CS189 Summer 2018

Convolutional Neural Networks

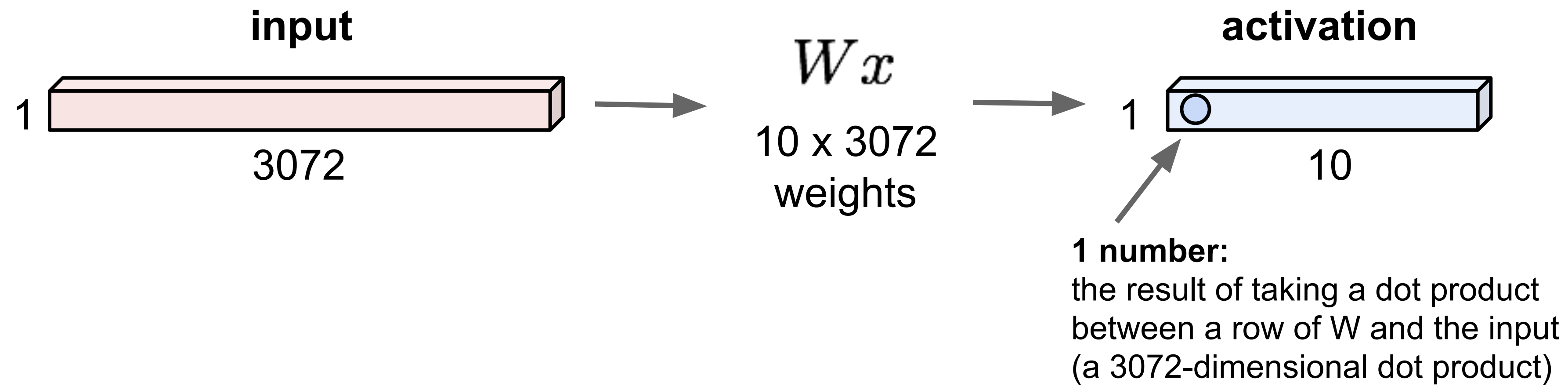
Josh Tobin

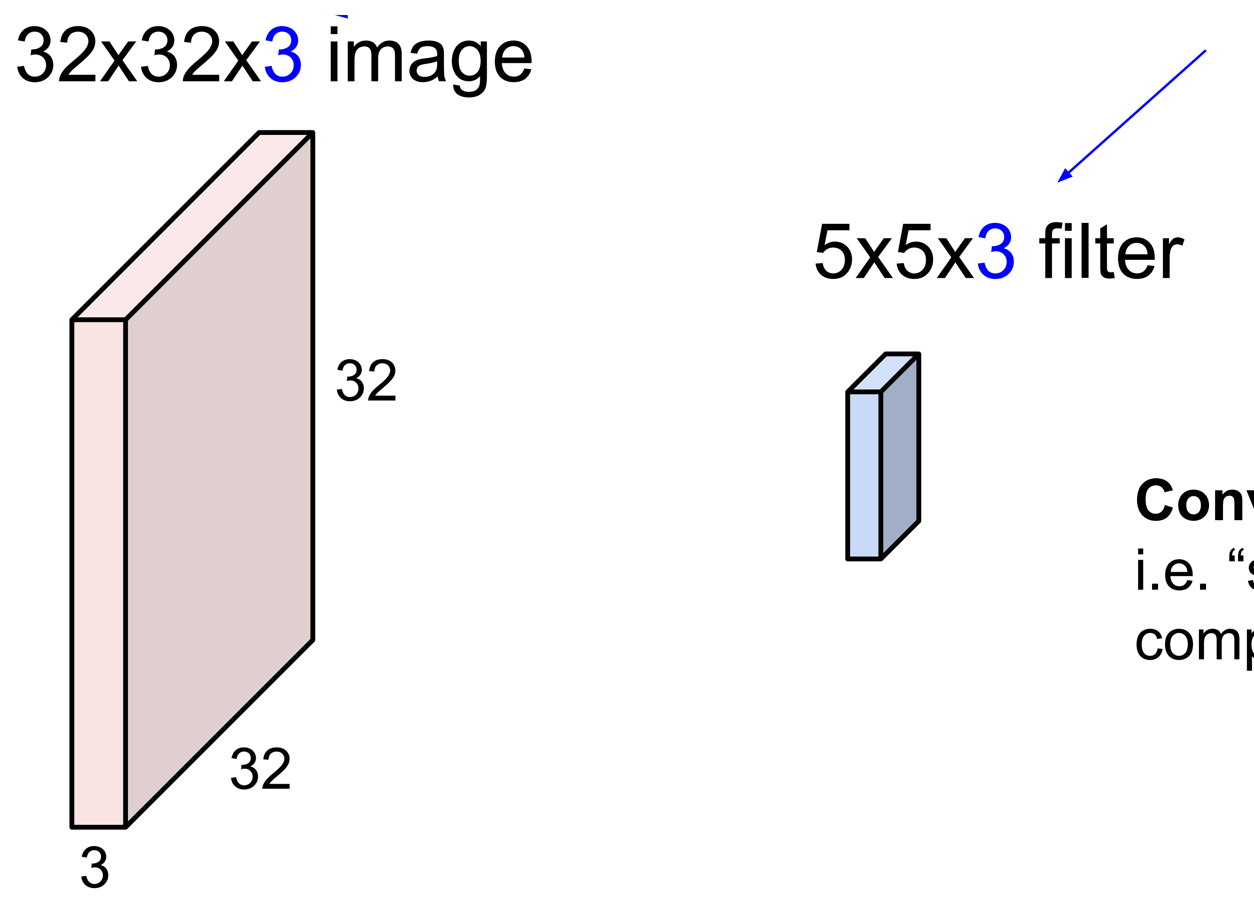
Agenda

1. **Overview of convolutions**
2. Other ConvNet operations
3. Basic ConvNet architectures

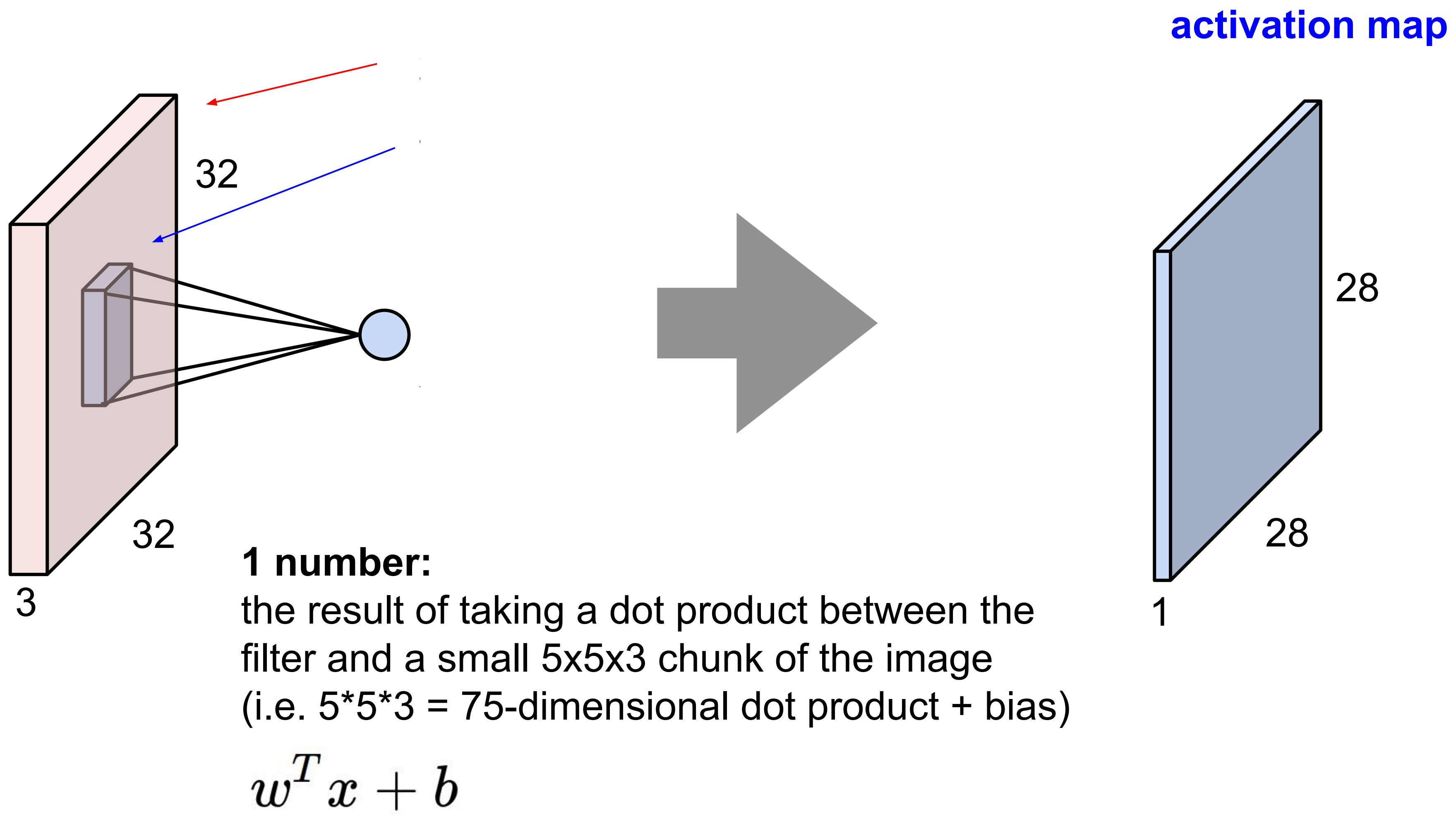
Why convolutions?

32x32x3 image -> stretch to 3072 x 1



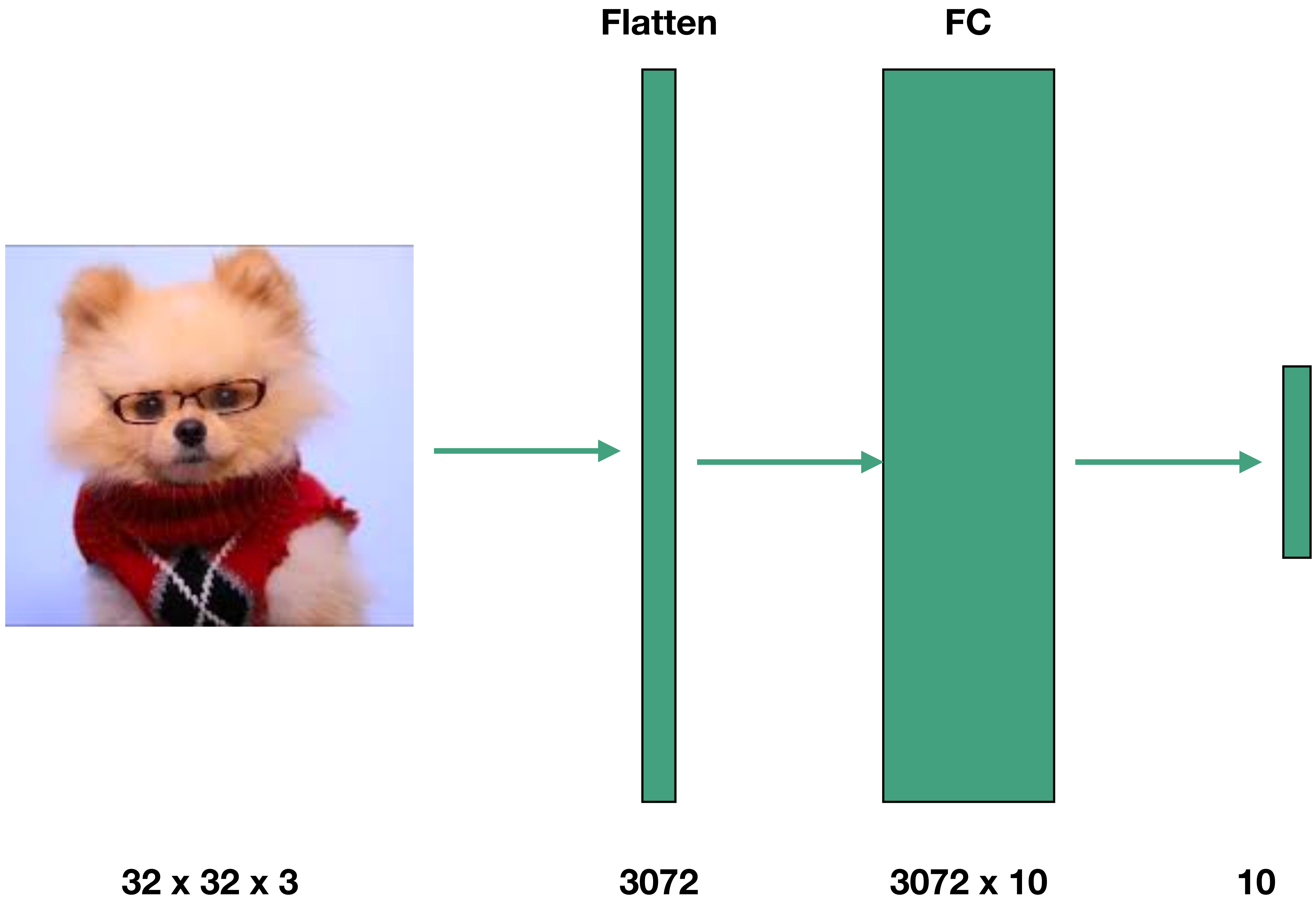


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

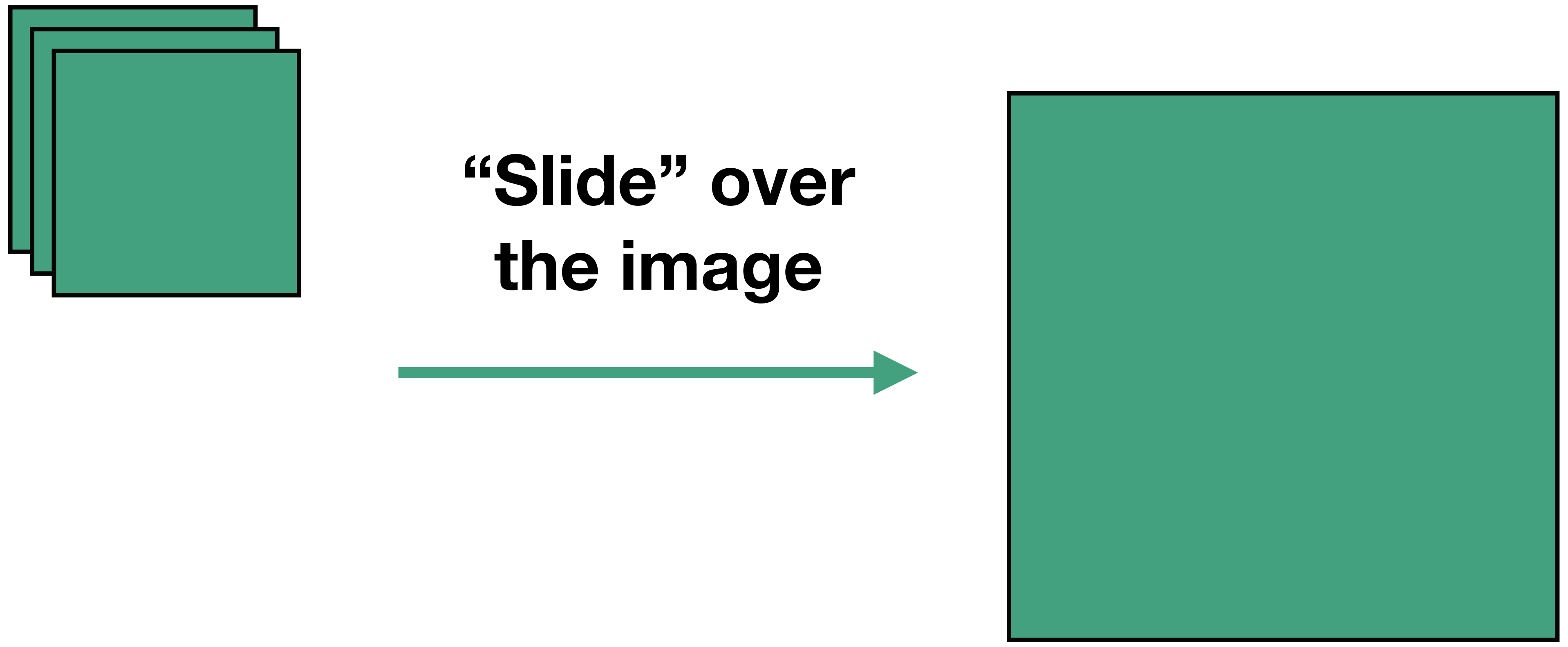


http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

Fully connected vs Conv



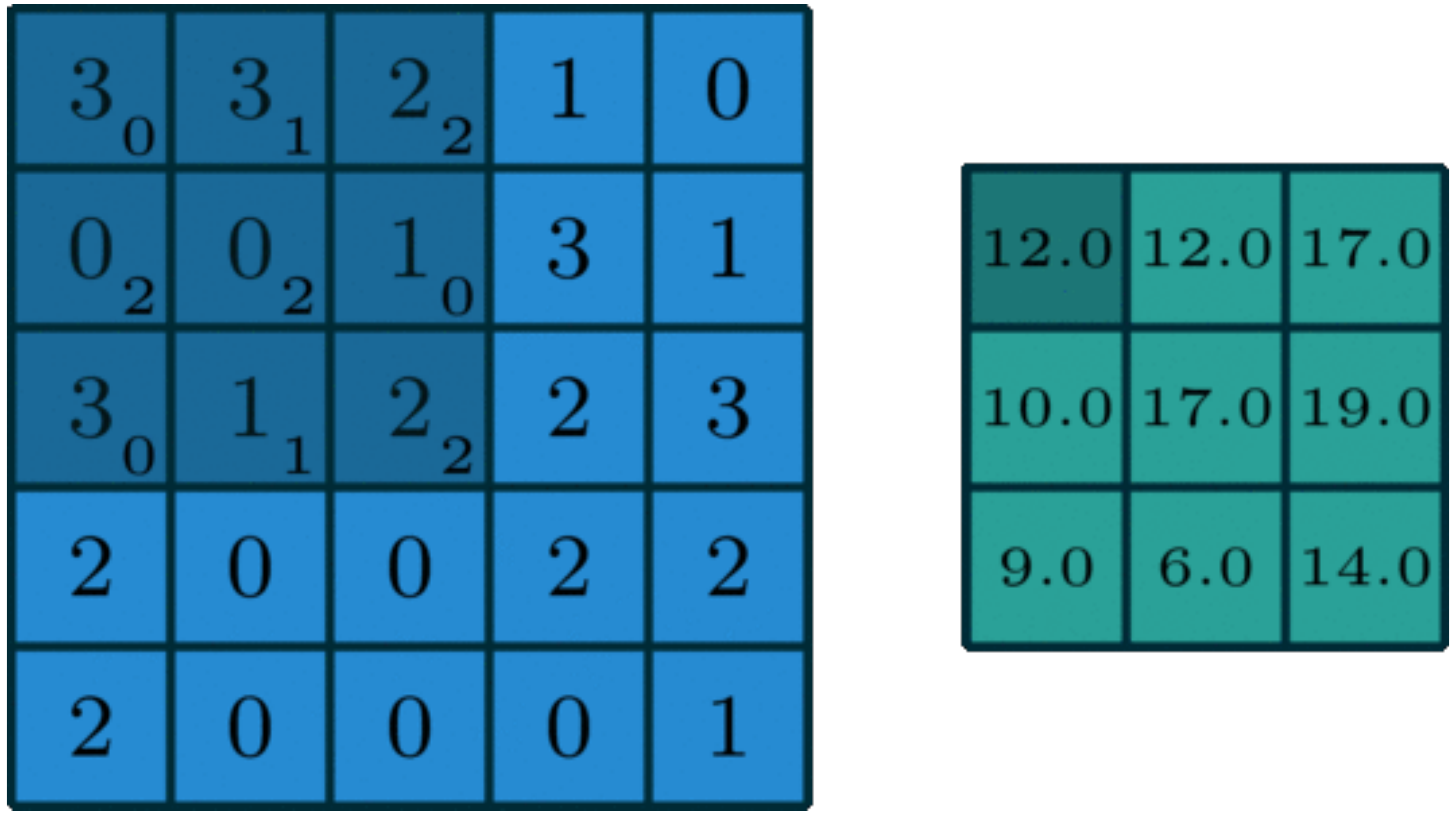
Fully connected vs Conv



32 x 32 x 3

3072

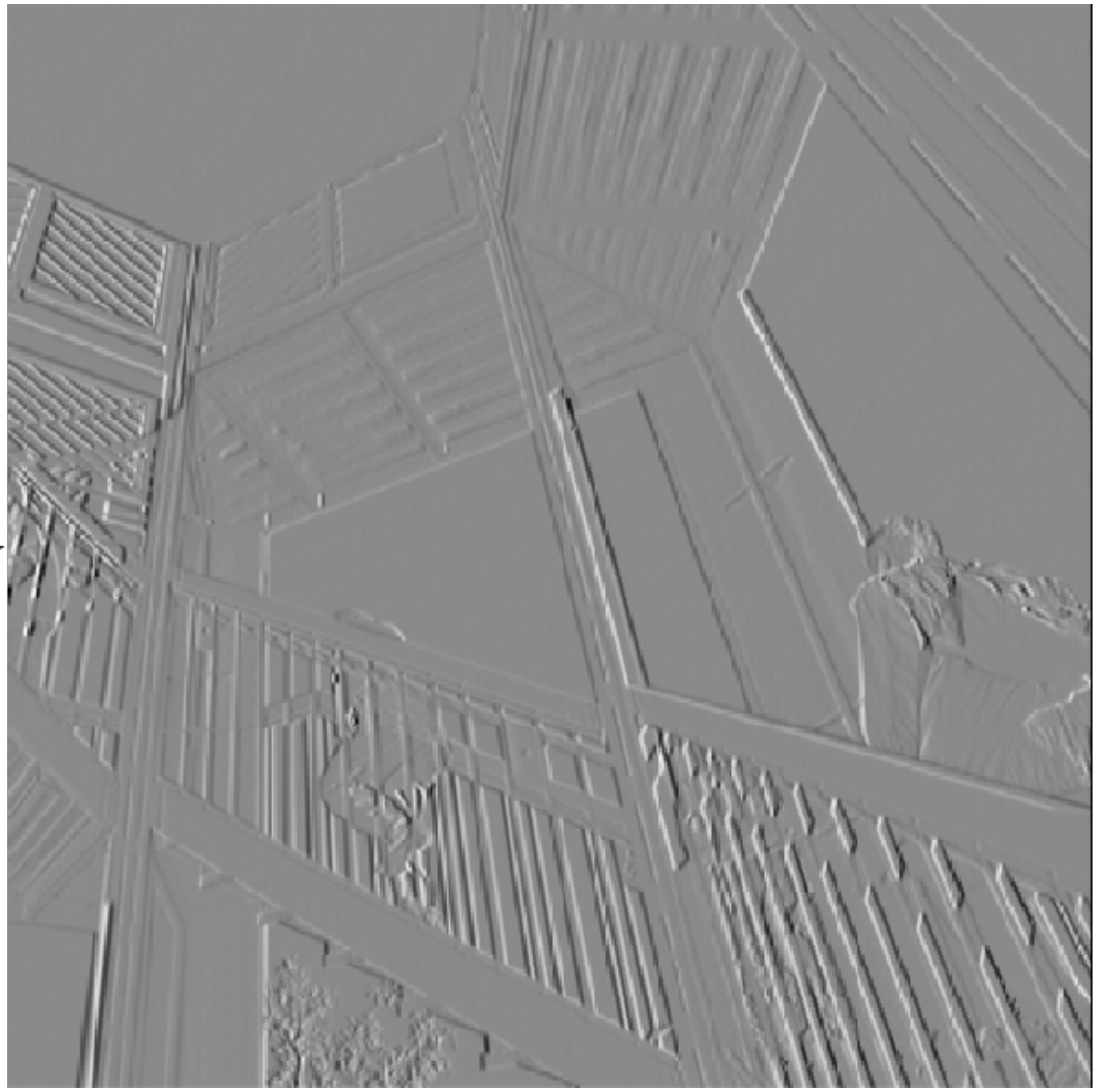
The convolution operation



What can a conv filter do?

$$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$$

Horizontal Sobel kernel



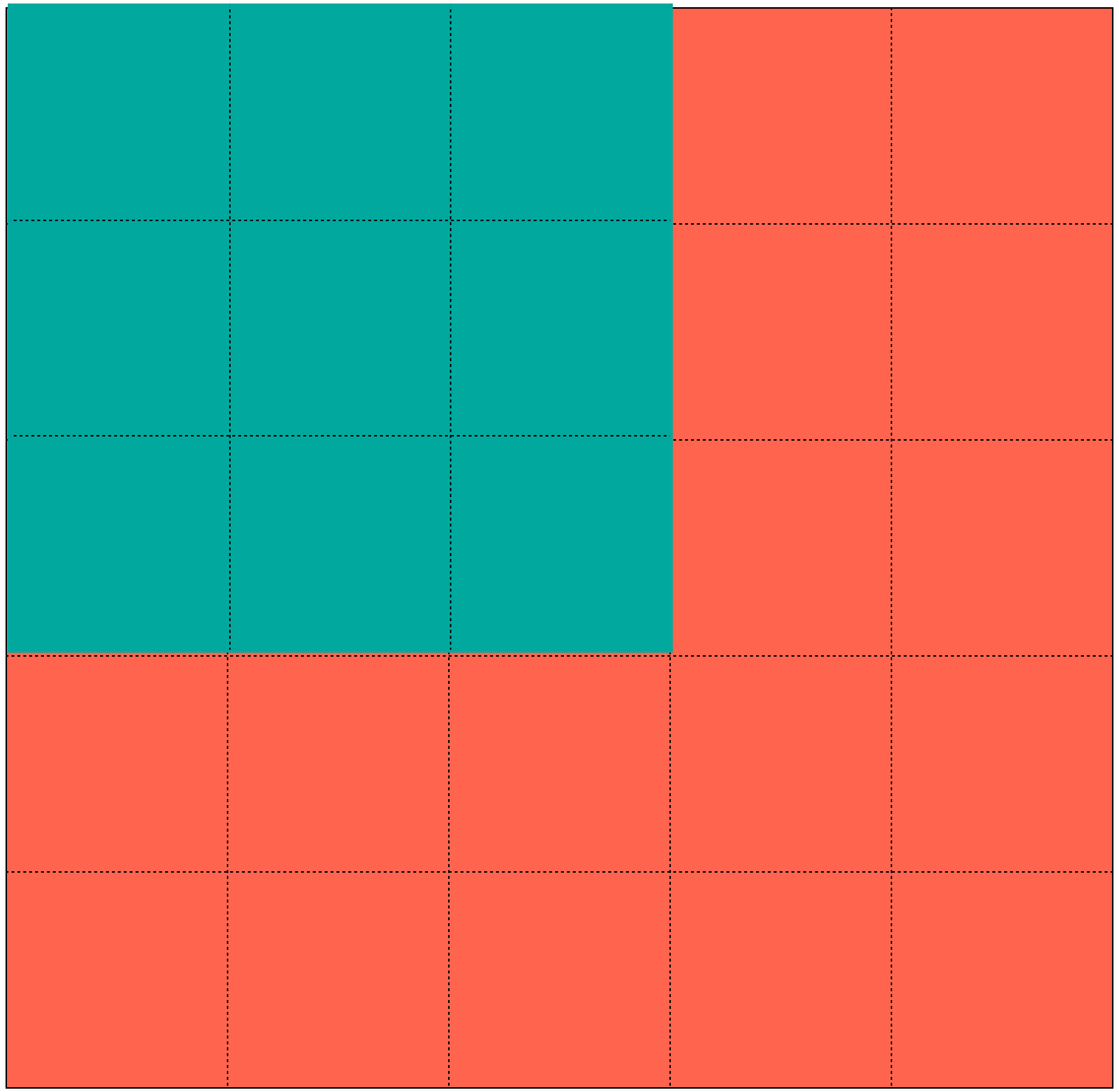
Instead of hard-coding the weights,
we can learn them!

https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

Strides

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

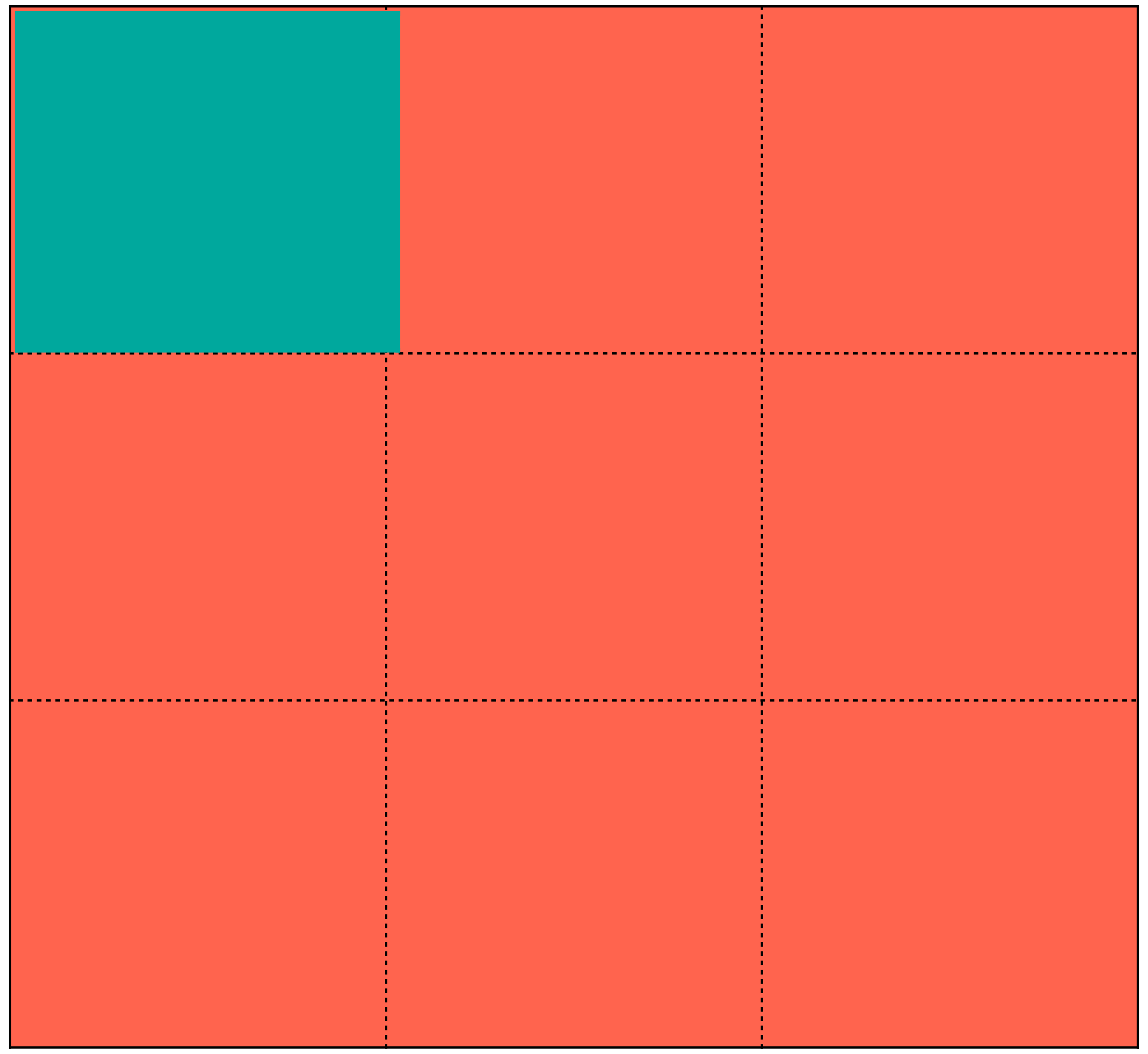
Strides



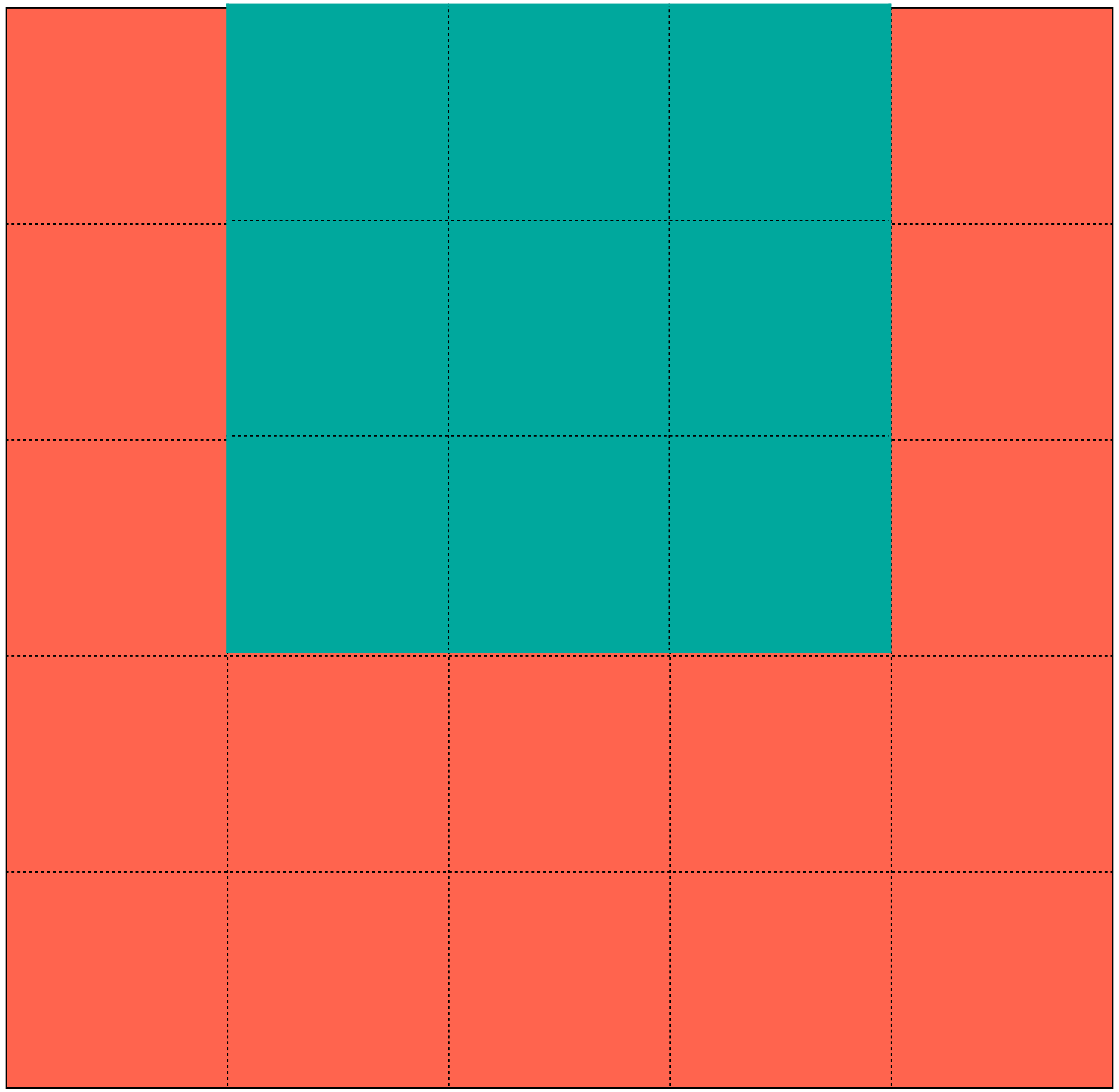
Conv2D

Filter = (3, 3)

Stride = (1, 1)



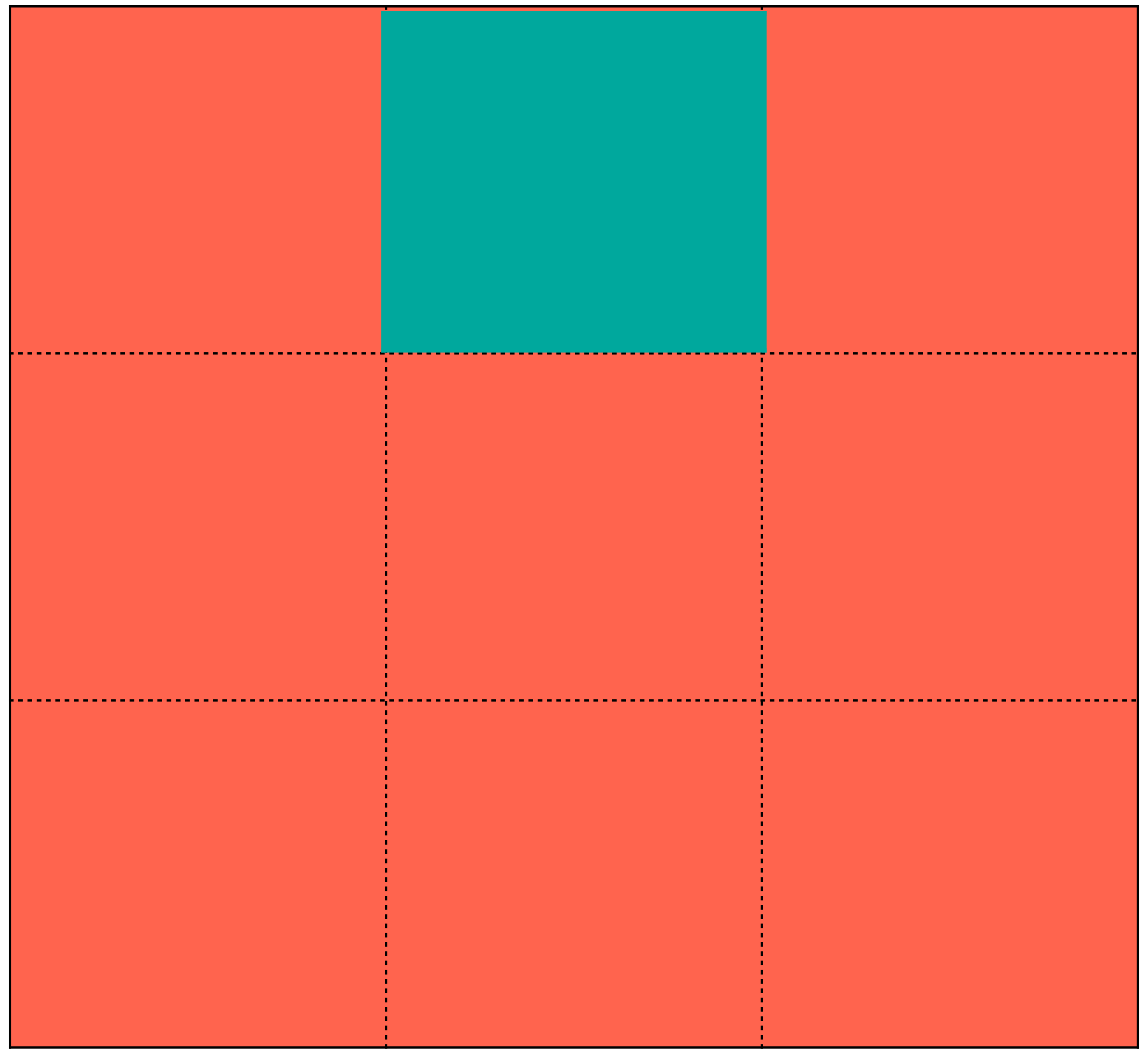
Strides



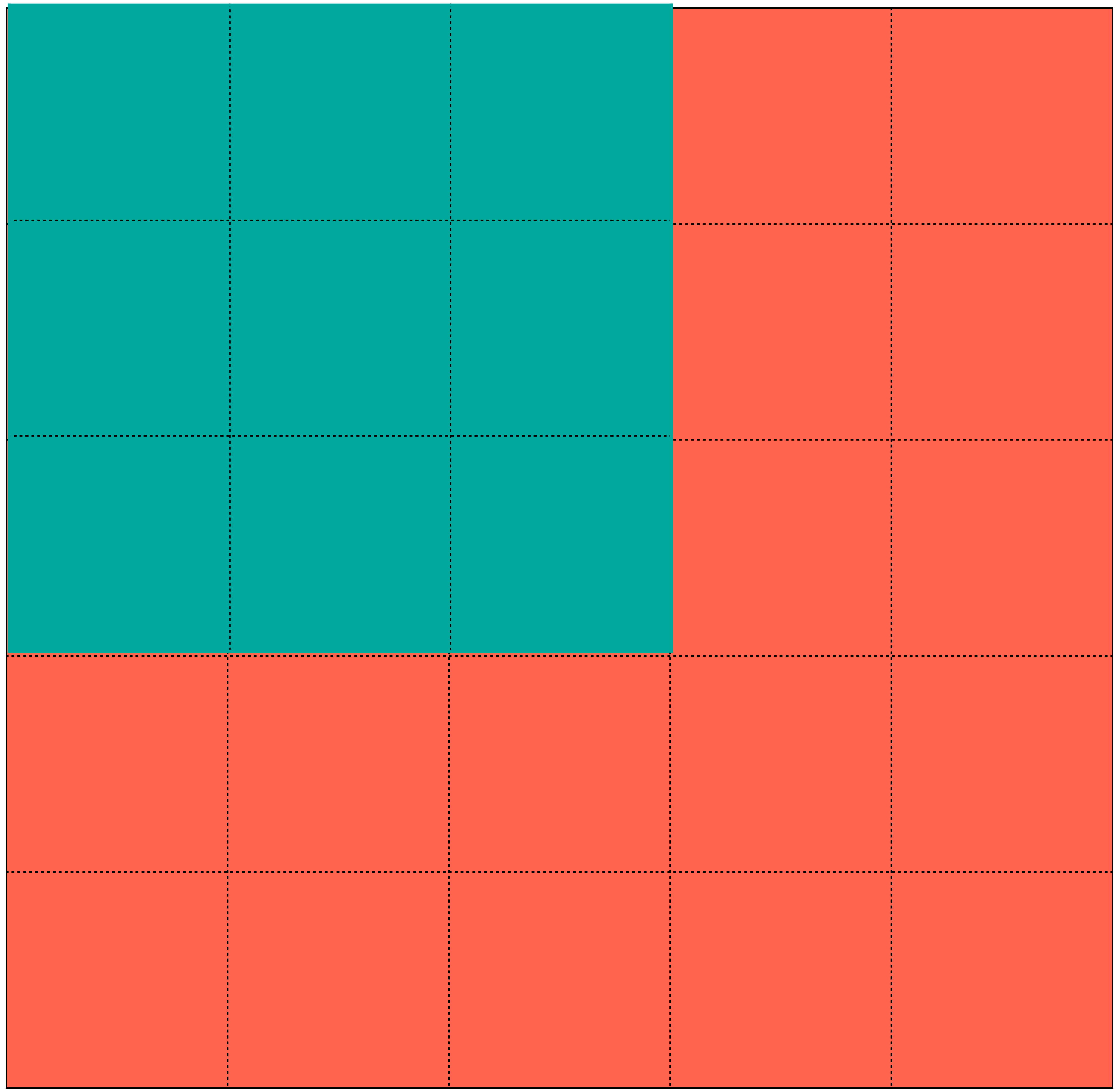
Conv2D

Filter = (3, 3)

Stride = (1, 1)



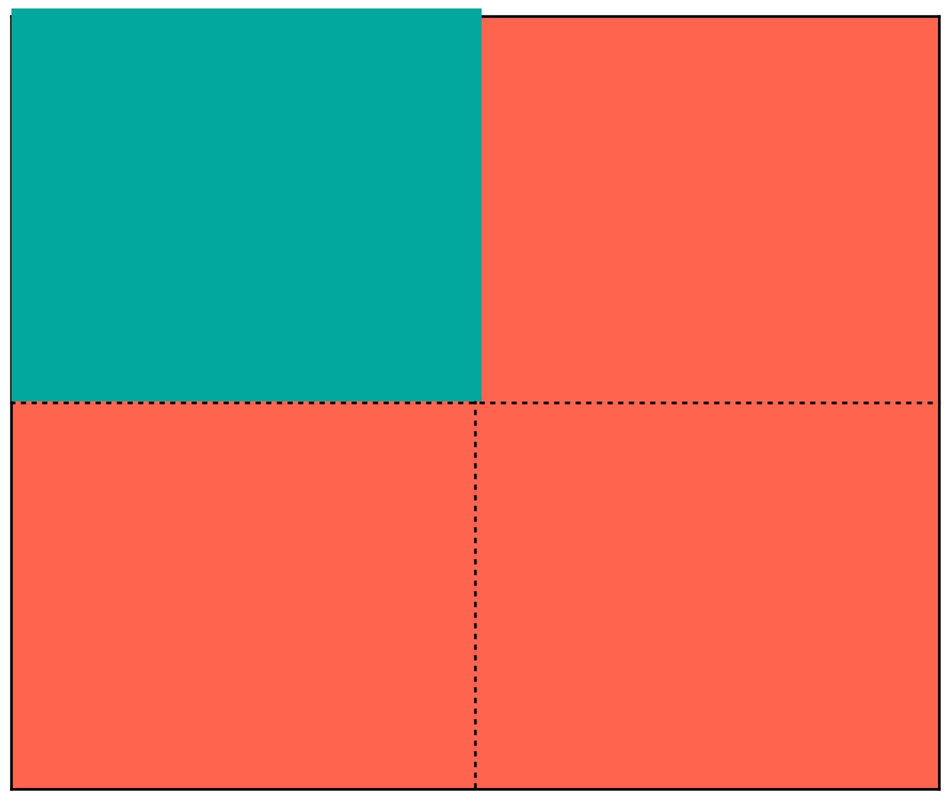
Strides



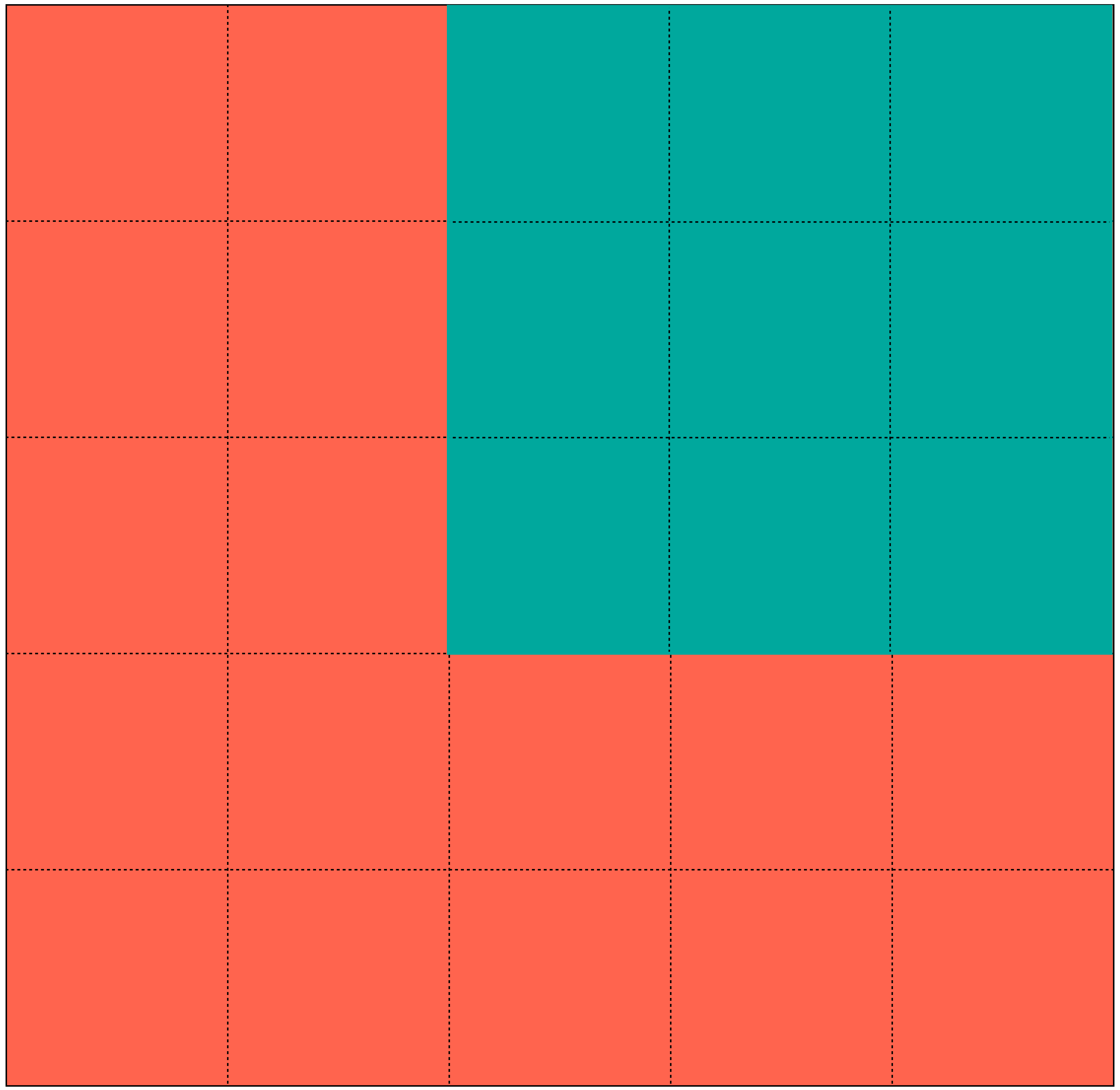
Conv2D

Filter = (3, 3)

Stride = (2, 2)



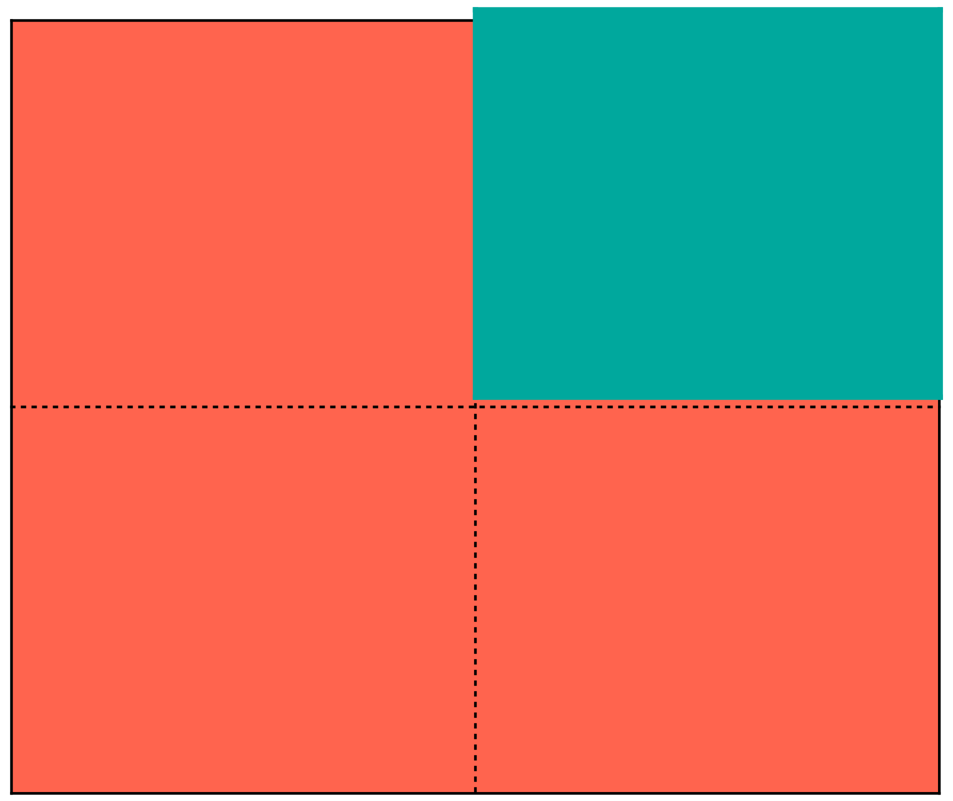
Strides



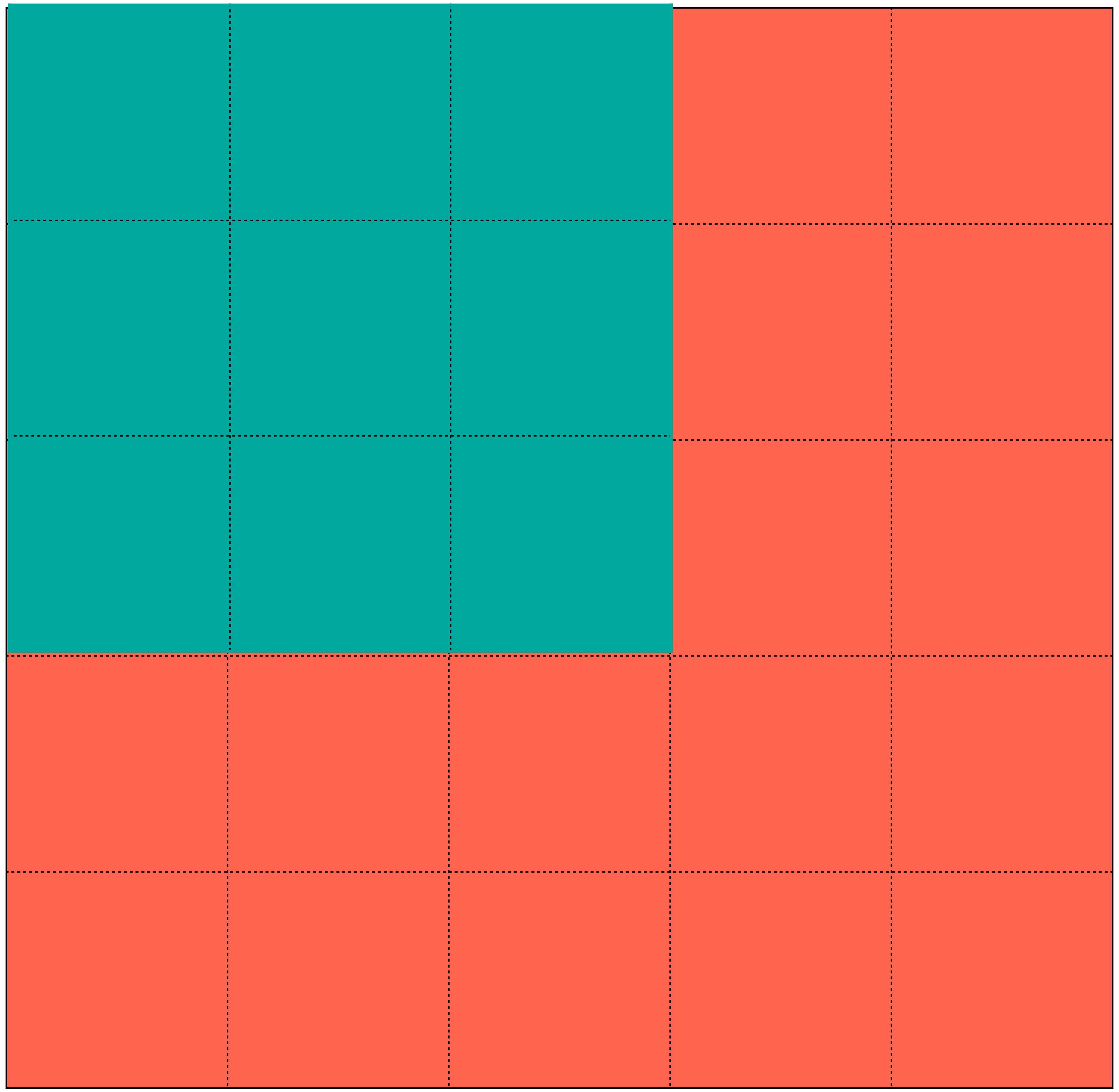
Conv2D

Filter = (3, 3)

Stride = (2, 2)



Strides



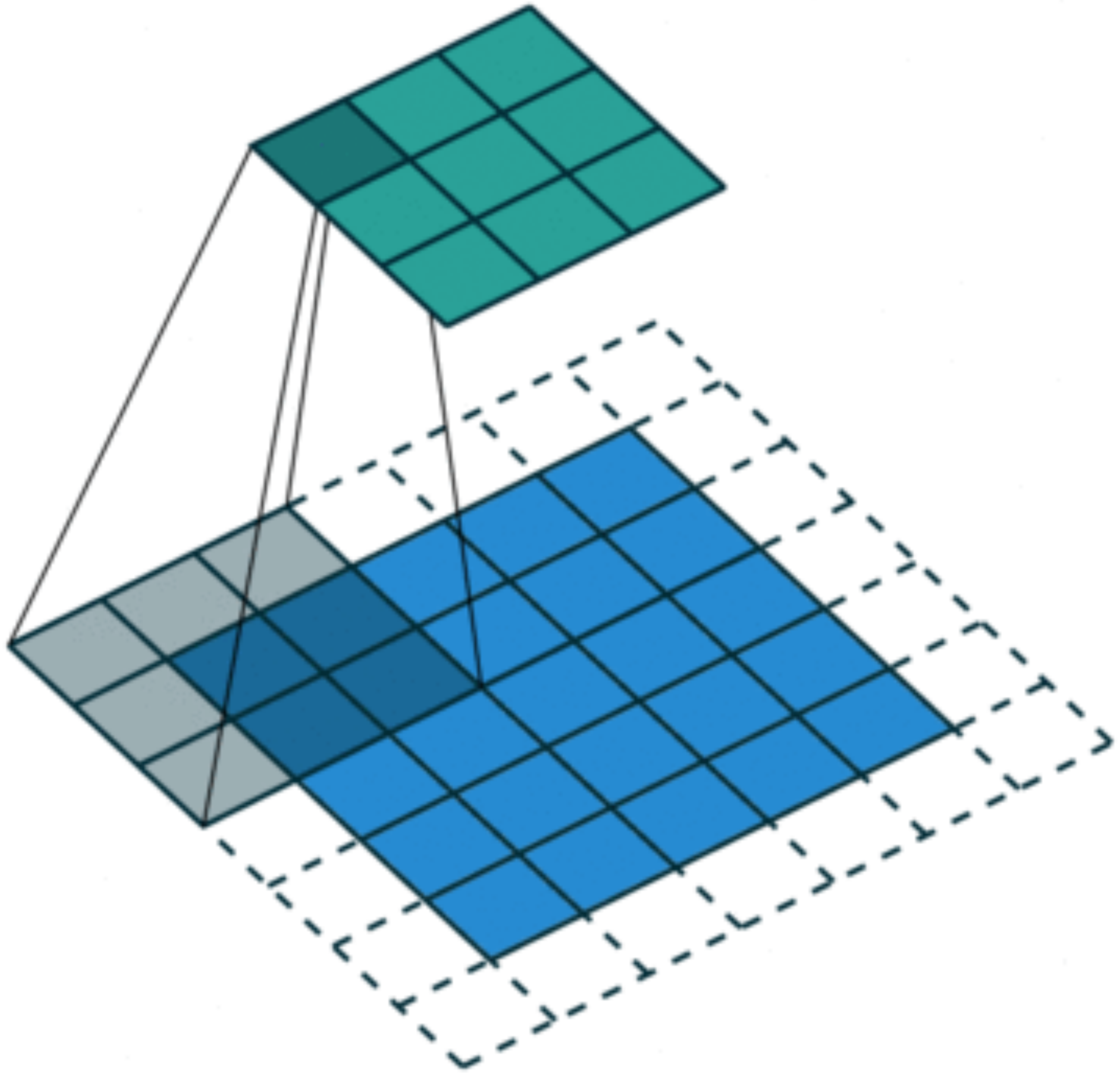
Conv2D

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

Conv2D Math

- Input: WxHxD volume
- Parameters:
  - K is the number of filters, each one of size (F_w, F_h)…
  - …moving at stride (S_w, S_h)
  - …with the input padded symmetrically with P all-zero rows/cols
- Output: W'xH'xK volume
  - W' = (W - F_w + 2P) / S_w + 1
  - H' = (H - F_h + 2P) / S_h + 1
- Each filter has (F_w * F_h * D) parameters, for K * (F_w * F_h * D) total in the layer

Conv2D Math

- Input: WxHxD volume
- Parameters:
  - K is the number of filters…
  - …each one of size (F_w, F_h)
  - …moving at stride (S_w, S_h)
  - …with the input padded symmetrically with P all-zero rows/cols
- Output: W’xH’xK volume
  - W’ = (W - F_w + 2P) / S_w + 1
  - H’ = (H - F_h + 2P) / S_h + 1
- Each filter has (F_w * F_h * D) parameters, for K * (F_w * F_h * D) total in the layer

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

Conv2D Math

- Input: WxHxD volume
- Parameters:
  - K is the number of filters…
  - …each one of size (F_w, F_h)
  - …moving at stride (S_w, S_h)
  - …with the input padded symmetrically with P all-zero rows/cols
- Output: W’xH’xK volume
  - $W' = (W - F_w + 2P) / S_w + 1$
  - $H' = (H - F_h + 2P) / S_h + 1$
- Each filter has (F_w * F_h * D) parameters, for K * (F_w * F_h * D) total in the layer

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

Conv2D Math

- Input: WxHxD volume
- Parameters:
  - K is the number of filters…
  - …each one of size (F_w, F_h)
  - …moving at stride (S_w, S_h)
  - …with the input padded symmetrically with P all-zero rows/cols
- Output: W'xH'xK volume
  - W' = (W - F_w + 2P) / S_w + 1
  - H' = (H - F_h + 2P) / S_h + 1
- Each filter has (F_w * F_h * D) parameters, for K * (F_w * F_h * 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

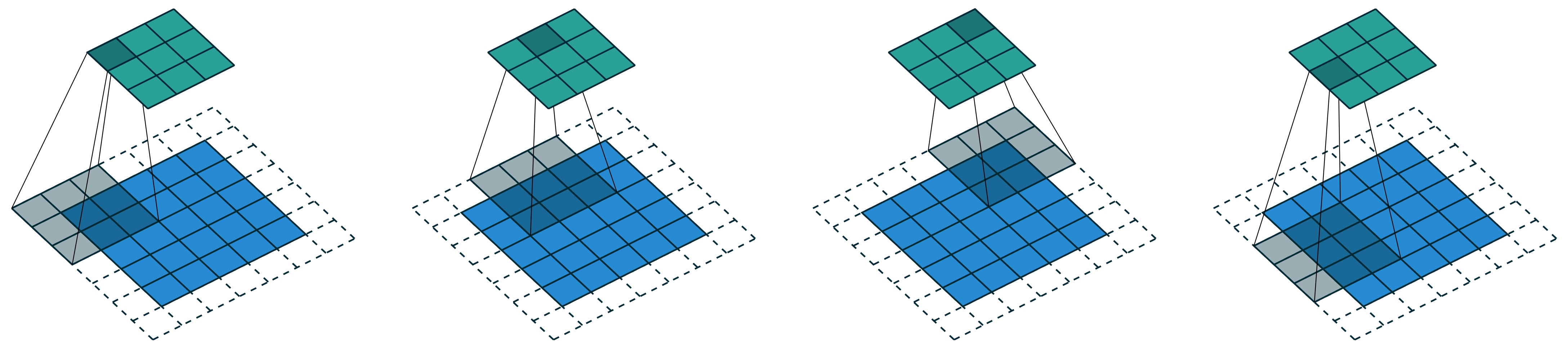


Figure 2.6: (Arbitrary padding and strides) Convolving a 3×3 kernel over a 5×5 input padded with a 1×1 border of zeros using 2×2 strides (i.e., $i = 5$, $k = 3$, $s = 2$ and $p = 1$).

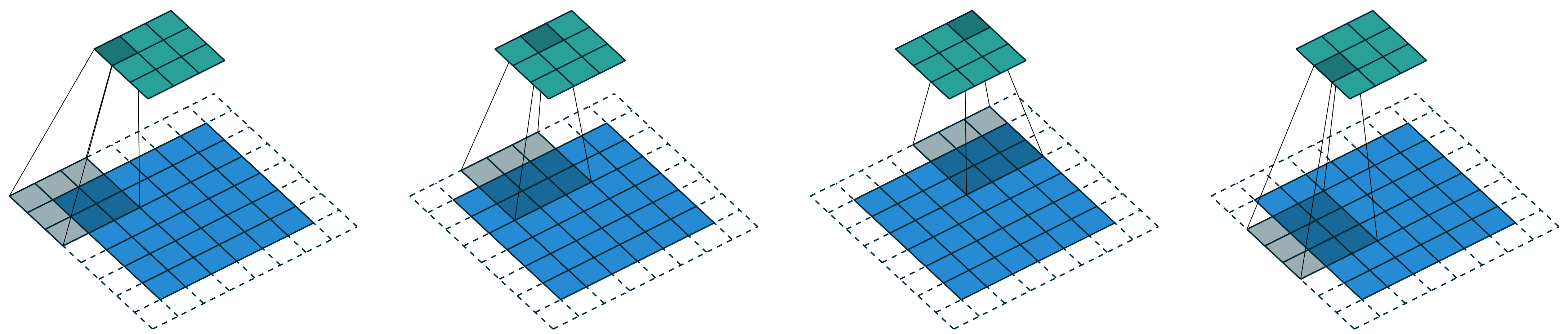
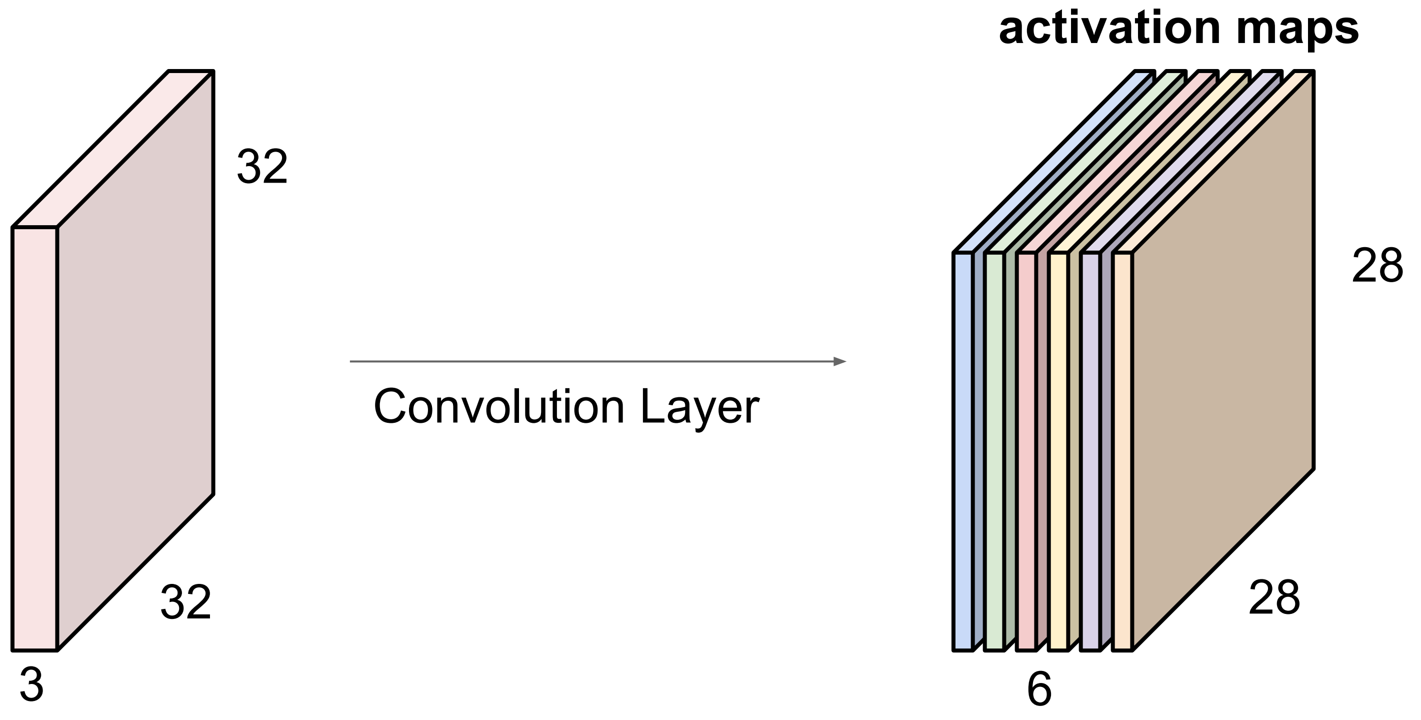


Figure 2.7: (Arbitrary padding and strides) Convolving a 3×3 kernel over a 6×6 input padded with a 1×1 border of zeros using 2×2 strides (i.e., $i = 6$, $k = 3$, $s = 2$ and $p = 1$). In this case, the bottom row and right column of the zero padded input are not covered by the kernel.

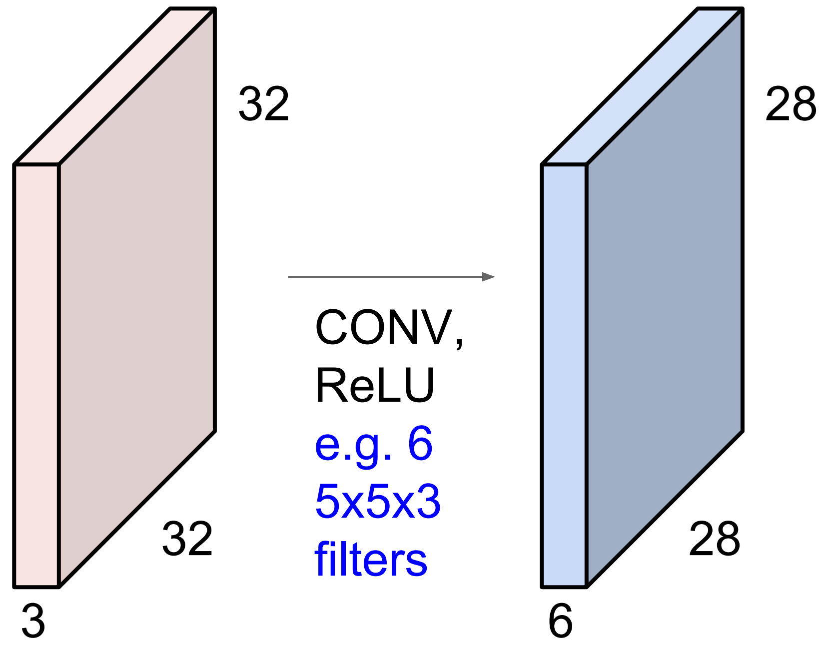
https://github.com/vdumoulin/conv_arithmetic

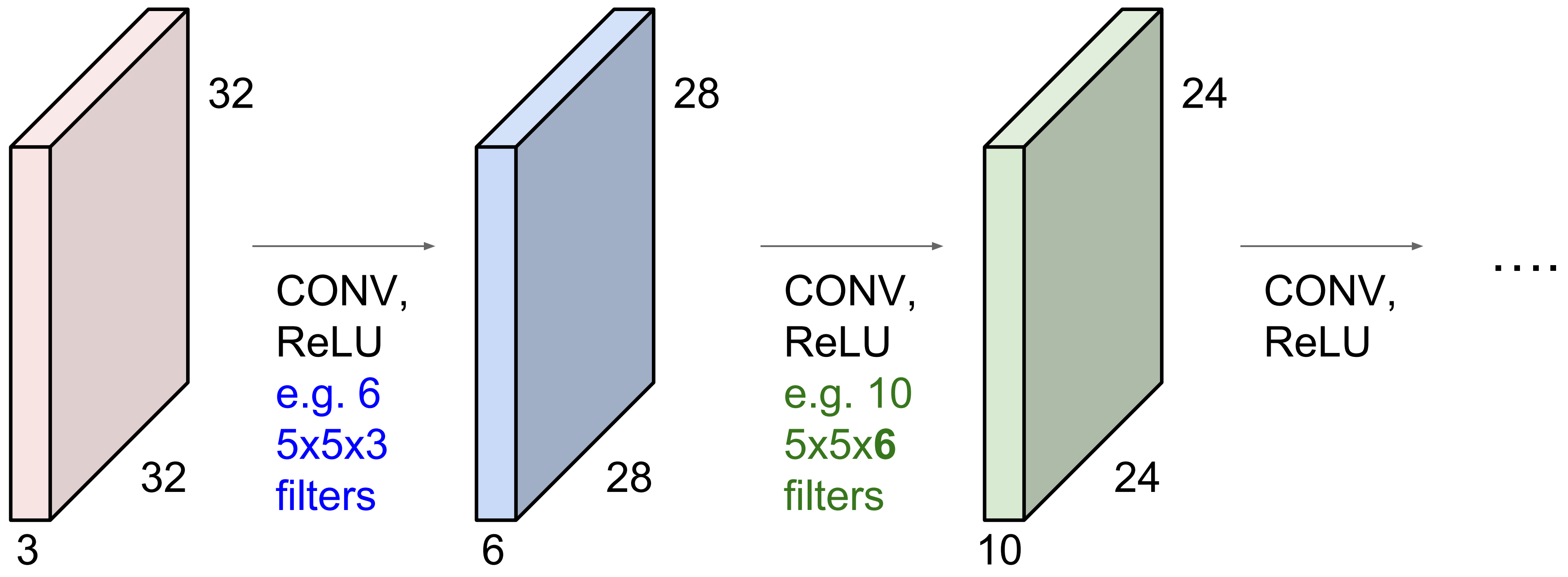
Conv2D output is another “image”

For example, if we had 6 5x5 filters, we’ll get 6 separate activation maps



We stack these up to get a “new image” of size 28x28x6!



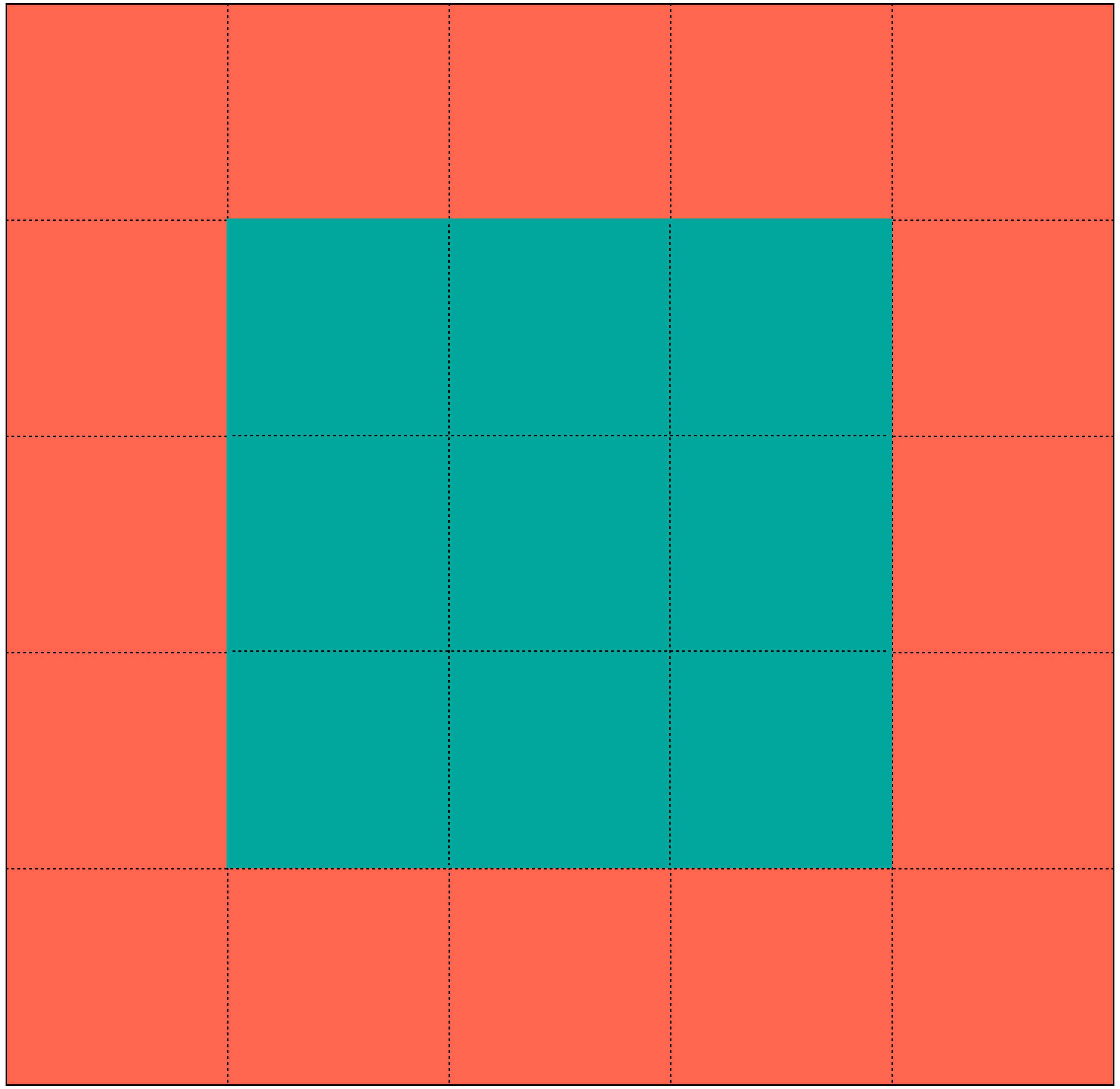


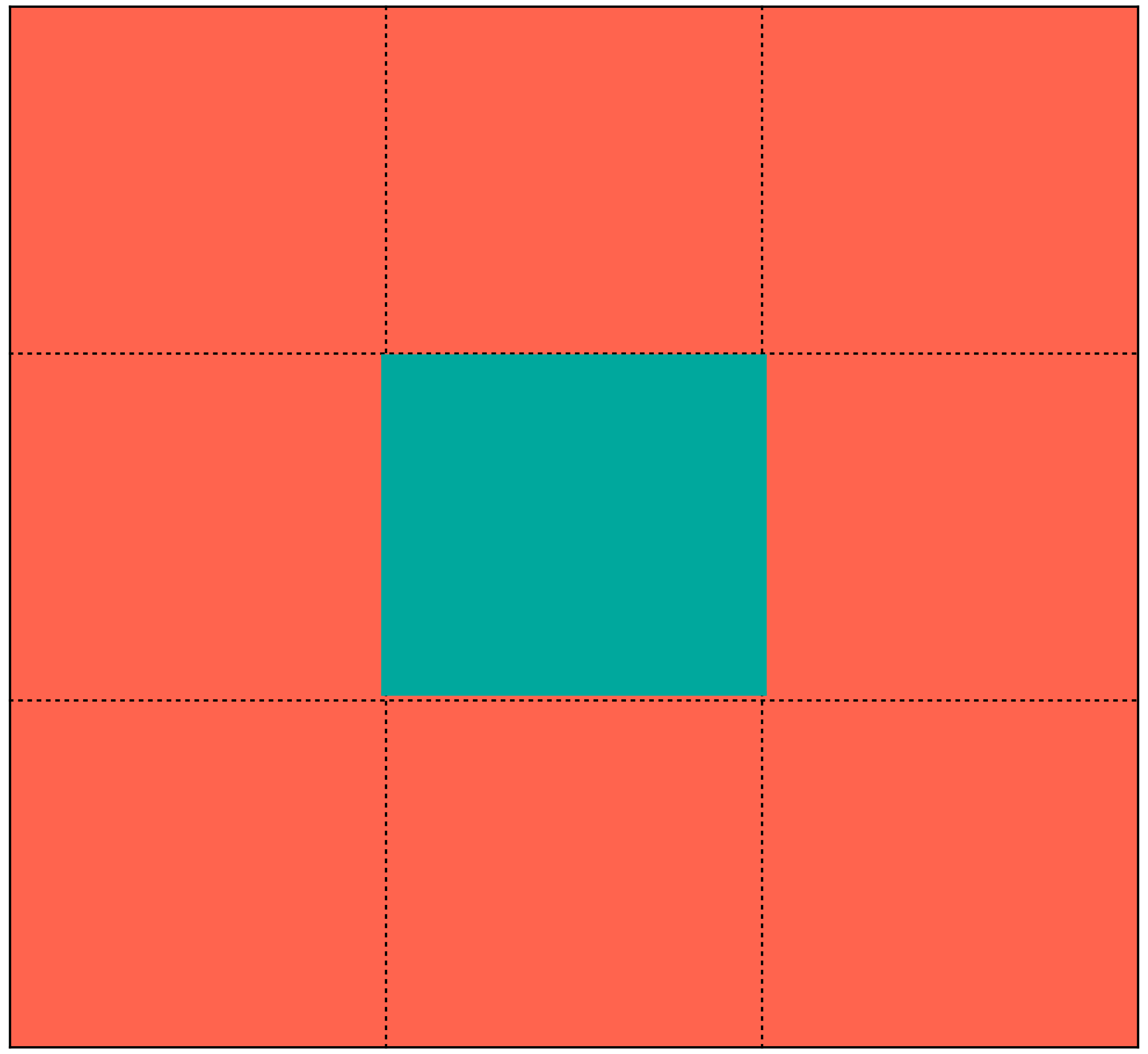
Build a network by repeating the conv operation, with activation function in between

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

Receptive fields

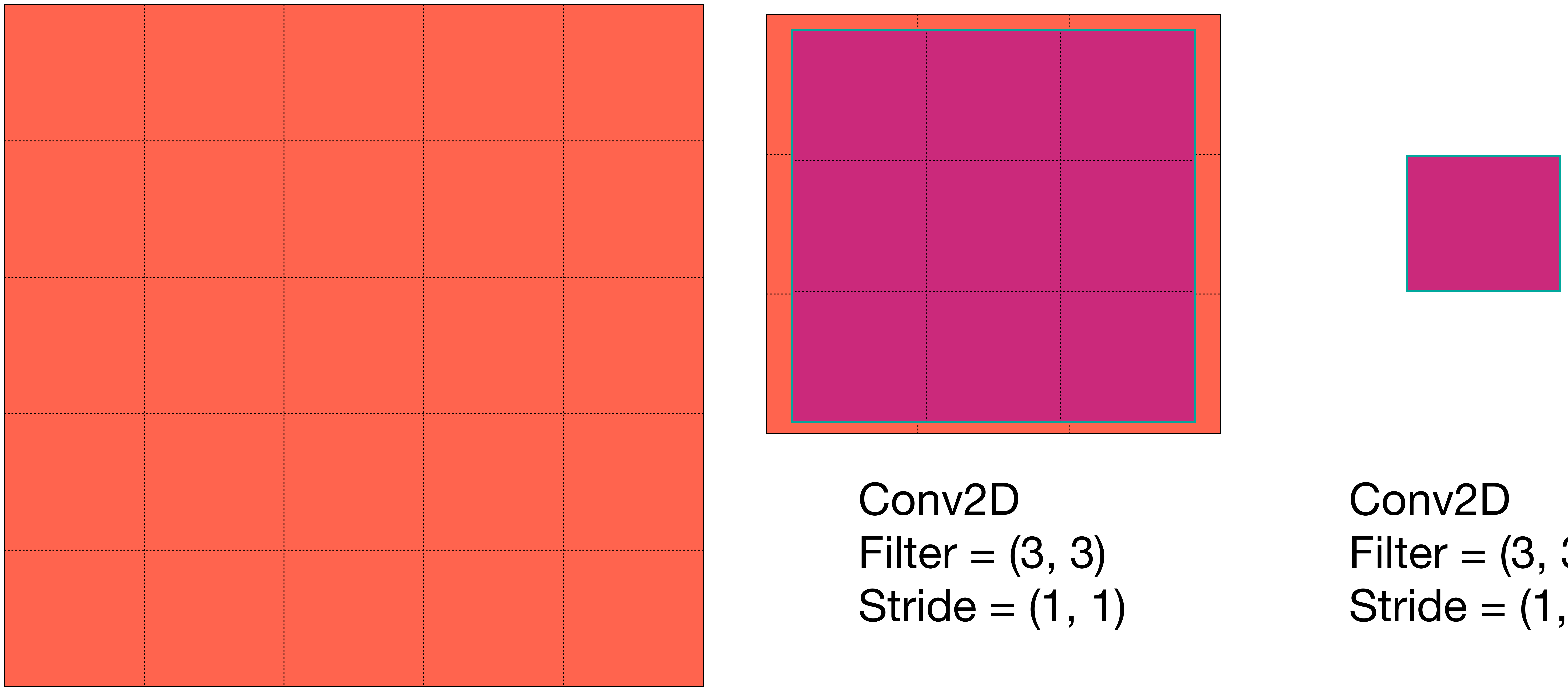
Receptive field: 3x3





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

Receptive fields

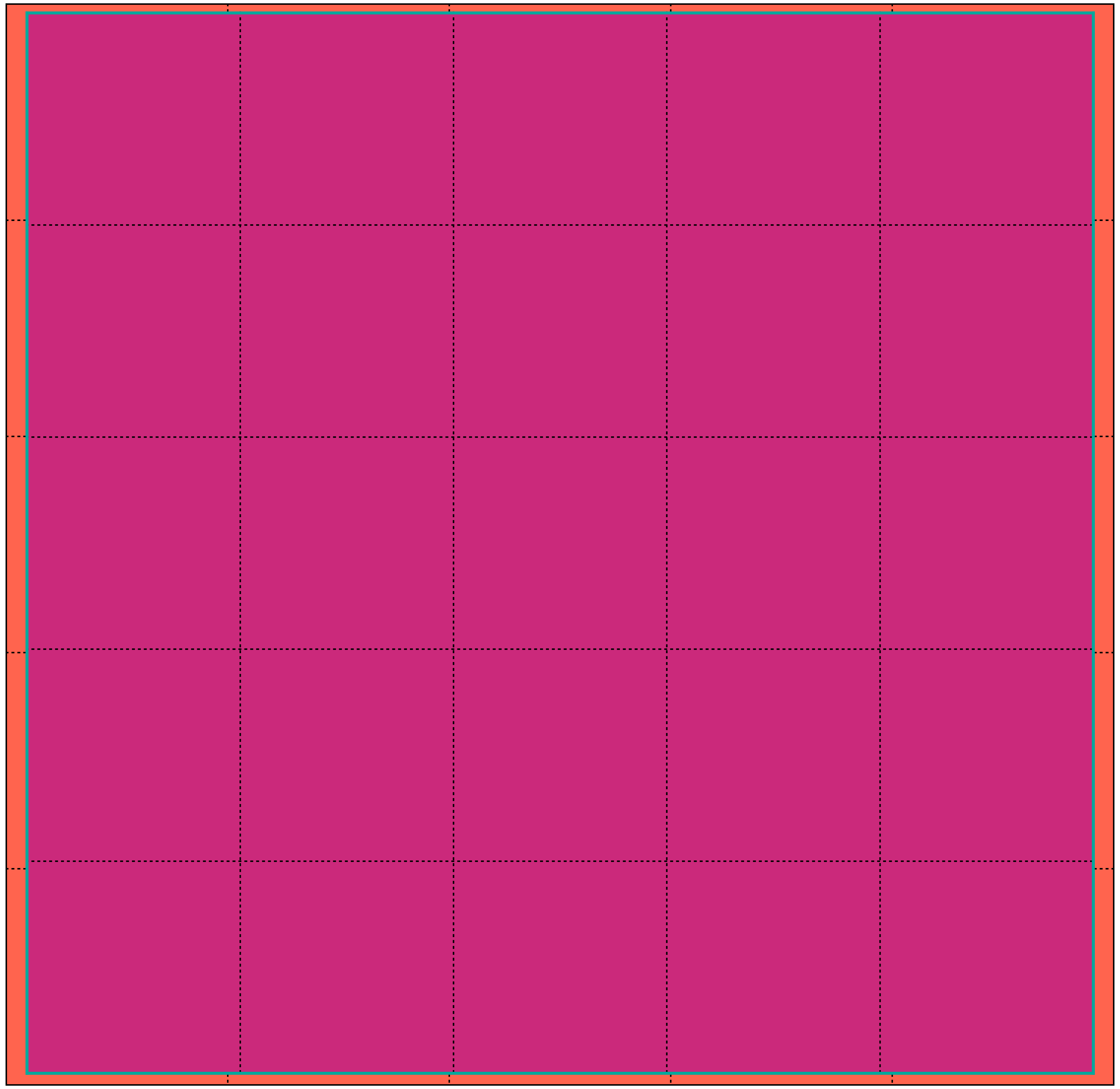


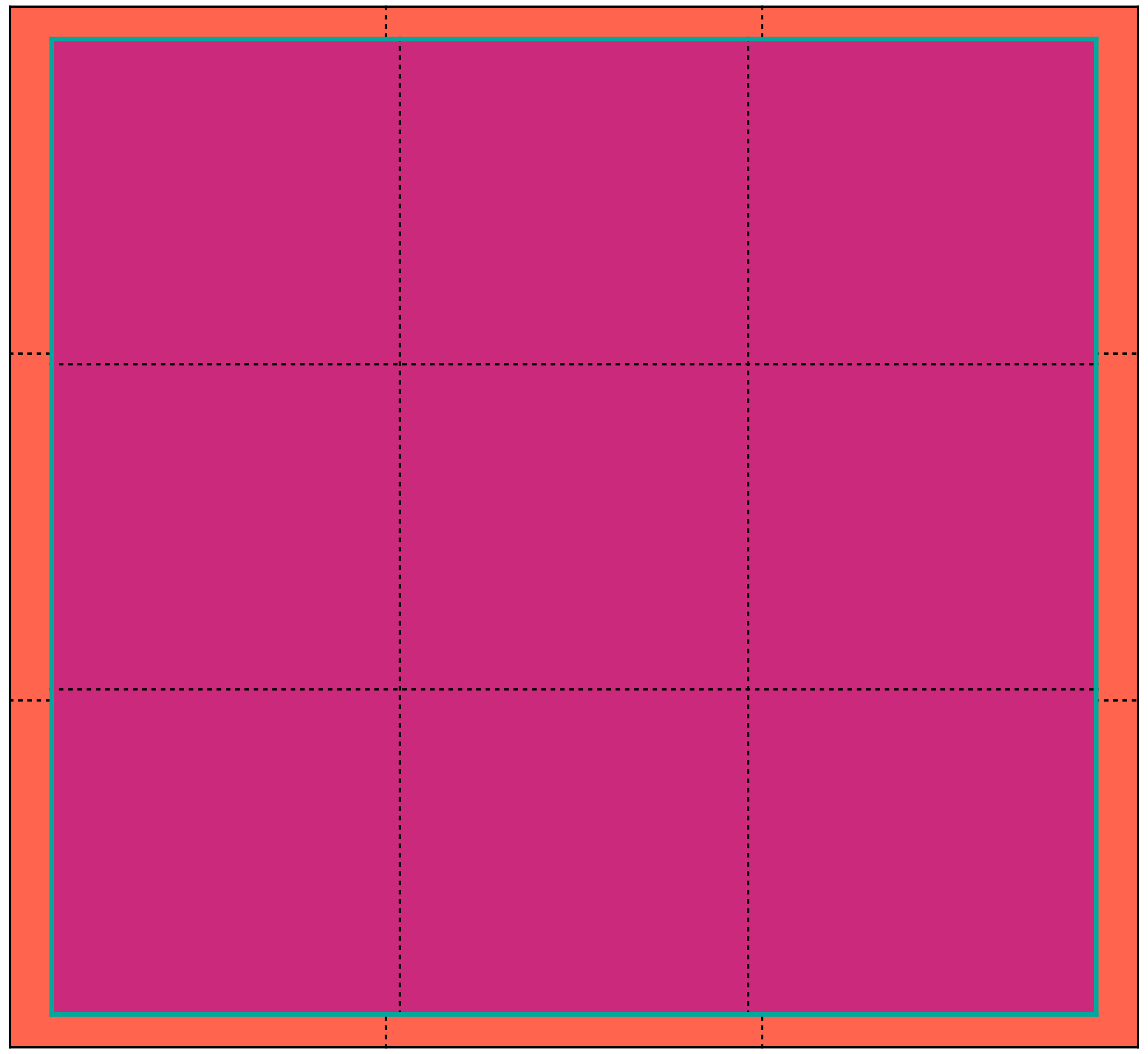
Conv2D
Filter = (3, 3)
Stride = (1, 1)

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

Receptive fields

Original receptive field: 5x5





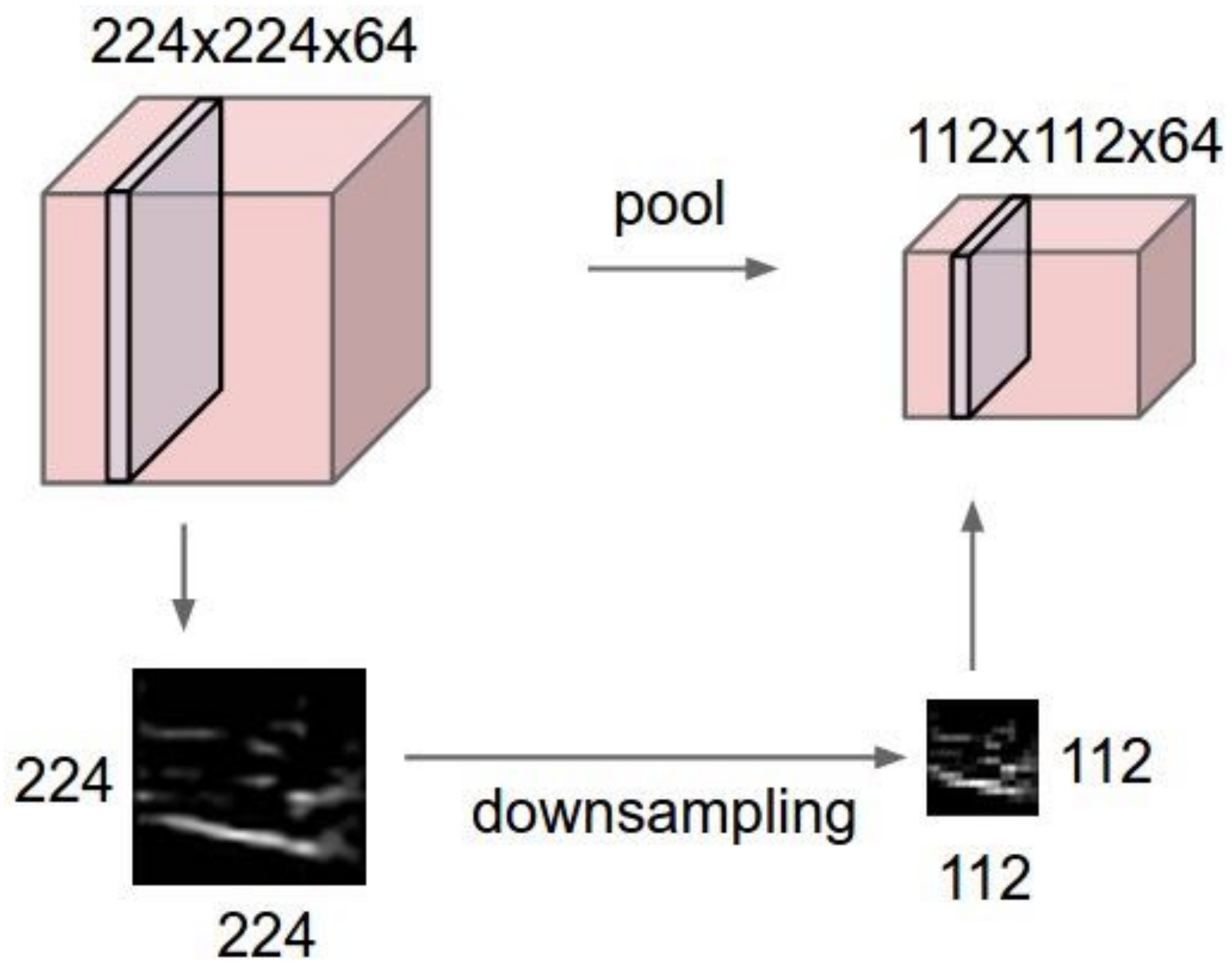
Conv2D
Filter = (3, 3)
Stride = (1, 1)

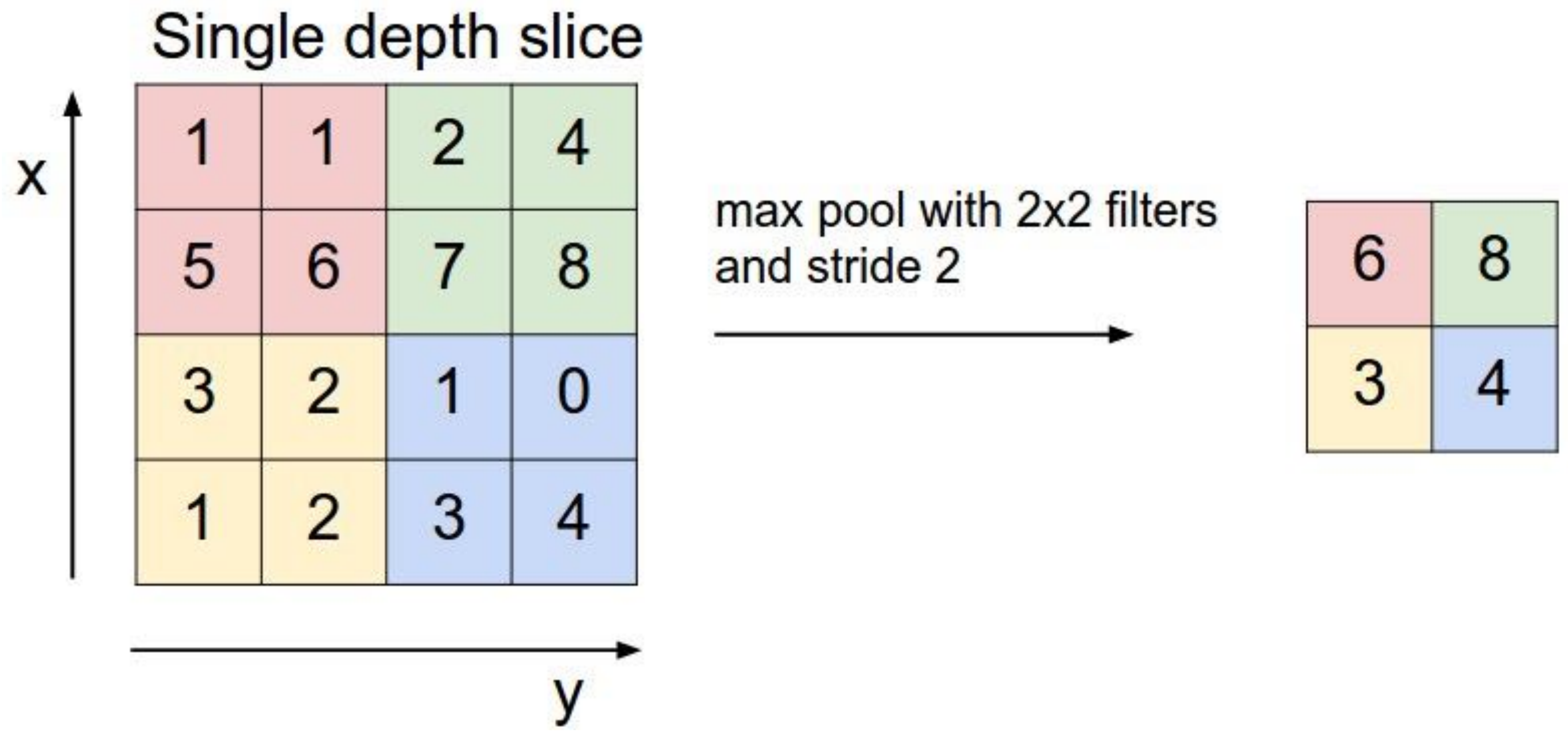
Conv2D
Filter = (3, 3)
Stride = (1, 1)

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)

Pooling





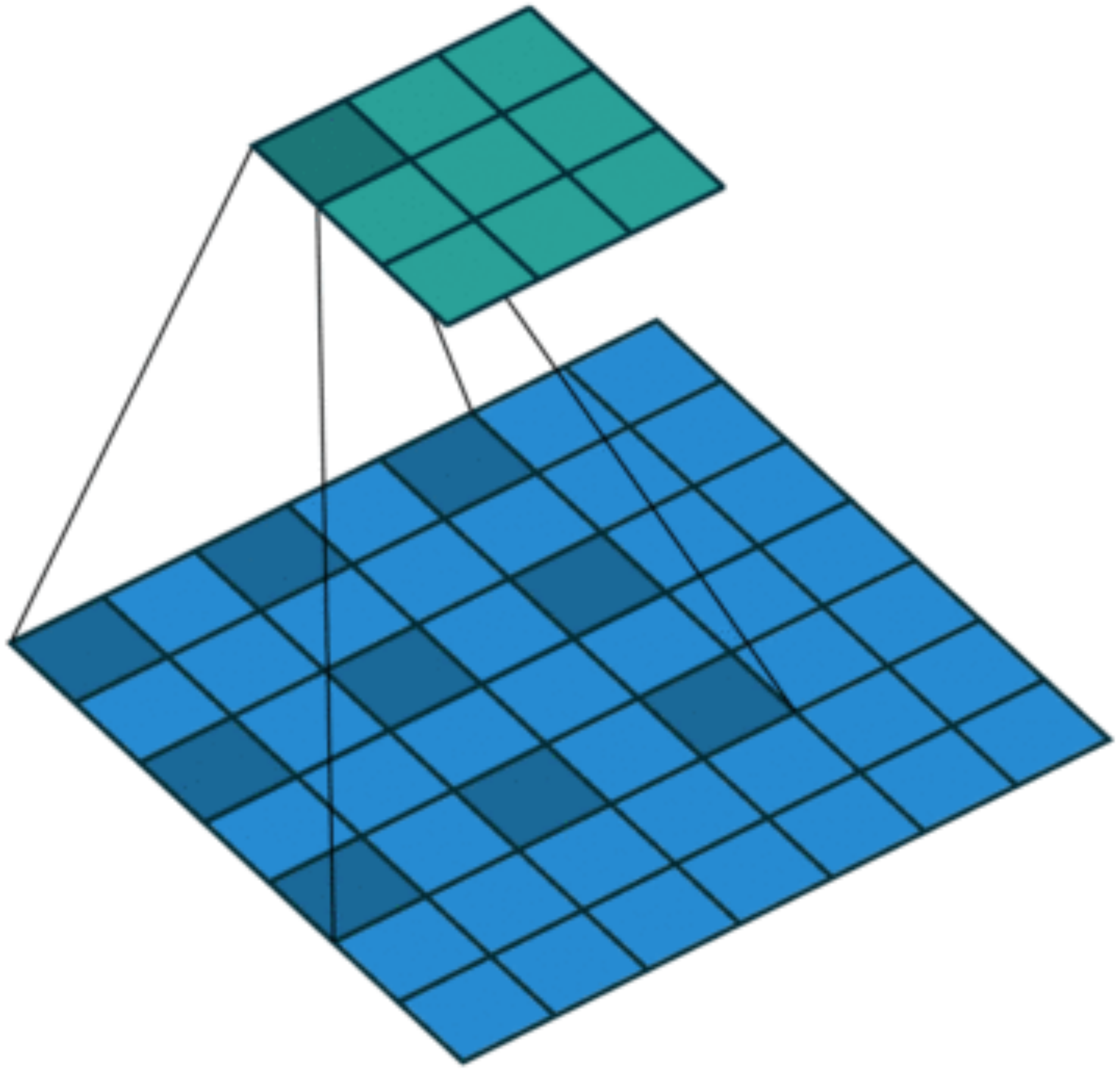
- Pooling subsamples the image by taking average or max value of a pooling region
- 2x2 max pooling is illustrated, and probably most common
- Very important in early convnet applications, but has recently fallen out of favor (strided convolutions can subsample, and tend to work better in GAN applications).

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

Agenda

1. Basics of convolutions, and filter/stride/pooling math
2. **More advanced convolution types, and computational considerations**
3. Classic convnet architectures

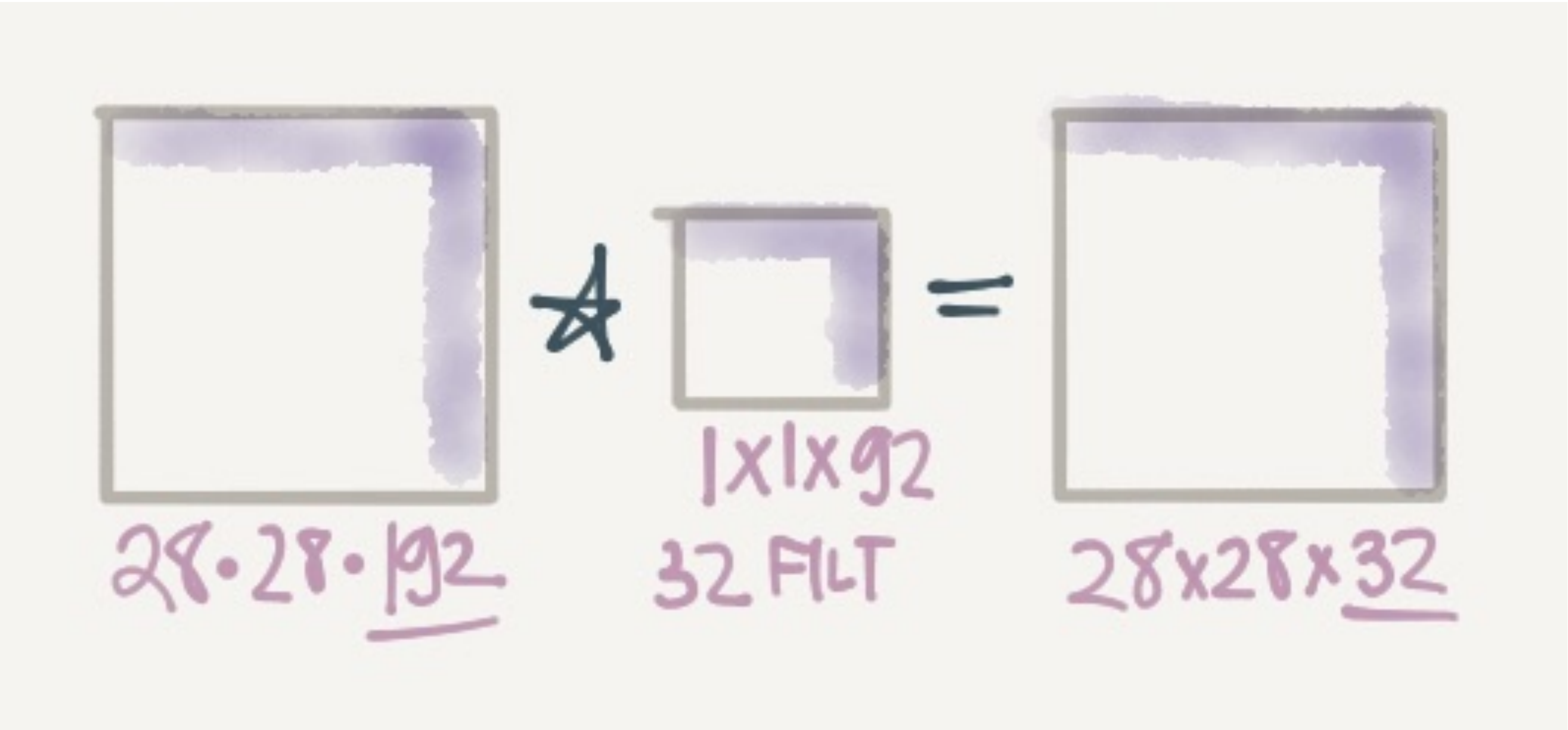
Dilated Convolution



- 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

1x1 Convolution

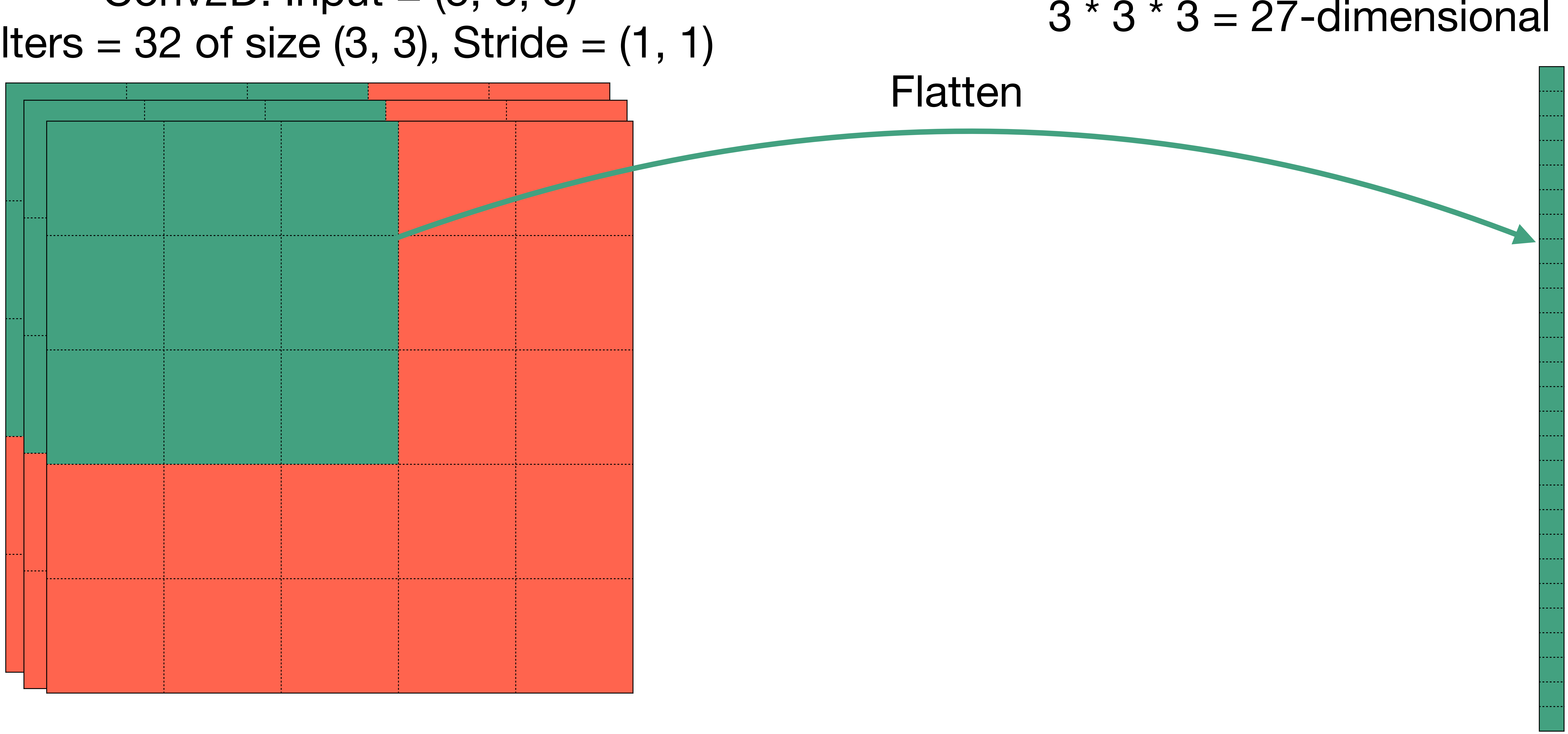


- A way to reduce the “depth” dimension of convolutional outputs
- Corresponds to applying an MLP to every pixel in the convolutional output
- Crucial to popular convnet architectures like Inception (GoogleNet)

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

Implementing the math

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

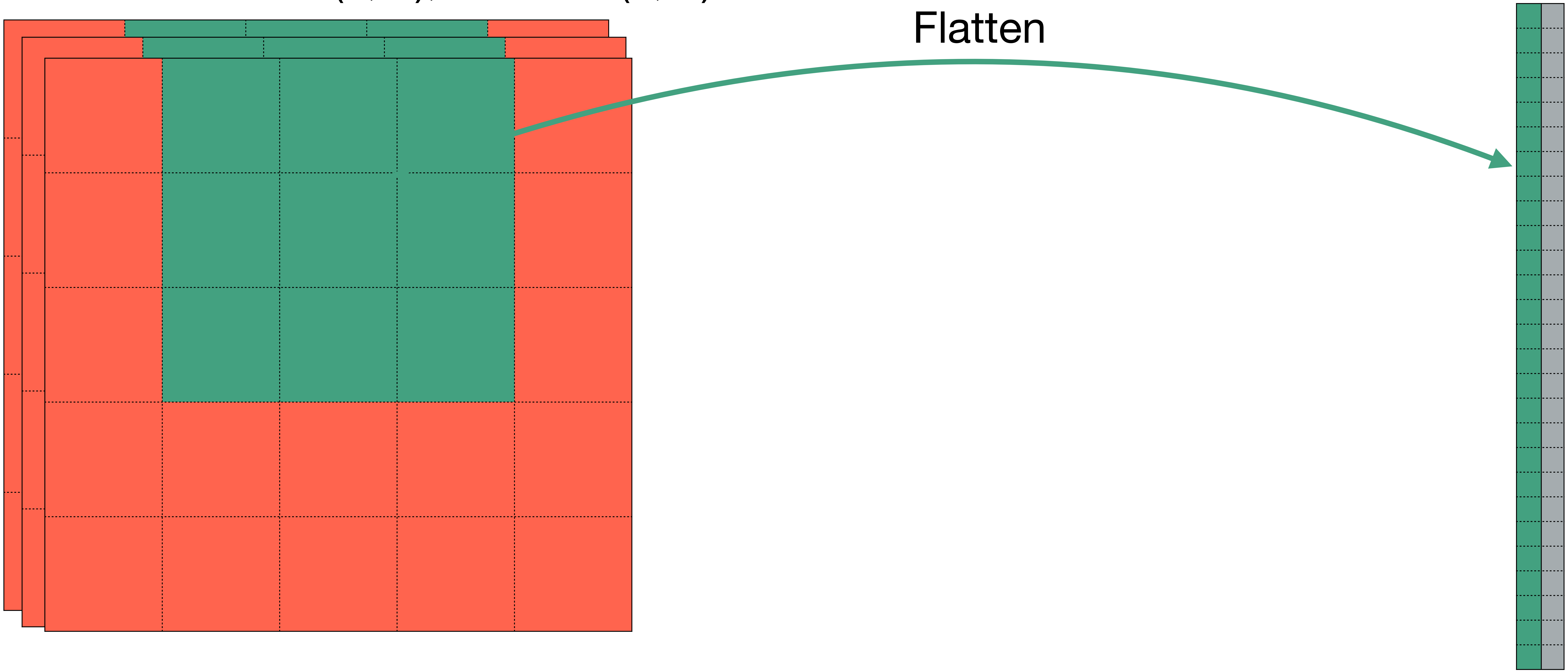


http://cs231n.github.io/convolutional-networks/

Implementing the math

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

3 * 3 * 3 = 27-dimensional

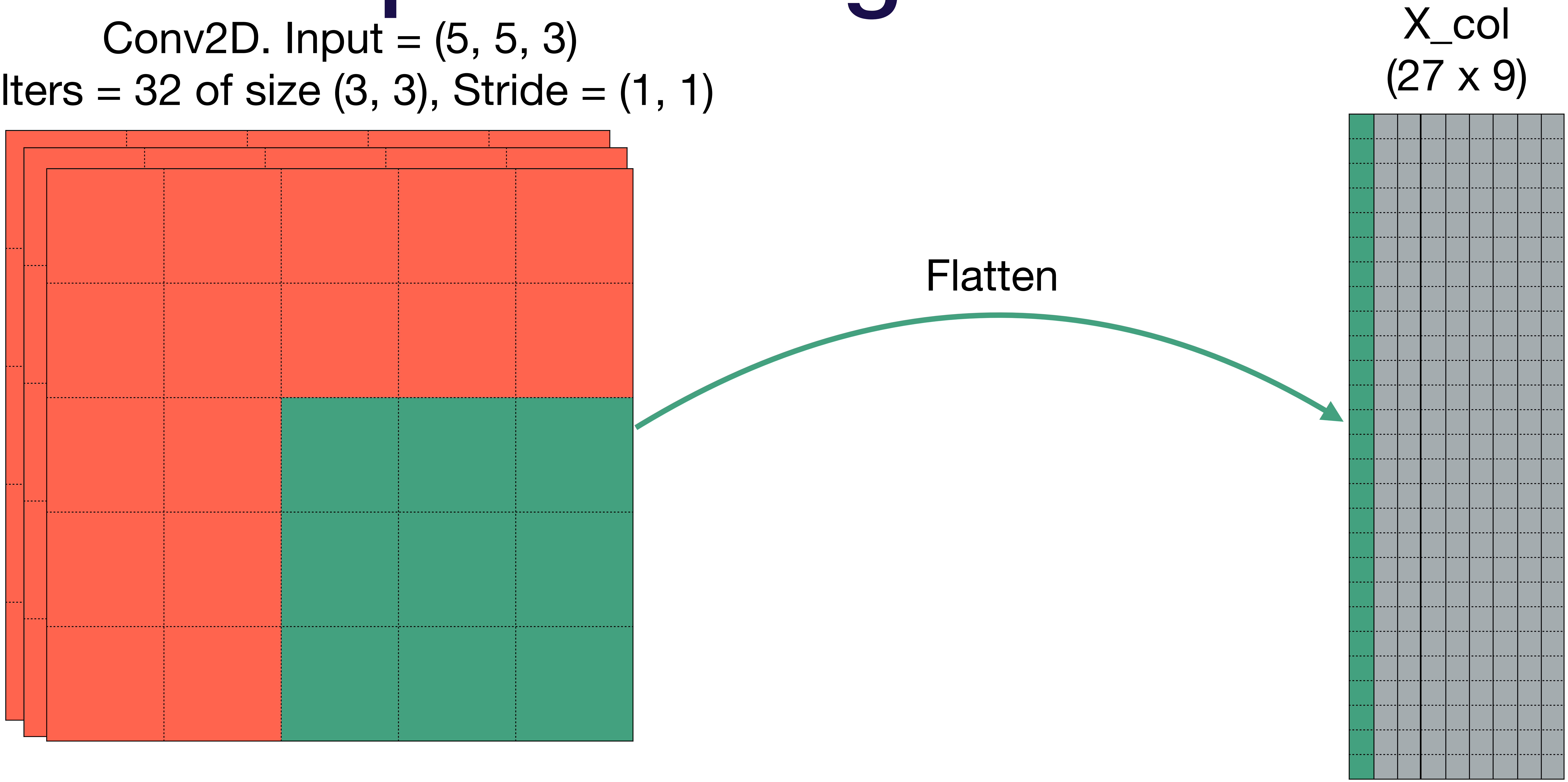


http://cs231n.github.io/convolutional-networks/

Implementing the math

Conv2D. Input = (5, 5, 3)

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



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

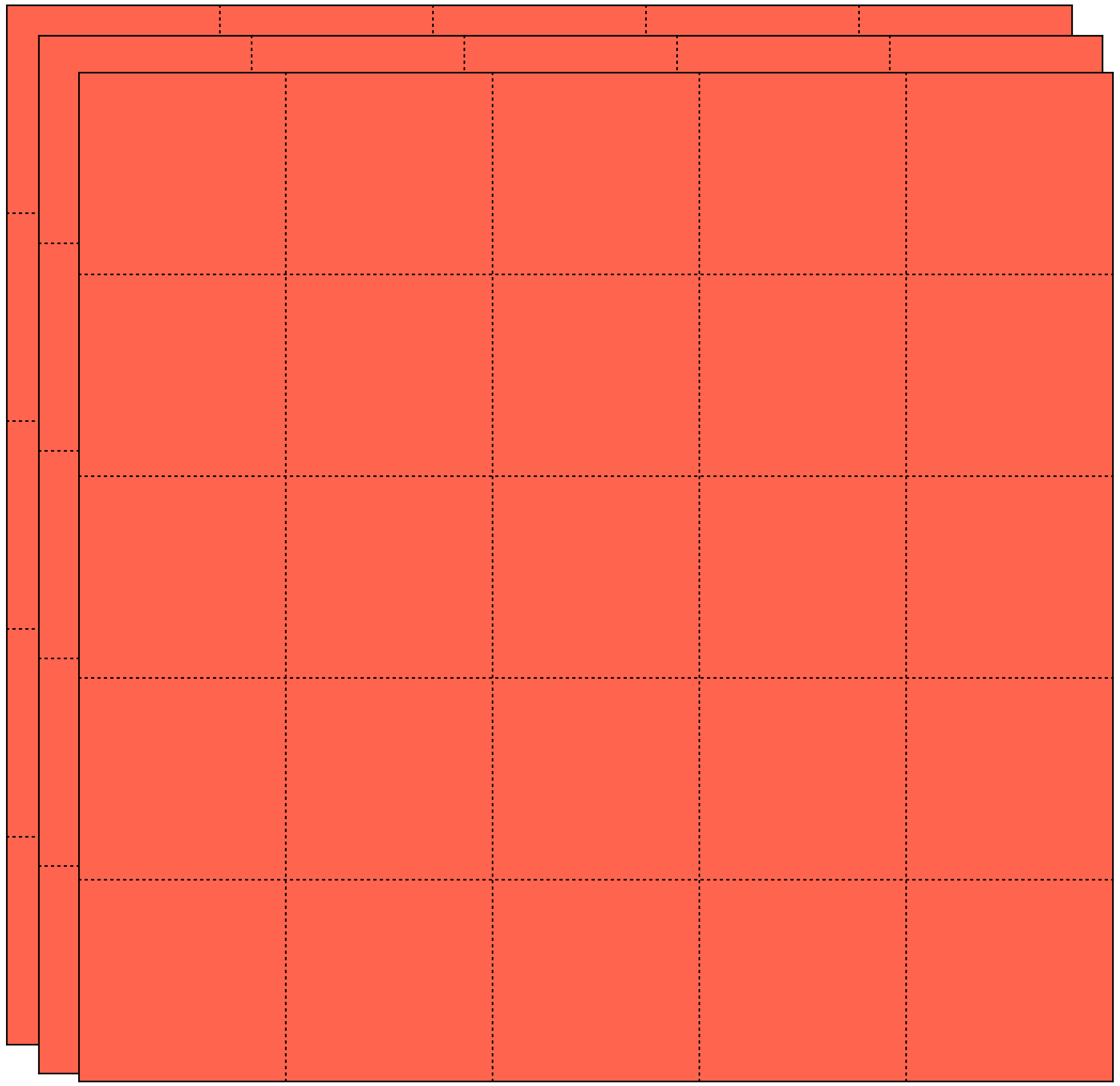
Implementing the math

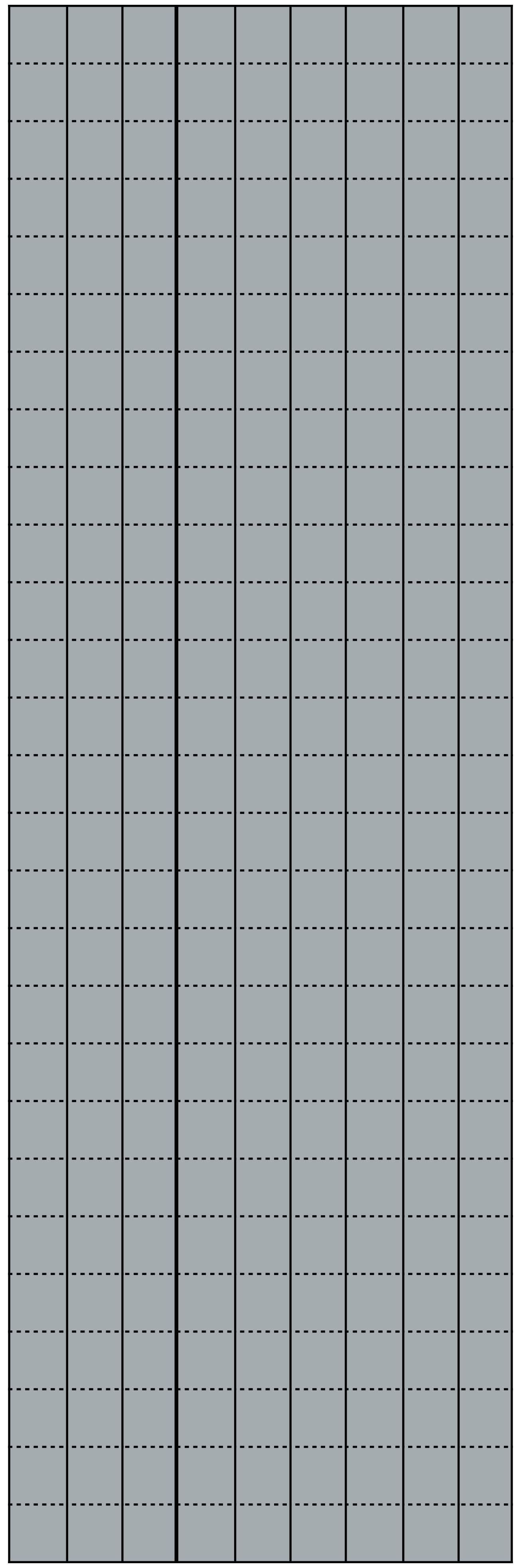
Conv2D. Input = (5, 5, 3)

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

W_row
(32 x 27)

X_col
(27 x 9)



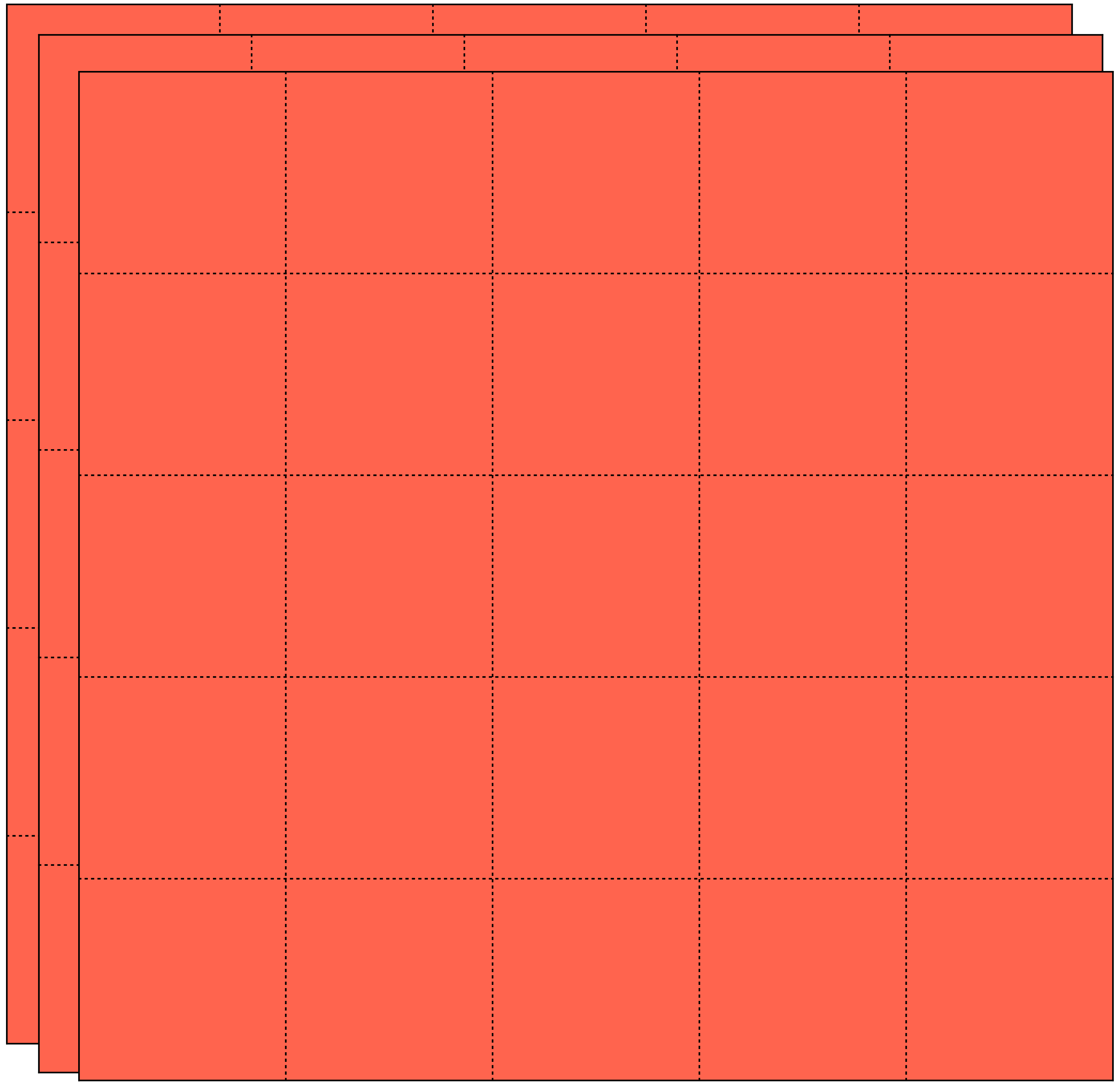


http://cs231n.github.io/convolutional-networks/

Implementing the math

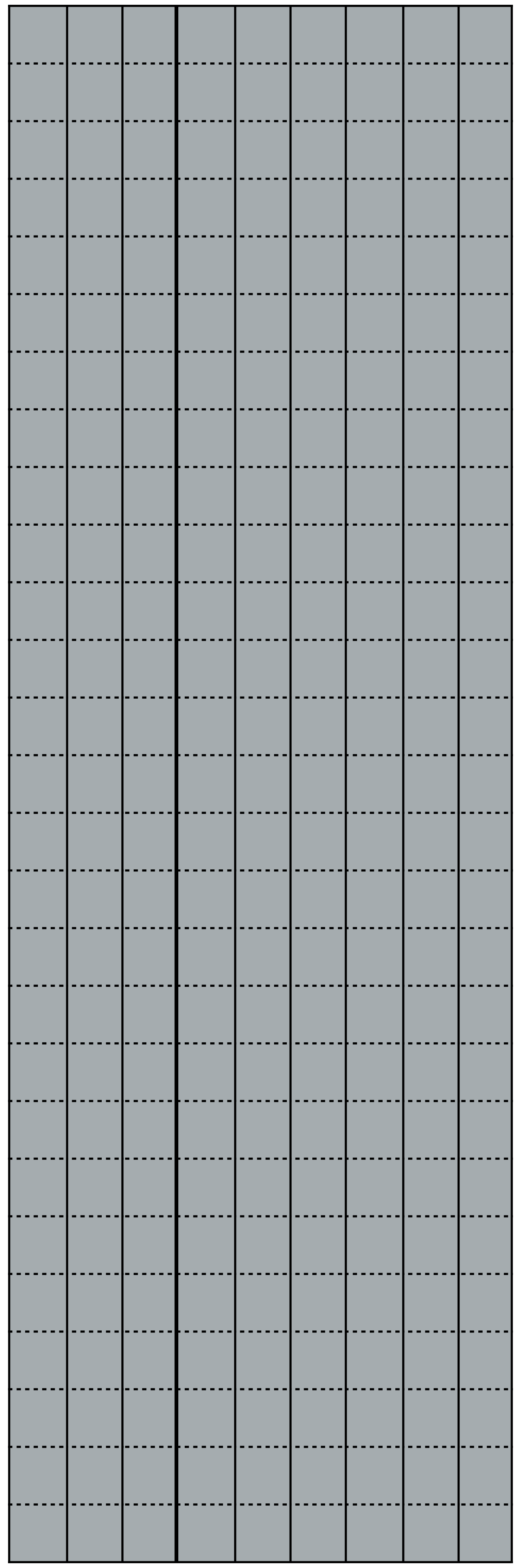
Conv2D. Input = (5, 5, 3)

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



W_row
(32 x 27)

X_col
(27 x 9)



W_row.X_col
(32 x 9)

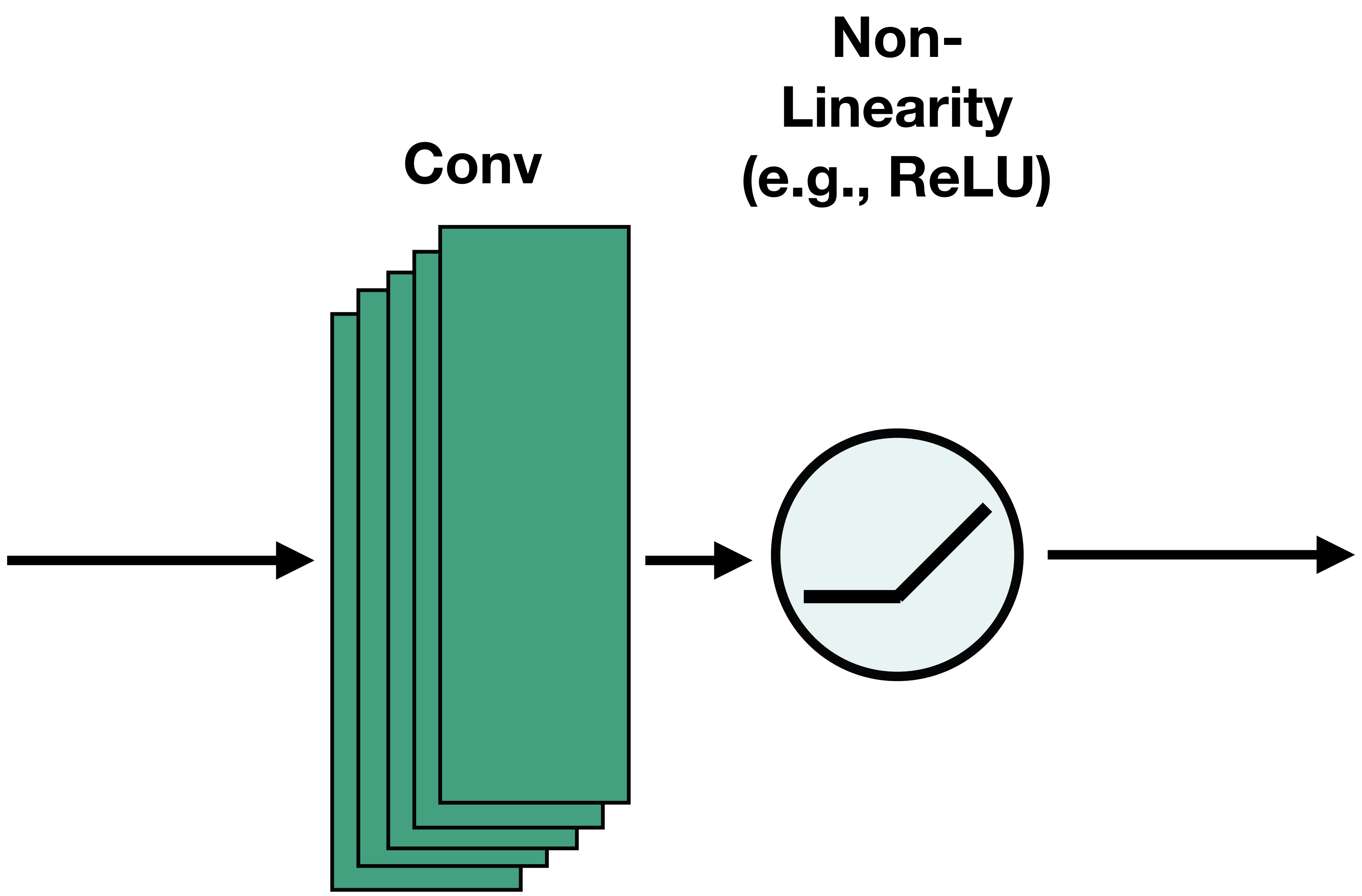
Reshape
(3 x 3 x 32)

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

Agenda

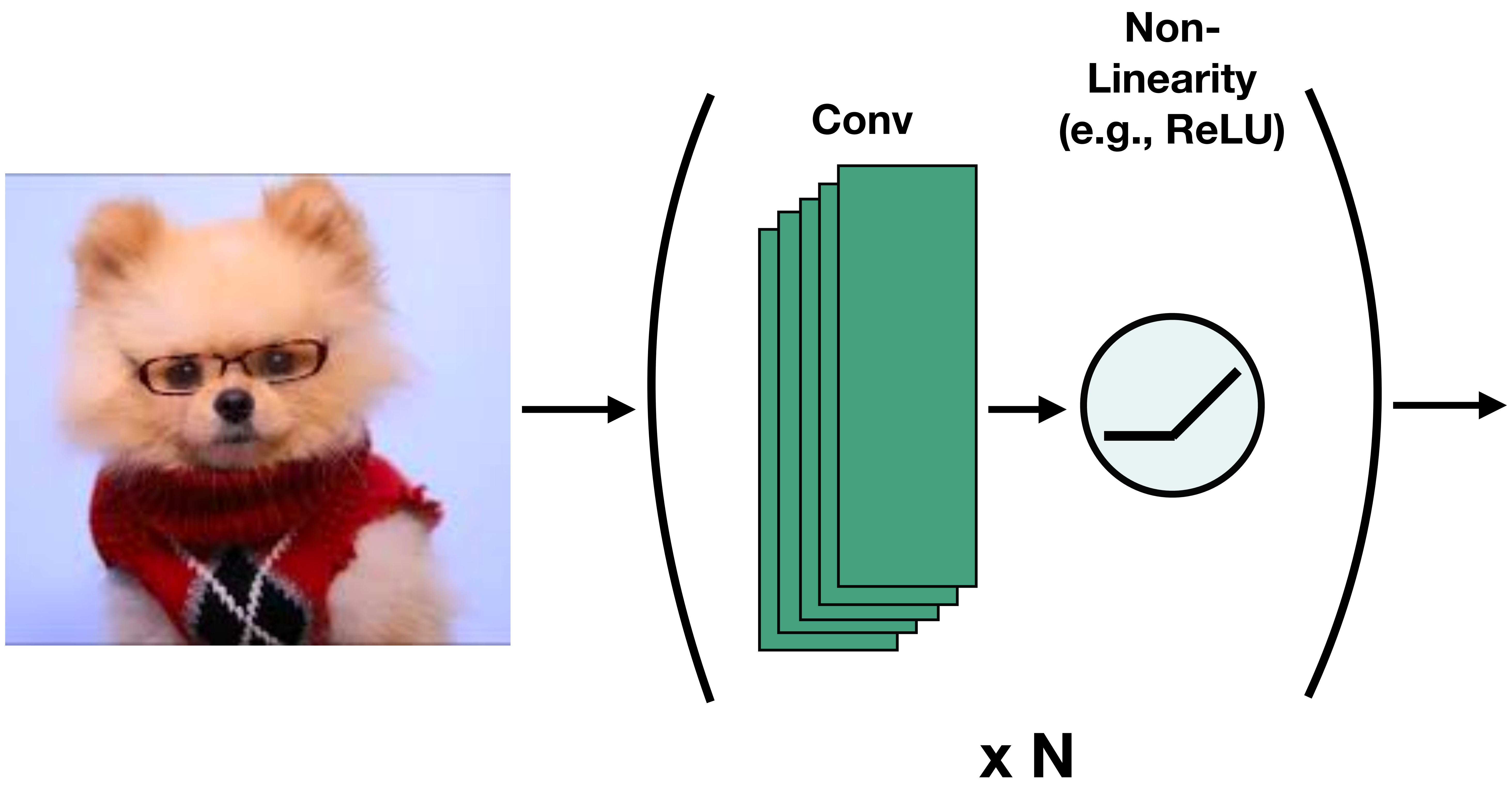
1. Basics of convolutions, and filter/stride/pooling math
2. More advanced convolution types, and computational considerations
3. **Classic ConvNet architectures**

Most standard: LeNet(-like) architectures



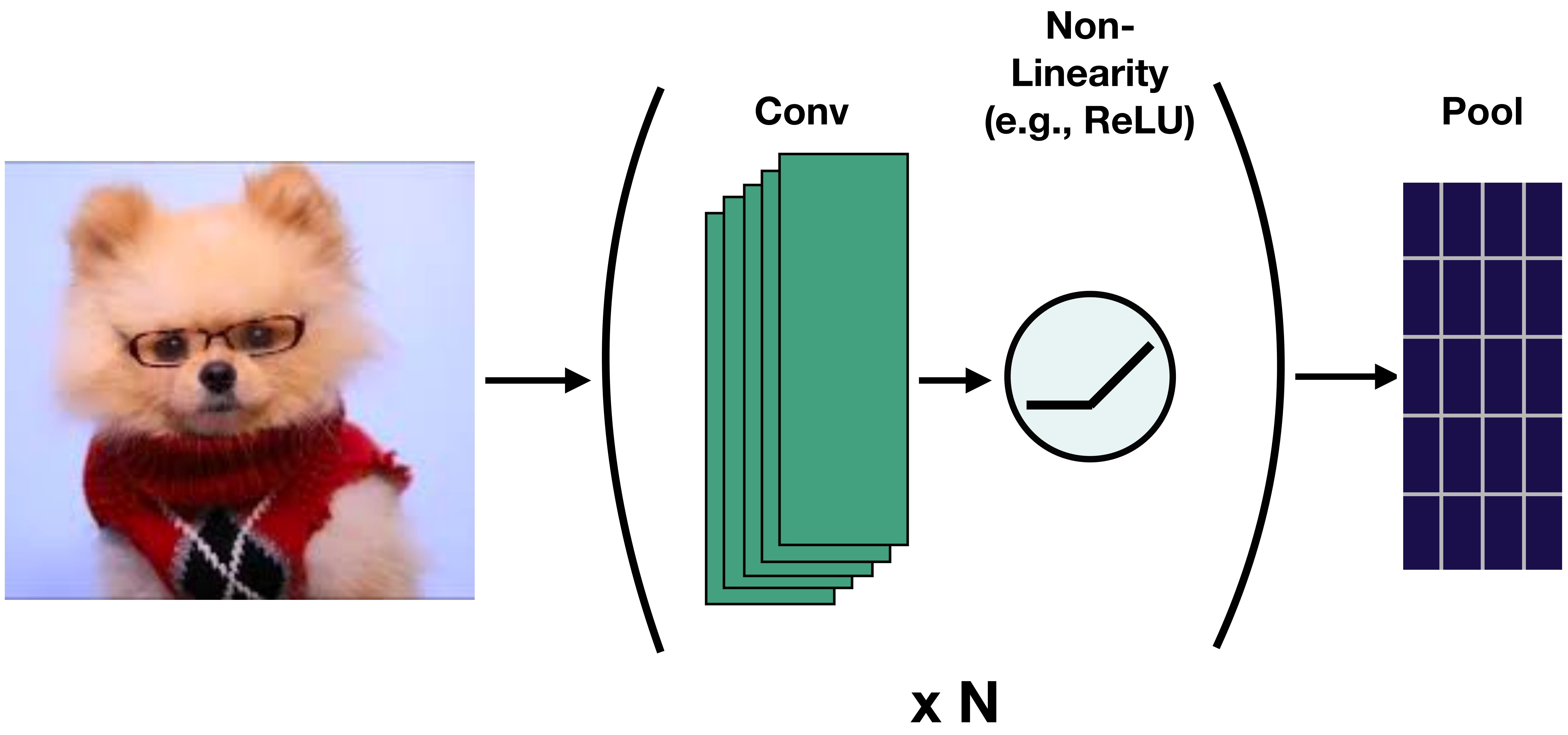
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

Most standard: LeNet(-like) architectures



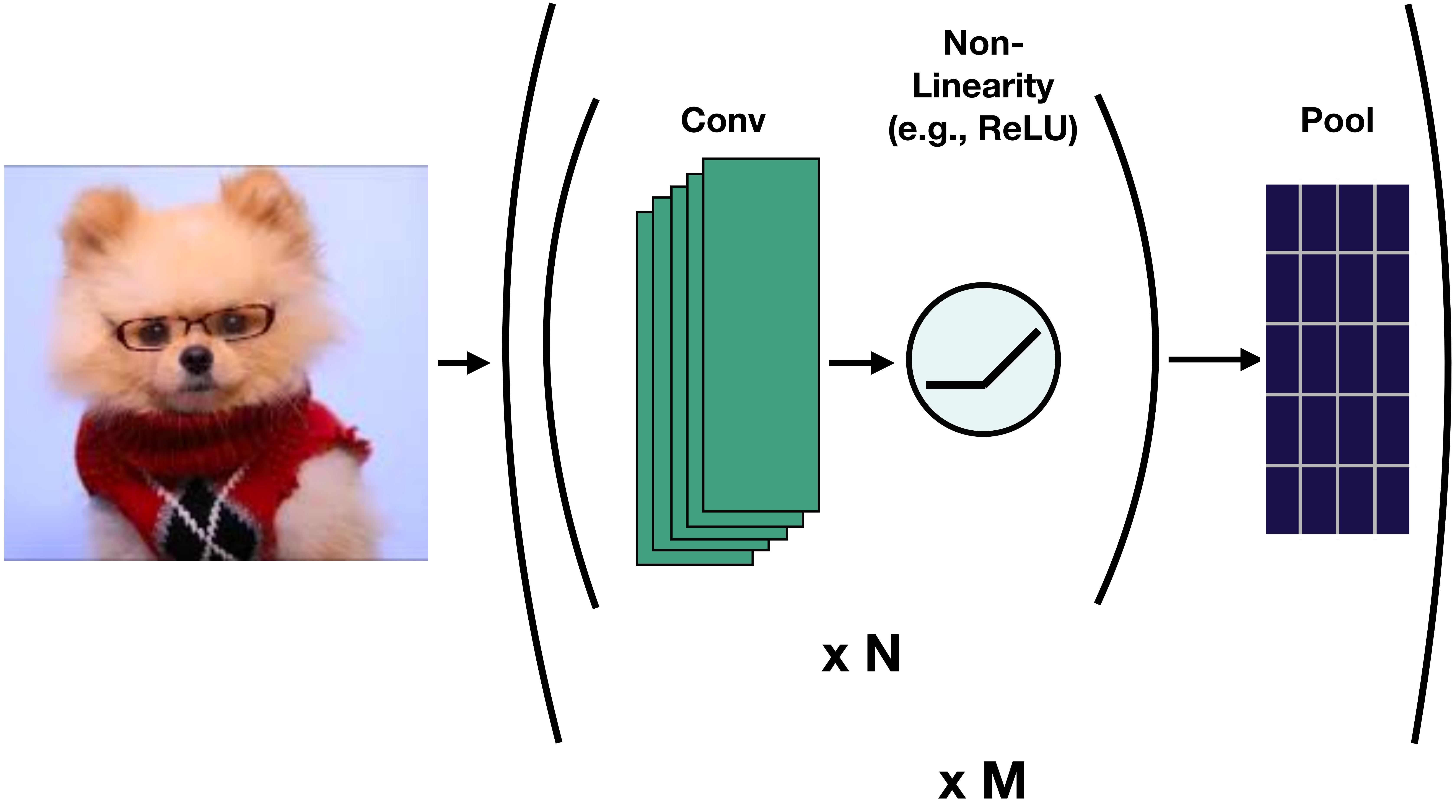
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

Most standard: LeNet(-like) architectures



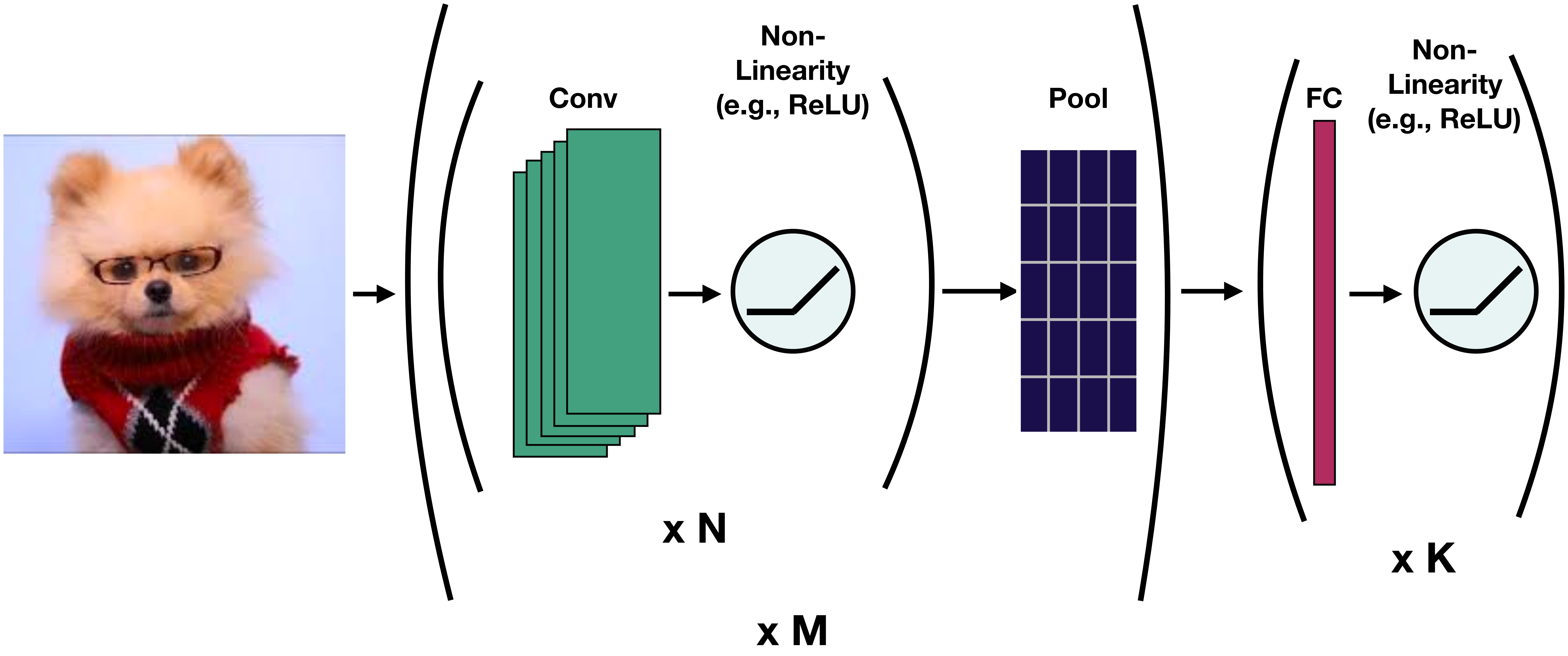
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

Most standard: LeNet(-like) architectures



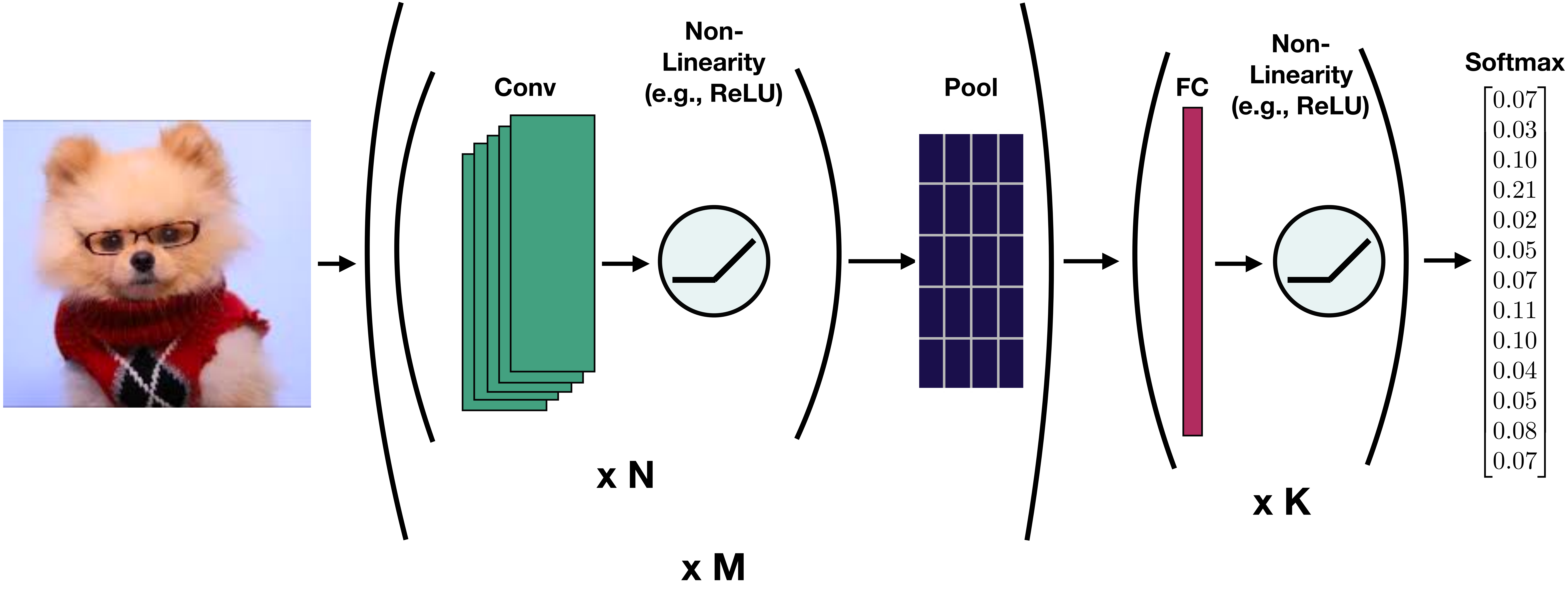
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

Most standard: LeNet(-like) architectures



http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

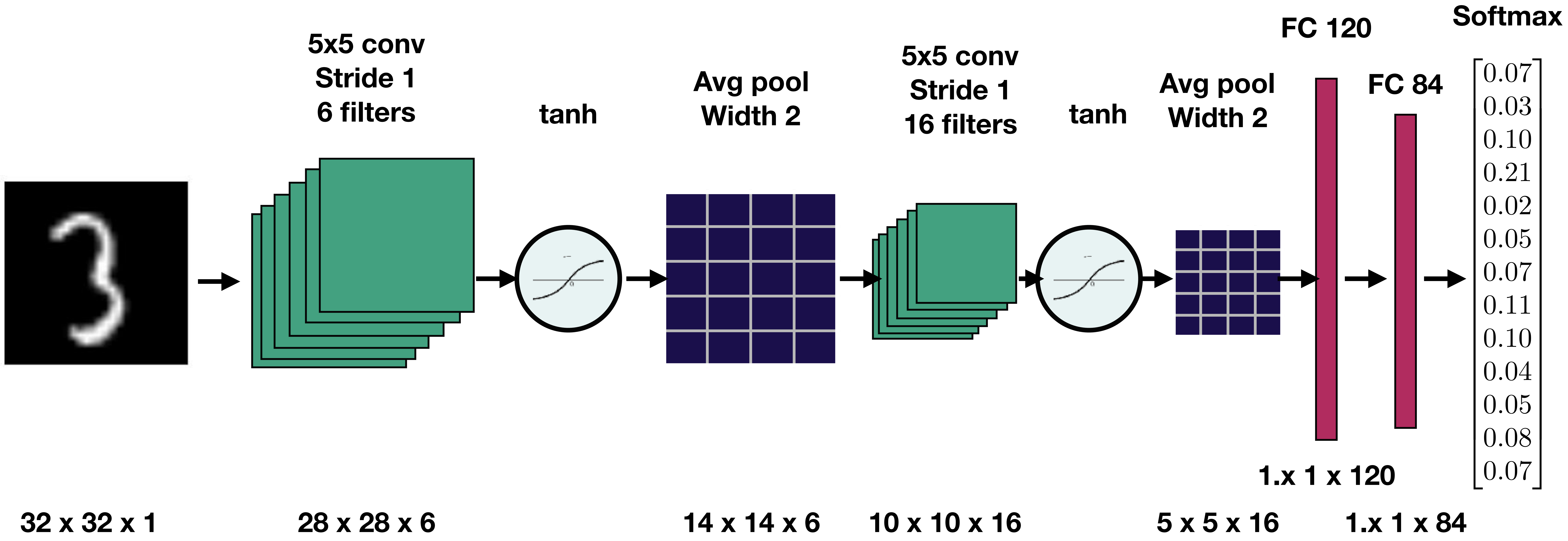
Most standard: LeNet(-like) architectures



$$\begin{bmatrix}0.07\\0.03\\0.10\\0.21\\0.02\\0.05\\0.07\\0.11\\0.10\\0.04\\0.05\\0.08\\0.07\end{bmatrix}$$

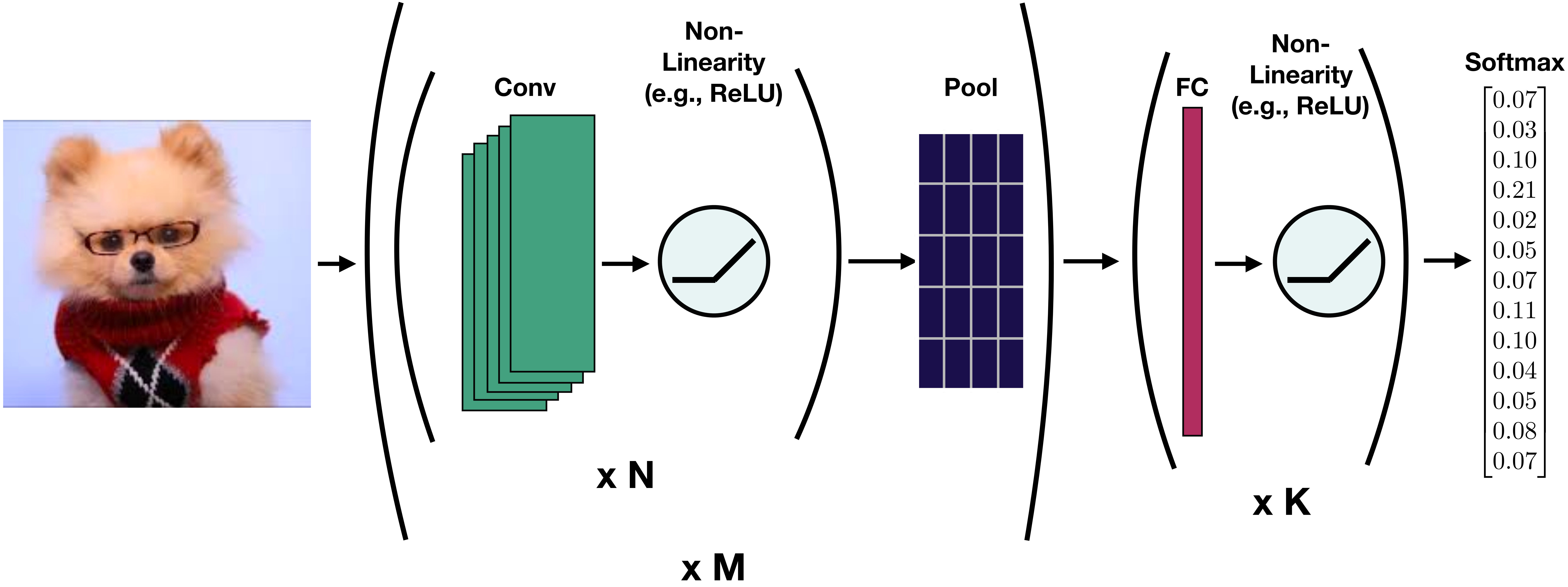
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

Classic Convnet Architecture: LeNet



http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

Most standard: LeNet(-like) architectures



http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

More modern LeNet-like architectures

- N = up to 5
- M large
- 0 <= K <= 2
- ReLU instead of Tanh

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

Where to go to learn more?

- Stanford’s CS231n