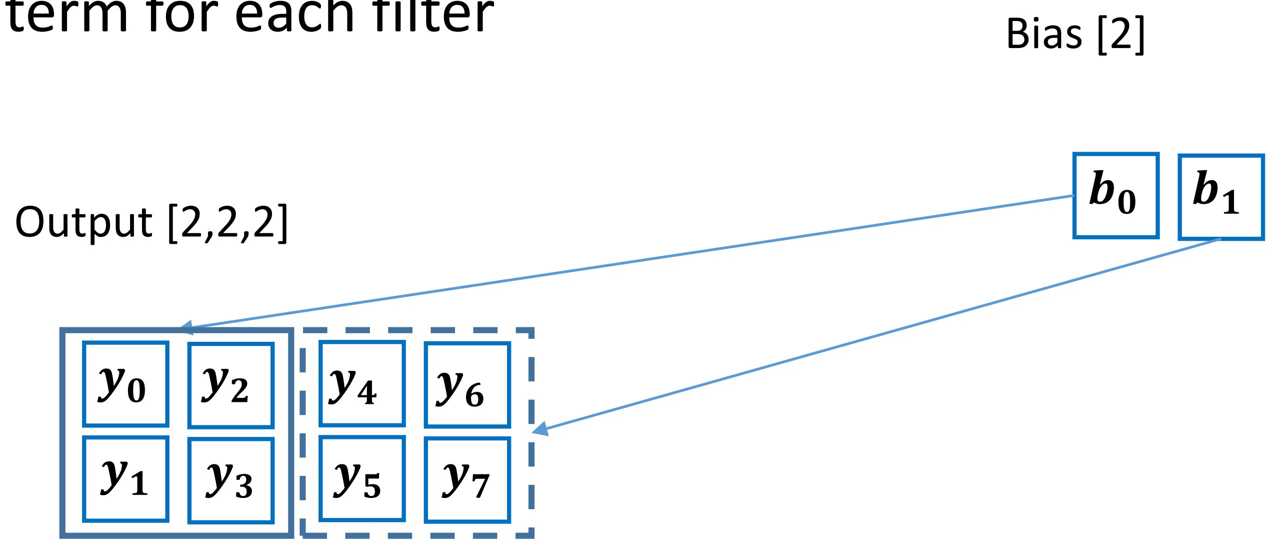
Filter^1

$$\begin{matrix} f_0^1 & f_2^1 \\ f_1^1 & f_3^1 \end{matrix} \qquad \begin{matrix} f_4^1 & f_6^1 \\ f_5^1 & f_7^1 \end{matrix}$$

Filter^2

Parameters of the Conv2D Layer

- A bias term for each filter



Parameters of the Conv2D Layer

- Additional Parameters:
  - Padding (values: None, Half, Full), explained later
  - Size of the input Tensor
  - Dilation, explained later (optional)
  - Stride (in x and y direction, optional)
  - Shape of the kernel tensor or kernelsize and amount kernels
- First question: How do we calculate the output size of the convolution layer?

Output-Shape of Convolution Layers

- If we perform a standard convolution of an [x,y] image, using a $[k_x, k_y]$ kernel, we get:

$$o_x = (x - k_x) + 1$$
$$o_y = (y - k_y) + 1$$

Convolution-Example: Forward

Input [3,3,2]

x_0	x_3	x_6
x_1	x_4	x_7
x_2	x_5	x_8

Channel 0

x_9	x_{12}	x_{15}
x_{10}	x_{13}	x_{16}
x_{11}	x_{14}	x_{17}

Channel 1

Output [2,2,2]

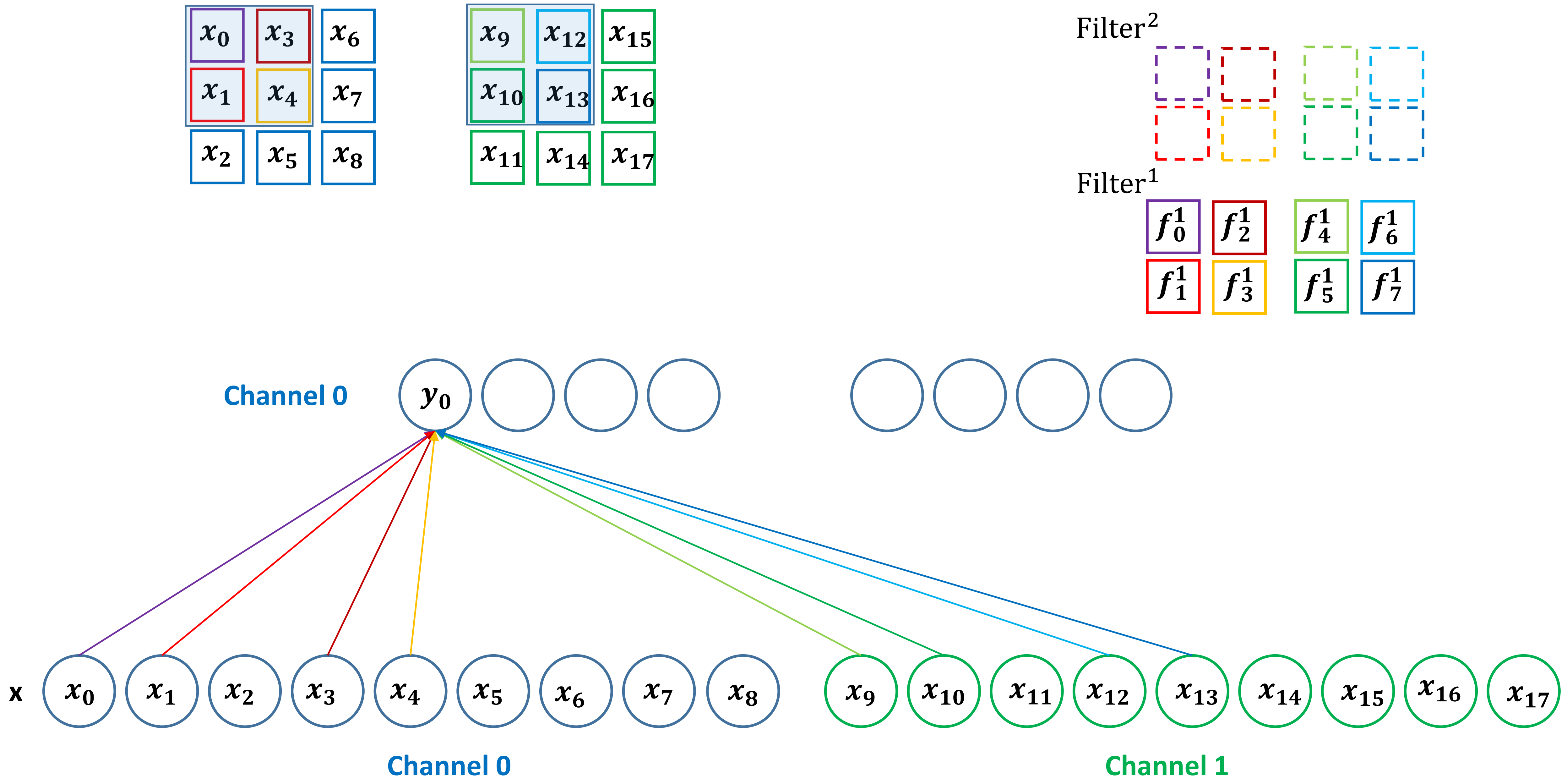
y_0	y_2	y_4	y_6
y_1	y_3	y_5	y_7

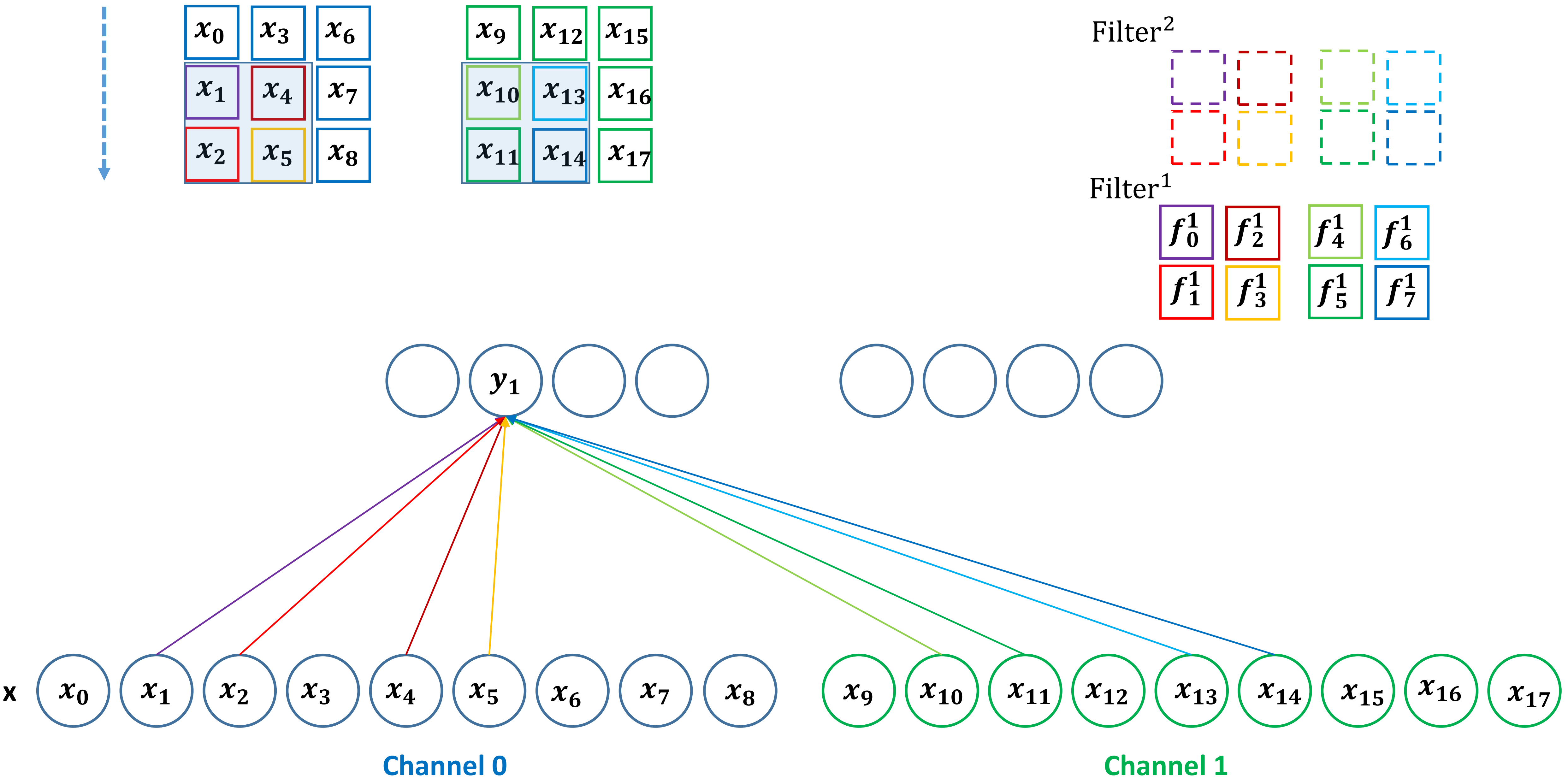
Filter [2,2,2,2]

Filter^1

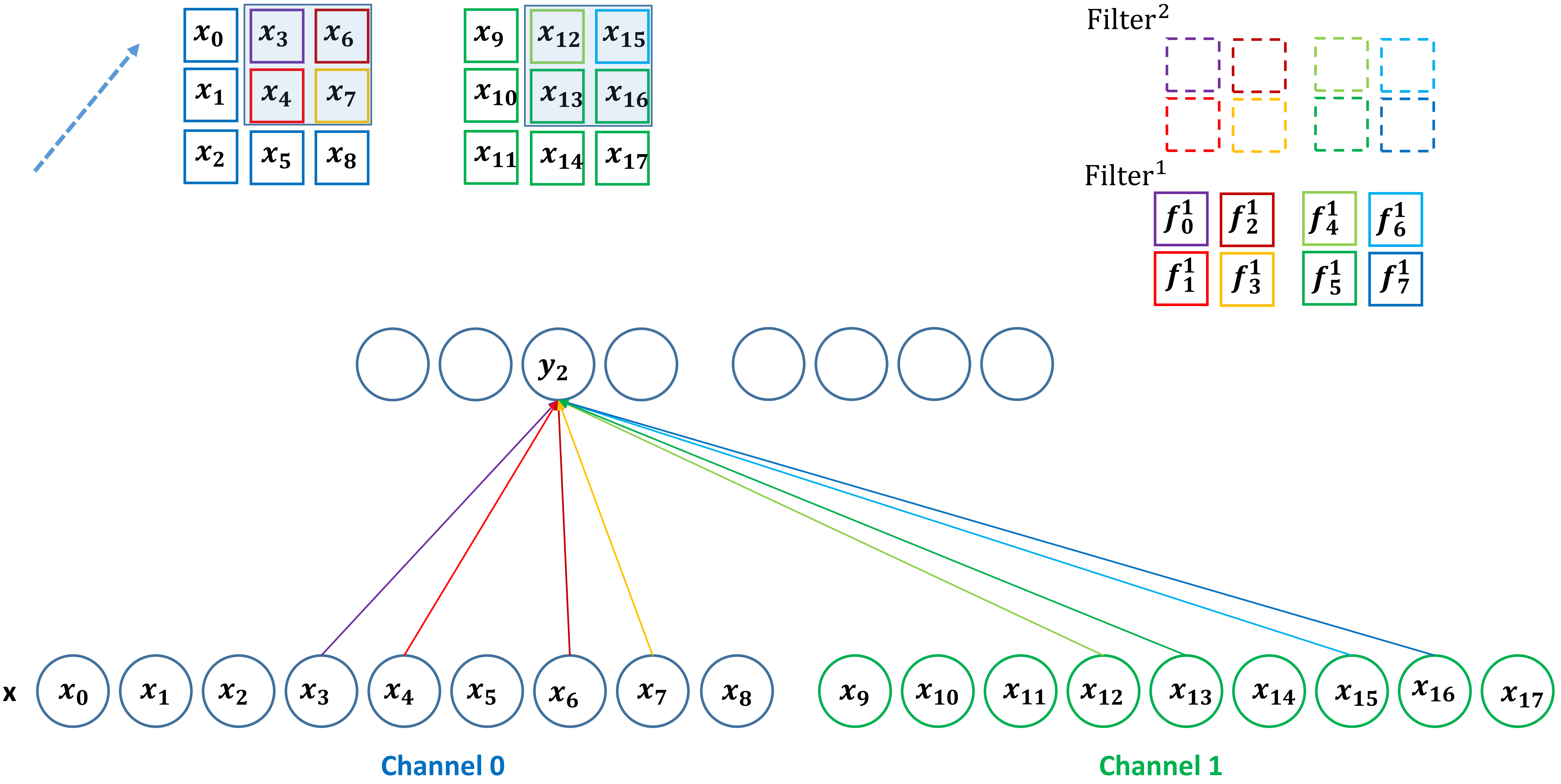
f_0^1	f_2^1	f_4^1	f_6^1
f_1^1	f_3^1	f_5^1	f_7^1

Filter^2

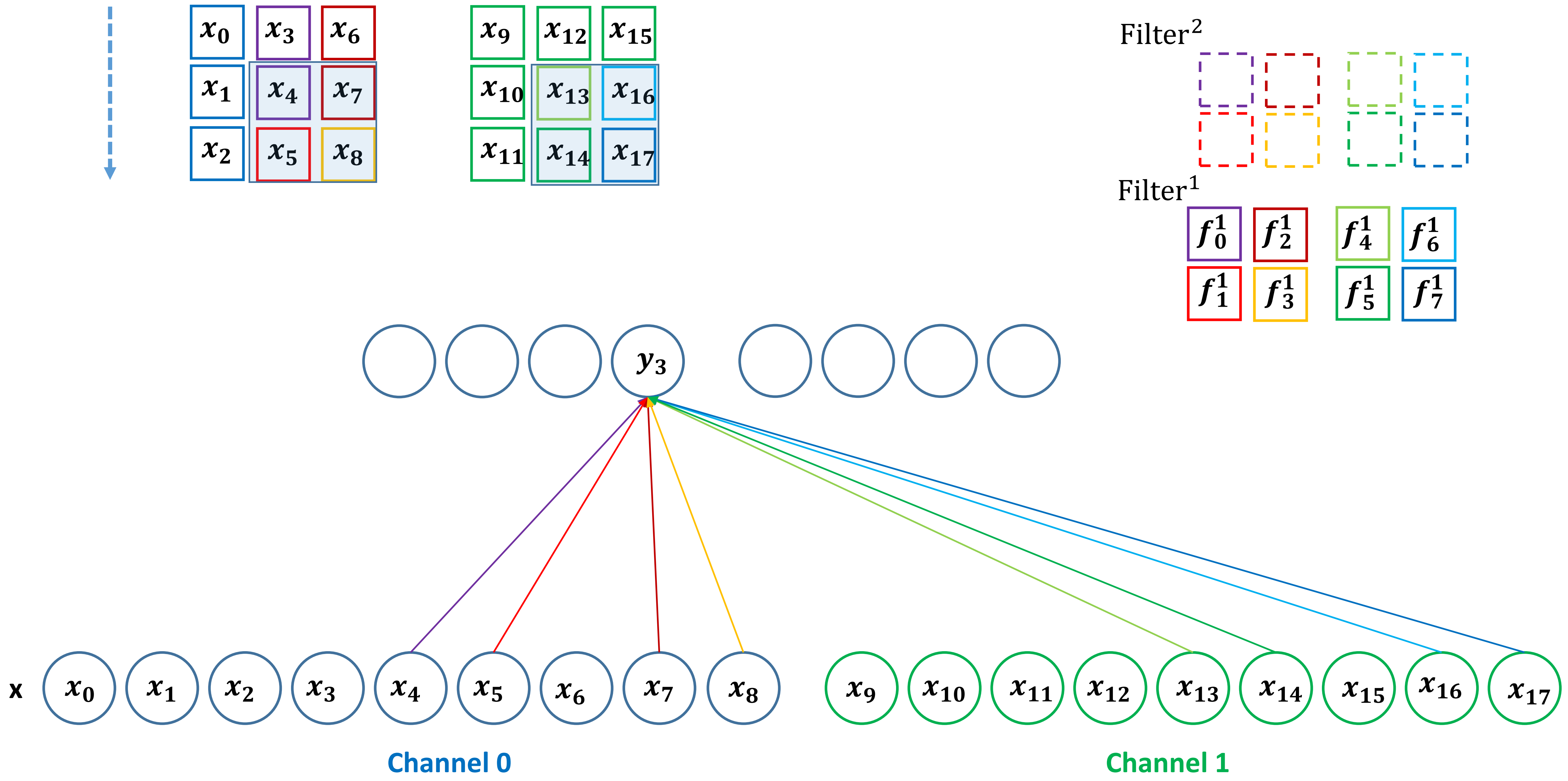

x_0 x_3 x_6
x_1 x_4 x_7
x_2 x_5 x_8
x_9 x_{12} x_{15}
x_{10} x_{13} x_{16}
x_{11} x_{14} x_{17}
Filter^2
Filter^1
f_0^1 f_2^1 f_4^1 f_6^1
f_1^1 f_3^1 f_5^1 f_7^1
Channel 0
y_0
x
x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15} x_{16} x_{17}
Channel 0
Channel 1

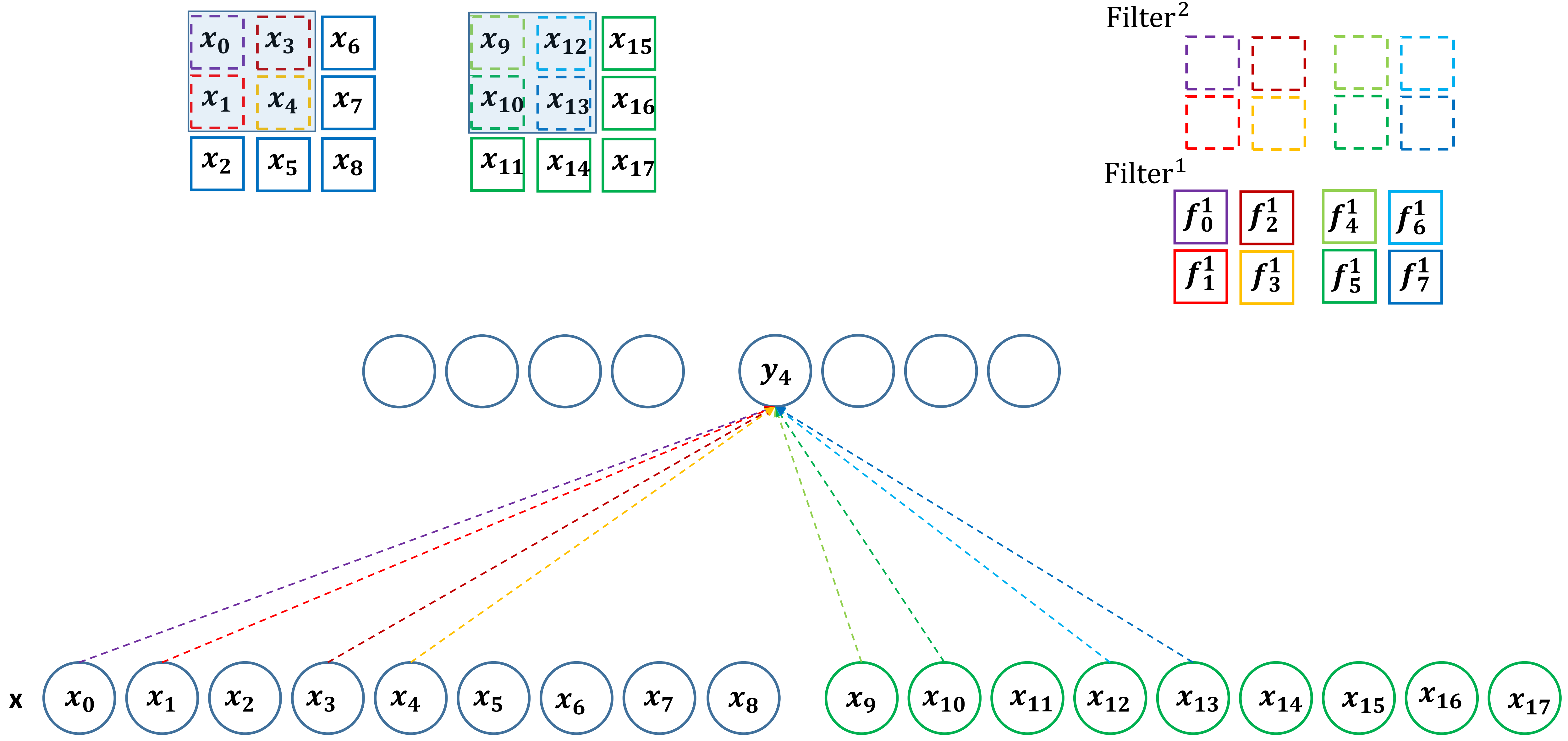


x_0 x_3 x_6
x_1 x_4 x_7
x_2 x_5 x_8
x_9 x_{12} x_{15}
x_{10} x_{13} x_{16}
x_{11} x_{14} x_{17}
Filter^2
Filter^1
f_0^1 f_2^1 f_4^1 f_6^1
f_1^1 f_3^1 f_5^1 f_7^1
y_1
x
x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8
x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15} x_{16} x_{17}
Channel 0
Channel 1



x_0 x_3 x_6
x_1 x_4 x_7
x_2 x_5 x_8
x_9 x_{12} x_{15}
x_{10} x_{13} x_{16}
x_{11} x_{14} x_{17}
Filter^2
Filter^1
f_0^1 f_2^1 f_4^1 f_6^1
f_1^1 f_3^1 f_5^1 f_7^1
y_2
x
x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8
x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15} x_{16} x_{17}
Channel 0
Channel 1


x_0 x_3 x_6
x_1 x_4 x_7
x_2 x_5 x_8
x_9 x_{12} x_{15}
x_{10} x_{13} x_{16}
x_{11} x_{14} x_{17}
Filter^2
Filter^1
f_0^1 f_2^1 f_4^1 f_6^1
f_1^1 f_3^1 f_5^1 f_7^1
y_3
x
x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8
x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15} x_{16} x_{17}
Channel 0
Channel 1


x_0 x_3 x_6
x_1 x_4 x_7
x_2 x_5 x_8
x_9 x_{12} x_{15}
x_{10} x_{13} x_{16}
x_{11} x_{14} x_{17}
Filter^2
Filter^1
f_0^1 f_2^1 f_4^1 f_6^1
f_1^1 f_3^1 f_5^1 f_7^1
y_4
x
x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8
x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15} x_{16} x_{17}
Channel 0
Channel 1

Forward: Convolution2D

- The forward pass of a convolution layer can be implemented rather simple

$$Y = X * F + \mathrm{bias}$$

- With X being the input tensor, F the kernel tensor, Y the output tensor and $*$ denotes the convolution operator for images with a depth.

Convolution-Example: Backward

Input: Deltas of output[2,2,2]

δy_0	δy_2	δy_4	δy_6
δy_1	δy_3	δy_5	δy_7

Output: Deltas of input [3,3,2]

δx_0	δx_3	δx_6
δx_1	δx_4	δx_7
δx_2	δx_5	δx_8

Channel 0

δx_9	δx_{12}	δx_{15}
δx_{10}	δx_{13}	δx_{16}
δx_{11}	δx_{14}	δx_{17}

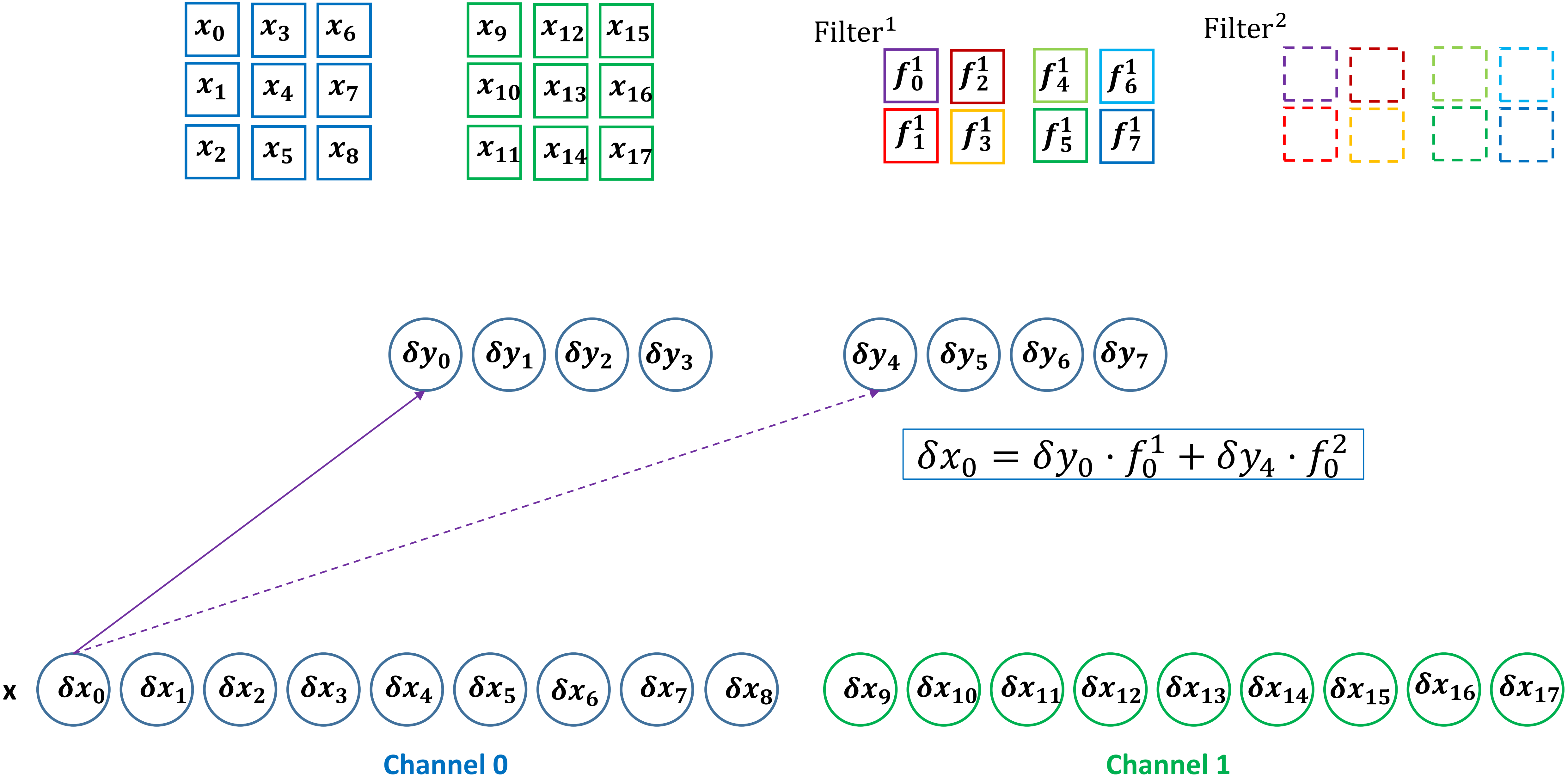
Channel 1

Filter [2,2,2,2]

Filter^1

f_0^1	f_2^1	f_4^1	f_6^1
f_1^1	f_3^1	f_5^1	f_7^1

Filter^2



$$\delta x_0 = \delta y_0 \cdot f_0^1 + \delta y_4 \cdot f_0^2$$

Why?

- We start with the calculus:

Our deltas:
$[\delta y_0 \; \dots \delta y_7]$

$$\frac{\partial L}{\partial x_0} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial x_0}$$

Let us derive $\frac{\partial y}{\partial x_0}$

$$\begin{aligned}
y_0 &= f_0^1 \cdot x_0 + \; \dots \\
y_1 &= \text{something without } x_0 \\
y_2 &= \text{something without } x_0 \\
y_3 &= \text{something without } x_0 \\
y_4 &= f_0^2 \cdot x_0 + \dots \\
y_5 &= \text{something without } x_0 \\
y_6 &= \text{something without } x_0 \\
y_7 &= \text{something without } x_0
\end{aligned}$$

$$\begin{aligned}
\frac{\partial y_0}{\partial x_0} &= f_0^1 \\
\frac{\partial y_1}{\partial x_0} &= 0 \\
\frac{\partial y_2}{\partial x_0} &= 0 \\
\frac{\partial y_3}{\partial x_0} &= 0 \\
\frac{\partial y_4}{\partial x_0} &= f_0^2 \\
\frac{\partial y_5}{\partial x_0} &= 0 \\
\frac{\partial y_6}{\partial x_0} &= 0 \\
\frac{\partial y_7}{\partial x_0} &= 0
\end{aligned}$$

$$\frac{\partial y}{\partial x_0} = [f_0^1, 0,0,0, f_0^2, 0,0,0]$$

Why?

Our deltas:
$[\delta y_0 \ \ldots \delta y_7]$

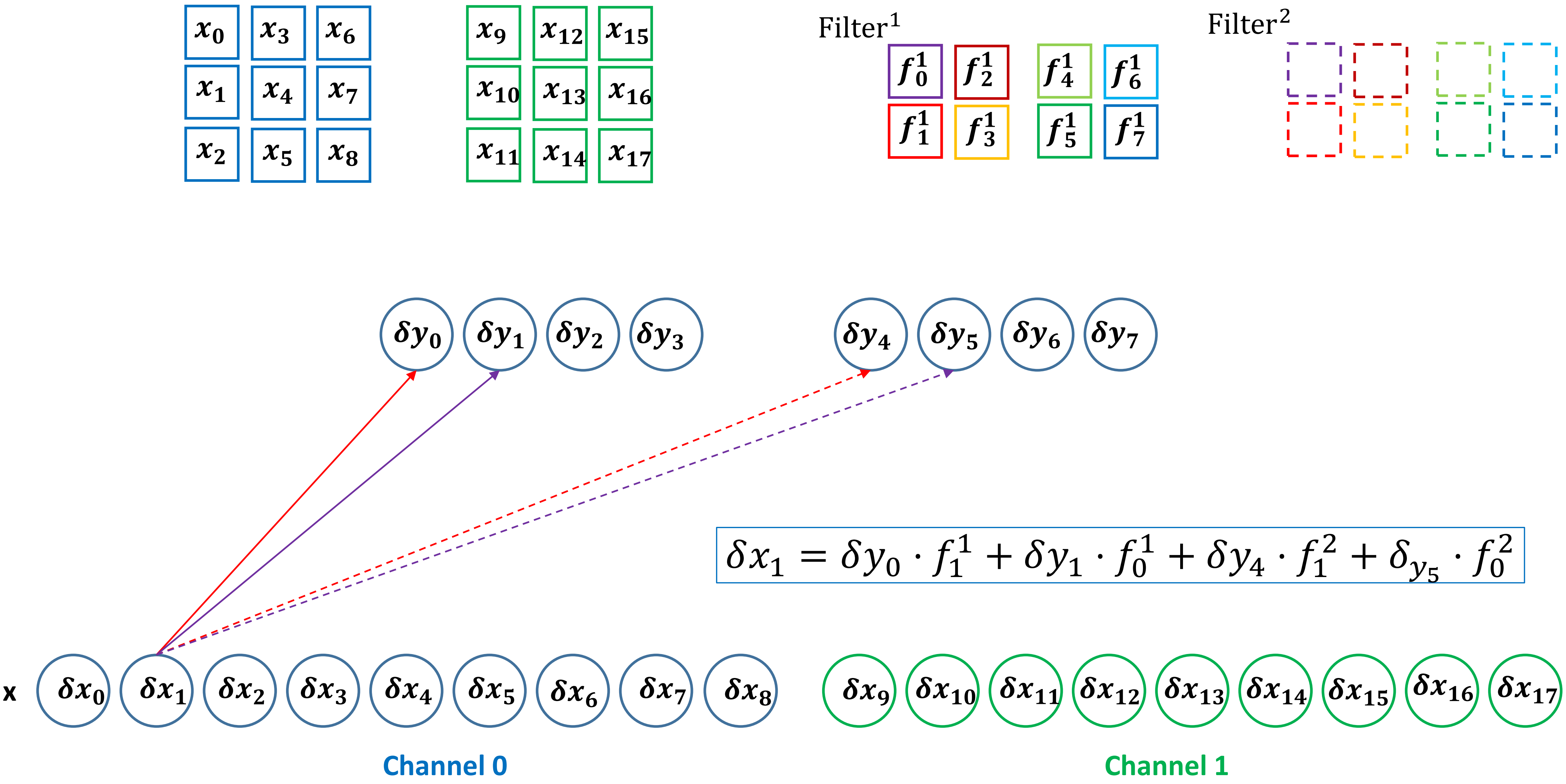
- We start with the calculus:

$$\frac{\partial L}{\partial x_0} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial x_0}$$

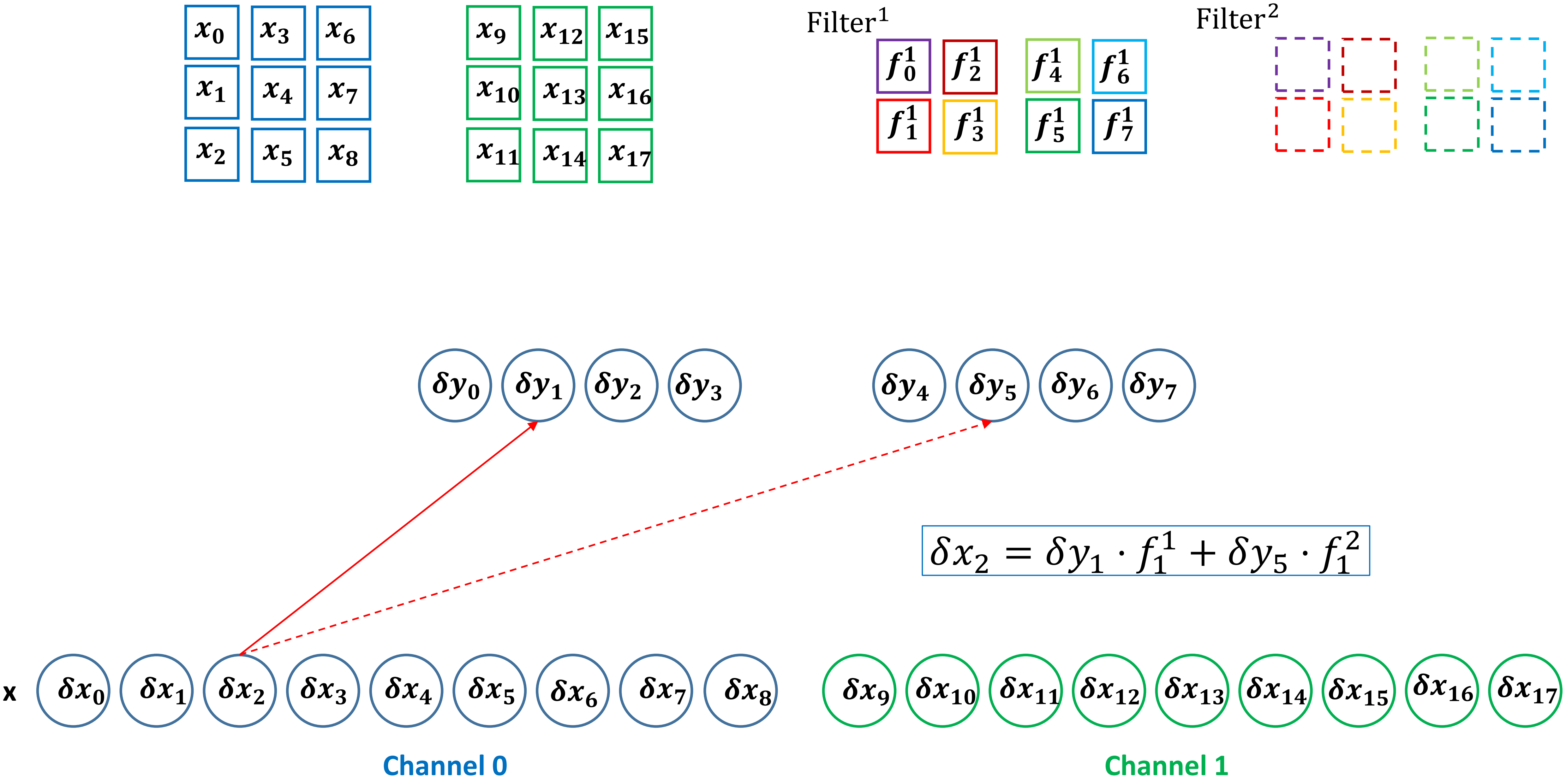
$$\frac{\partial y}{\partial x_0} = [f_0^1, 0{,}0{,}0, f_0^2, 0{,}0{,}0]$$

If we multiply (elementwise), we get:

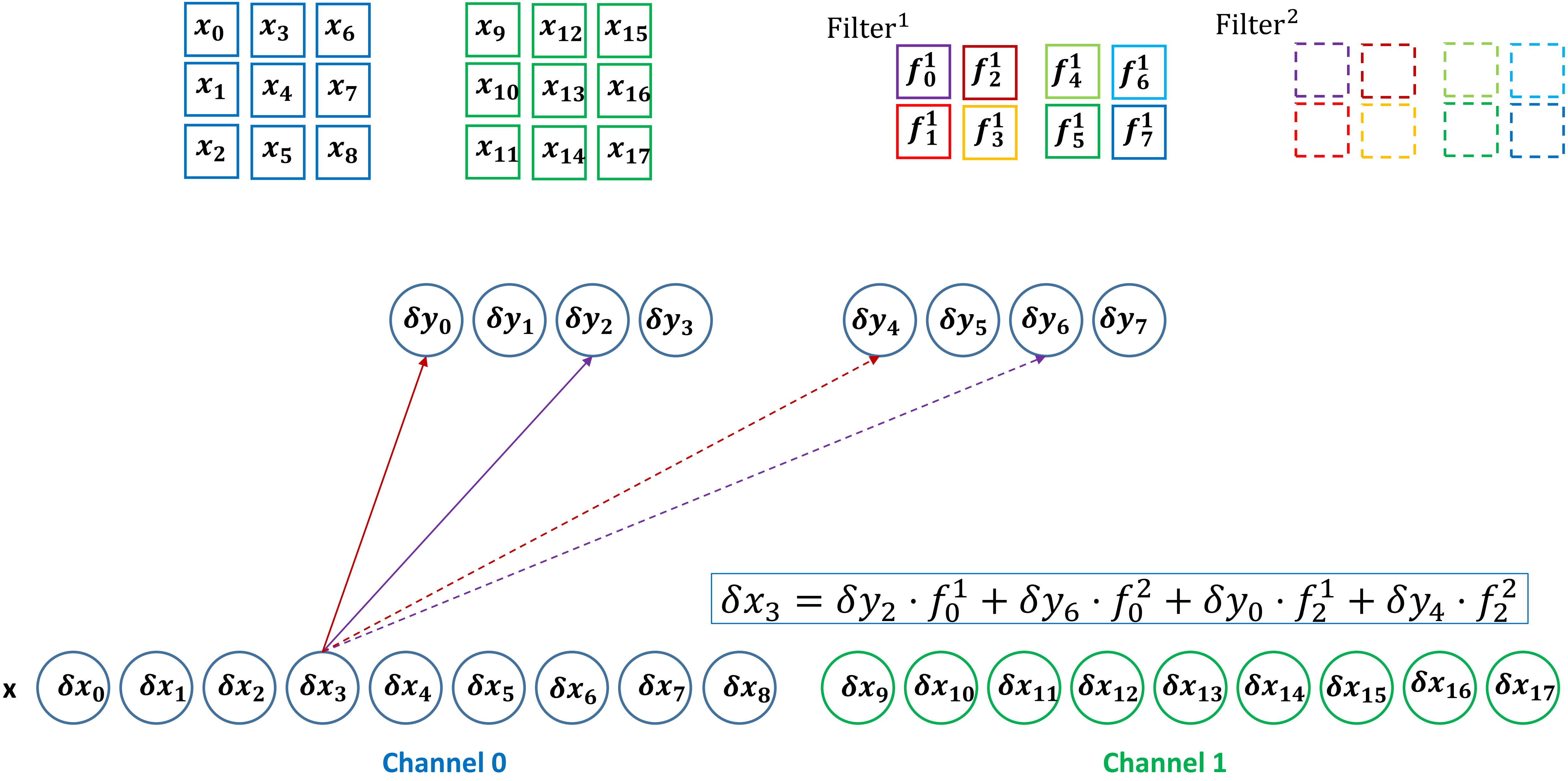
$$\delta x_0 = \frac{\partial L}{\partial x_0} = \delta y_0 \cdot f_0^1 + \delta y_4 \cdot f_0^2$$



$$\delta x_1 = \delta y_0 \cdot f_1^1 + \delta y_1 \cdot f_0^1 + \delta y_4 \cdot f_1^2 + \delta_{y_5} \cdot f_0^2$$



x_0 x_3 x_6 x_1 x_4 x_7 x_2 x_5 x_8
x_9 x_{12} x_{15} x_{10} x_{13} x_{16} x_{11} x_{14} x_{17}
Filter^1
f_0^1 f_2^1 f_1^1 f_3^1 f_4^1 f_6^1 f_5^1 f_7^1
Filter^2
δy_0 δy_1 δy_2 δy_3
δy_4 δy_5 δy_6 δy_7
$\delta x_2 = \delta y_1 \cdot f_1^1 + \delta y_5 \cdot f_1^2$
x
δx_0 δx_1 δx_2 δx_3 δx_4 δx_5 δx_6 δx_7 δx_8
δx_9 δx_{10} δx_{11} δx_{12} δx_{13} δx_{14} δx_{15} δx_{16} δx_{17}
Channel 0
Channel 1



$$\delta x_3 = \delta y_2 \cdot f_0^1 + \delta y_6 \cdot f_0^2 + \delta y_0 \cdot f_2^1 + \delta y_4 \cdot f_2^2$$

x_0	x_3	x_6
x_1	x_4	x_7
x_2	x_5	x_8

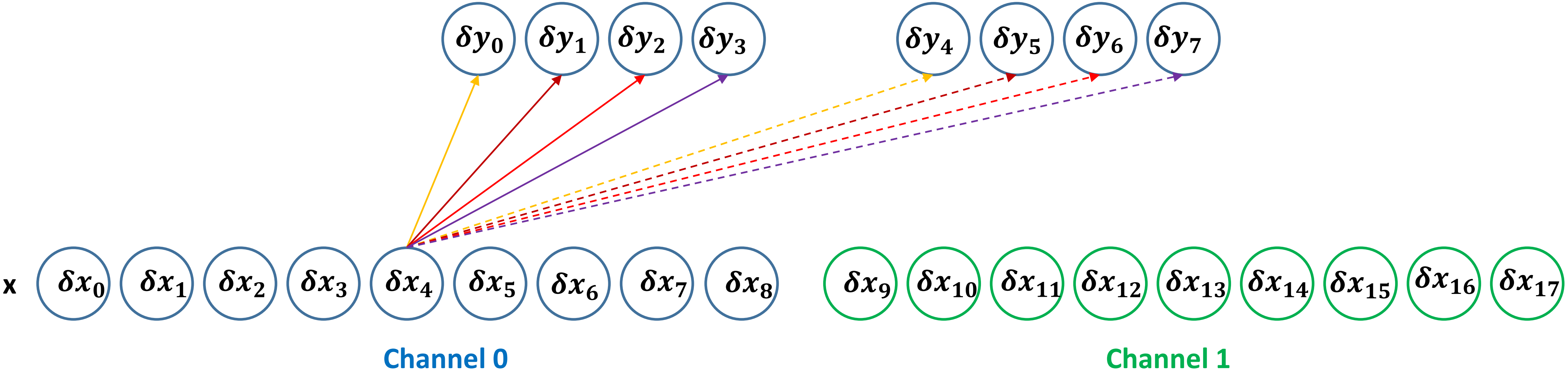
x_9	x_{12}	x_{15}
x_{10}	x_{13}	x_{16}
x_{11}	x_{14}	x_{17}

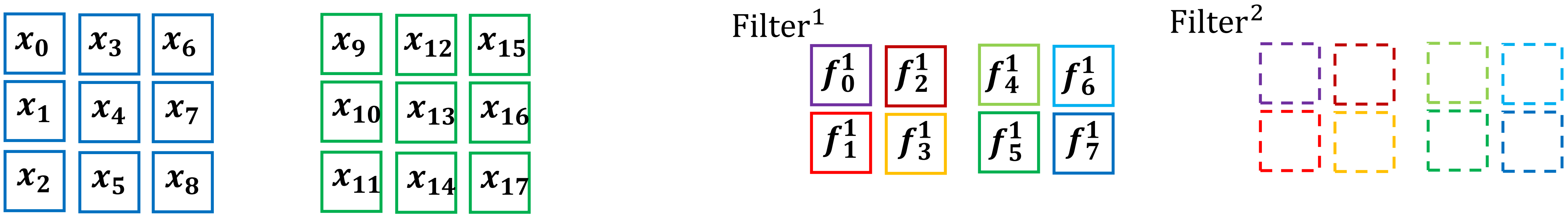
Filter^1

f_0^1	f_2^1	f_4^1	f_6^1
f_1^1	f_3^1	f_5^1	f_7^1

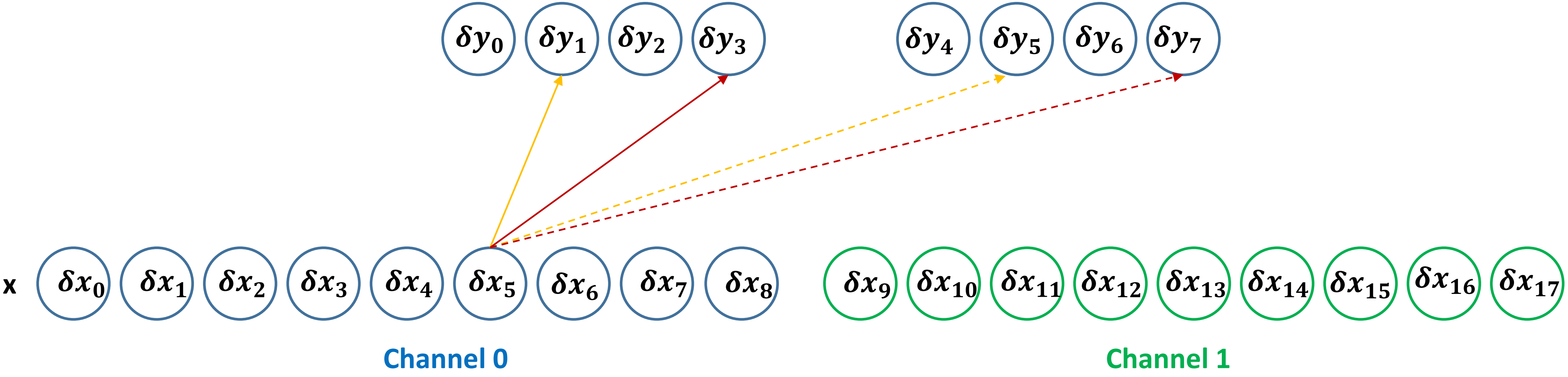
Filter^2

$$\delta x_4 = \delta y_0 \cdot f_3^1 + \delta y_1 \cdot f_2^1 + \delta y_2 \cdot f_1^1 + \delta y_3 \cdot f_3^1 + \delta y_4 \cdot f_3^2 + \delta y_5 \cdot f_2^2 + \delta y_6 \cdot f_1^2 + \delta y_7 \cdot f_0^2$$



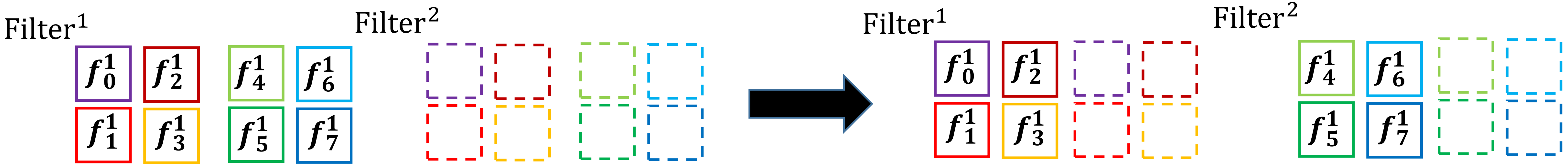

x_0 x_3 x_6
x_1 x_4 x_7
x_2 x_5 x_8
x_9 x_{12} x_{15}
x_{10} x_{13} x_{16}
x_{11} x_{14} x_{17}
Filter^1
f_0^1 f_2^1 f_4^1 f_6^1
f_1^1 f_3^1 f_5^1 f_7^1
Filter^2

$$\delta x_5 = ???$$


δy_0 δy_1 δy_2 δy_3 δy_4 δy_5 δy_6 δy_7
x
δx_0 δx_1 δx_2 δx_3 δx_4 δx_5 δx_6 δx_7 δx_8
δx_9 δx_{10} δx_{11} δx_{12} δx_{13} δx_{14} δx_{15} δx_{16} δx_{17}
Channel 0
Channel 1

Observations (1):

- For the first channel of the input, we only need the first channel of the filters!



- In general this means we transpose the depth with the amount of filters $[x, y, c, f] \Rightarrow [x, y, f, c]$

Observations (2):

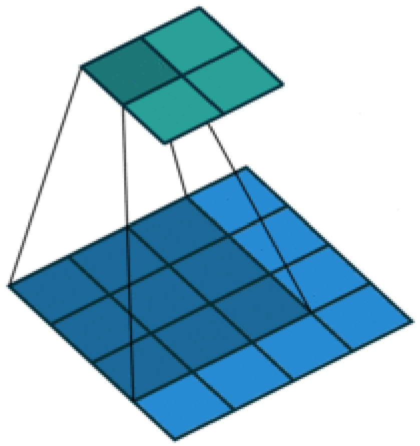
- A regular convolution shrinks the input!

➔ But to obtain the deltas of the input we need to increase the size of the output

- Captain padding to the rescue! We differentiate three forms of padding

Observations (2): Padding

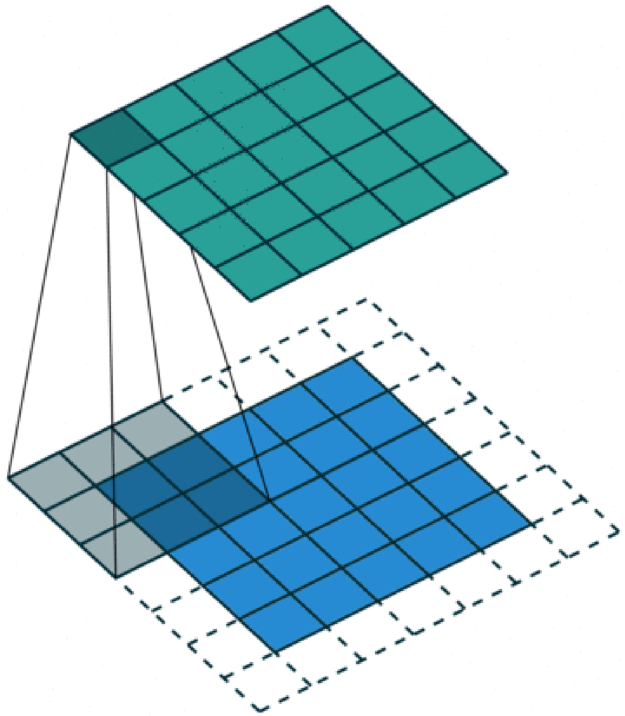
- No Padding (p=0):



➔ The output image is smaller than the input image

Observations (2): Padding

- Half Padding $p_x = \left\lfloor \frac{k_x}{2} \right\rfloor, p_y = \left\lfloor \frac{k_y}{2} \right\rfloor$:



➔Keeps the output image at the same size as the input

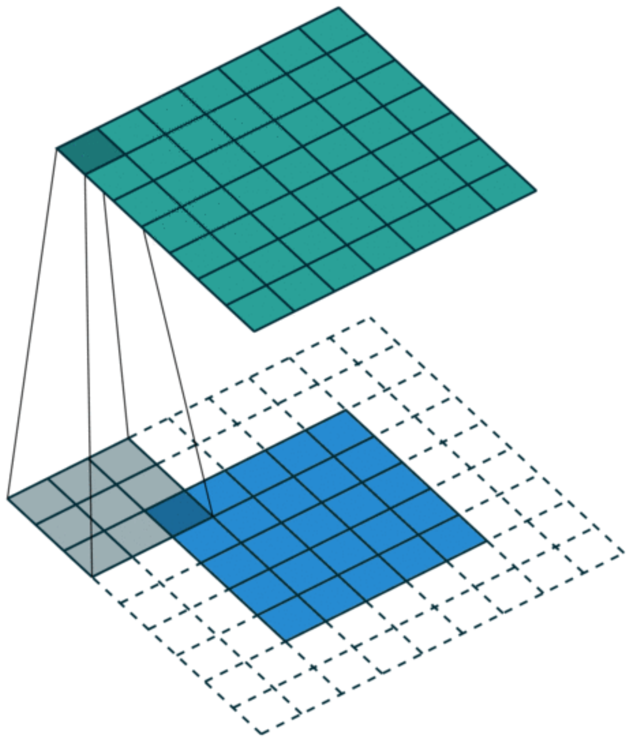
$$o_x = (x + 2p_x) - (k_x - 1)$$
$$o_y = (y + 2p_y) - (k_y - 1)$$

Observations (2): Padding

- Full Padding $p_x = k_x - 1$:

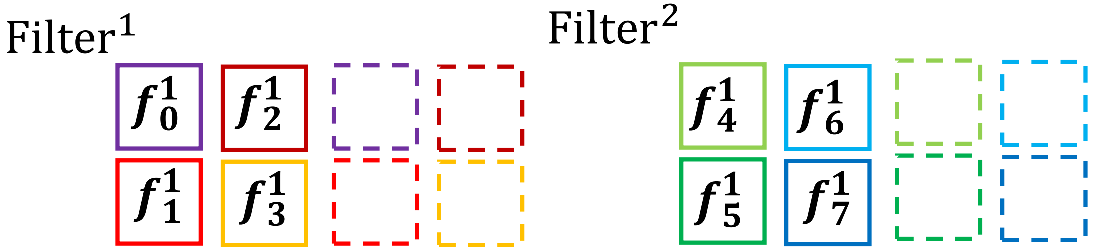
➔Increases the size of the output image

➔We call a convolution with full padding „Full Convolution“

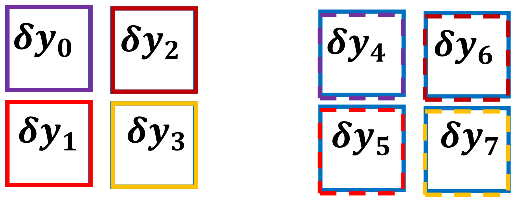


$$o_x = (x + 2p_x) - (k_x - 1)$$
$$o_y = \left(y + 2p_y\right) - (k_y - 1)$$

Observations (3):



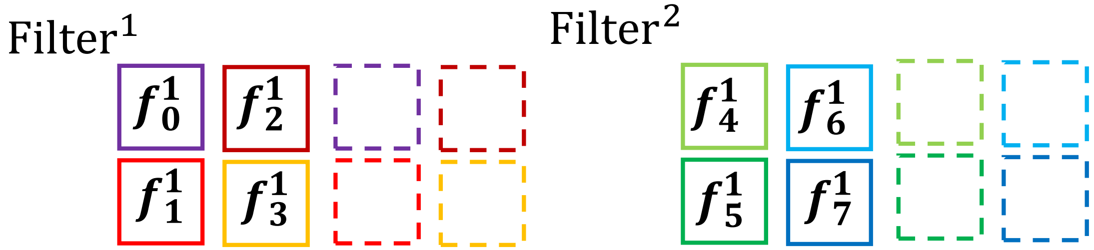
- We need a rotated form of the filters! A regular convolution wont cut it!



- We would get: $\boxed{\delta x_4 = \delta y_0 \cdot f_0^1 + \delta y_1 \cdot f_1^1 + \cdots}$

- But we need: $\boxed{\delta x_4 = \delta y_0 \cdot f_3^1 + \delta y_1 \cdot f_2^1 \ldots}$

Observations (3):



- Rotate the filter in x-y axis by 180 degrees:

Backward: Convolution2D

- The backward pass of a convolution layer is more tricky!

$$\delta X = \delta Y \ *_F \, rot_{x,y}^{180}(trans_{0,1,3,2}(F))$$

- With δX being the deltas of the input tensor, F the kernel tensor, δY the deltas of the output tensor and $*_F$ denotes the full convolution operator for images with a depth

- The backward pass is also called „**Transposed Convolution**“

Convolution-Example: Weight Updates

We got access to: Input [3,3,2]

x_0	x_3	x_6
x_1	x_4	x_7
x_2	x_5	x_8

Channel 0

x_9	x_{12}	x_{15}
x_{10}	x_{13}	x_{16}
x_{11}	x_{14}	x_{17}

Channel 1

And:Deltas of the Output [2,2,2]

δy_0	δy_2	δy_4	δy_6
δy_1	δy_3	δy_5	δy_7

And need an equation for our filter weights [2,2,2,2]

Filter^1

f_0^1	f_2^1	f_4^1	f_6^1
f_1^1	f_3^1	f_5^1	f_7^1

Filter^2

➔ Somehow we need to get an extra dimension to our output!

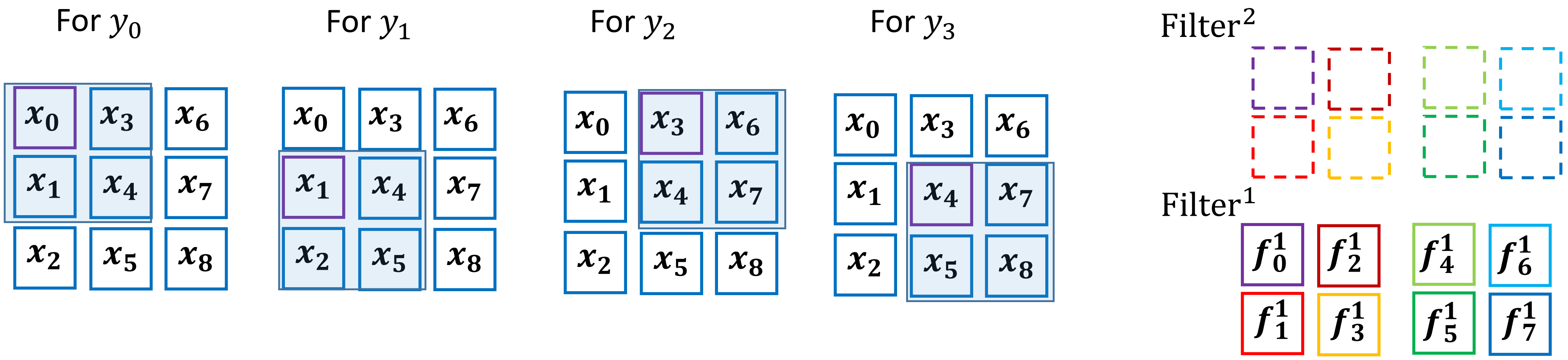
Weight Updates

$$\frac{\partial L}{\partial f_j} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial f_j}$$

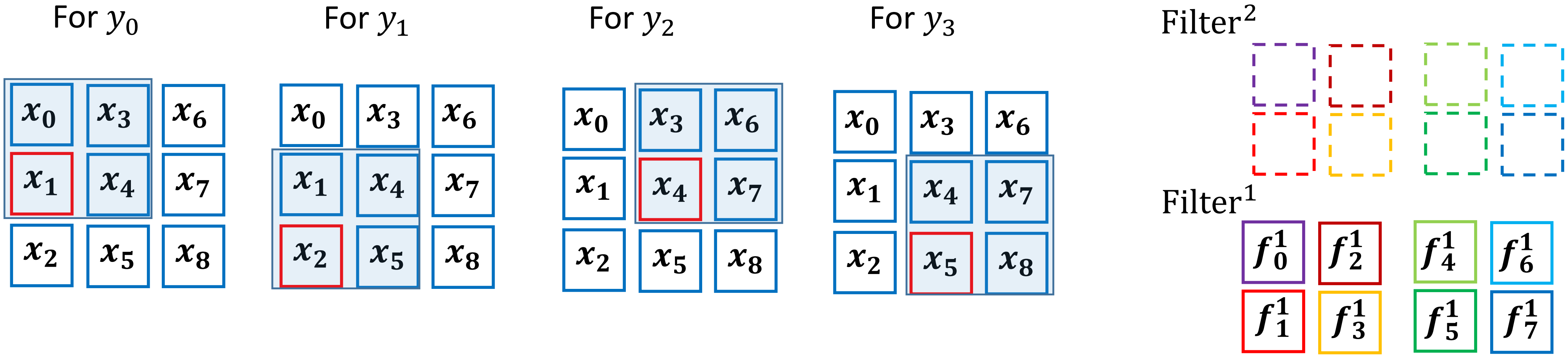
- But since our f_j influences many y‘s („weight sharing“) we need to sum:

$$\frac{\partial L}{\partial f_j} = \sum_i \frac{\partial L}{\partial y_i} \cdot \frac{\partial y_i}{\partial f_j}$$

$$\frac{\partial y}{\partial f_0^1} = \delta y_0 x_0 + \delta y_1 x_1 + \delta y_2 x_3 + \delta y_3 x_4$$



$$\frac{\partial y}{\partial f_1^1} = \delta y_0 x_1 + \delta y_1 x_2 + \delta y_2 x_4 + \delta y_3 x_5$$



Convolution-Example: Weight Updates

Filter[1]

$$\begin{array}{cc} \boxed{f_0^1} & \boxed{f_2^1} \\ \boxed{f_1^1} & \boxed{f_3^1} \end{array}$$

- In general, the equations for the first channel of the first filter:

$$\frac{\partial y}{\partial f_0^1} = \delta y_0 x_0 + \delta y_1 x_1 + \delta y_2 x_3 + \delta y_3 x_4$$
$$\frac{\partial y}{\partial f_1^1} = \delta y_0 x_1 + \delta y_1 x_2 + \delta y_2 x_4 + \delta y_3 x_5$$
$$\frac{\partial y}{\partial f_2^1} = \delta y_0 x_3 + \delta y_1 x_4 + \delta y_2 x_6 + \delta y_3 x_7$$
$$\frac{\partial y}{\partial f_3^1} = \delta y_0 x_4 + \delta y_1 x_5 + \delta y_2 x_7 + \delta y_3 x_8$$

➔ For the weight updates of the first channel of the first filter, we need:

- The first channel of δy as well as the first channel of x

Convolution-Example: Weight Updates

Filter[1]

f_4^1	f_6^1
f_5^1	f_7^1

- In general, the equations for the second channel of the first filter:

$$\frac{\partial y}{\partial f_0^1} = \delta y_0 x_9 + \delta y_1 x_{10} + \delta y_2 x_{12} + \delta y_3 x_{13}$$

$$\frac{\partial y}{\partial f_1^1} = \delta y_0 x_{10} + \delta y_1 x_{11} + \delta y_2 x_{13} + \delta y_3 x_{14}$$

$$\frac{\partial y}{\partial f_2^1} = \delta y_0 x_{12} + \delta y_1 x_{13} + \delta y_2 x_{15} + \delta y_3 x_{16}$$

$$\frac{\partial y}{\partial f_3^1} = \delta y_0 x_{13} + \delta y_1 x_{14} + \delta y_2 x_{16} + \delta y_3 x_{17}$$

➔ For the weight updates of the second channel of the first filter, we need:

- The first channel of δy as well as the second channel of x

Convolution-Example: Weight Updates

- So we get:

Filter 1, Channel 1 ➔ $\mathrm{x}[\mathrm{Channel}_1] * \delta y[Channel_1]$
Filter 1, Channel 2 ➔ $\mathrm{x}[\mathrm{Channel}_2] * \delta y[Channel_1]$

Filter 2, Channel 1 ➔ $\mathrm{x}[\mathrm{Channel}_1] * \delta y[Channel_2]$
Filter 2, Channel 2 ➔ $\mathrm{x}[\mathrm{Channel}_2] * \delta y[Channel_2]$

Convolution-Example: Weight Updates

In our case this results in a shape of [2,2]

- So we get:

Filter 1, Channel 1 ➔ $\mathrm{x}[\mathrm{Channel}_1] * \delta y[Channel_1]$
Filter 1, Channel 2 ➔ $\mathrm{x}[\mathrm{Channel}_2] * \delta y[Channel_1]$

Filter 2, Channel 1 ➔ $\mathrm{x}[\mathrm{Channel}_1] * \delta y[Channel_2]$
Filter 2, Channel 2 ➔ $\mathrm{x}[\mathrm{Channel}_2] * \delta y[Channel_2]$

Convolution-Example: Weight Updates

2 of those per filter

- So we get:

Filter 1, Channel 1 ➔ x[Channel_1] $*\ \delta y[Channel_1]$
Filter 1, Channel 2 ➔ x[Channel_2] $*\ \delta y[Channel_1]$

Filter 2, Channel 1 ➔ x[Channel_1] $*\ \delta y[Channel_2]$
Filter 2, Channel 2 ➔ x[Channel_2] $*\ \delta y[Channel_2]$

Convolution-Example: Weight Updates

- So we get:

Filter 1, Channel 1 ➔ $\mathrm{x}[\mathrm{Channel}_1] \ * \ \delta y[Channel_1]$
Filter 1, Channel 2 ➔ $\mathrm{x}[\mathrm{Channel}_2] \ * \ \delta y[Channel_1]$

And 2 filter

Filter 2, Channel 1 ➔ $\mathrm{x}[\mathrm{Channel}_1] \ * \ \delta y[Channel_2]$
Filter 2, Channel 2 ➔ $\mathrm{x}[\mathrm{Channel}_2] \ * \ \delta y[Channel_2]$

➔ In total this results in a shape of [2,2,2,2], exactly as we need!

➔ Let us introduce $*_{ch}$ for „channelwise“ convolution

Convolution-Layer: Equations

- Forward:

$$Y = X \ * F + \mathrm{bias}$$

- Backward (transposed convolution):

$$\delta X = \delta Y \ *_F rot_{x,y}^{180}(trans_{0,1,3,2}(F))$$

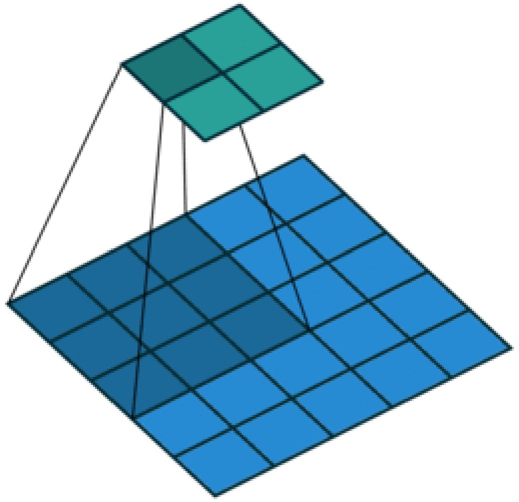
- Weight Updates:

$$\frac{\partial L}{\partial f} = X \ *_{ch} \ \delta Y$$

$$\frac{\partial L}{\partial bias_f} = \sum_i \delta y_{i,f=c} \qquad \text{(sum all } \delta y_i \text{ of the according channel)}$$

Even crazier convolutions - Strides

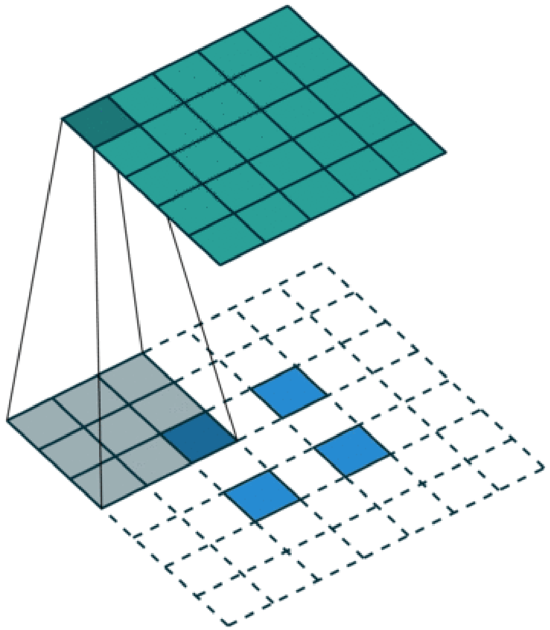
- So far we introduced padding, but there is more…
    - Stride („How much the filter is shifted per convolution, in x and y direction“)
    - Example: Stride-x and Stride-y are set to 2



Even crazier convolutions - Strides

- Dealing with strides in the backward pass is tricky!

- Introduces additional 0's in the output while performing the full convolution!



$$o_x = \frac{(x + 2p_x) - k_x}{stride_x} + 1$$

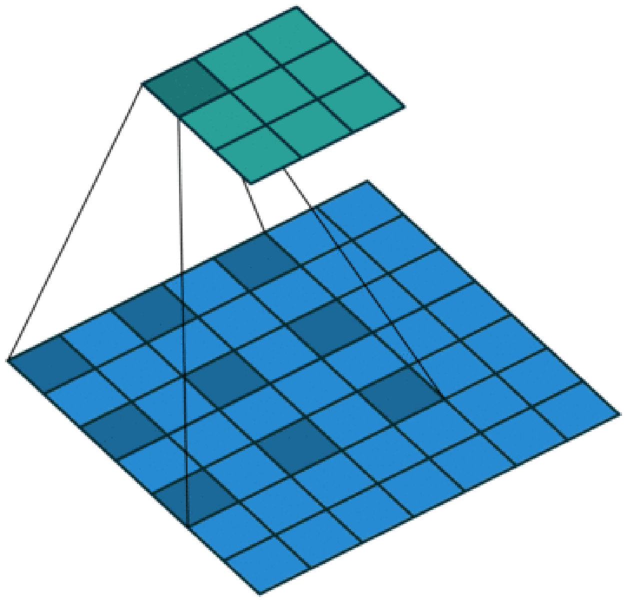
$$o_y = \frac{(y + 2p_y) - k_y}{stride_y}$$

Even crazier convolutions - Dilations

- A dilated convolution introduces additional (amount of d) 0‘s in the filter!

$$o_x = \left\lfloor \frac{(x + 2p_x) - k_x - (k_x - 1) \cdot (d_x - 1)}{stride_x} \right\rfloor + 1$$

$$o_y = \left\lfloor \frac{(y + 2p_y) - k_y - (k_y - 1) \cdot (d_y - 1)}{stride_y} \right\rfloor + 1$$



Things to pay attention to!

- Never blindly trust any Blog-Posts!

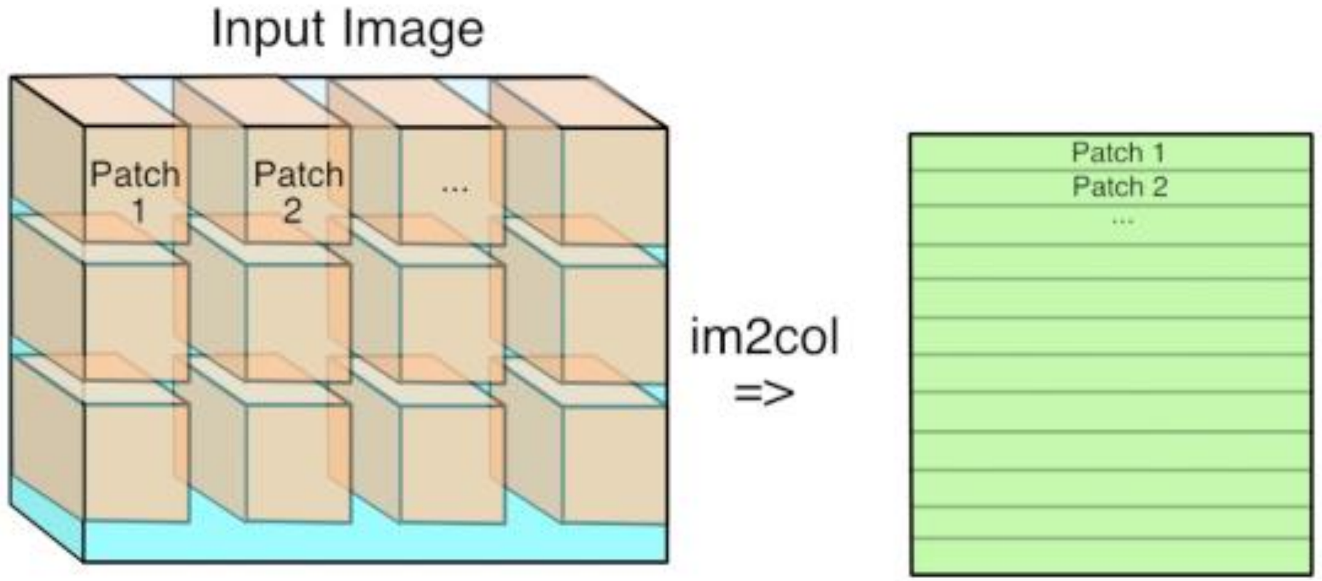
➔Many times convolution and cross correlation confused

➔Usually no filter/image depth respected, so you will not get the correct equations as we did

➔Best is to derive and verify the equations for yourself ☺

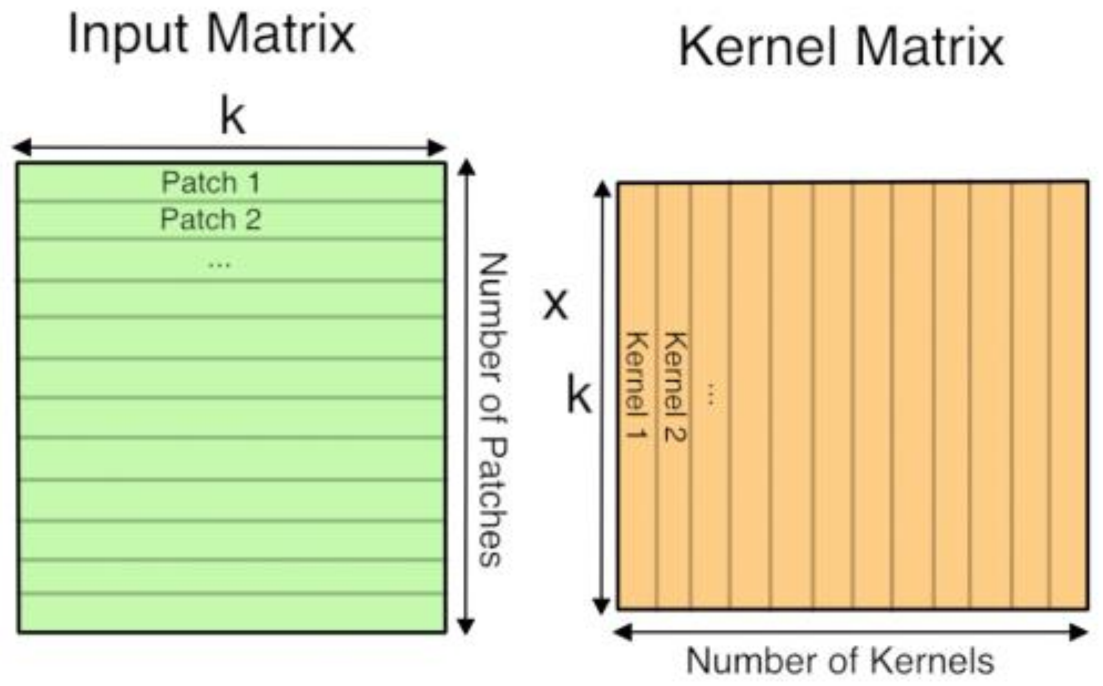
Faster Convolution

- Cast the convolution as Matrix-multiplication using the **Toeplitz** Matrix



Faster Convolution

- Multiply with an unrolled kernel-tensor (and reshape at the end)



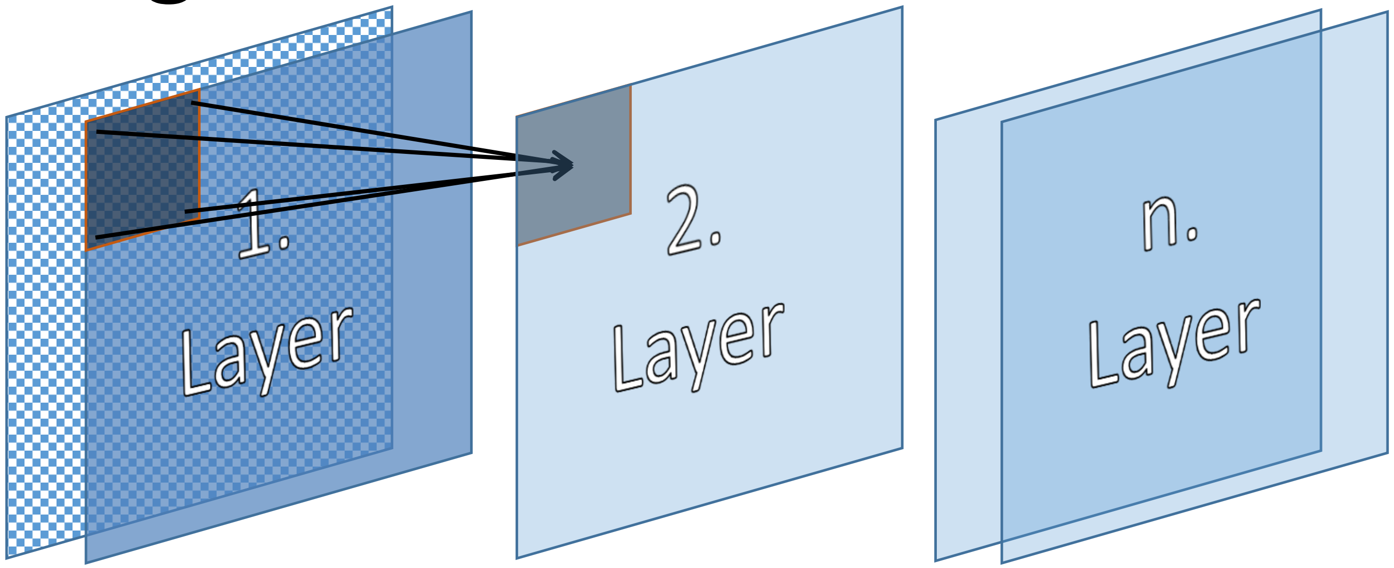
What you should be capable of

- Even if you do not implement it yourself, we expect you to:

1. Be able to calculate the output shape after the convolution
2. Perform the calculations of Y, given a kernel and an input x <u>by hand</u>
3. Peform the calculations of δX, given δY and a kernel <u>by hand</u>
4. Perform the weight updates of the kernel weights <u>by hand</u>

➔ Dont let your brain get convoluted by thinking in 4 dimensions!

Max-Pooling



$$\max \circ \begin{bmatrix} 4 & 1 & 1 & 3 & 3 & 4 \\ 8 & 6 & 0 & 5 & 2 & 2 \\ 0 & 6 & 9 & 5 & 1 & 8 \\ 4 & 8 & 5 & 7 & 4 & 2 \\ 9 & 8 & 5 & 9 & 3 & 0 \\ 3 & 6 & 7 & 4 & 7 & 5 \end{bmatrix} = \begin{bmatrix} 8 & 5 & 4 \\ 8 & 9 & 8 \\ 9 & 9 & 7 \end{bmatrix}$$

Backpropagation and Max-Pooling

- To extend our previous framework, we will:
  - Create a new implementation of our Layer interface
  - Define its parameters as well as its:
    1. Forward Pass
    2. Backward Pass
    3. ~~Calculation of the Weight Derivatives~~ (No parameter)

Max-Pooling: Forward

- Allocate a mask of the size of the output: [0,0,…,0]
- Slide the filter through the image, and remember:
  - The index of the winning element of the input data, and store the index of the input element in the according entry in the mask!

  - Max-function:
    - $y = \max(values)$

    - Derivative:

    $\delta x_i = \{\delta y, \text{if } x_i \text{ was max, else } 0\}$

Max-Pooling: Forward-Example

$$\max \circ \begin{bmatrix} 4 & 1 & 3 & 3 \\ 8 & 6 & 5 & 2 \\ 9 & 8 & 9 & 3 \\ 3 & 6 & 4 & 7 \end{bmatrix} = \begin{bmatrix} 8 & 5 \\ 9 & 9 \end{bmatrix}$$

Mask: [0,0,0,0]

Max-Pooling: Forward-Example

$$\max \circ \begin{bmatrix} 4 & 1 & 3 & 3 \\ 8 & 6 & 5 & 2 \\ 9 & 8 & 9 & 3 \\ 3 & 6 & 4 & 7 \end{bmatrix} = \begin{bmatrix} 8 & 5 \\ 9 & 9 \end{bmatrix}$$

Mask: [1,0,0,0]

Max-Pooling: Forward-Example

$$\max \circ \begin{bmatrix} 4 & 1 & 3 & 3 \\ 8 & 6 & 5 & 2 \\ \color{red}{9} & 8 & 9 & 3 \\ 3 & 6 & 4 & 7 \end{bmatrix} = \begin{bmatrix} 8 & 5 \\ \color{red}{9} & 9 \end{bmatrix}$$

Mask: [1,2,0,0]

Max-Pooling: Forward-Example

$$\max \circ \begin{bmatrix} 4 & 1 & 3 & 3 \\ 8 & 6 & \color{red}{5} & 2 \\ 9 & 8 & 9 & 3 \\ 3 & 6 & 4 & 7 \end{bmatrix} = \begin{bmatrix} 8 & \color{red}{5} \\ 9 & 9 \end{bmatrix}$$

Mask: [1,2,9,0]

Max-Pooling: Forward-Example

$$\max \circ \begin{bmatrix} 4 & 1 & 3 & 3 \\ 8 & 6 & 5 & 2 \\ 9 & 8 & 9 & 3 \\ 3 & 6 & 4 & 7 \end{bmatrix} = \begin{bmatrix} 8 & 5 \\ 9 & 9 \end{bmatrix}$$

Mask: [1,2,9,10]

Max-Pooling: Backward-Example

$$\begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix} = \begin{bmatrix} \delta y_0 & \delta y_2 \\ \delta y_1 & \delta y_3 \end{bmatrix}$$

Mask: [1,2,9,10]

➔ Just use the mask!

Max-Pooling: Backward-Example

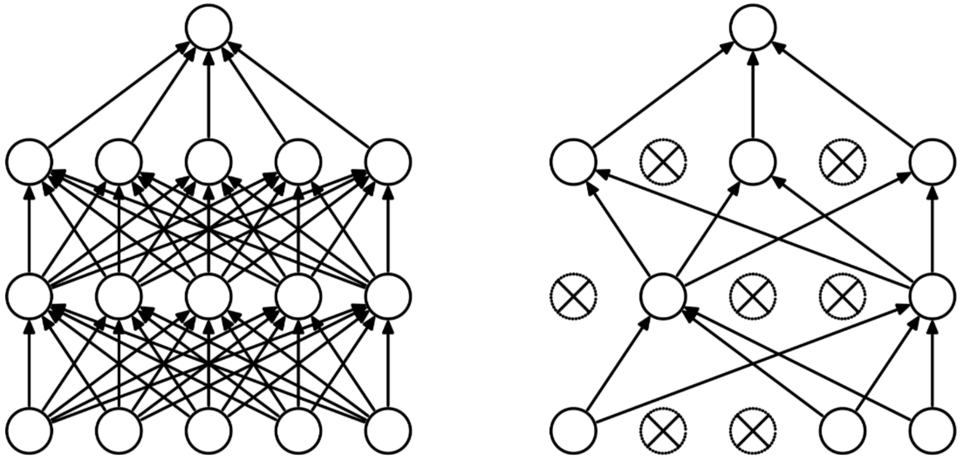
$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ \delta y_0 & 0 & \delta y_2 & 0 \\ \delta y_1 & 0 & \delta y_3 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} \delta y_0 & \delta y_2 \\ \delta y_1 & \delta y_3 \end{bmatrix}$$

Mask: [1,2,9,10]

➔ Just use the mask! Easiest backward of your life

Further Utility Layer– Dropout

- Dropout: More recent method (2014)
- **Idea**: During **training**, „remove“ a portion $0 \leq p < 1$ of neurons from the net
  → Force the net to rely on many features instead of few!



(a) Standard Neural Net (b) After applying dropout.

Srivastava, Nitish ; Hinton, Geoffrey E ; Krizhevsky, Alex ; Sutskever, Ilya ; Salakhutdinov, Ruslan: Dropout: a simple way to prevent neural networks from overfitting.. In: *Journal of machine learning research*, 15 (2014), Nr. 1, S. 1929--1958

Even happier coding!