Lecture 2.A
Convolutional Networks

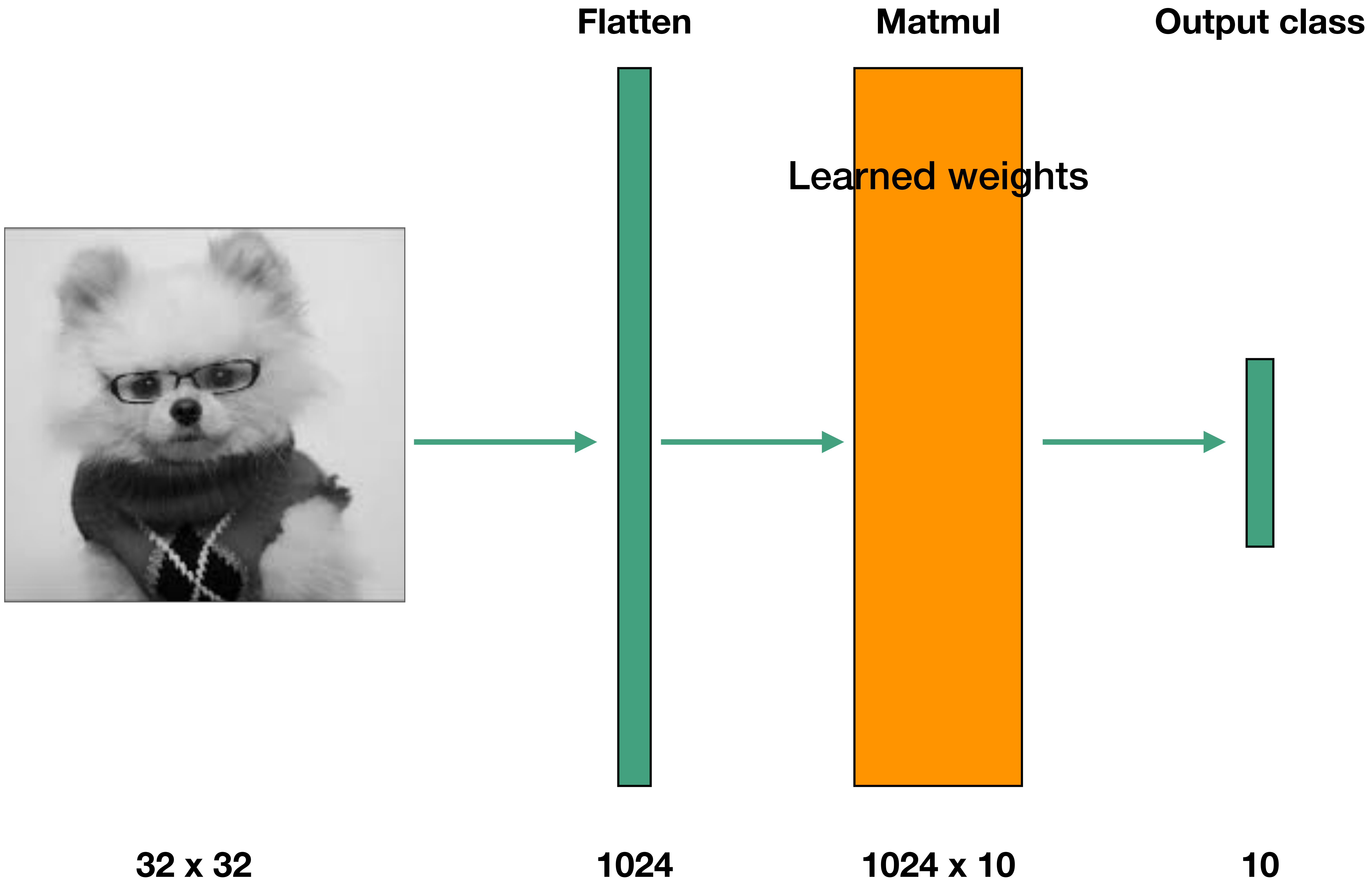
Agenda

1. **Review of the convolution operation**
2. Other important operations for ConvNets
3. Classic ConvNet architectures

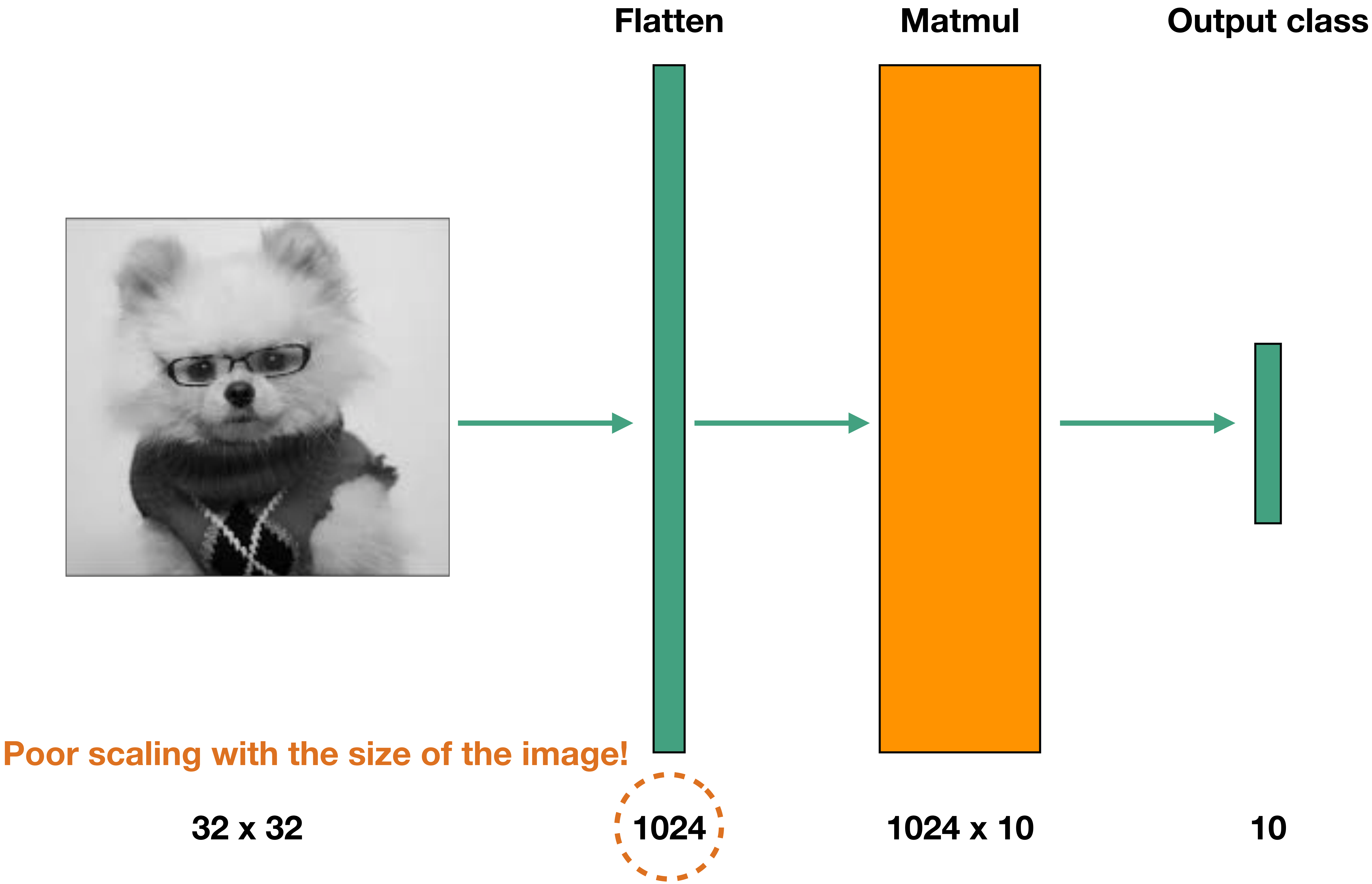
Review of convolutions

- **What's a convolutional filter?**
- Filter stacks and ConvNets
- Strides & padding
- Filter math
- Implementation notes

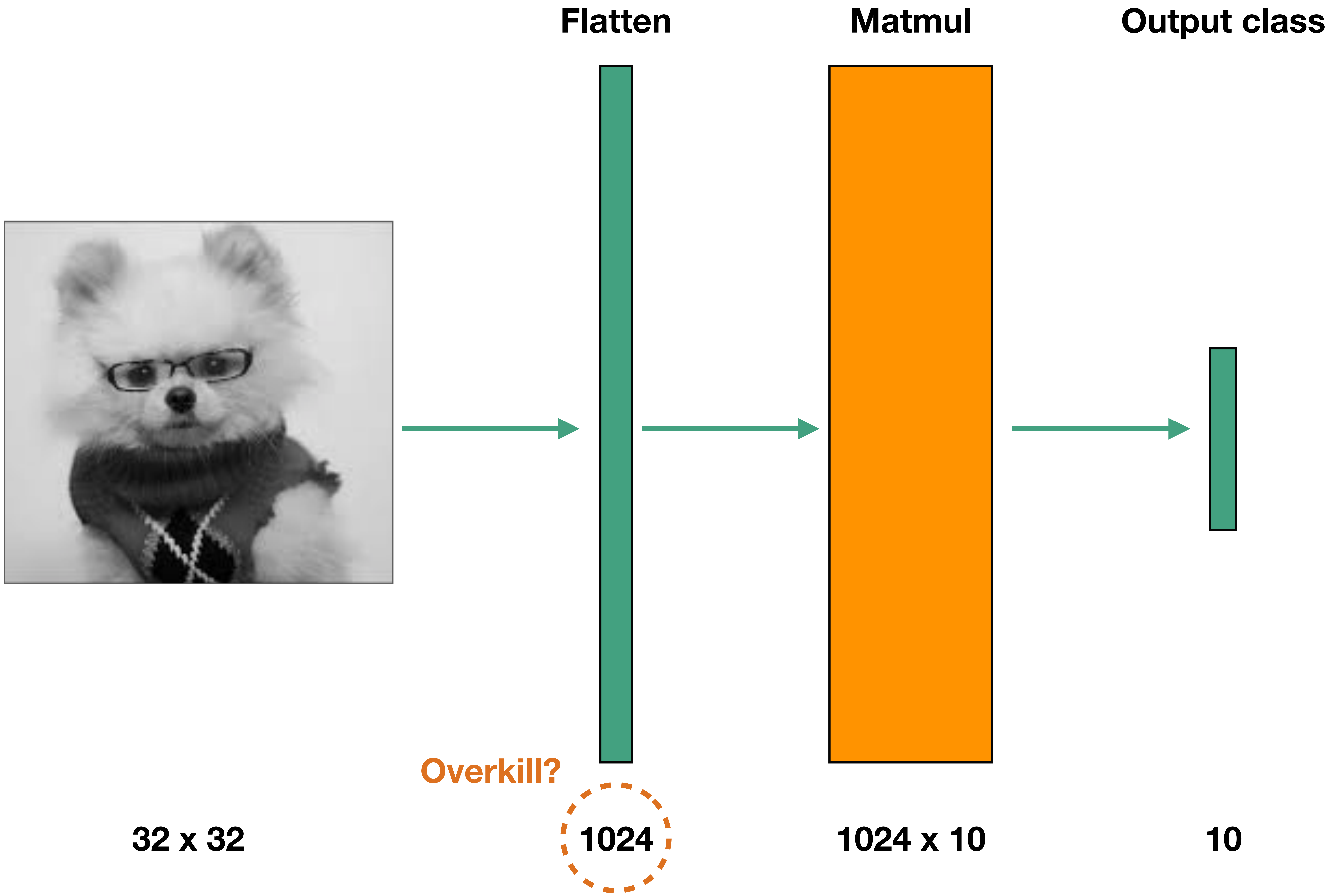
Why not use FC for images?



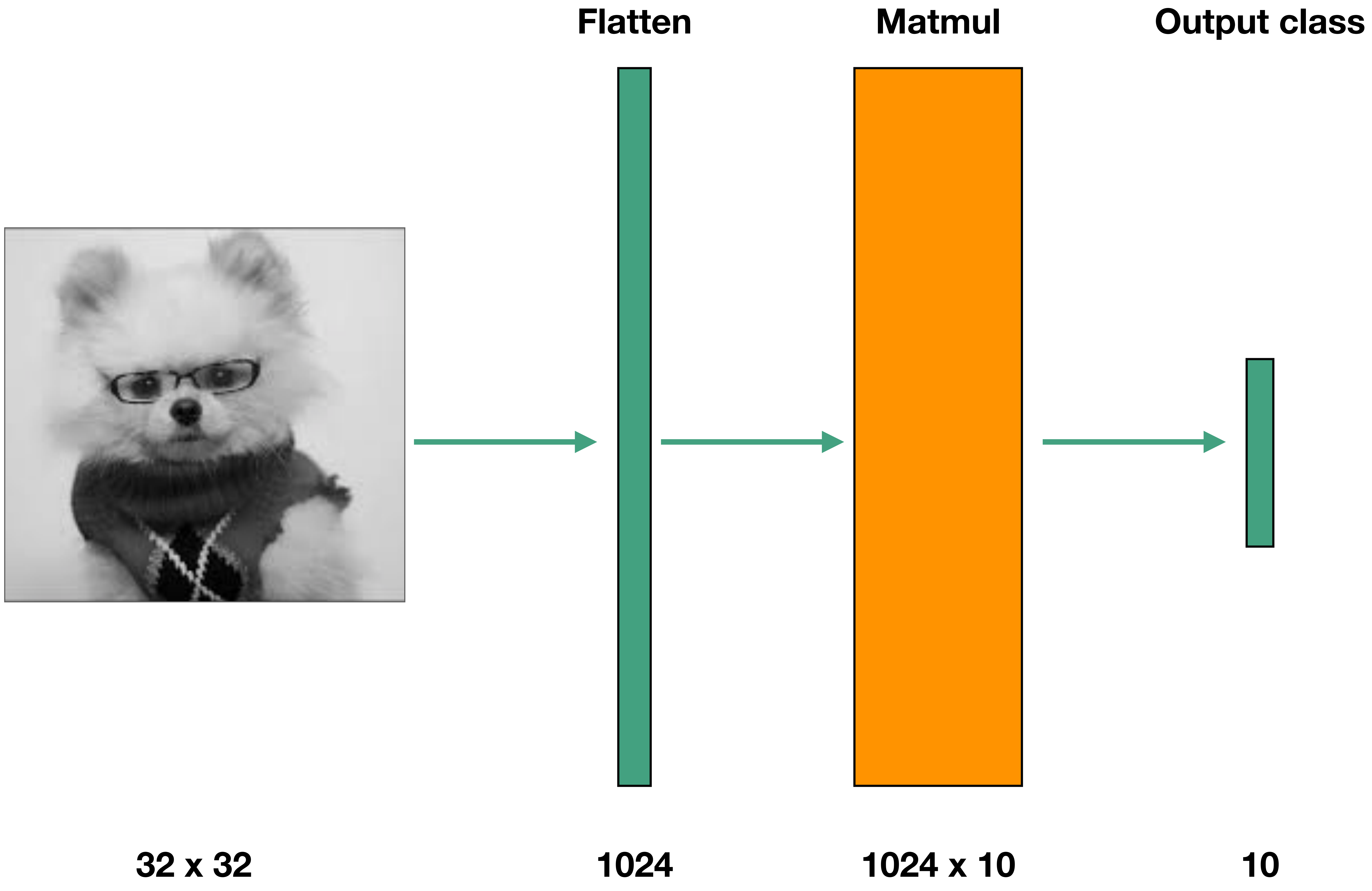
Why not use FC for images?



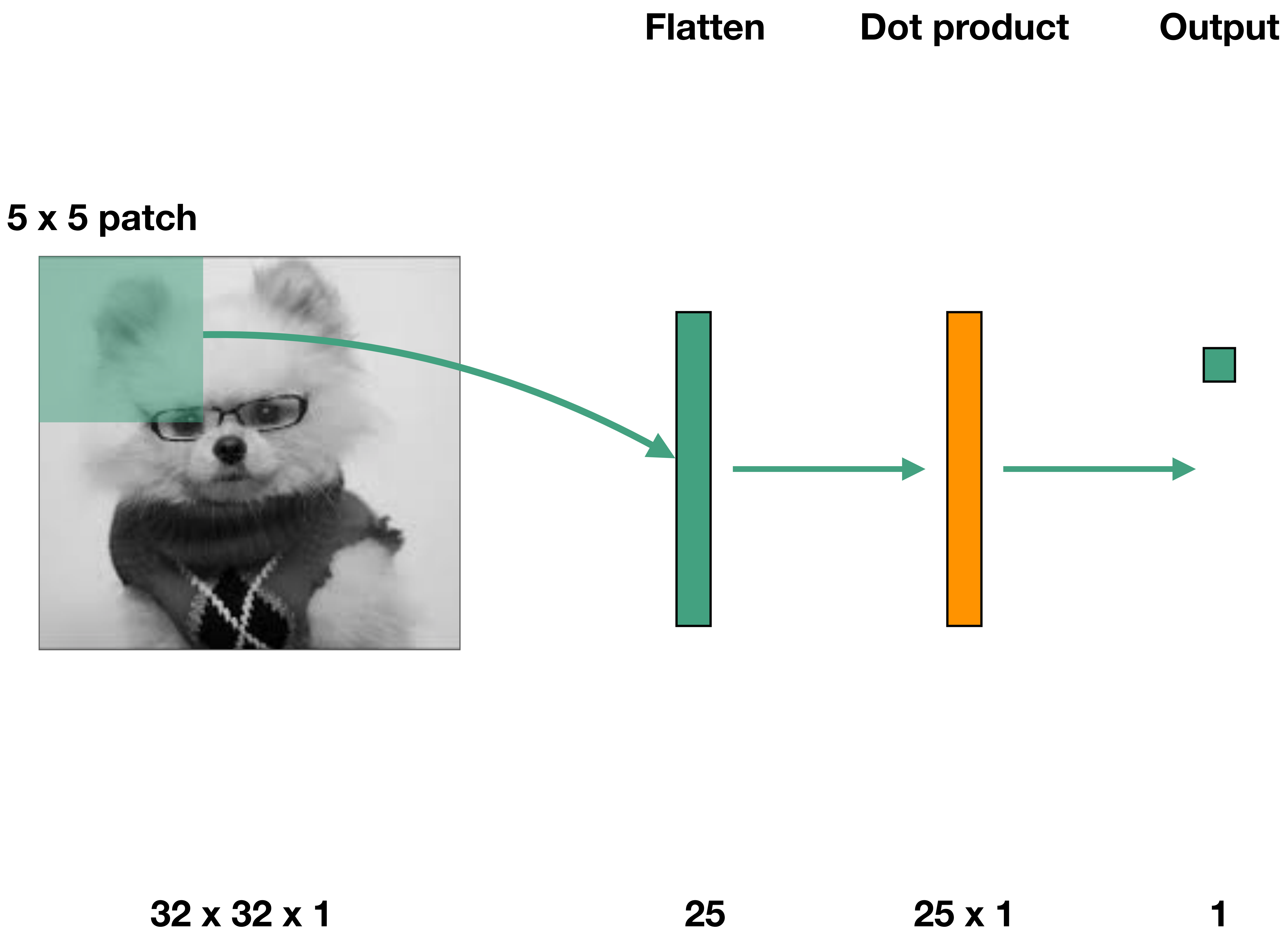
Why not use FC for images?



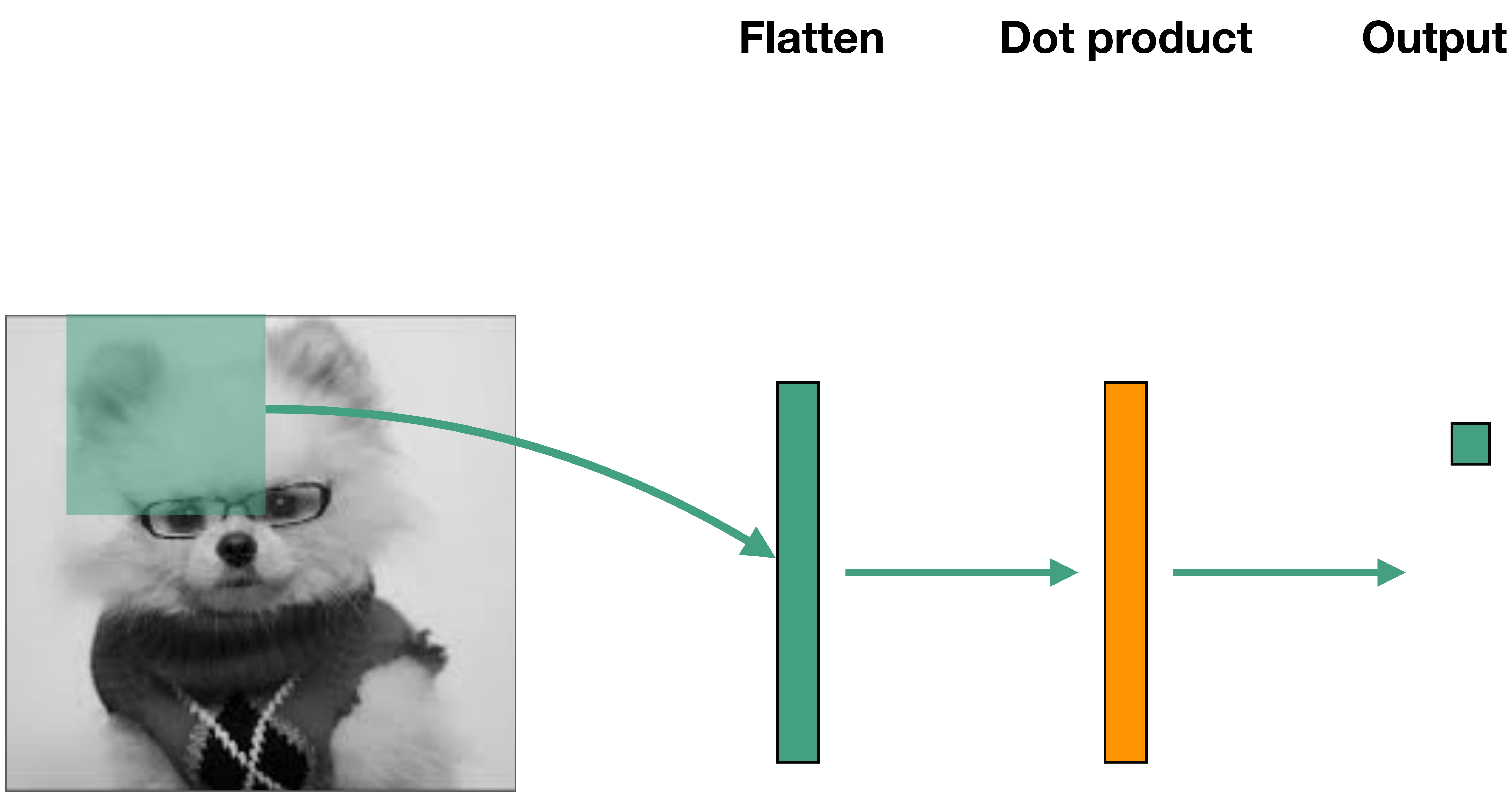
Why not use FC for images?



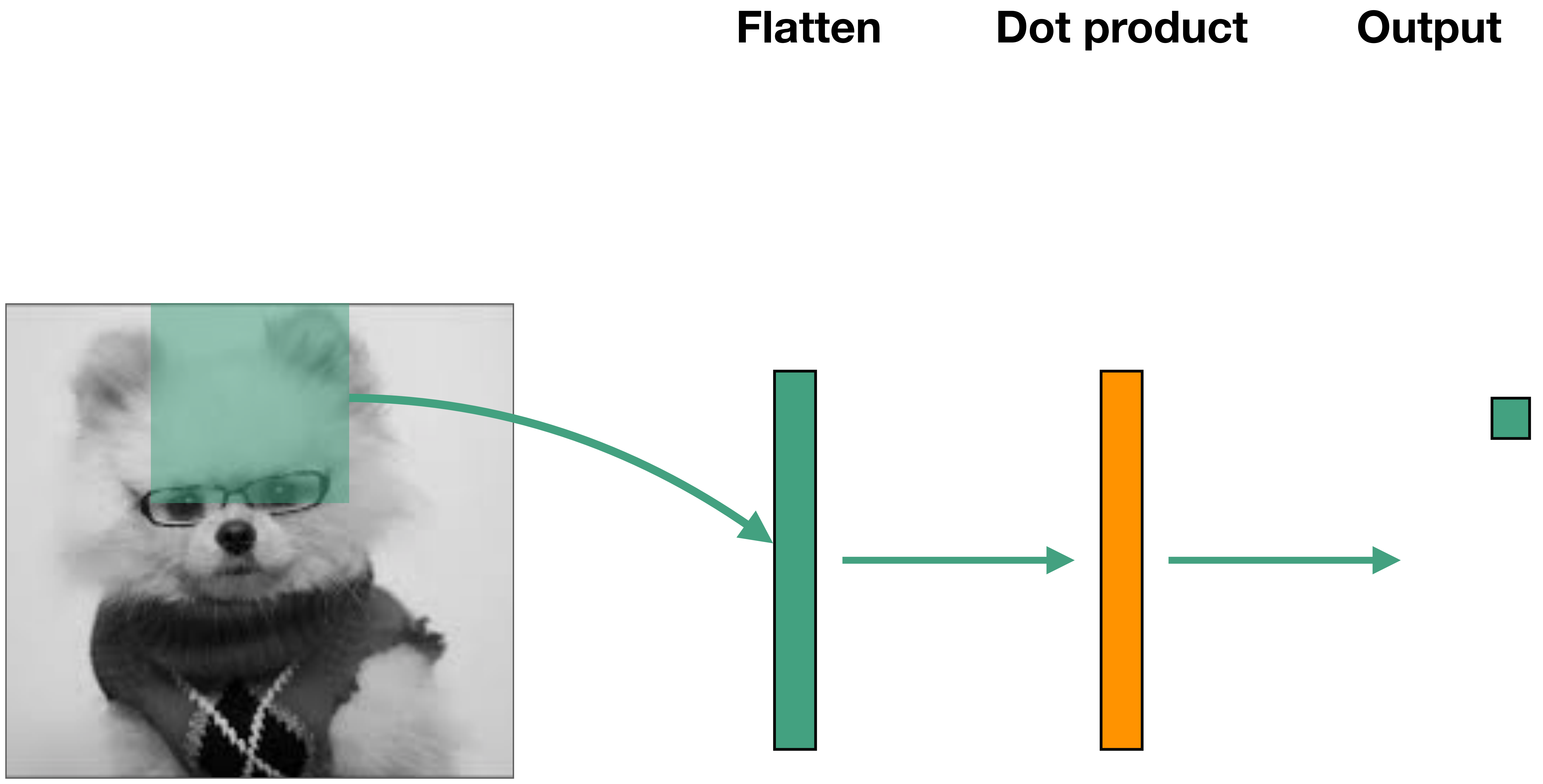
Convolutional filters



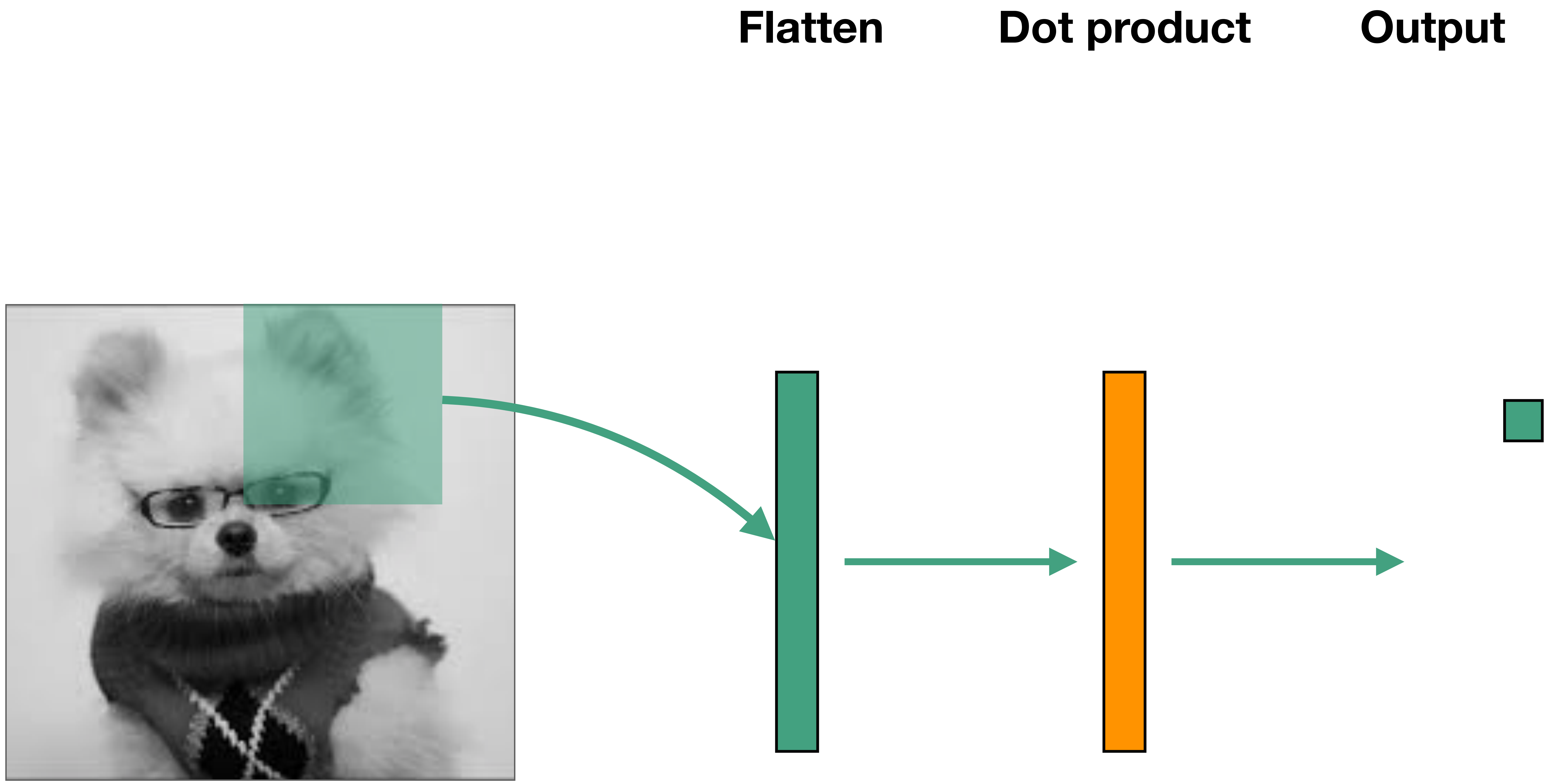
Convolutional filters



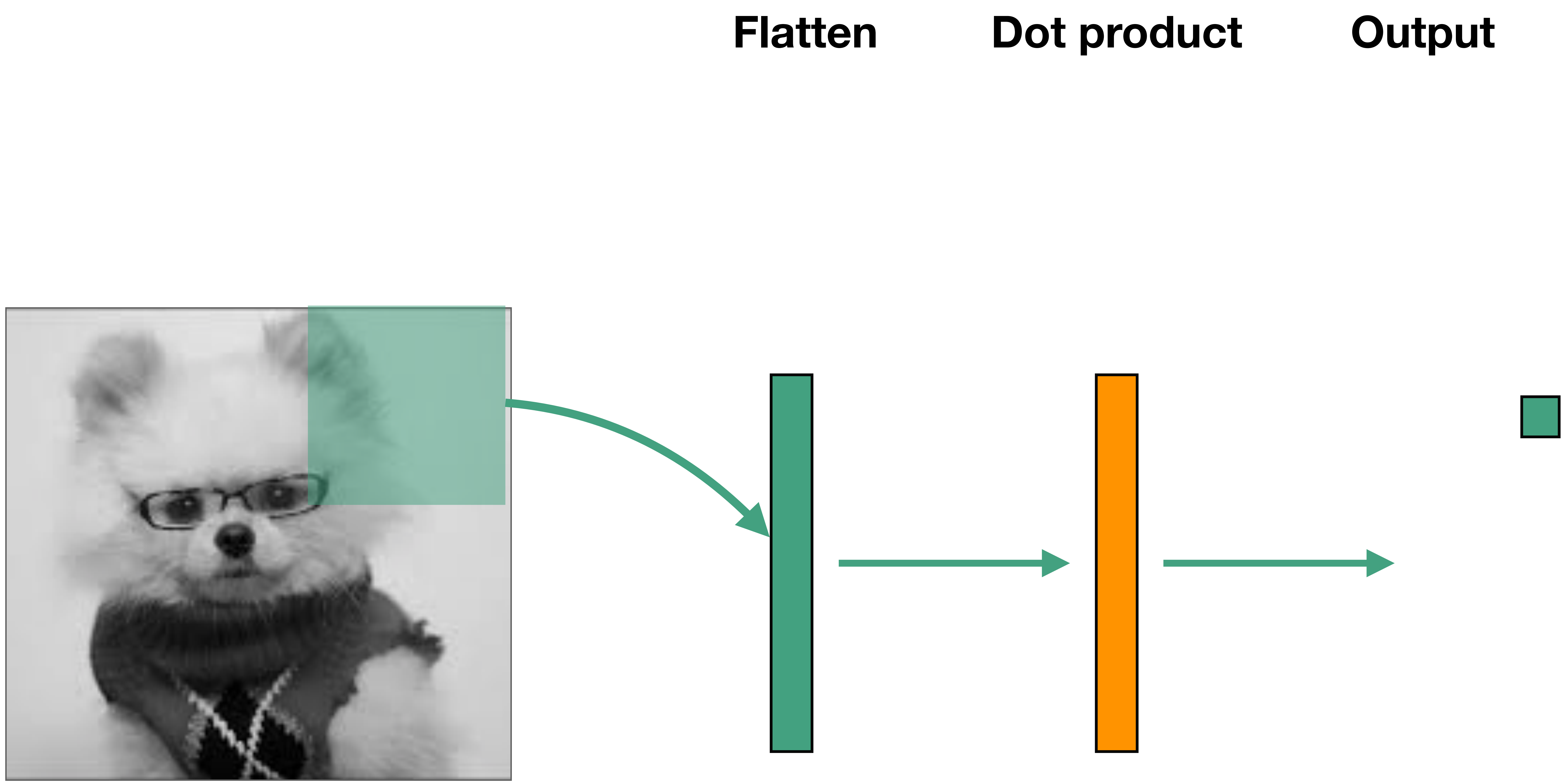
Convolutional filters



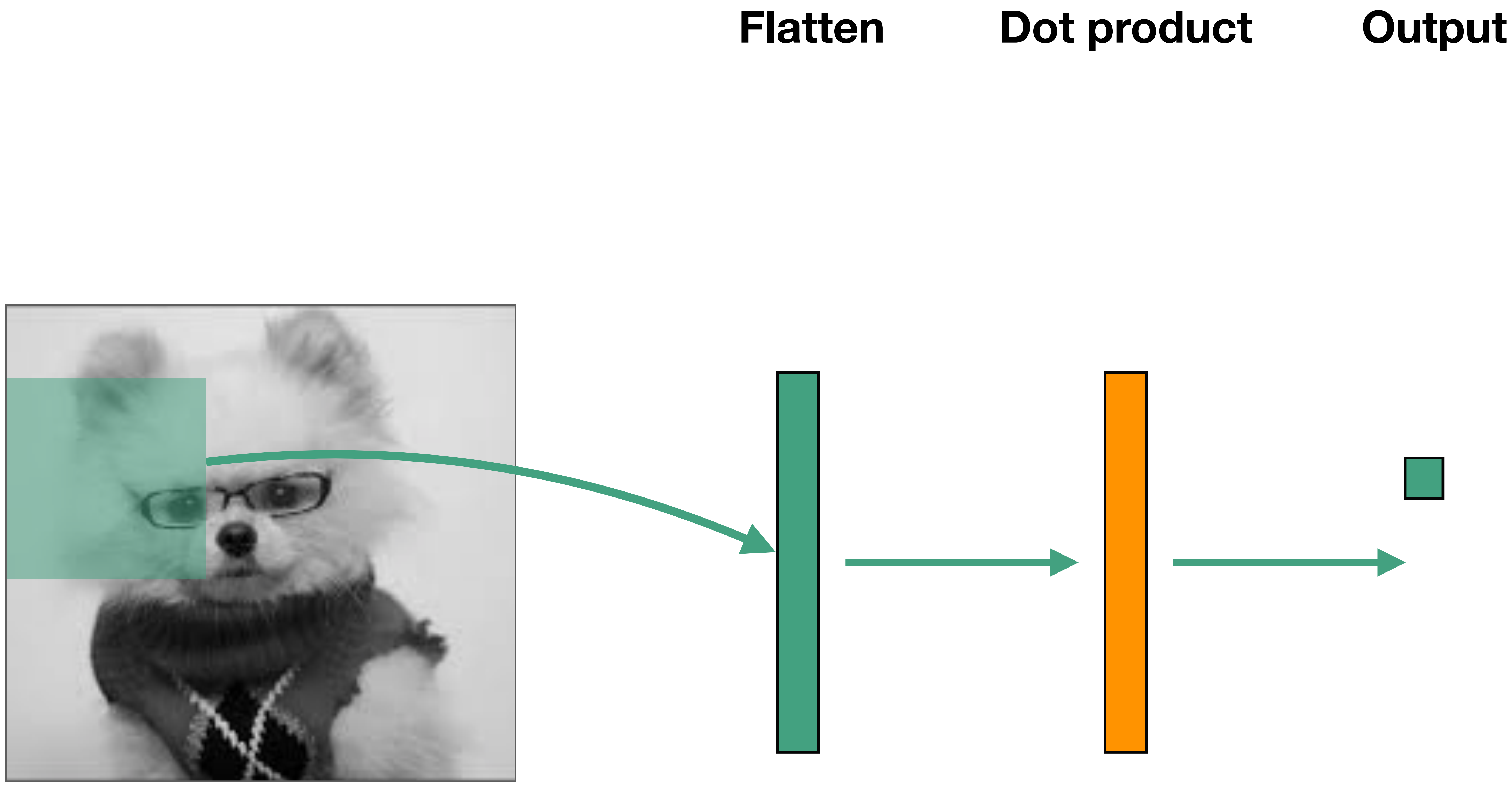
Convolutional filters



Convolutional filters

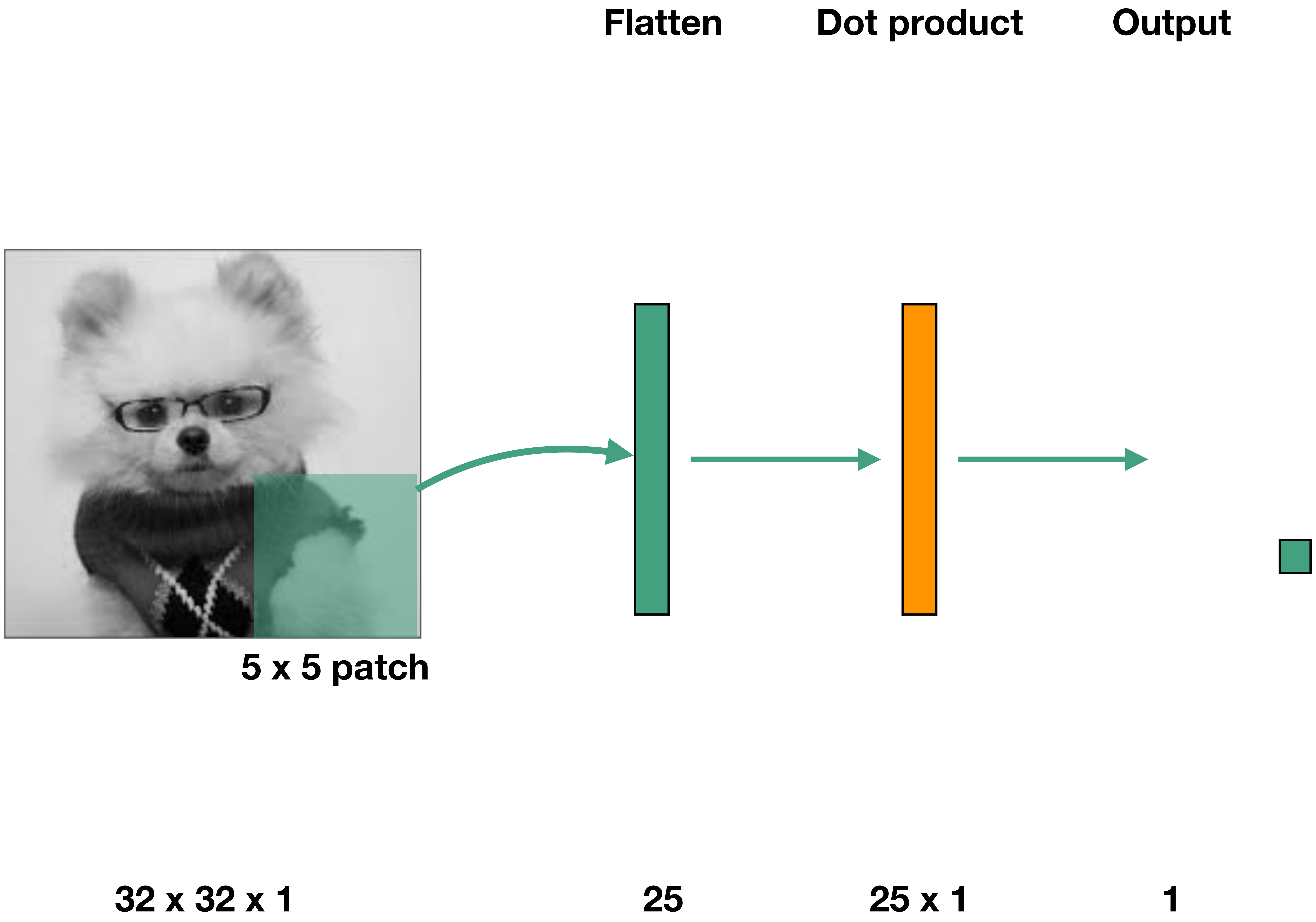


Convolutional filters

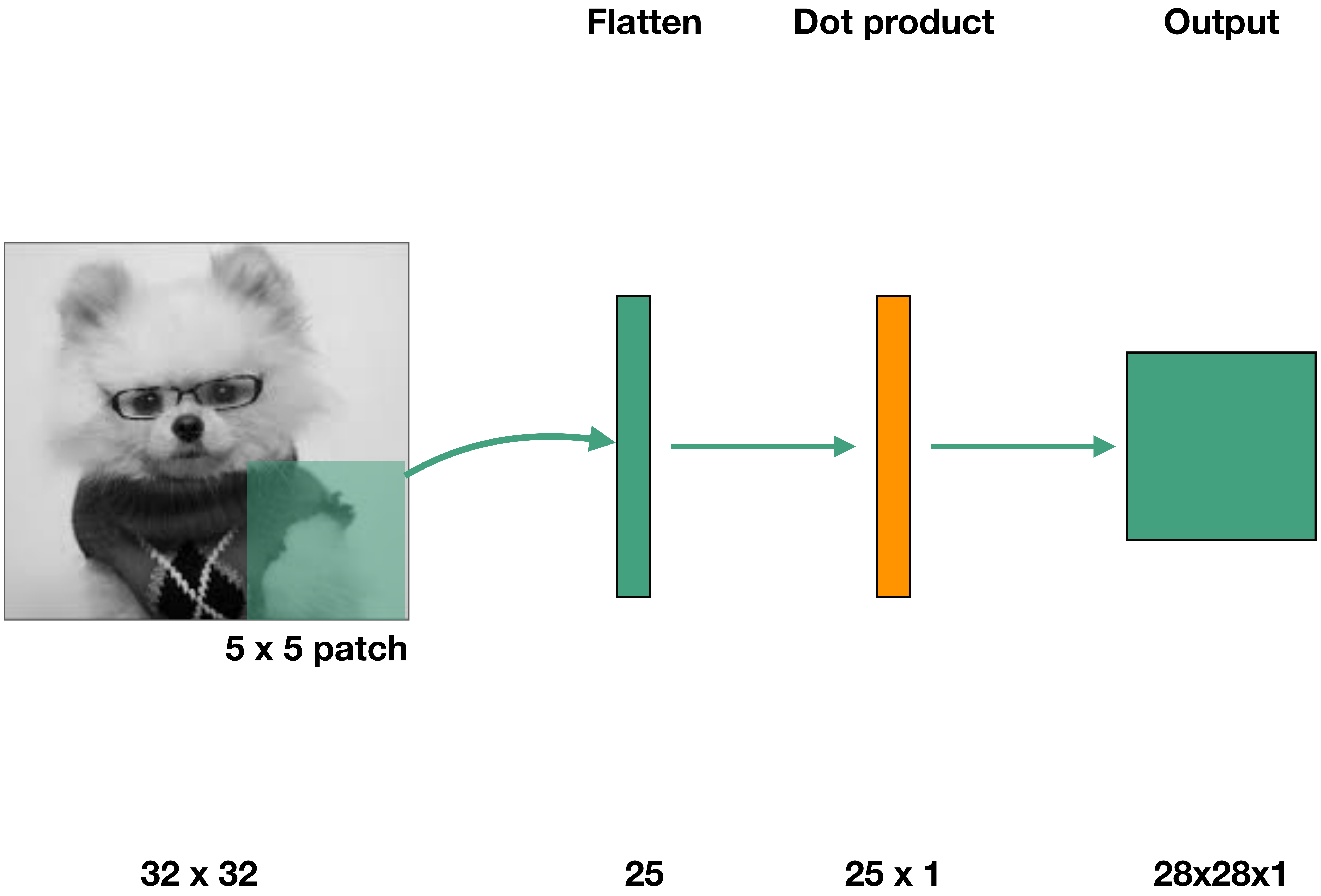


...sliding continues...

Convolutional filters



Convolutional filters



What can a conv filter do?

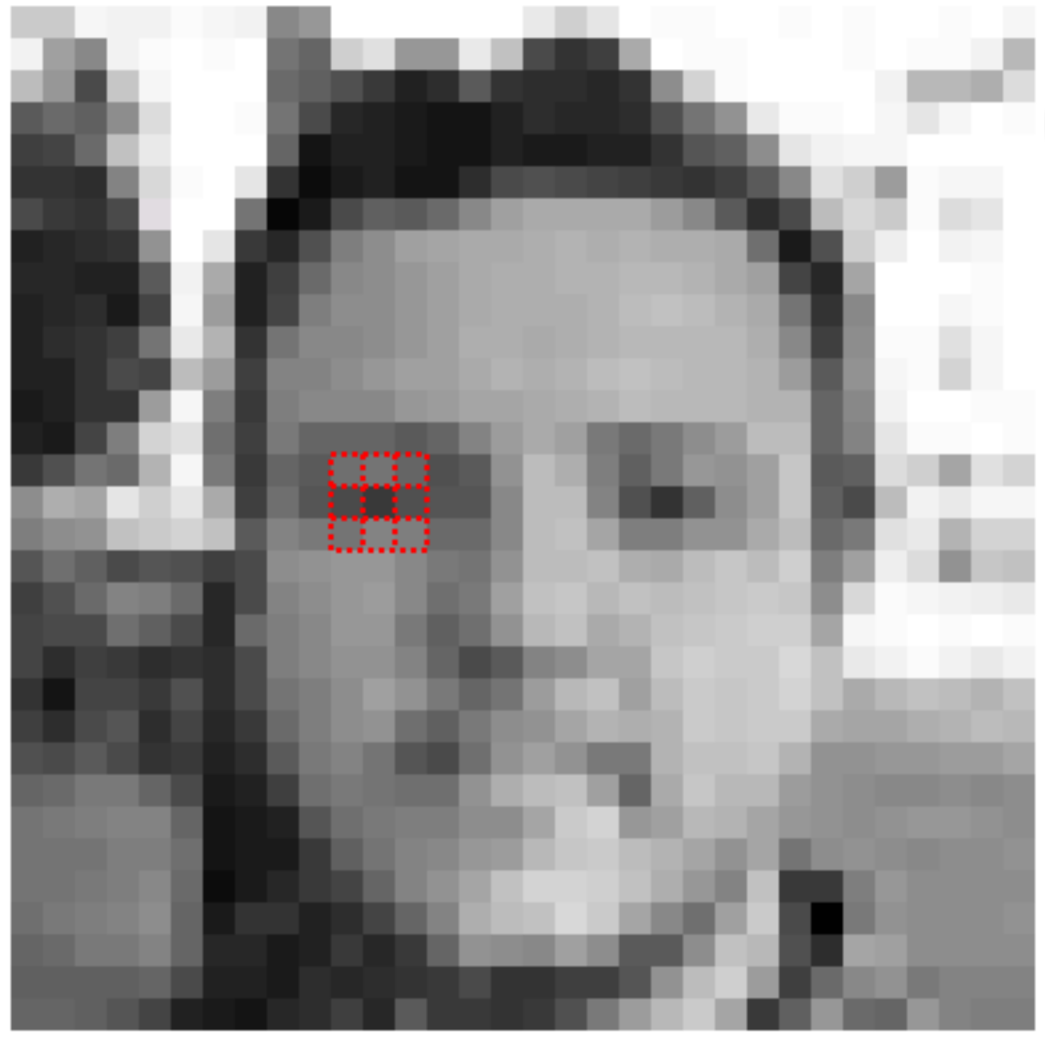
Let's walk through applying the following 3x3 **blur** kernel to the image of a face from above.

blur

$$\begin{pmatrix} 0.0625 & 0.125 & 0.0625 \\ 0.125 & 0.25 & 0.125 \\ 0.0625 & 0.125 & 0.0625 \end{pmatrix}$$

Instead of hard-coding the weights, we can learn them!

Below, for each 3x3 block of pixels in the image on the left, we multiply each pixel by the corresponding entry of the kernel and then take the sum. That sum becomes a new pixel in the image on the right. Hover over a pixel on either image to see how its value is computed.

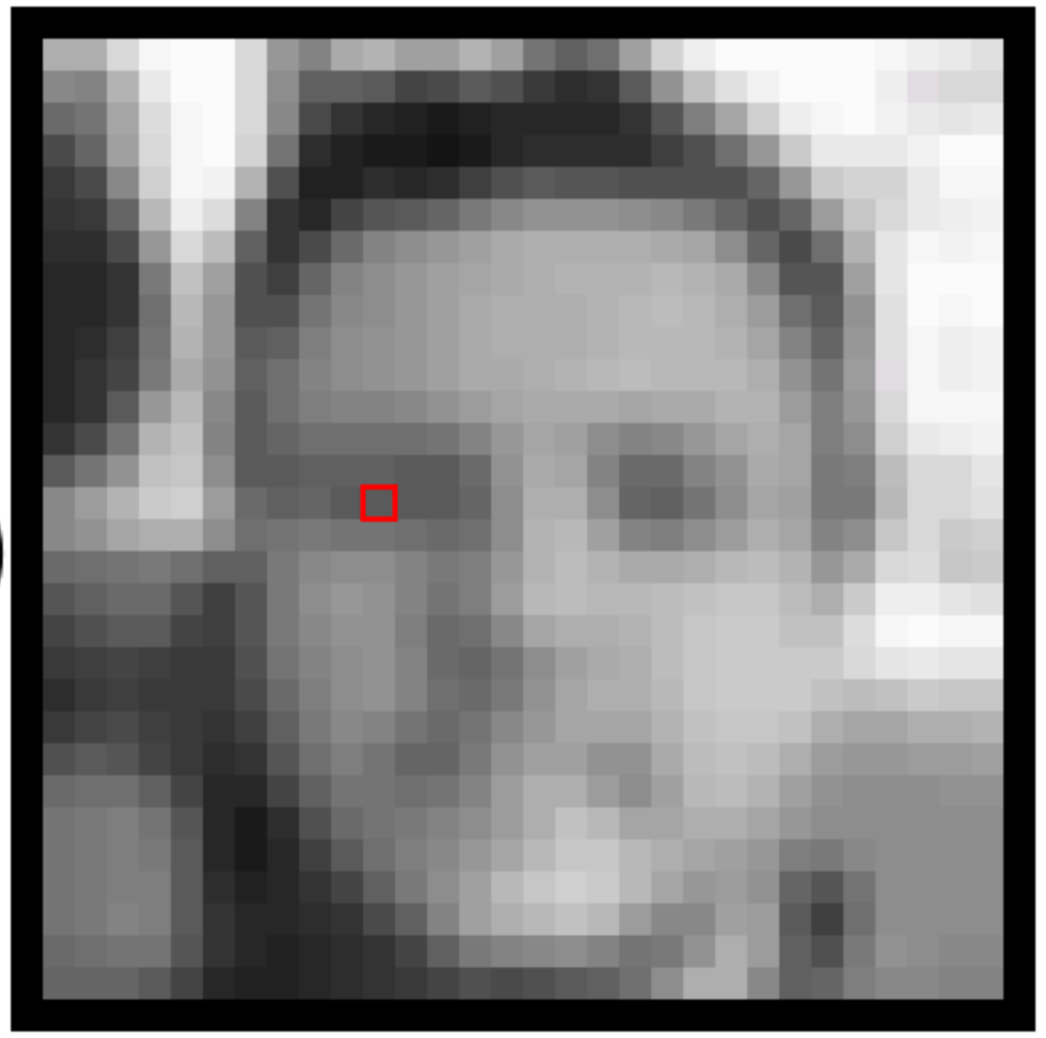


input image

$$\left(\begin{array}{l} 111 \times 0.0625 + 119 \times 0.125 + 103 \times 0.0625 \\ + 76 \times 0.125 + 62 \times 0.25 + 83 \times 0.125 \\ + 118 \times 0.0625 + 133 \times 0.125 + 126 \times 0.0625 \end{array} \right) = 96$$

kernel:

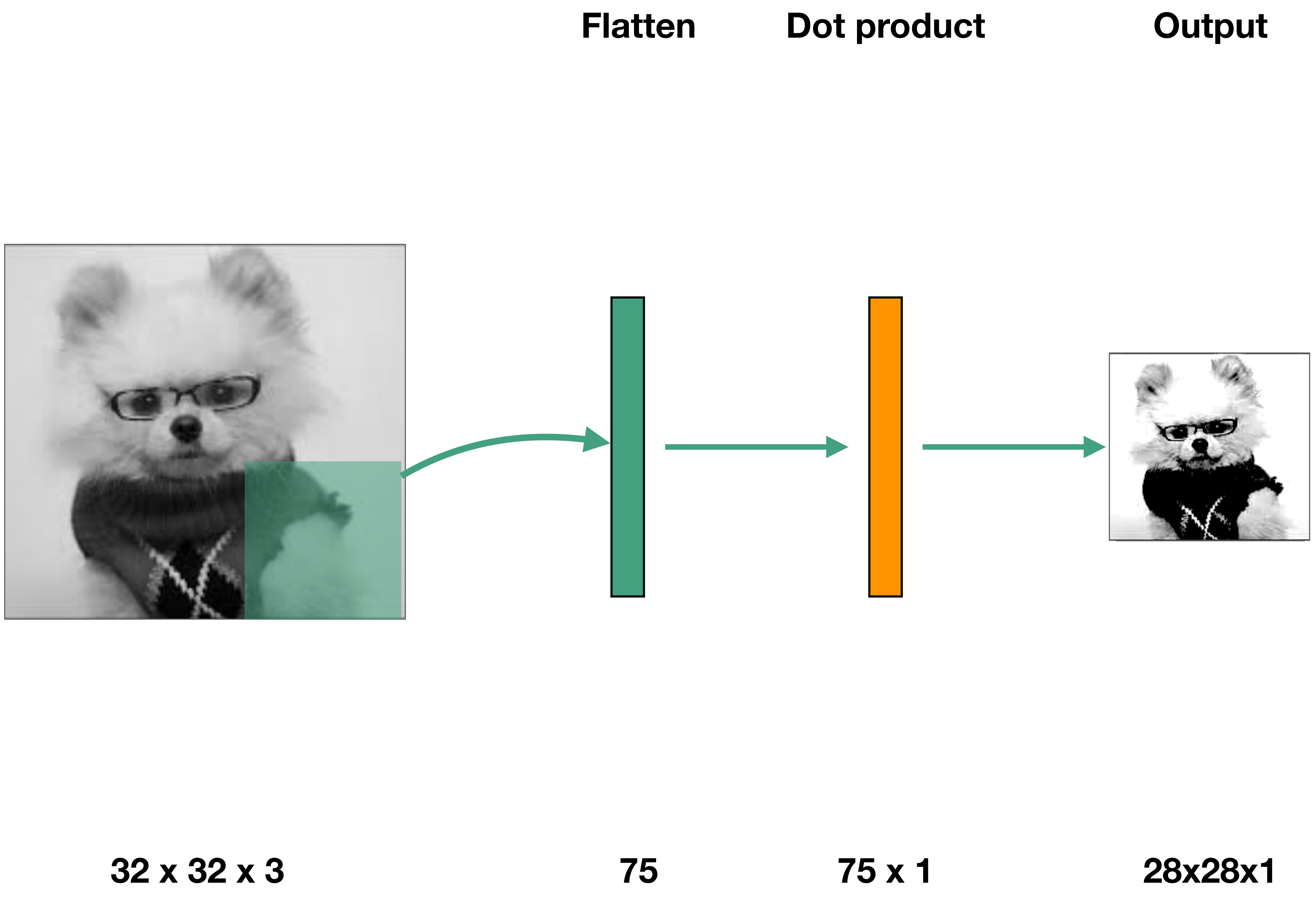
blur



output image

https://setosa.io/ev/image-kernels/

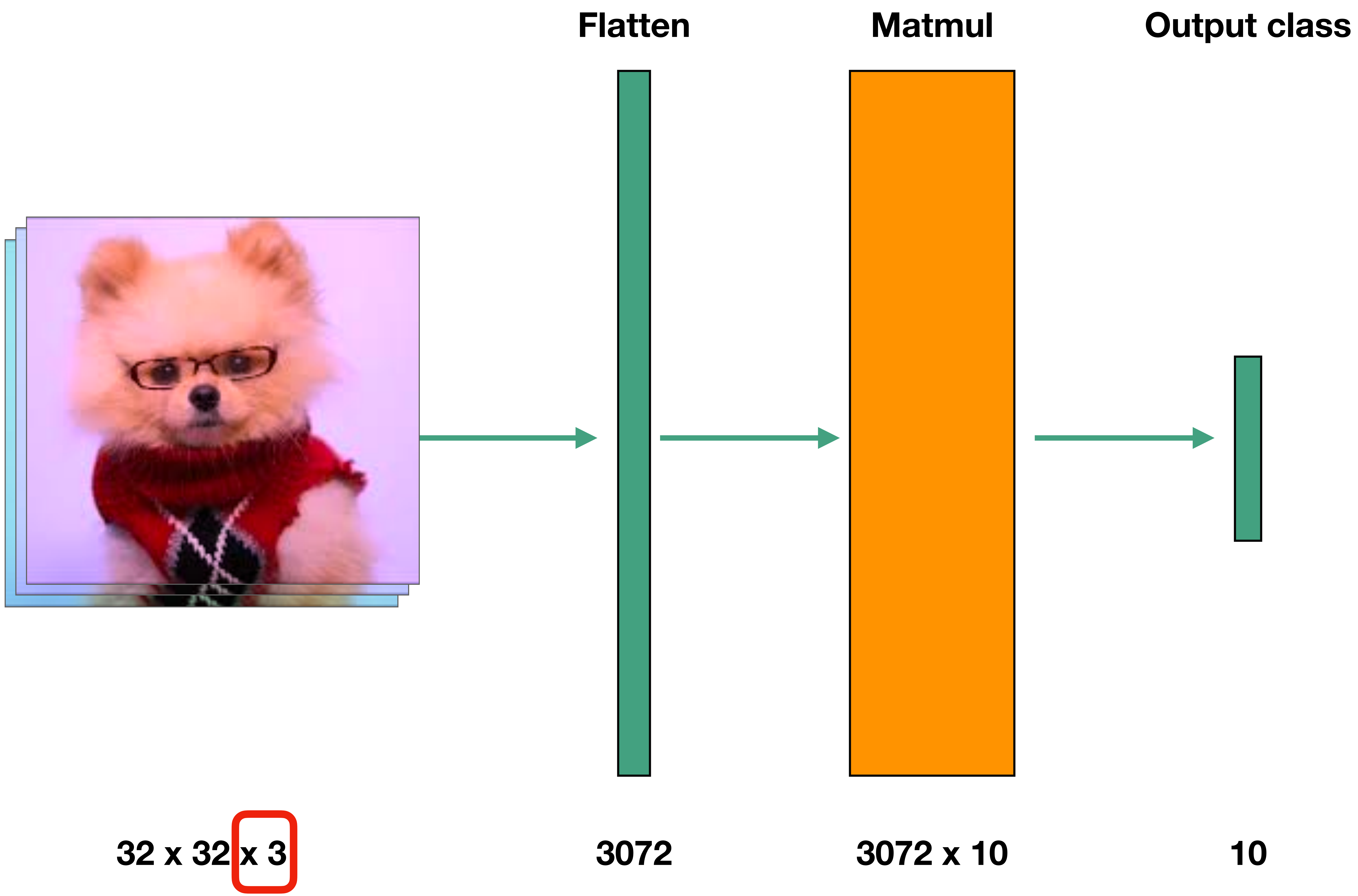
Convolutional filters



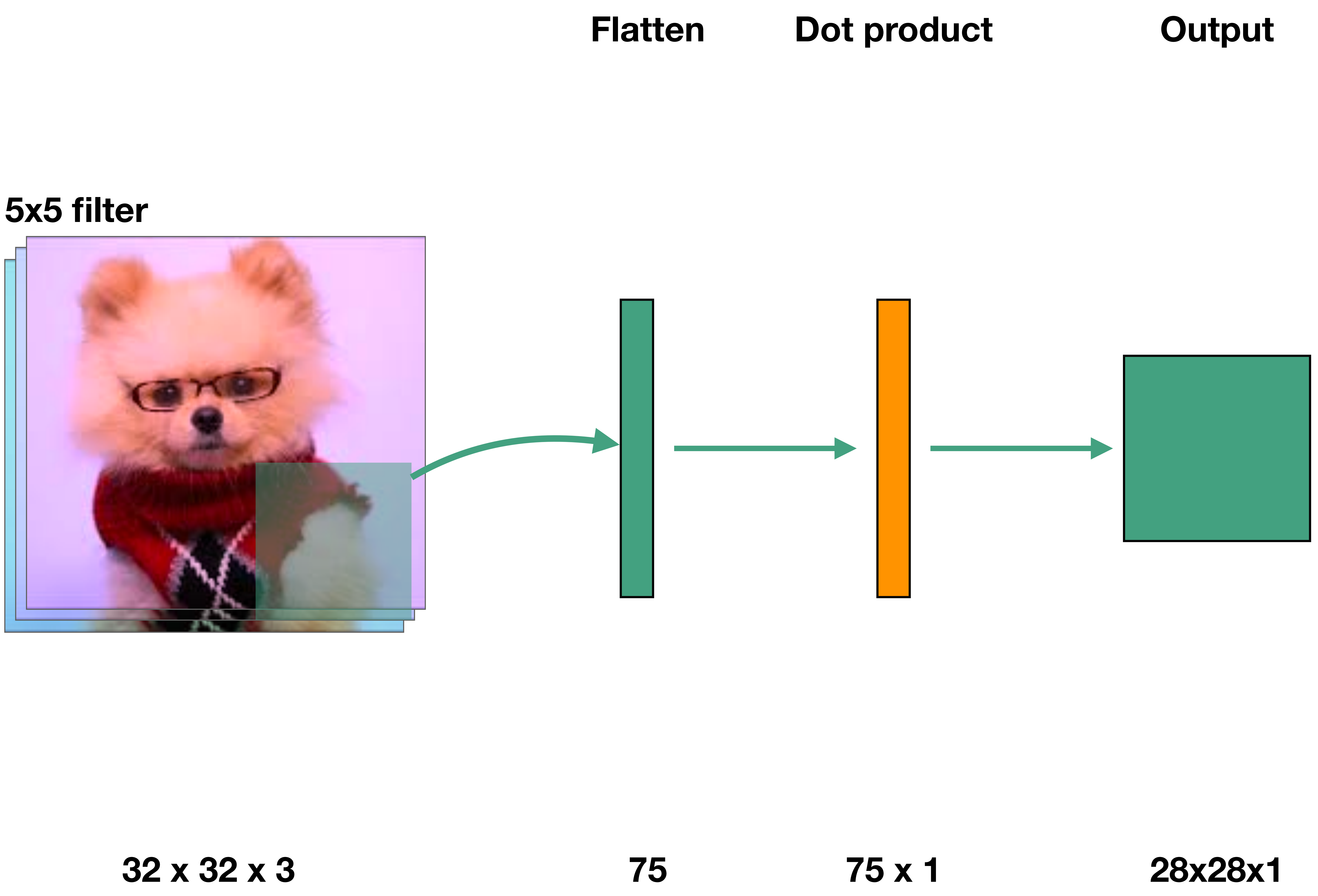
Review of convolutions

- What’s a convolutional filter?
- **Filter stacks and ConvNets**
- Strides & padding
- Filter math
- Implementation notes

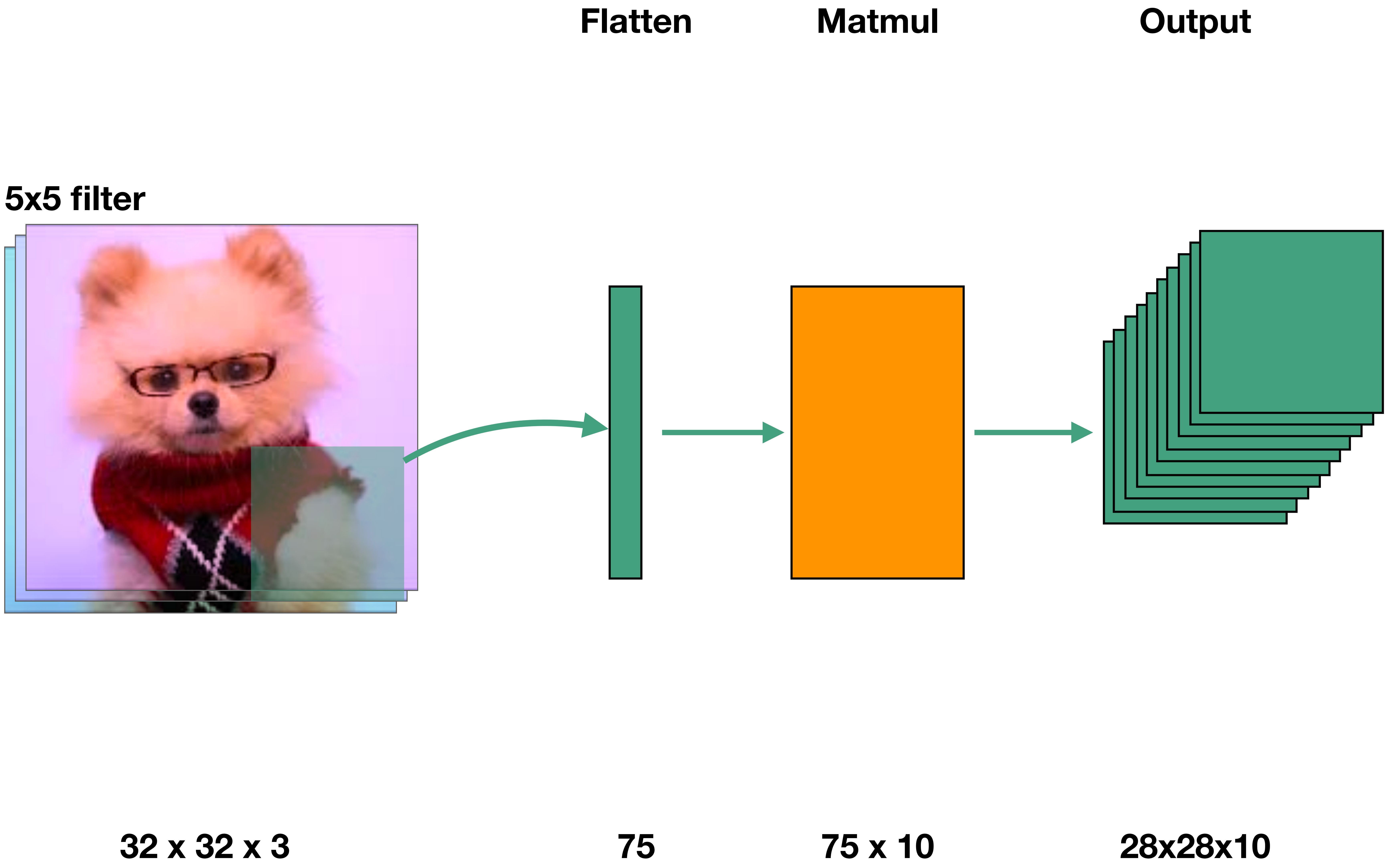
Input can have multiple channels



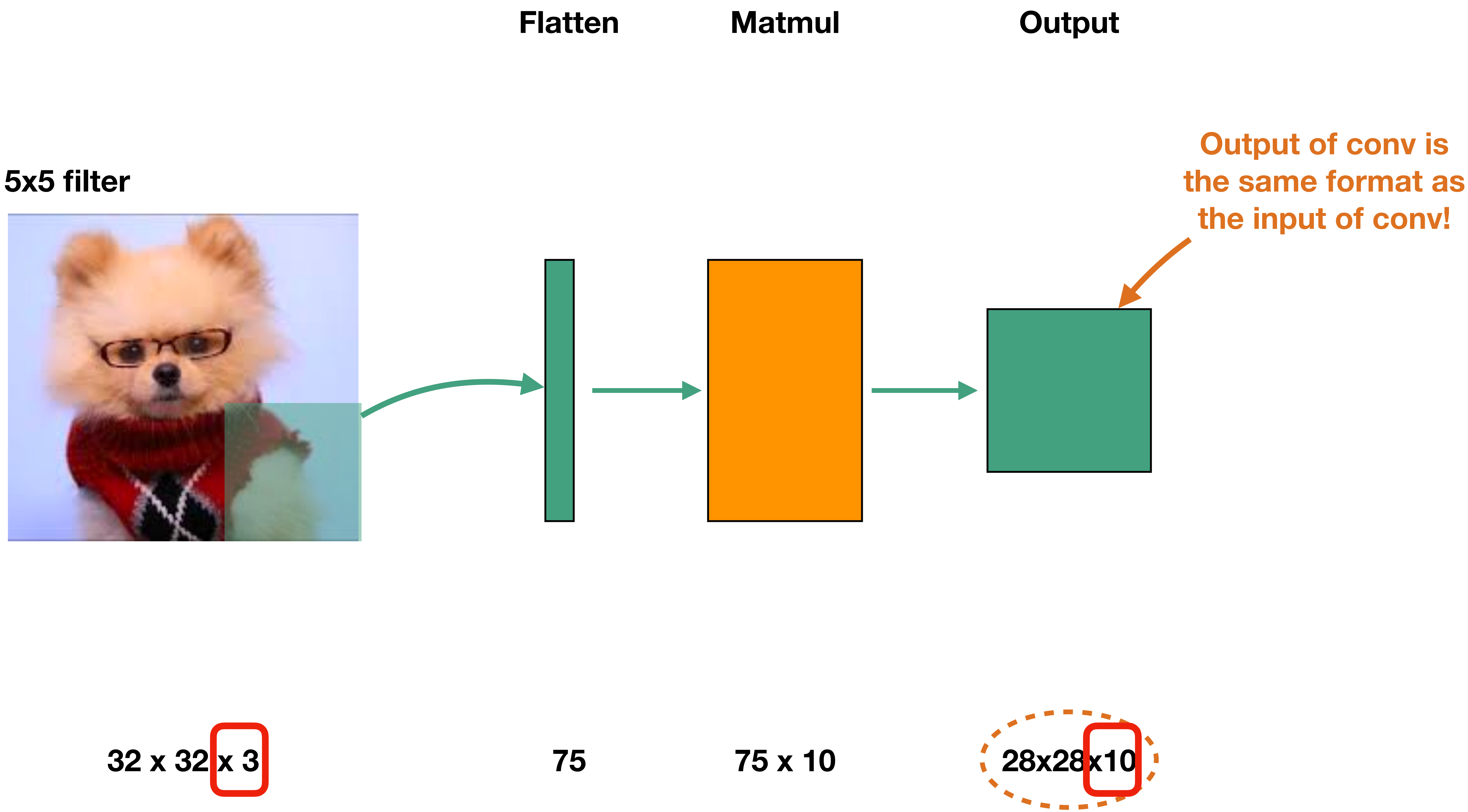
Input can have multiple channels



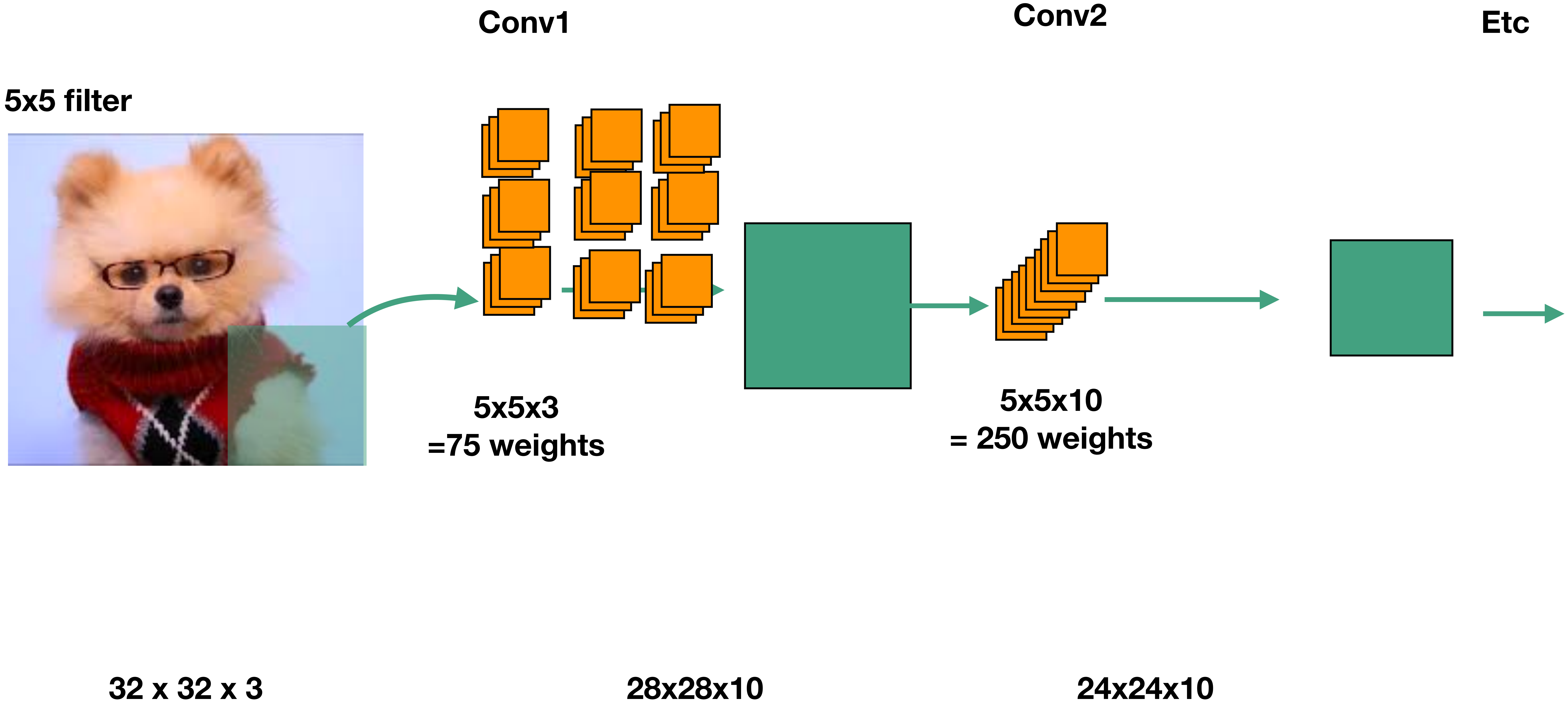
Output can have multiple channels



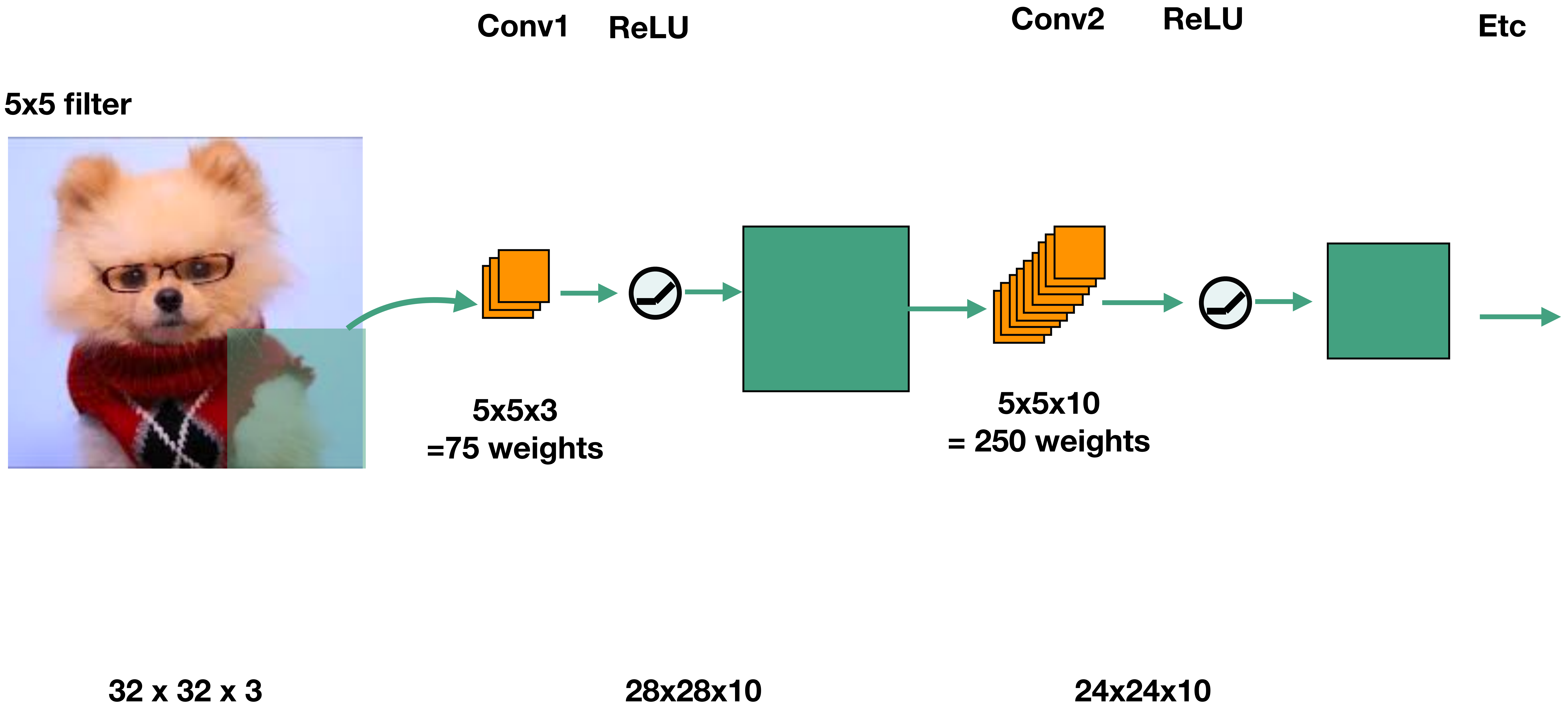
Convolutional filter stacks



Implication —> we can “stack” conv layers



Implication —> we can “stack” conv layers



Questions?

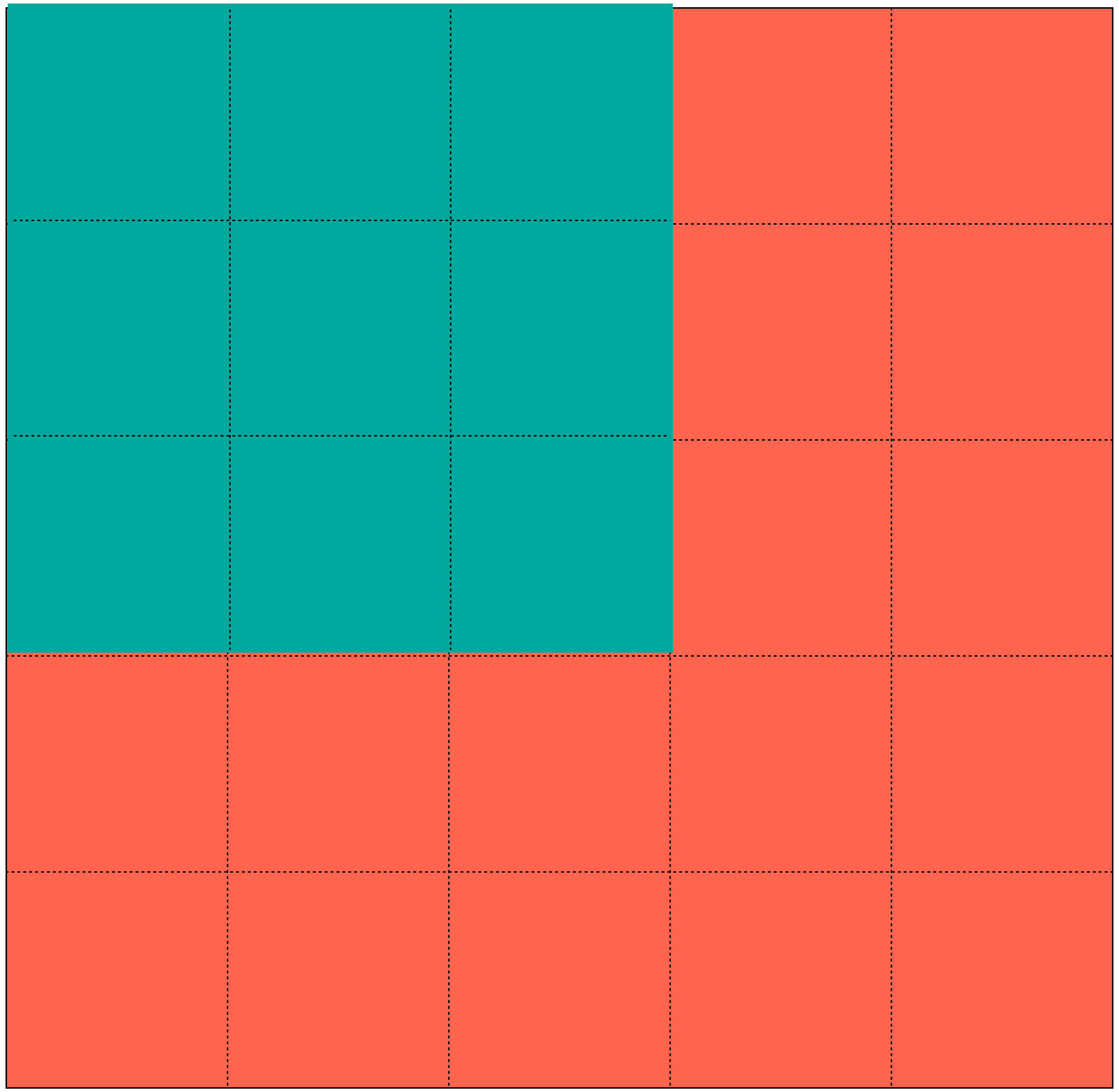
Review of convolutions

- What’s a convolutional filter?
- Filter stacks and ConvNets
- **Strides & padding**
- Filter math
- Implementation notes

Strides

- Convolutions can subsample the image by jumping across some locations — this is called ‘stride’

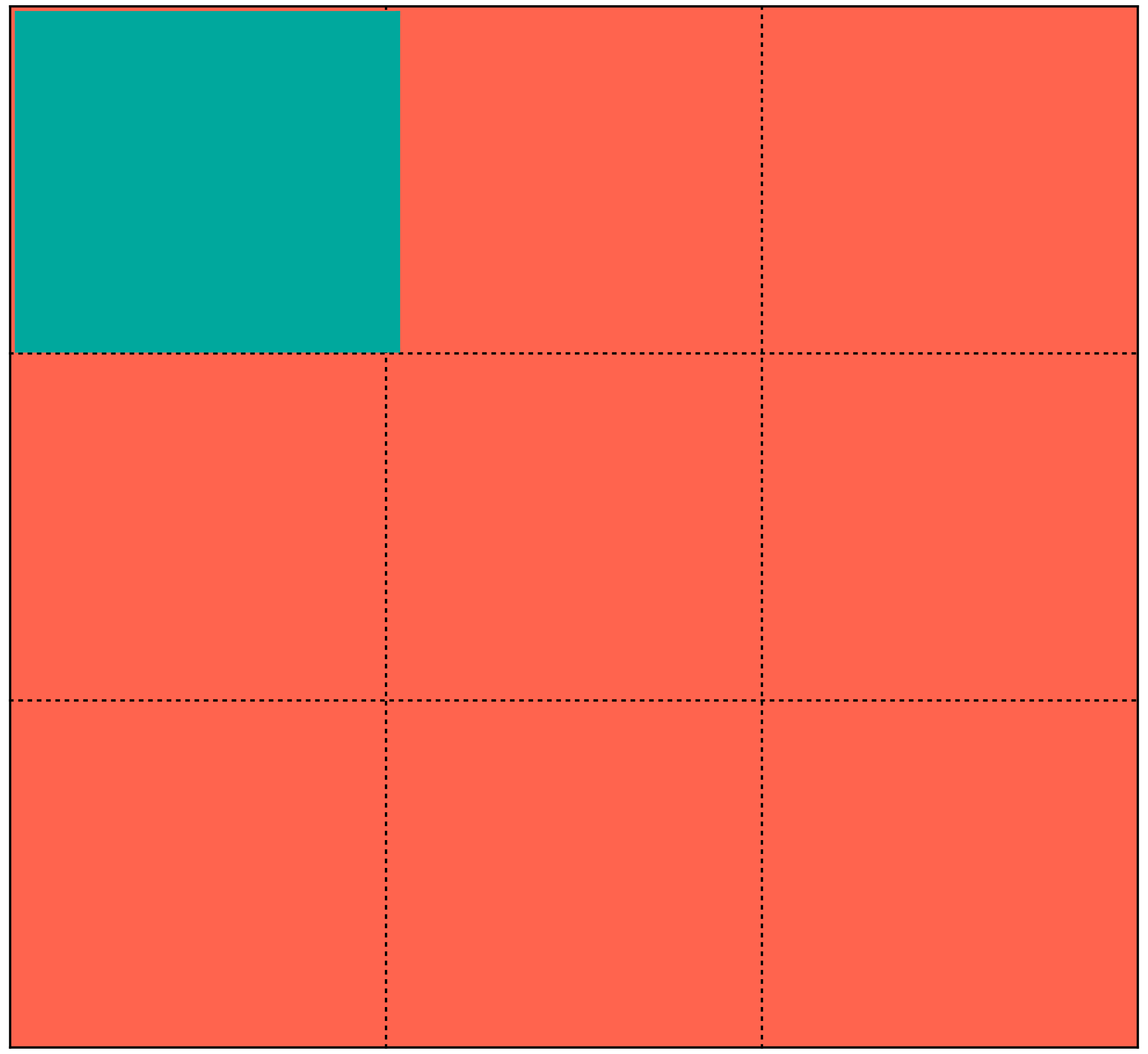
Strides



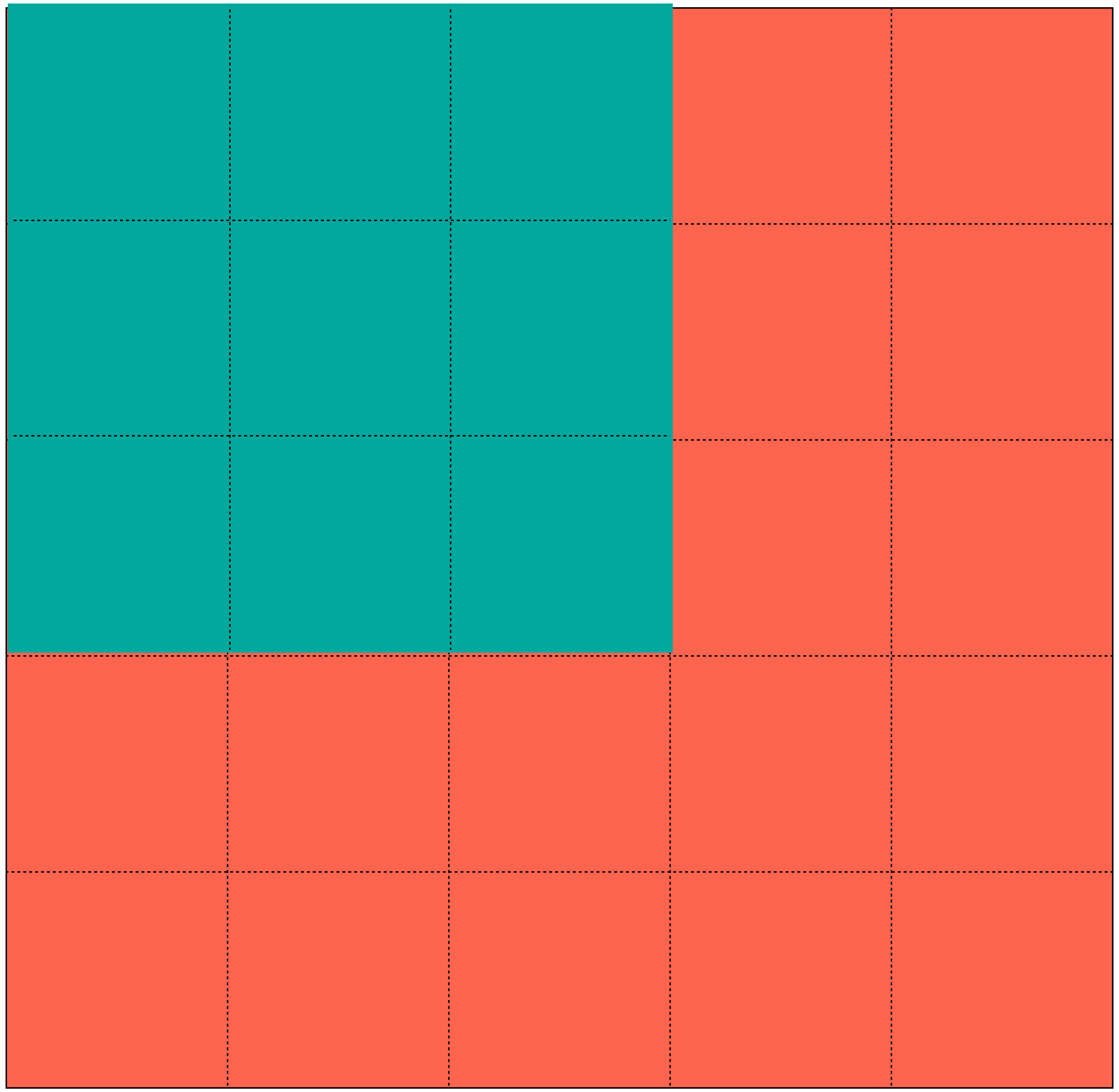
Conv2D

Filter = (3, 3)

Stride = (1, 1)



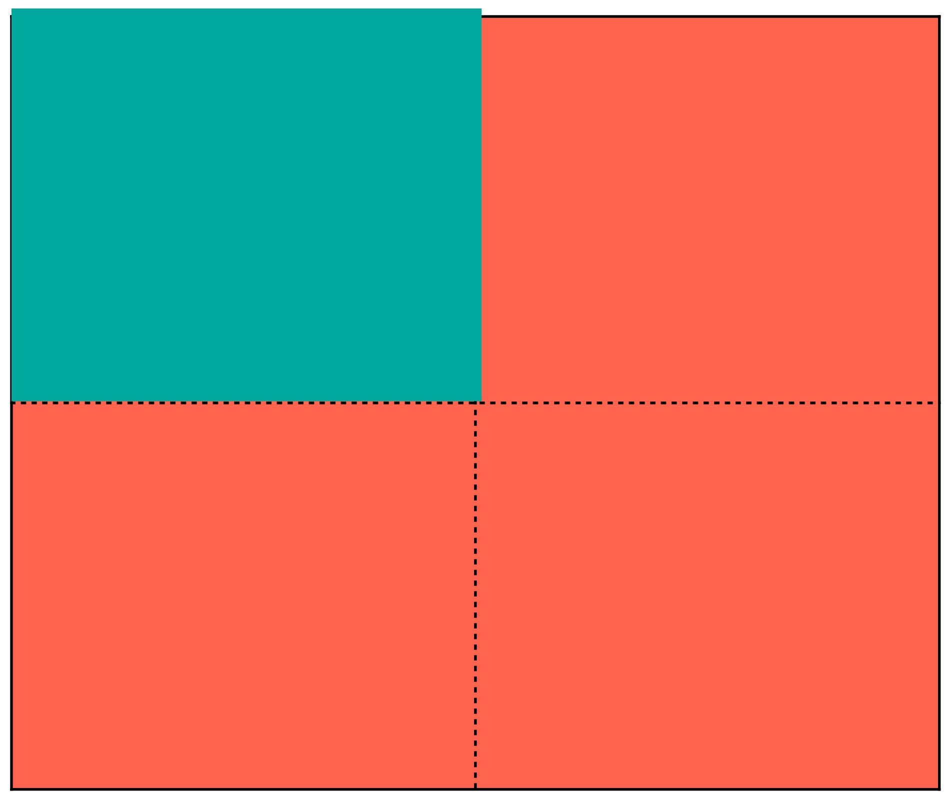
Strides



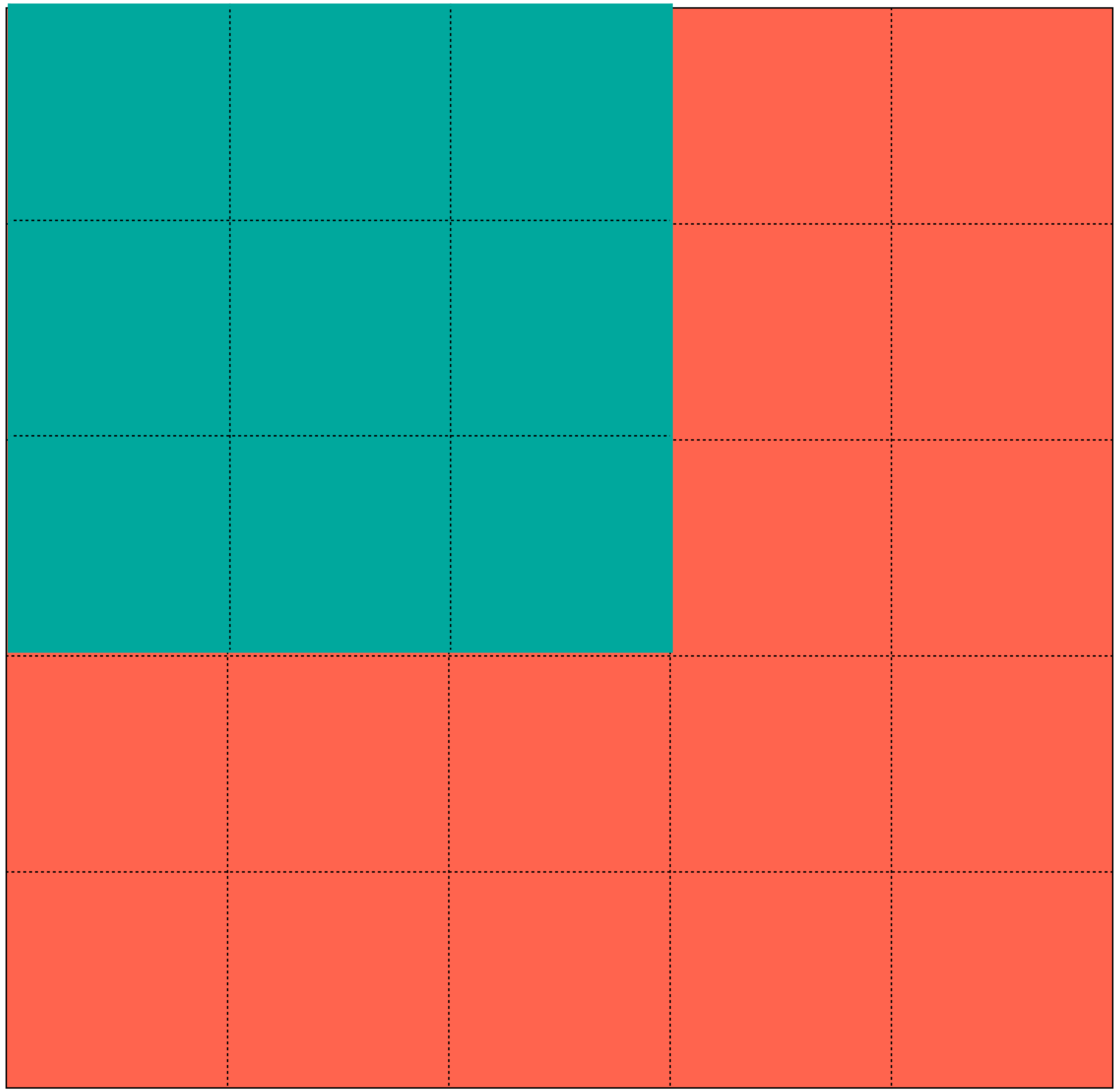
Conv2D

Filter = (3, 3)

Stride = (2, 2)



Strides



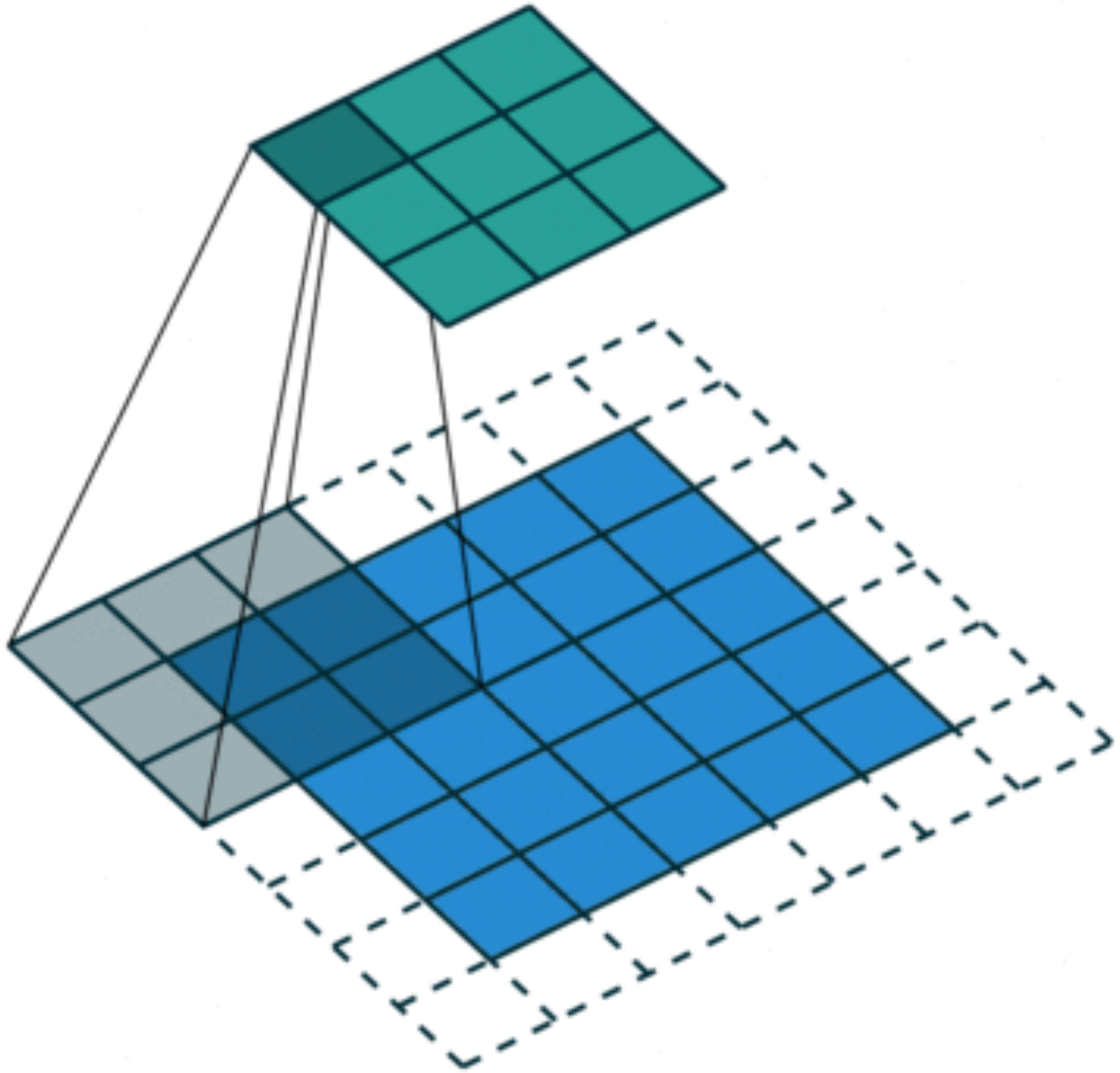
Conv2D

Filter = (3, 3)

Stride = (3, 3)

?

Padding



- Padding solves the problem of filters running out of image
- Done by adding extra rows/cols to the input (usually set to 0)
- 'SAME' padding is illustrated here for filter=(3,3) with stride=(2,2)
- Not padding is called 'VALID' padding

https://github.com/vdumoulin/conv_arithmetic

Review of convolutions

- What’s a convolutional filter?
- Filter stacks and ConvNets
- Strides & padding
- **Filter math**
- Implementation notes

Conv2D Math

- Input: WxHxD volume
- Parameters:
    - K filters, each with size (F, F)
    - …moving at stride (S, S)
    - …with padding P
- Output: W’xH’xK volume
    - $W' = (W - F + 2P) / S + 1$
    - $H' = (H - F + 2P) / S + 1$
- Each filter has $(F * F * D)$ parameters
- $K * (F * F * D)$ total in the layer

Conv2D Math

- Input: WxHxD volume
- Parameters:
  - K filters, each with size (F, F)
  - …moving at stride (S, S)
  - …with padding P
- Output: W’xH’xK volume
  - W’ = (W - F + 2P) / S + 1
  - H’ = (H - F + 2P) / S + 1
- Each filter has (F * F * D) parameters
- K * (F * F * D) total in the layer

- **Commonly set to powers of 2 (e.g. 32, 64, 128)**

Conv2D Math

- Input: WxHxD volume
- Parameters:
  - K filters, each with size (F, F)
  - …moving at stride (S, S)
  - …with padding P
- Output: W’xH’xK volume
  - W’ = (W - F + 2P) / S + 1
  - H’ = (H - F + 2P) / S + 1
- Each filter has (F * F * D) parameters
- K * (F * F * D) total in the layer

- **Commonly (5, 5), (3, 3), (2, 2), (1, 1)**

Conv2D Math

- Input: WxHxD volume
- Parameters:
  - K filters, each with size (F, F)
  - …moving at stride (S, S)
  - …with padding P
- Output: W’xH’xK volume
  - W’ = (W - F + 2P) / S + 1
  - H’ = (H - F + 2P) / S + 1
- Each filter has (F * F * D) parameters
- K * (F * F * D) total in the layer

- **‘SAME’ sets it automatically**

A guide to convolution arithmetic for deep learning

Vincent Dumoulin[1★] and Francesco Visin[2★†]

- Lots of cool visualizations and comforting equations

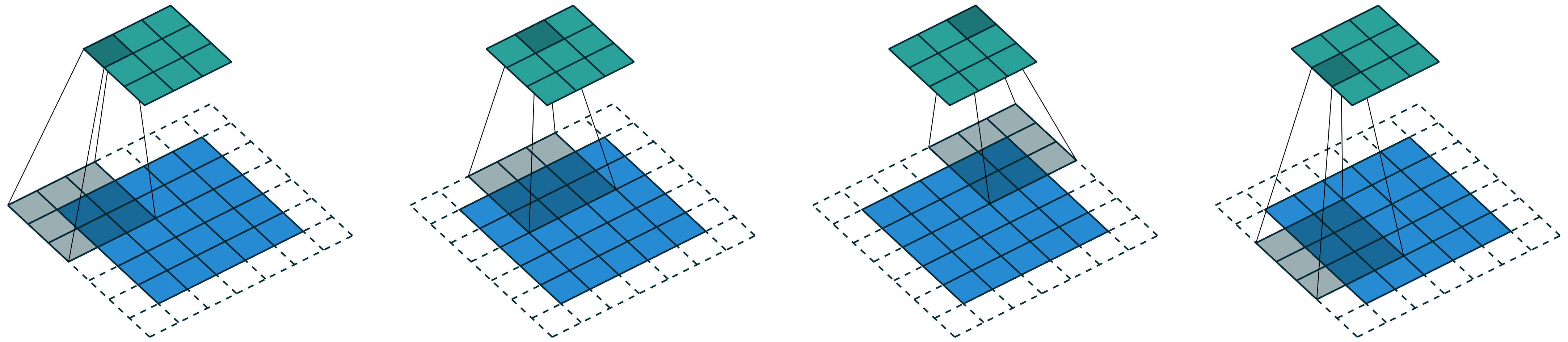


Figure 2.6: (Arbitrary padding and strides) Convolving a 3×3 kernel over a 5×5 input padded with a 1×1 border of zeros using 2×2 strides (i.e., $i = 5$, $k = 3$, $s = 2$ and $p = 1$).

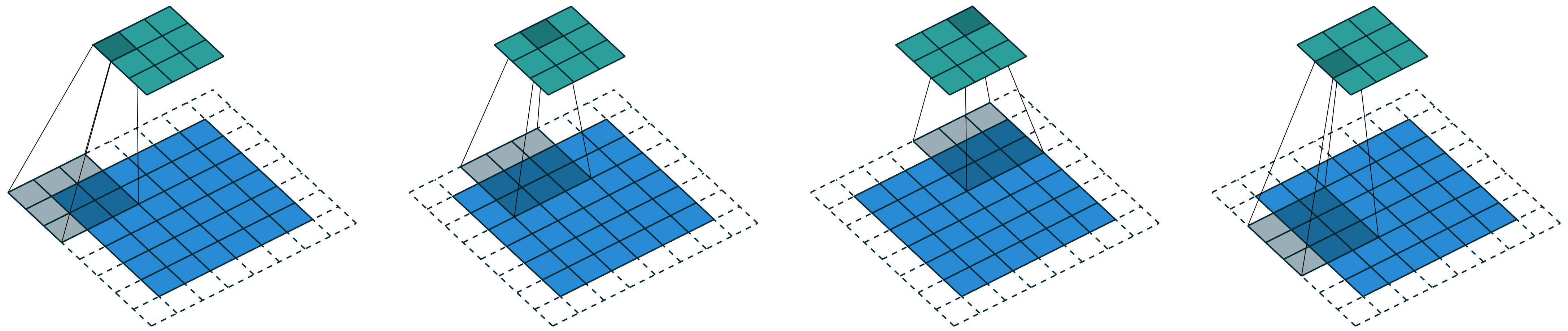


Figure 2.7: (Arbitrary padding and strides) Convolving a 3×3 kernel over a 6×6 input padded with a 1×1 border of zeros using 2×2 strides (i.e., $i = 6$, $k = 3$, $s = 2$ and $p = 1$). In this case, the bottom row and right column of the zero padded input are not covered by the kernel.

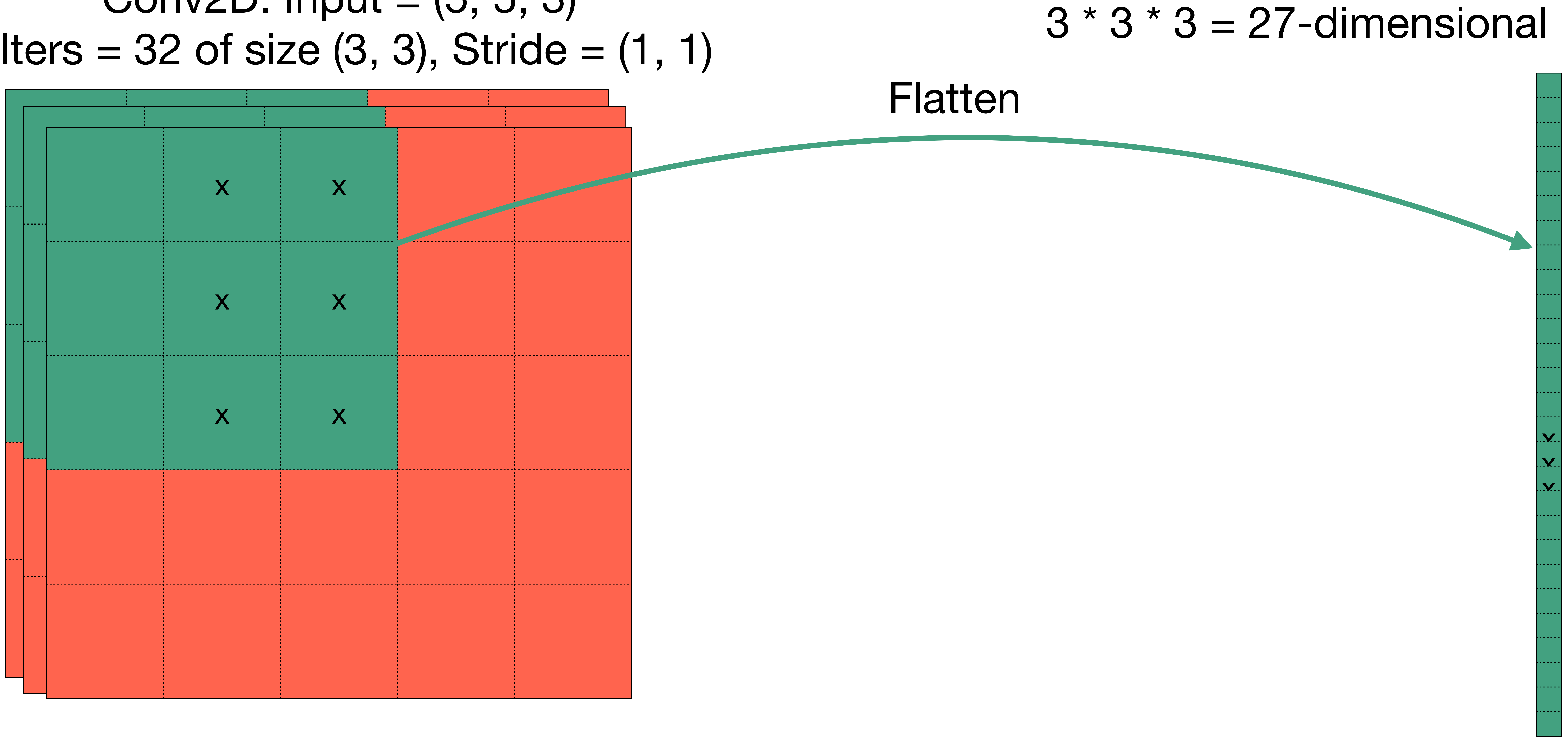
https://github.com/vdumoulin/conv_arithmetic

Review of convolutions

- What's a convolutional filter?
- Filter stacks and ConvNets
- Strides & padding
- Filter math
- **Implementation notes**

Convolution implementation

Conv2D. Input = (5, 5, 3)
Filters = 32 of size (3, 3), Stride = (1, 1)

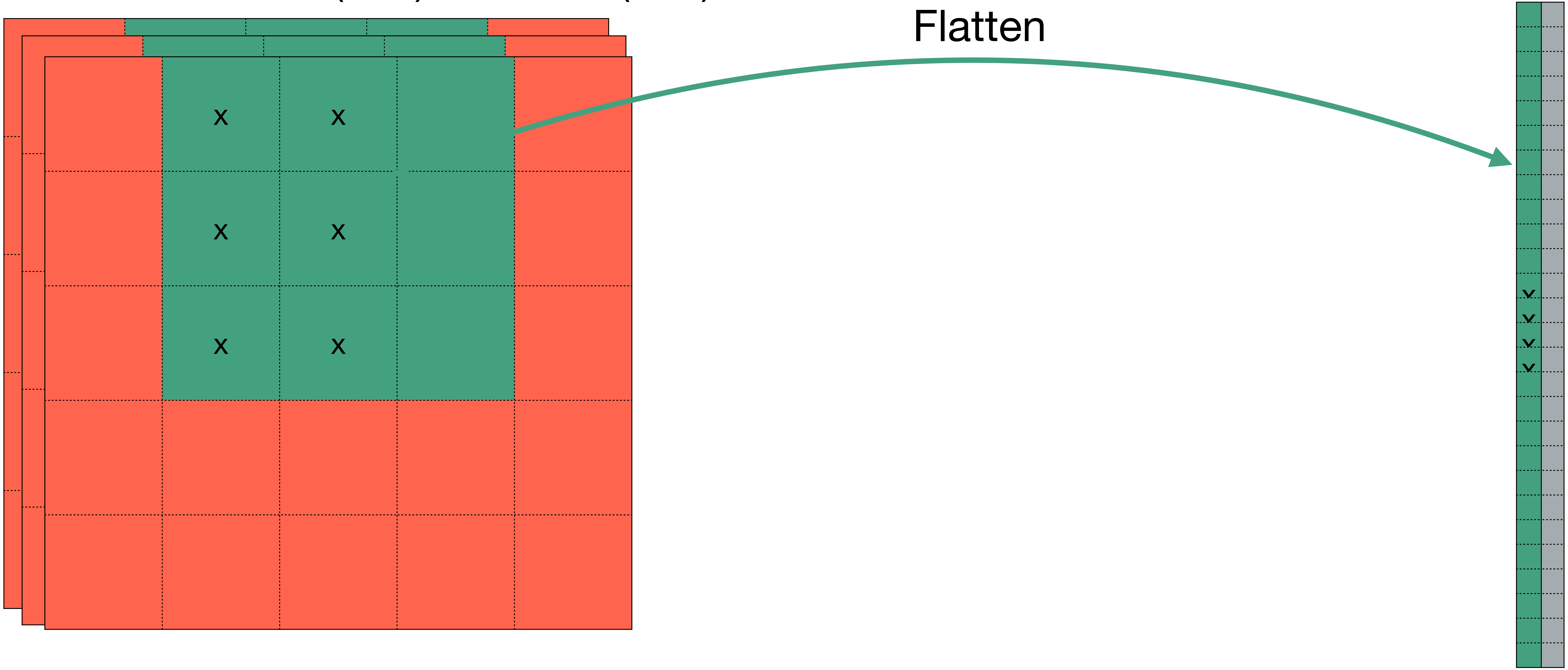


http://cs231n.github.io/convolutional-networks/

Convolution implementation

Conv2D. Input = (5, 5, 3)
Filters = 32 of size (3, 3), Stride = (1, 1)

3 * 3 * 3 = 27-dimensional

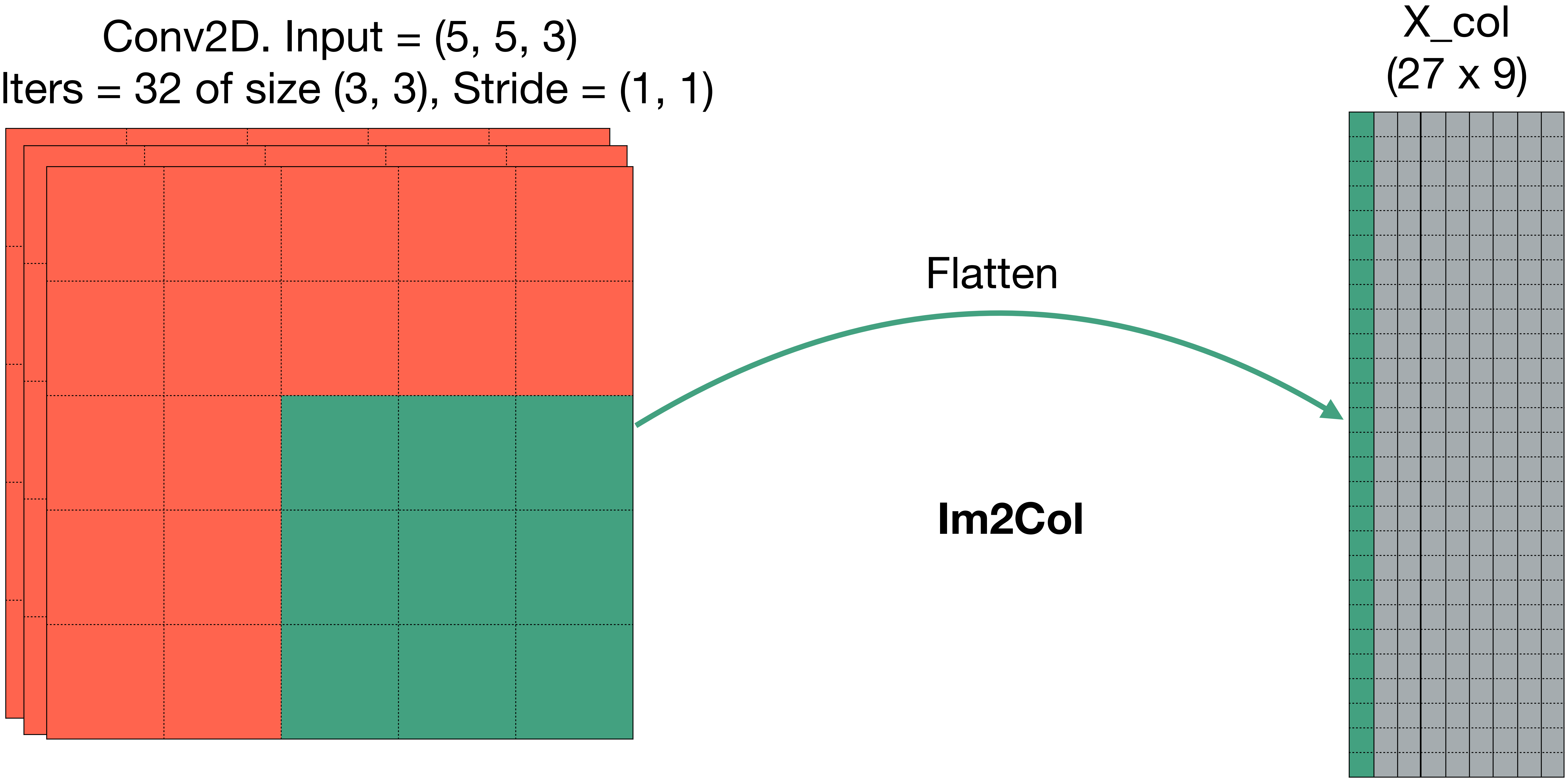


http://cs231n.github.io/convolutional-networks/

...sliding continues...

Convolution implementation

Conv2D. Input = (5, 5, 3)
Filters = 32 of size (3, 3), Stride = (1, 1)



http://cs231n.github.io/convolutional-networks/

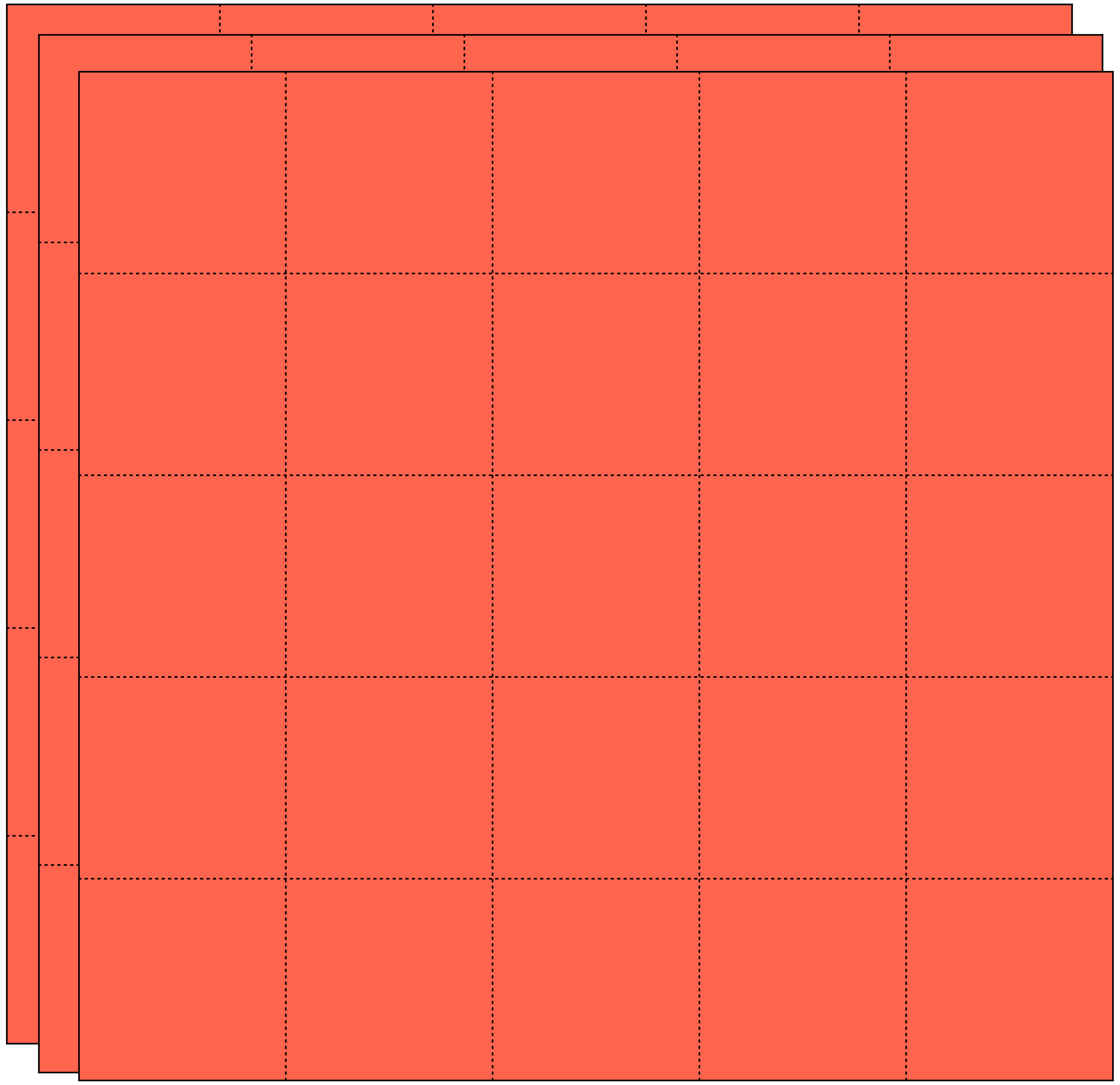
3x3 filter, 3-channel input

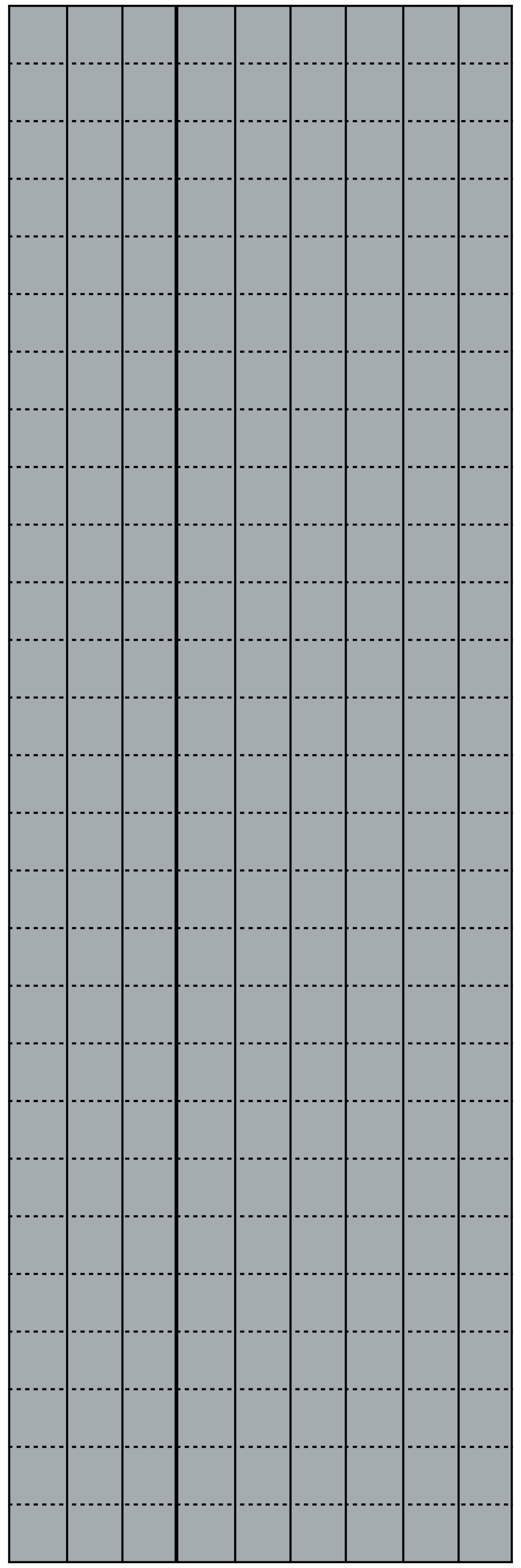
Conv2D. Input = (5, 5, 3)

Filters = 32 of size (3, 3), Stride = (1, 1)

W_row
(32 x 27)

X_col
(27 x 9)



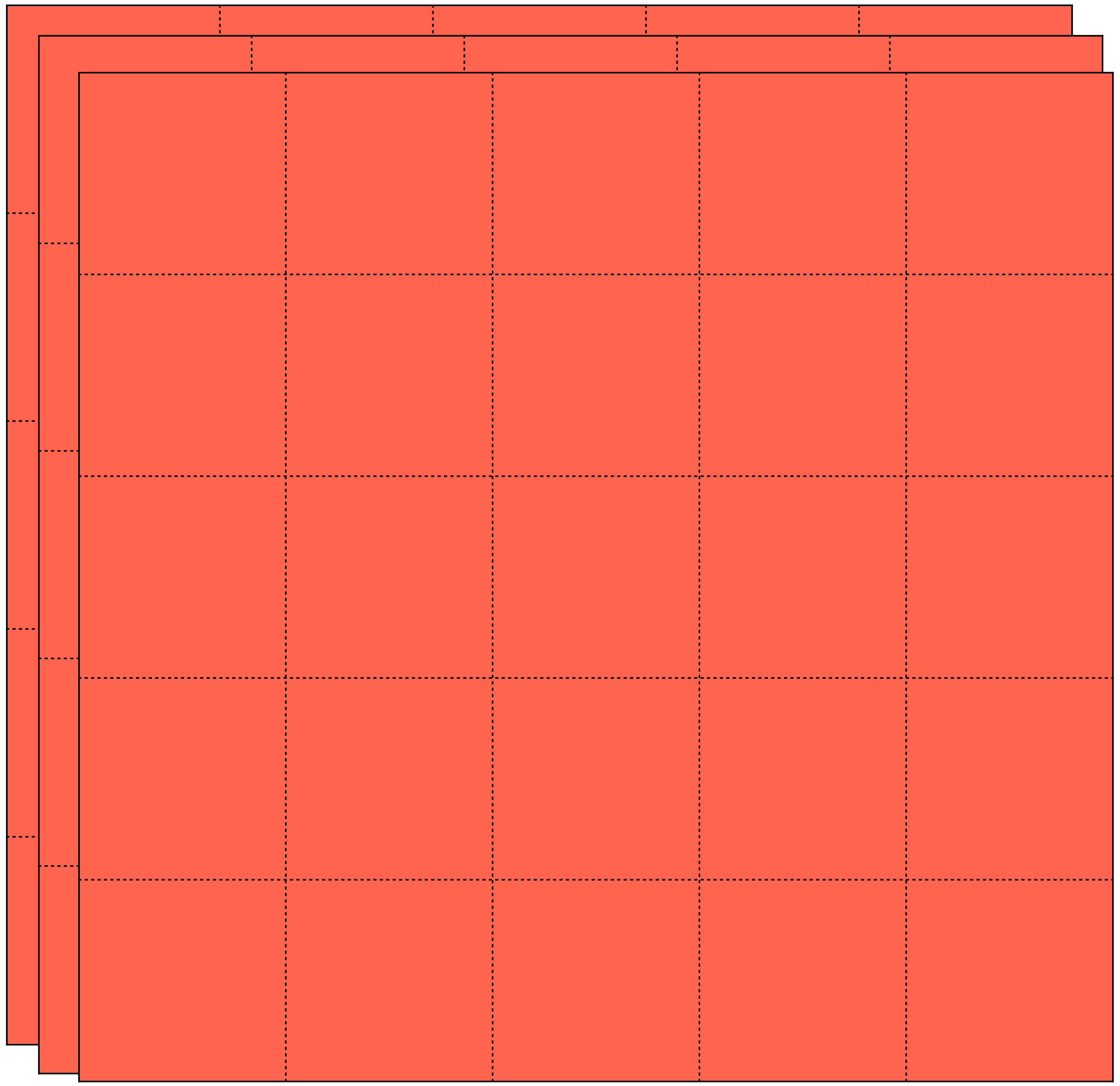


http://cs231n.github.io/convolutional-networks/

Convolution implementation

Conv2D. Input = (5, 5, 3)

Filters = 32 of size (3, 3), Stride = (1, 1)



W

(32 x 27)

X

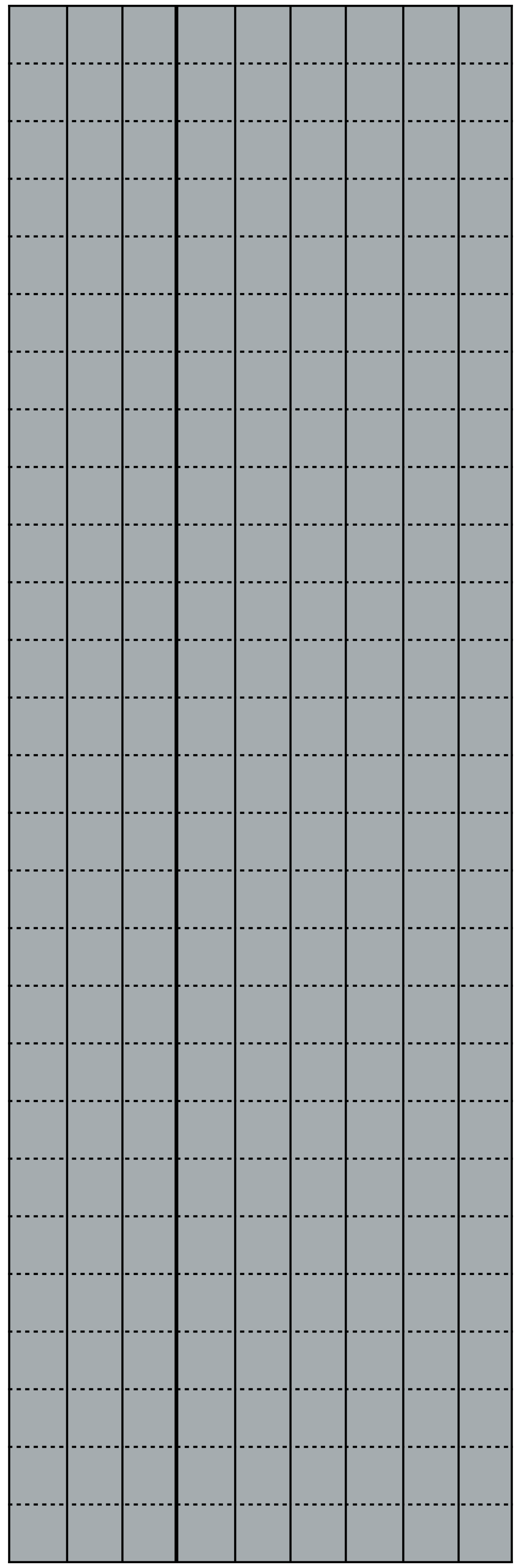
(27 x 9)

W @ X

(32 x 9)

Reshape

(3 x 3 x 32)



http://cs231n.github.io/convolutional-networks/

Questions?

Agenda

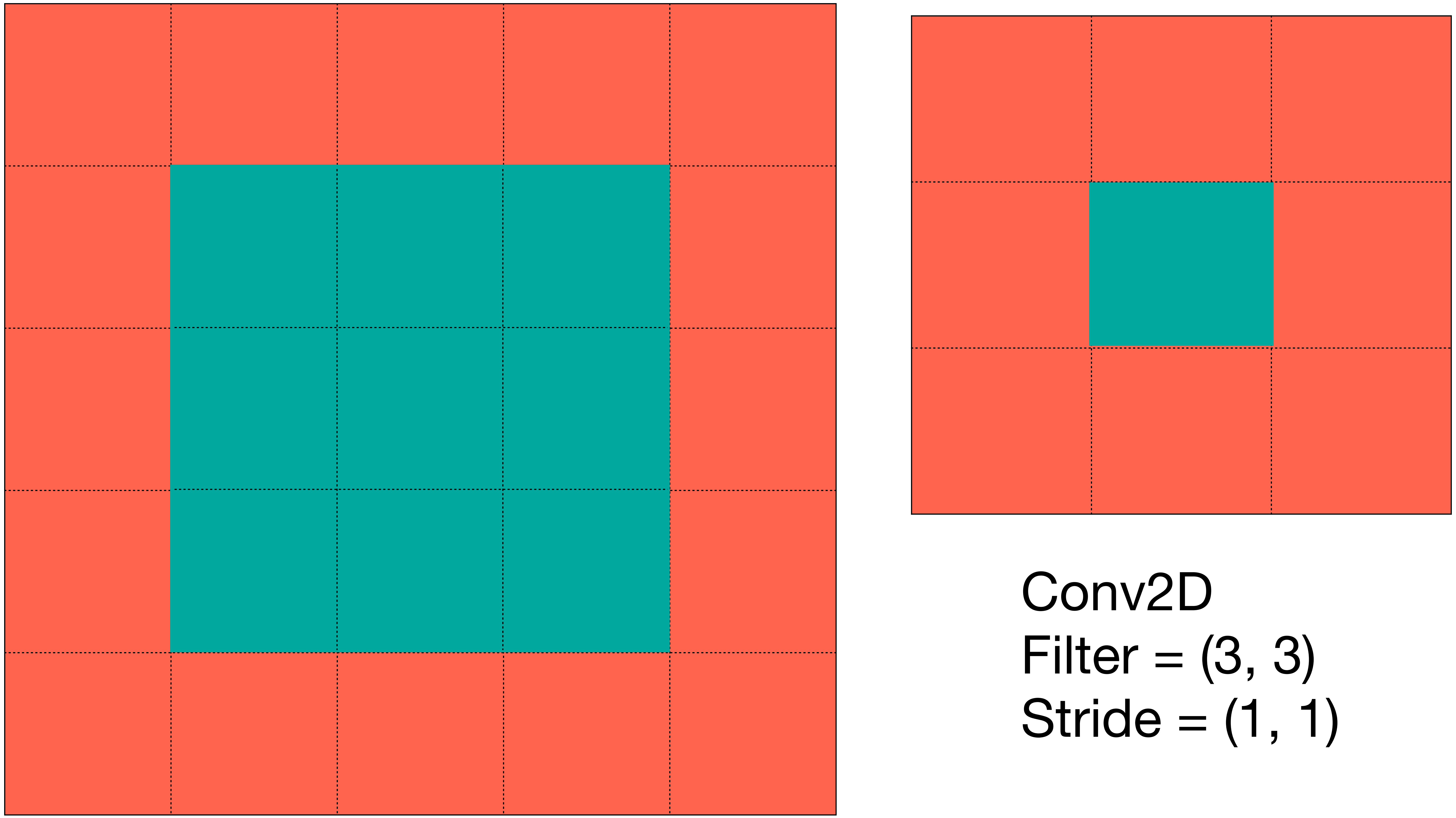
1. Review of the convolution operation
2. **Other important operations for ConvNets**
3. Classic ConvNet architectures

Other important ConvNet operations

- **Increasing the receptive field (dilated convolutions)**
- Decreasing the size of the tensor
  - Pooling
  - 1x1-convolutions

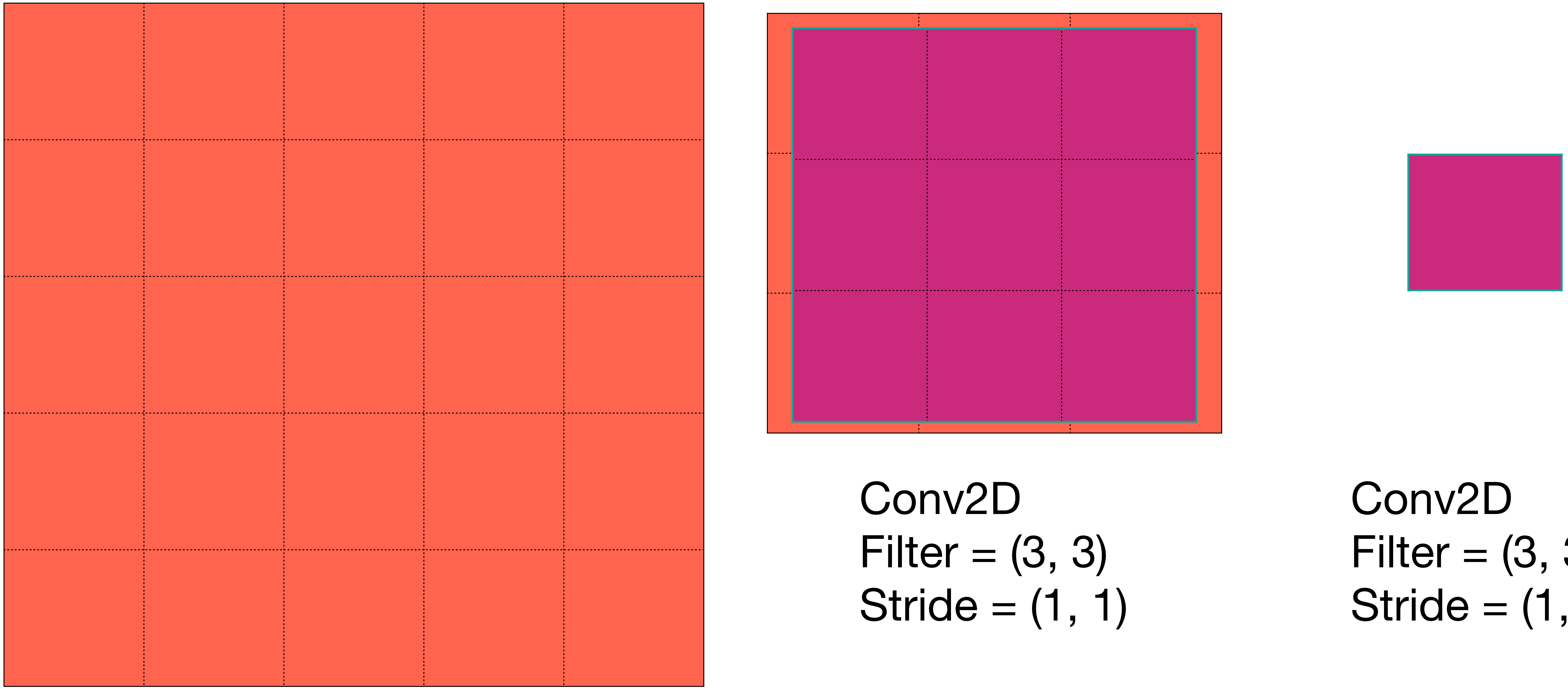
Receptive fields

Receptive field: 3x3



Conv2D
Filter = (3, 3)
Stride = (1, 1)

Receptive fields

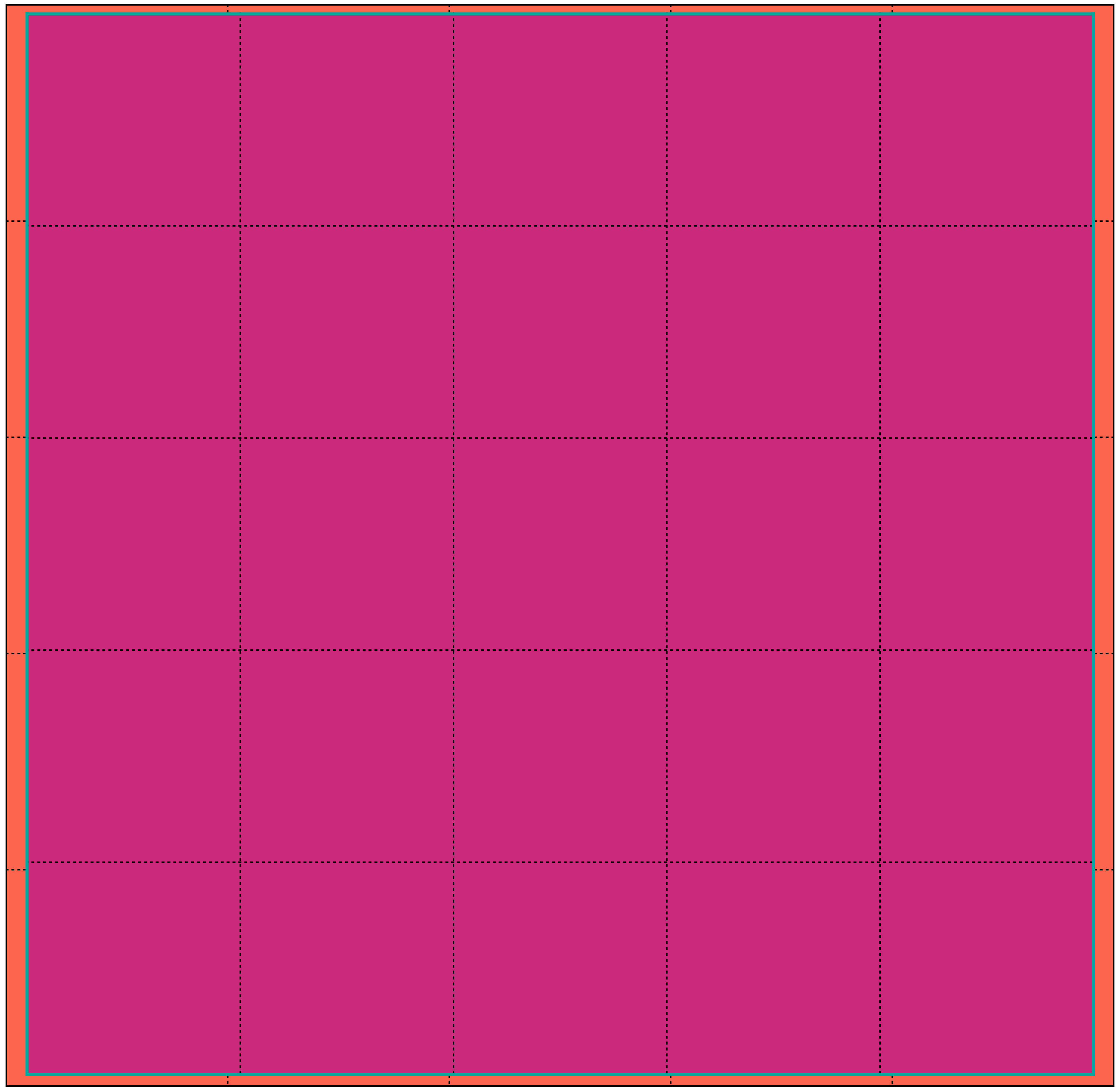


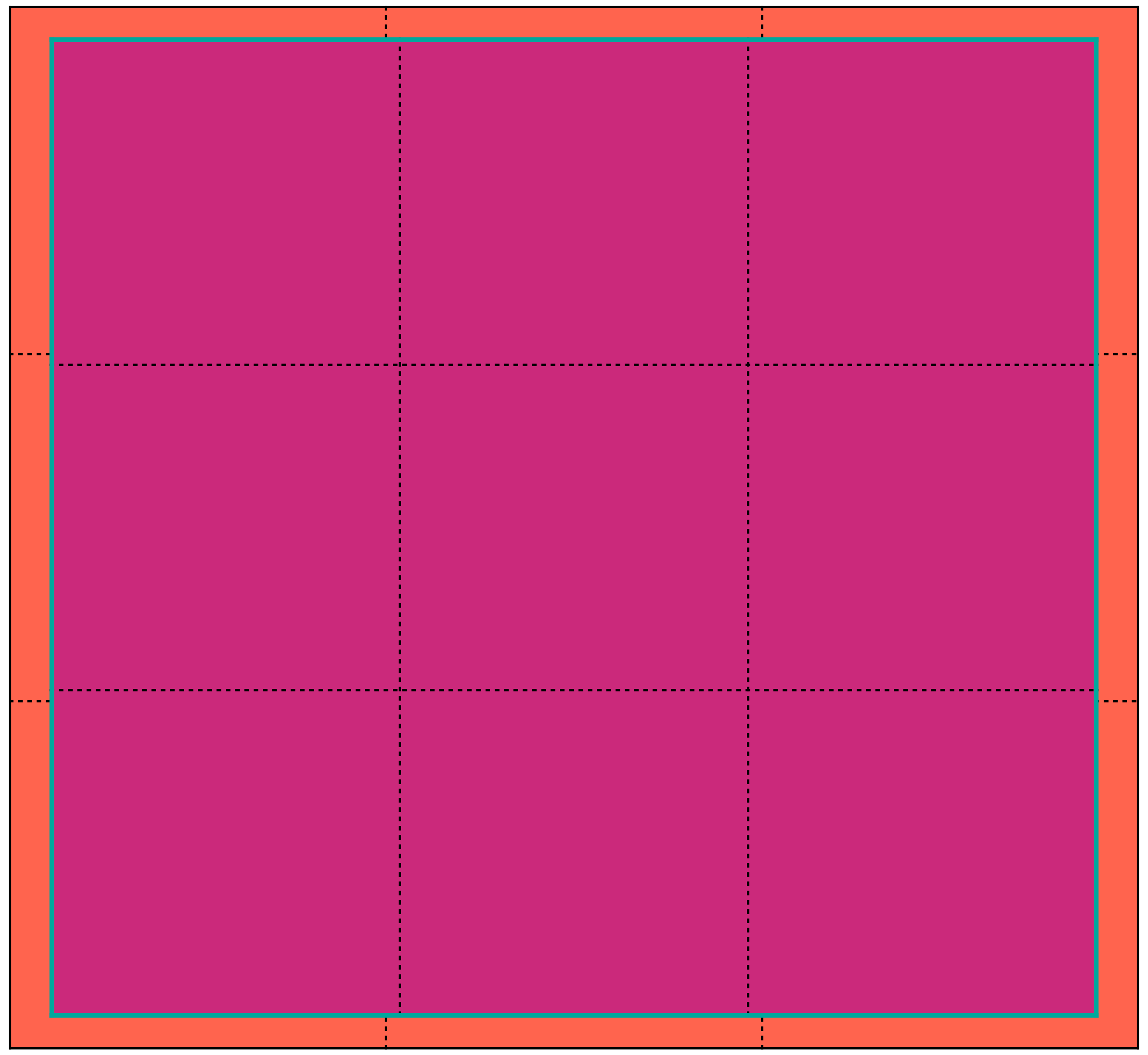
Conv2D
Filter = (3, 3)
Stride = (1, 1)

Conv2D
Filter = (3, 3)
Stride = (1, 1)

Receptive fields

Original receptive field: 5x5





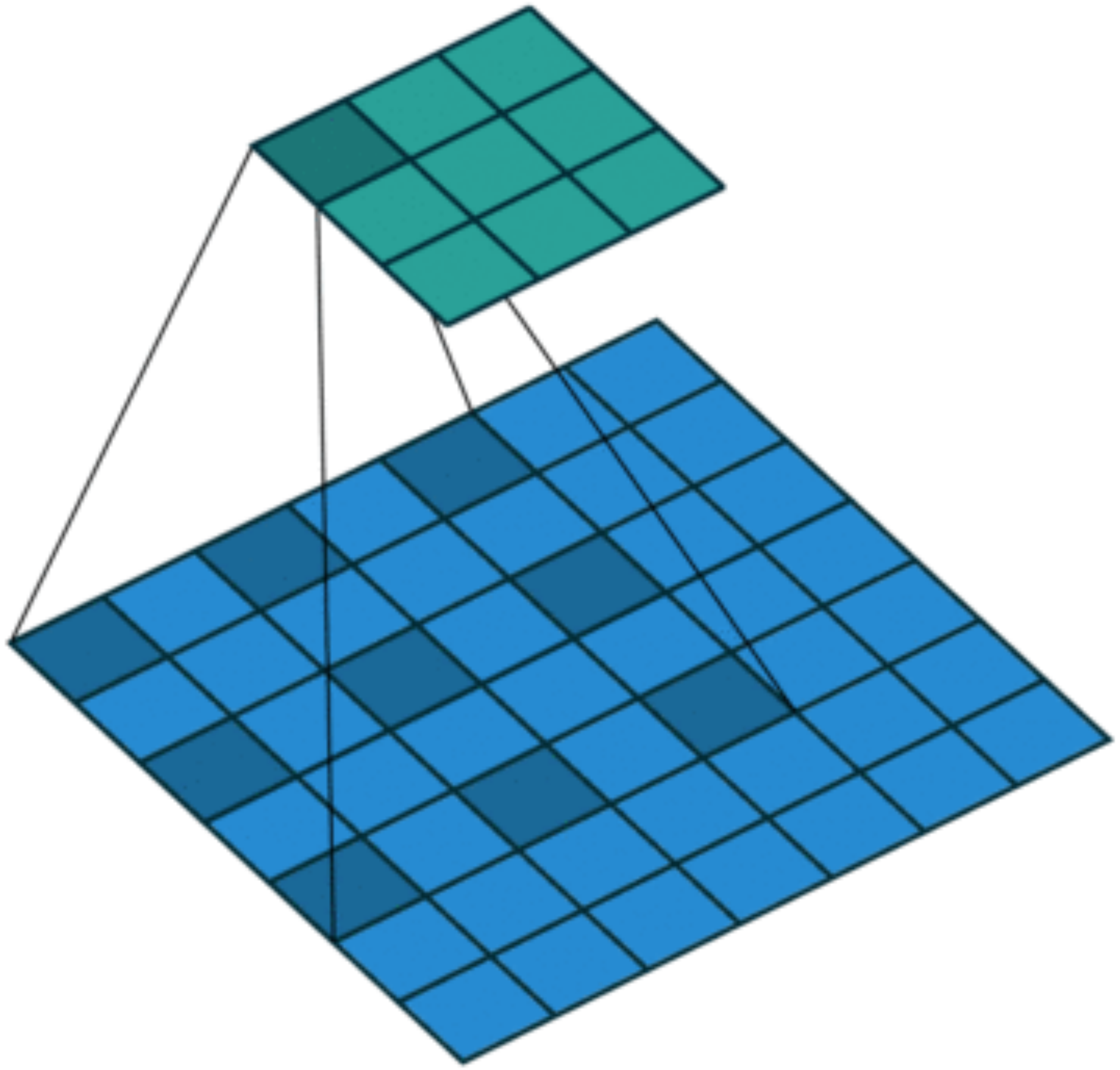
Conv2D
Filter = (3, 3)
Stride = (1, 1)

Conv2D
Filter = (3, 3)
Stride = (1, 1)

Receptive fields

- Stacking convolutions one after the other increases the original receptive field: two (3, 3) convs get to a (5, 5) receptive field
  - (and tend to perform better than a single (5, 5) conv)
  - (with fewer parameters!)

Dilated Convolution



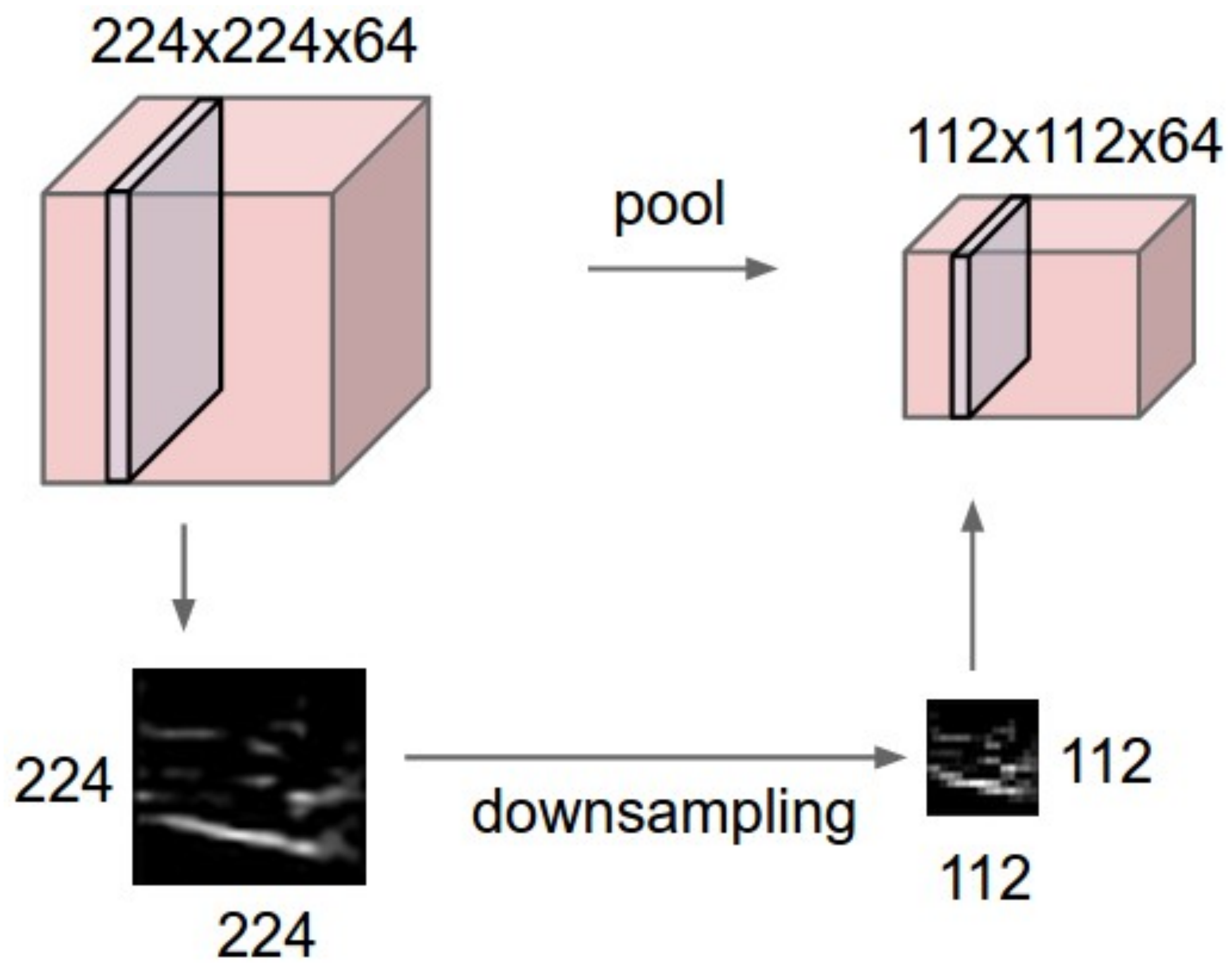
- Dilated convolutions can “see” a greater portion of the image by skipping pixels
- The (3, 3) 1-dilated convolution illustrated here has a (5, 5) receptive field
- Stacking dilated convolutions up quickly gets to large receptive fields

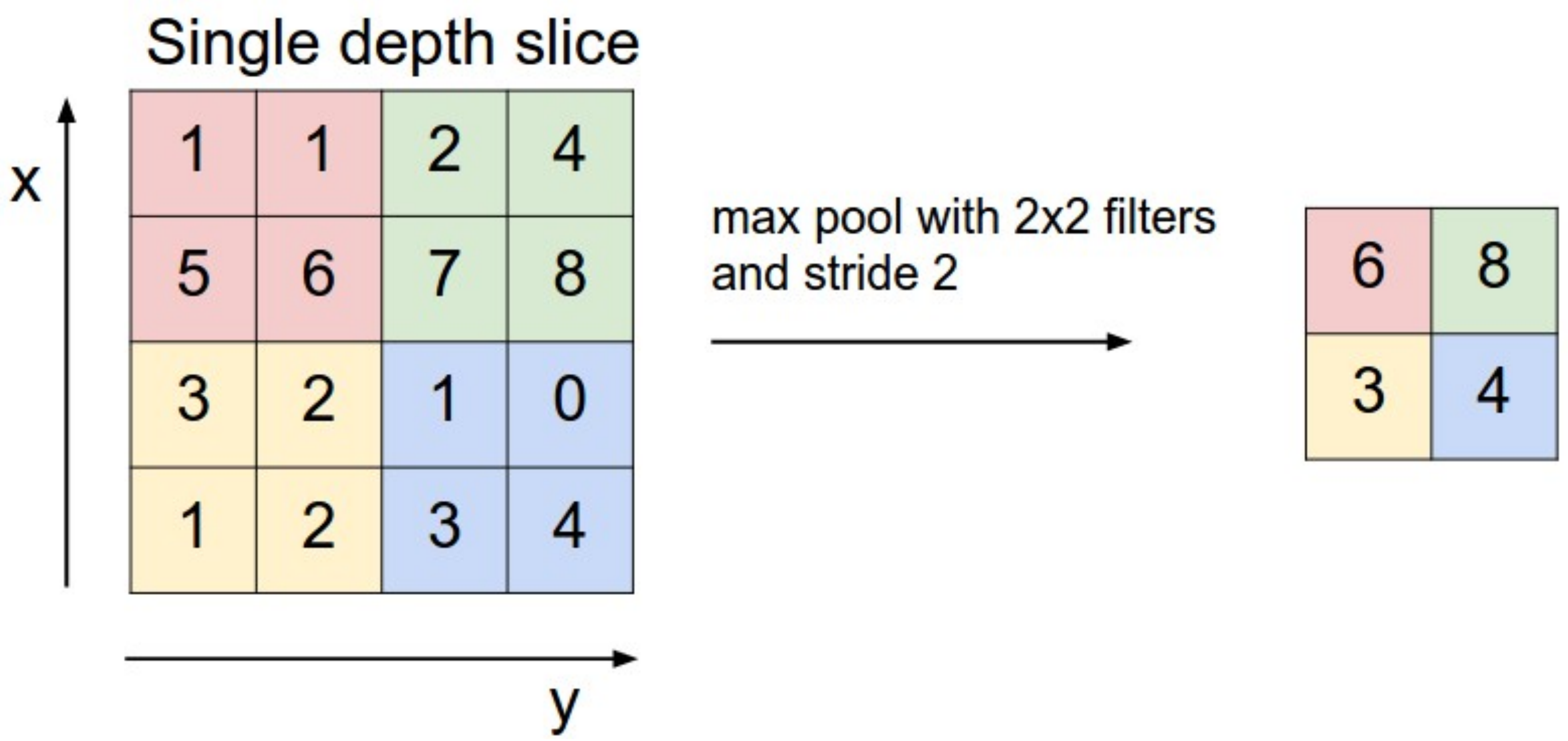
https://github.com/vdumoulin/conv_arithmetic

Other important ConvNet operations

- Increasing the receptive field
  (dilated convolutions)
- **Decreasing the size of the tensor**
  - Pooling
  - 1x1-convolutions

Pooling

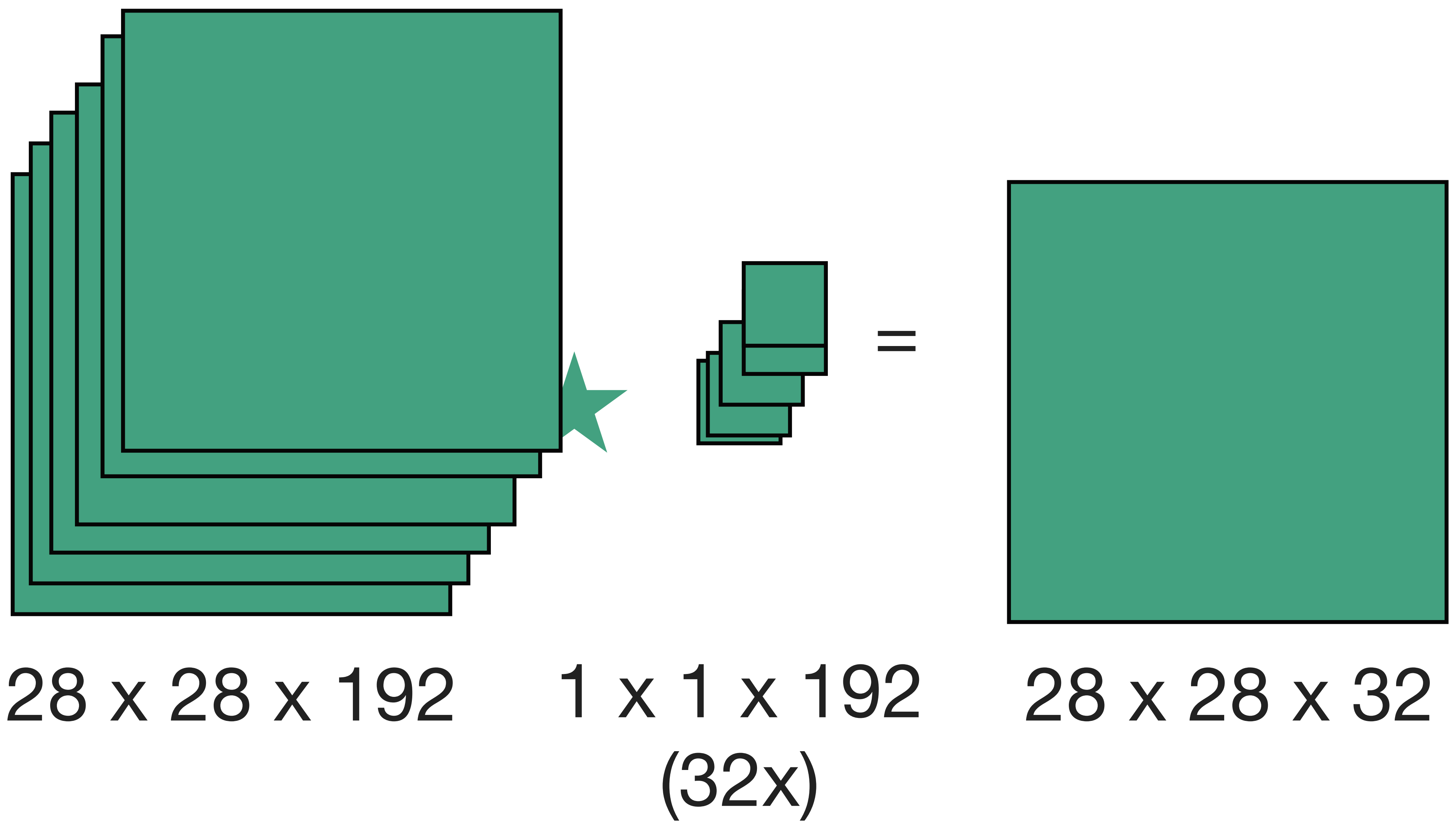




- Subsamples the image through average or max of region
- 2x2 max pooling is most common
- Recently fallen out of favor

http://cs231n.github.io/convolutional-networks/#pool

1x1 Convolution



- A way to reduce the "depth" dimension of convolutional outputs
- Corresponds to applying an MLP to every pixel in the convolutional output
- Crucial to popular convnet architectures like Inception (GoogleNet)

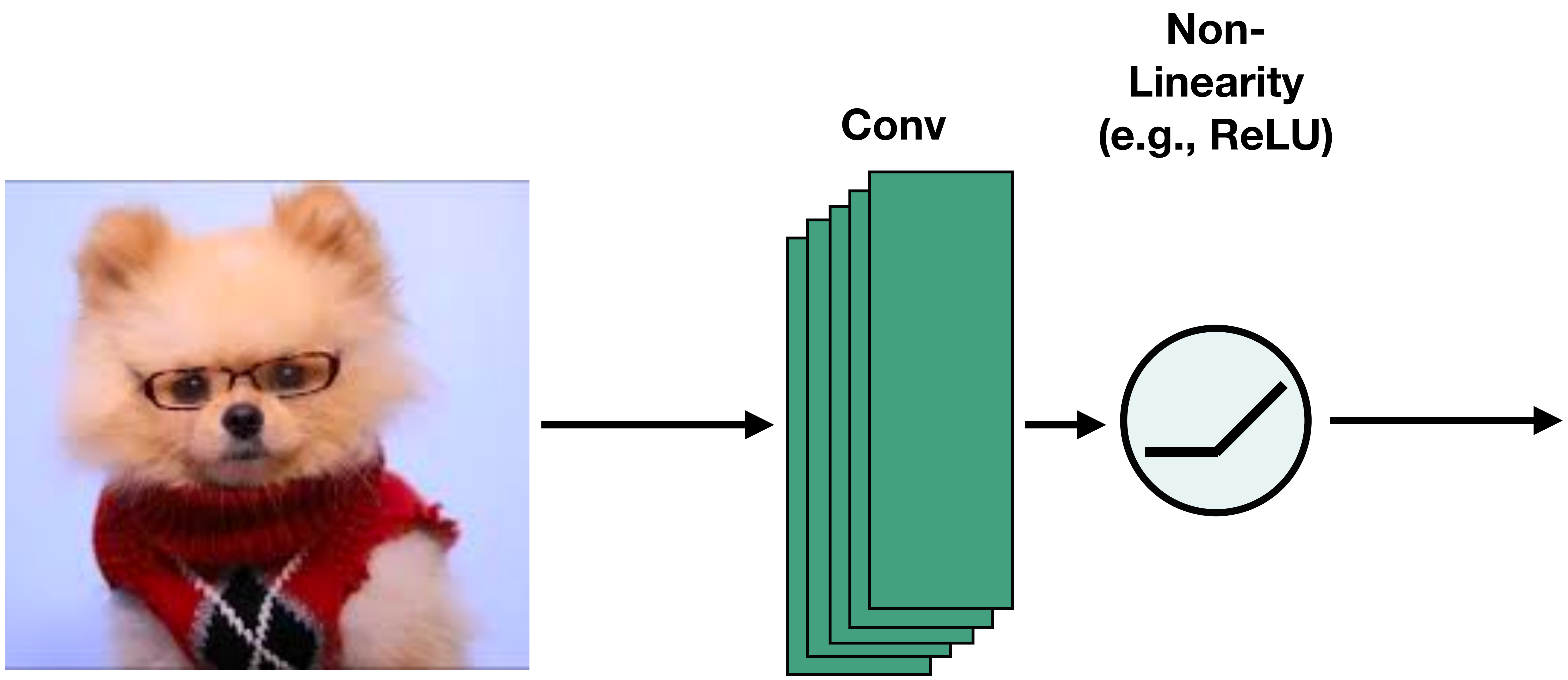
https://www.slideshare.net/TessFerrandez/notes-from-coursera-deep-learning-courses-by-andrew-ng

Questions?

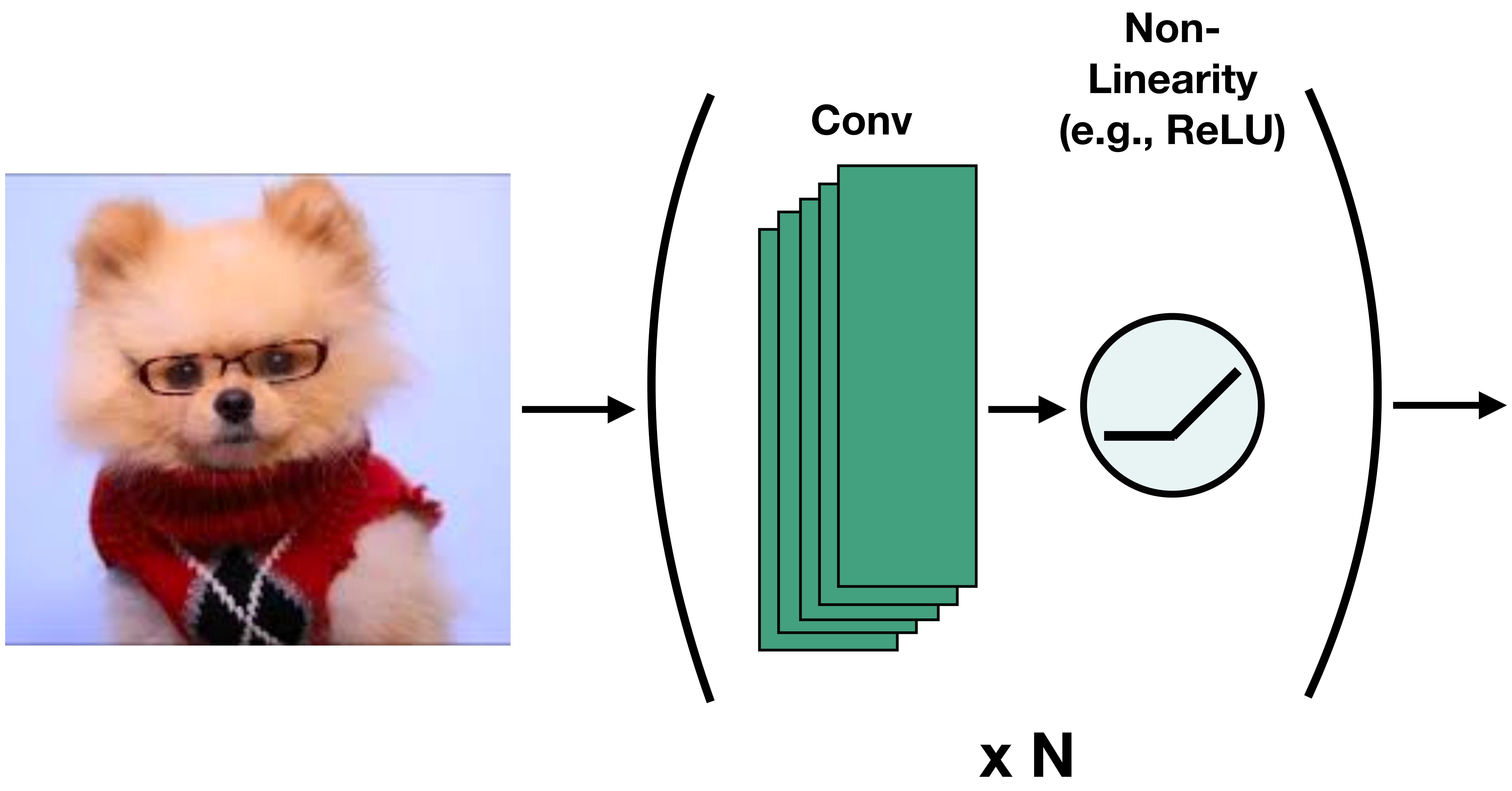
Agenda

1. Review of the convolution operation
2. Other important operations for ConvNets
3. **Classic ConvNet architectures**

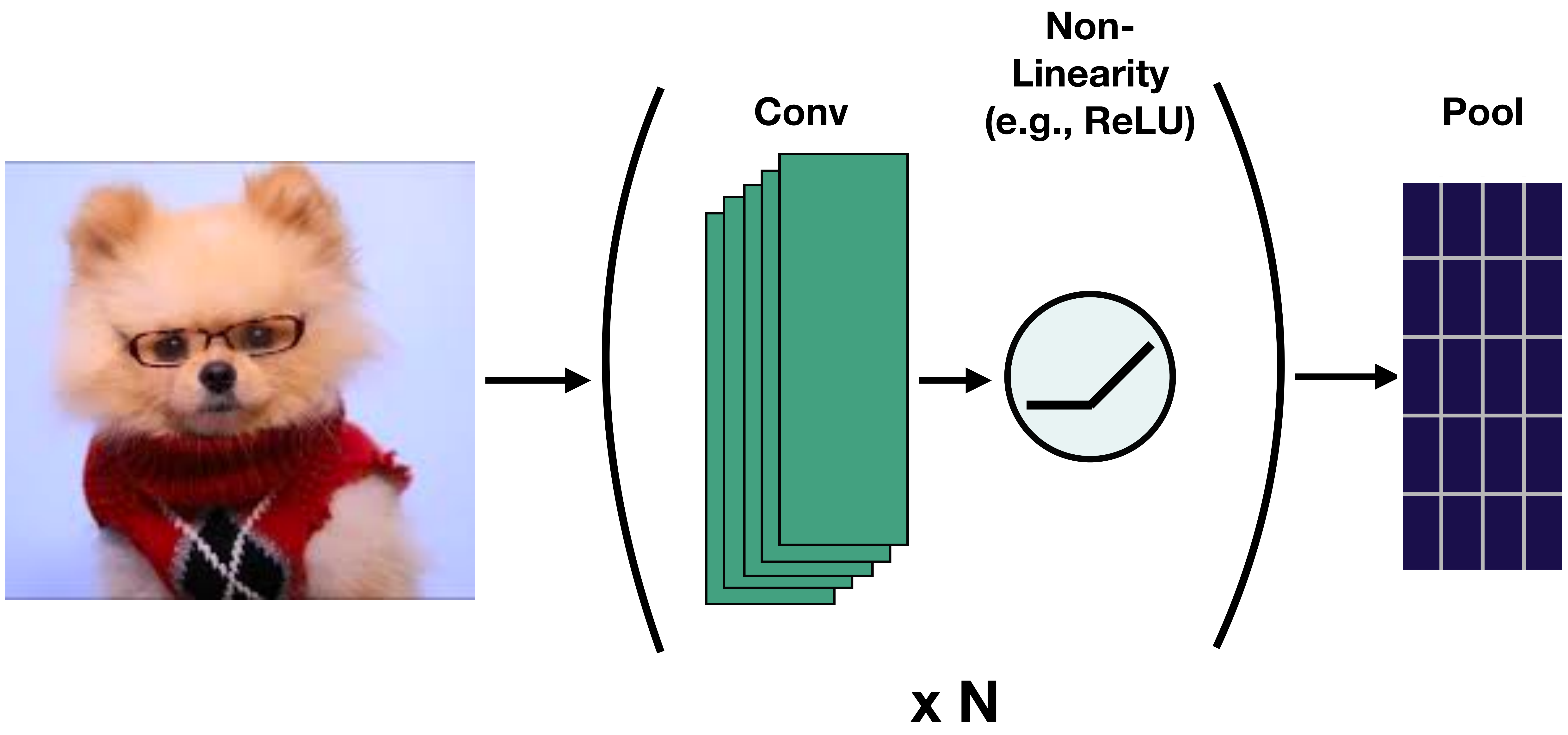
Most standard: LeNet(-like) architectures



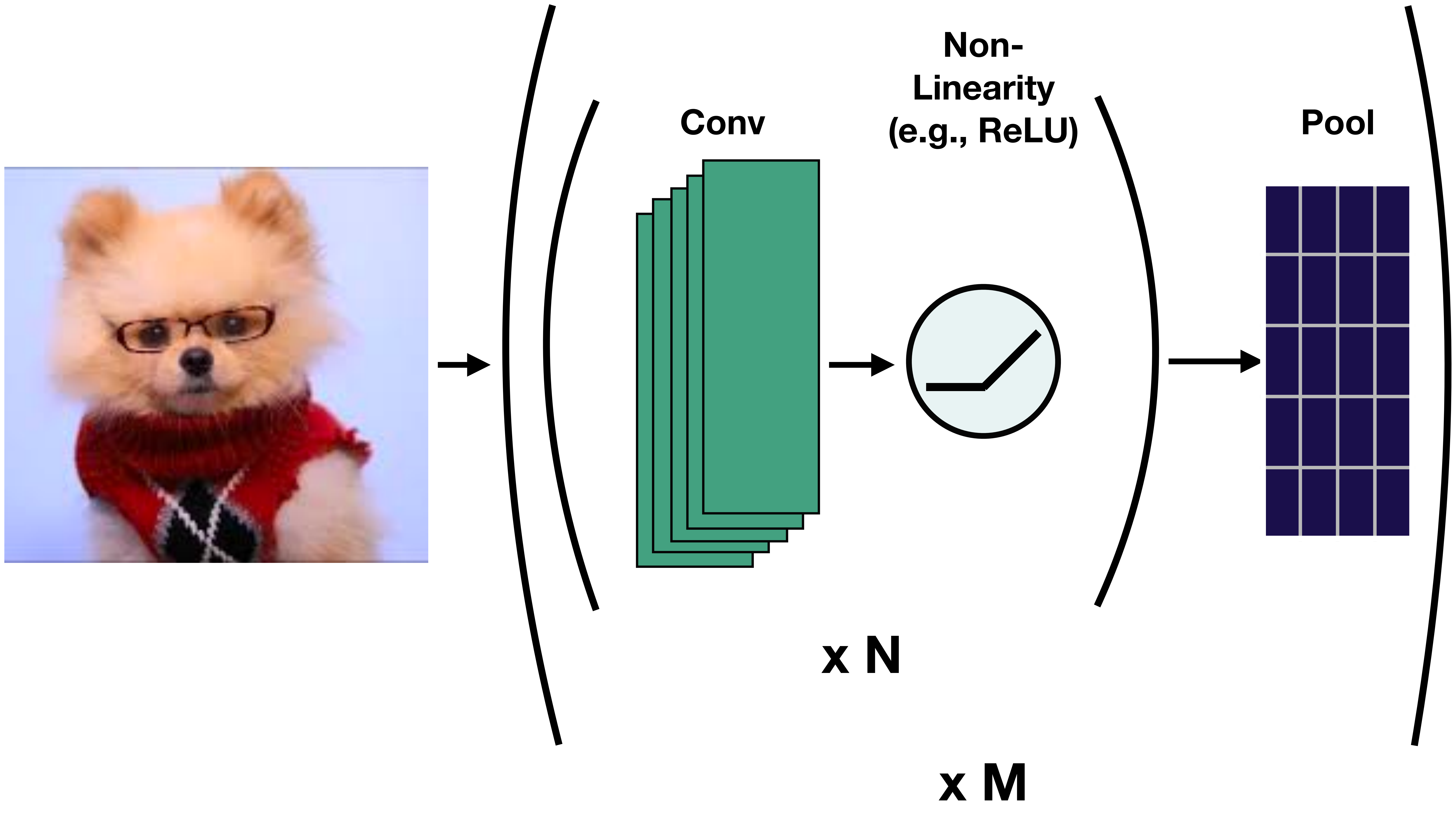
Most standard: LeNet(-like) architectures



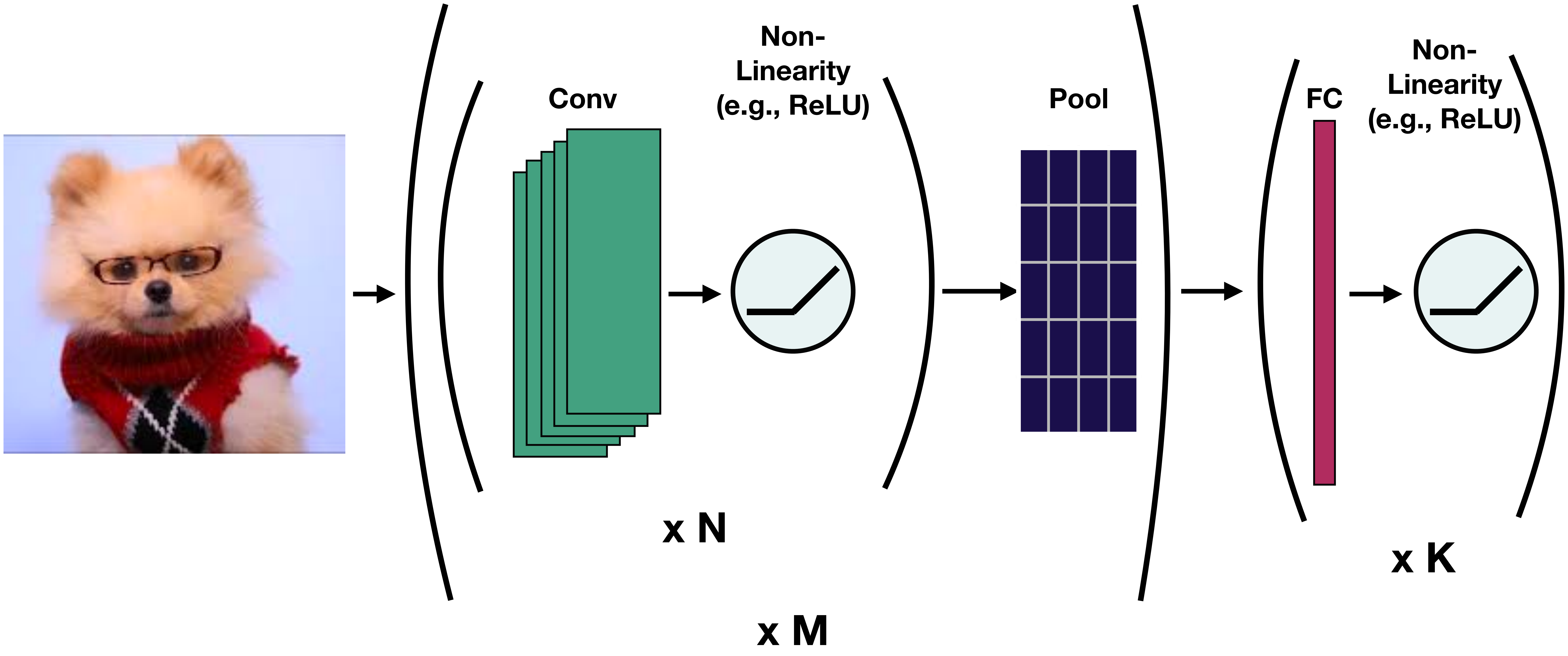
Most standard: LeNet(-like) architectures



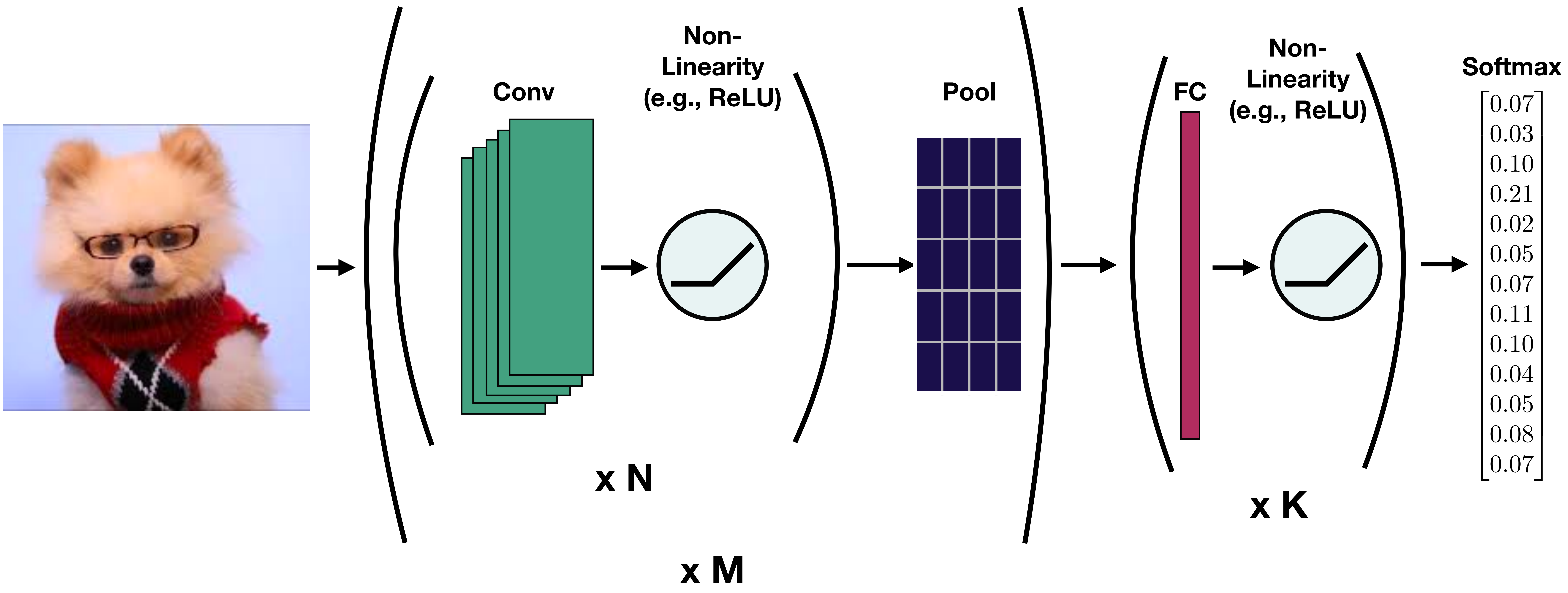
Most standard: LeNet(-like) architectures



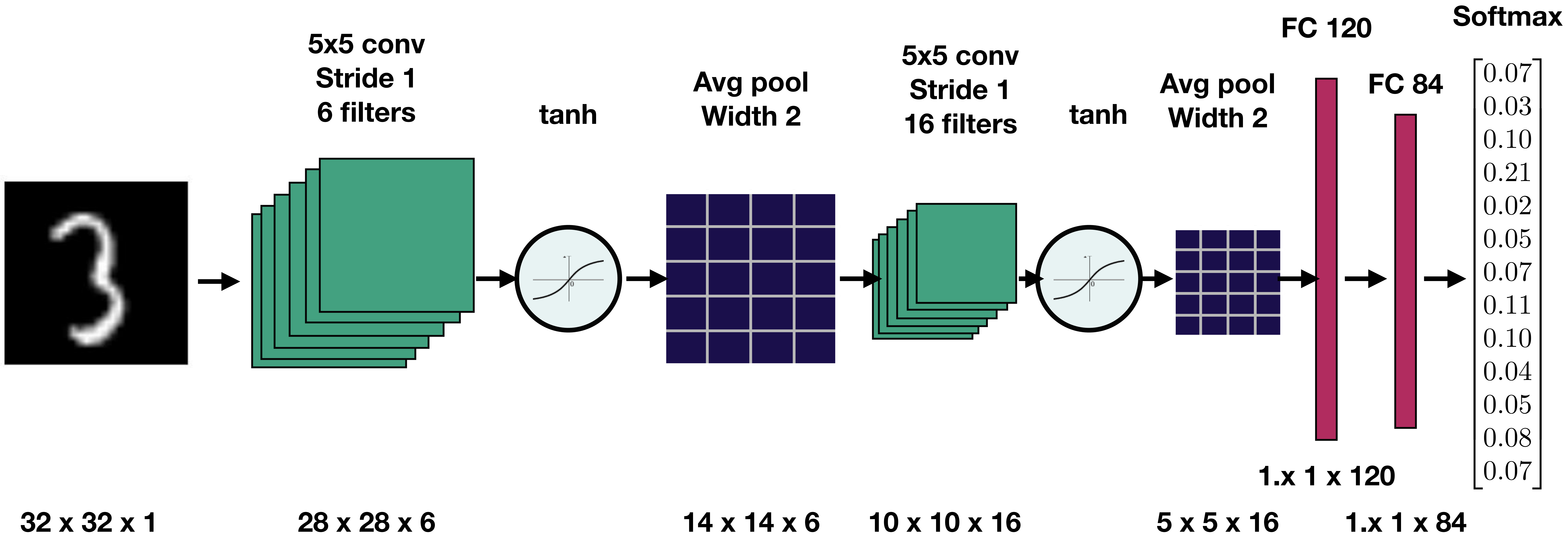
Most standard: LeNet(-like) architectures



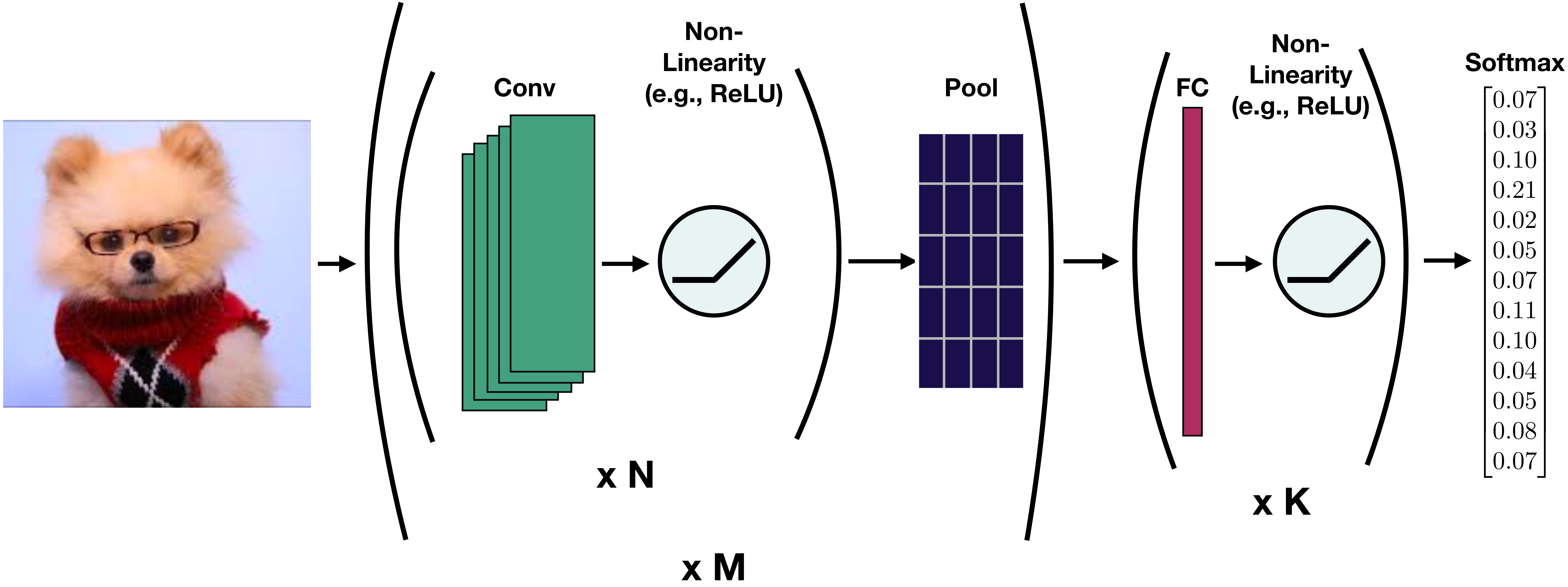
Most standard: LeNet(-like) architectures



Classic Convnet Architecture: LeNet



Most standard: LeNet(-like) architectures



More modern convnet architectures in the Vision Applications lecture!

Questions?