Dauphine | PSL
UNIVERSITÉ PARIS

Deep Learning for Image Analysis

Mehyar MLAWEH

🏫 Master BDIA - Dauphine Tunis
PhD student - Université PSL

👨‍💻 Generative AI Engineer - Wealthy Technology

Resources: https://mehyarmlaweh.github.io/mehyar-mlaweh/teaching

Module Overview

1. Computer Vision Foundations
2. CNNs
3. Transfer Learning
4. Vision Transformers
5. Object Detection
6. Segment Anything Model (SAM)
7. Final Project

Lecture 2 - Part 1:

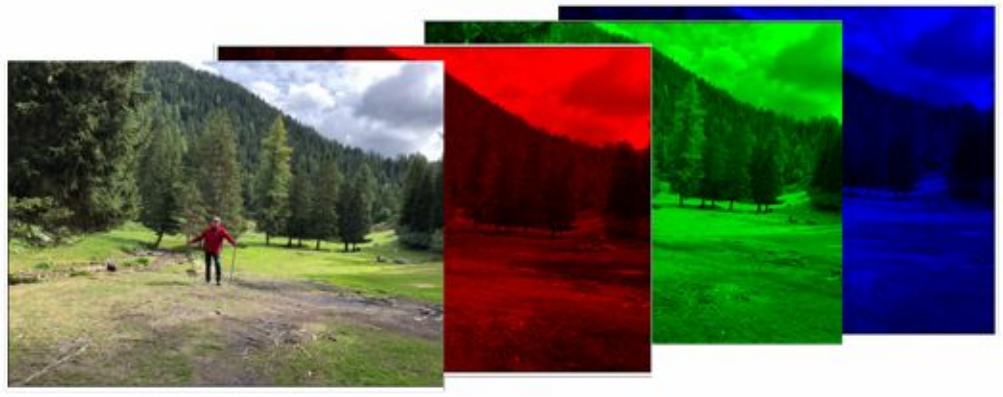
Convolutional neural networks

What is an image?

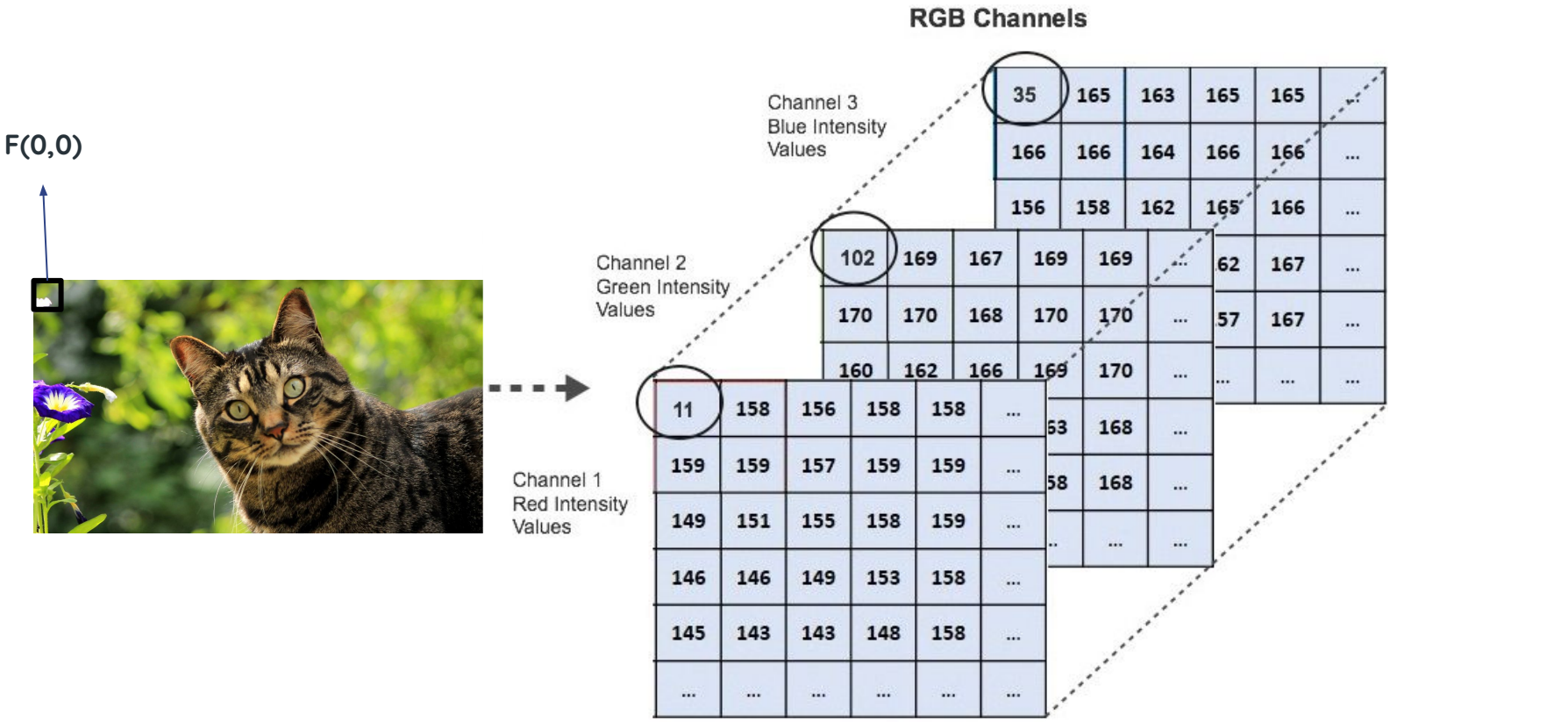
An image is a matrix of numbers are called pixels

3 channels RGB : Superposition of 3 matrices of numbers between 0 and 255

- A larger value means a brighter pixel.
- A smaller value means a darker pixel.

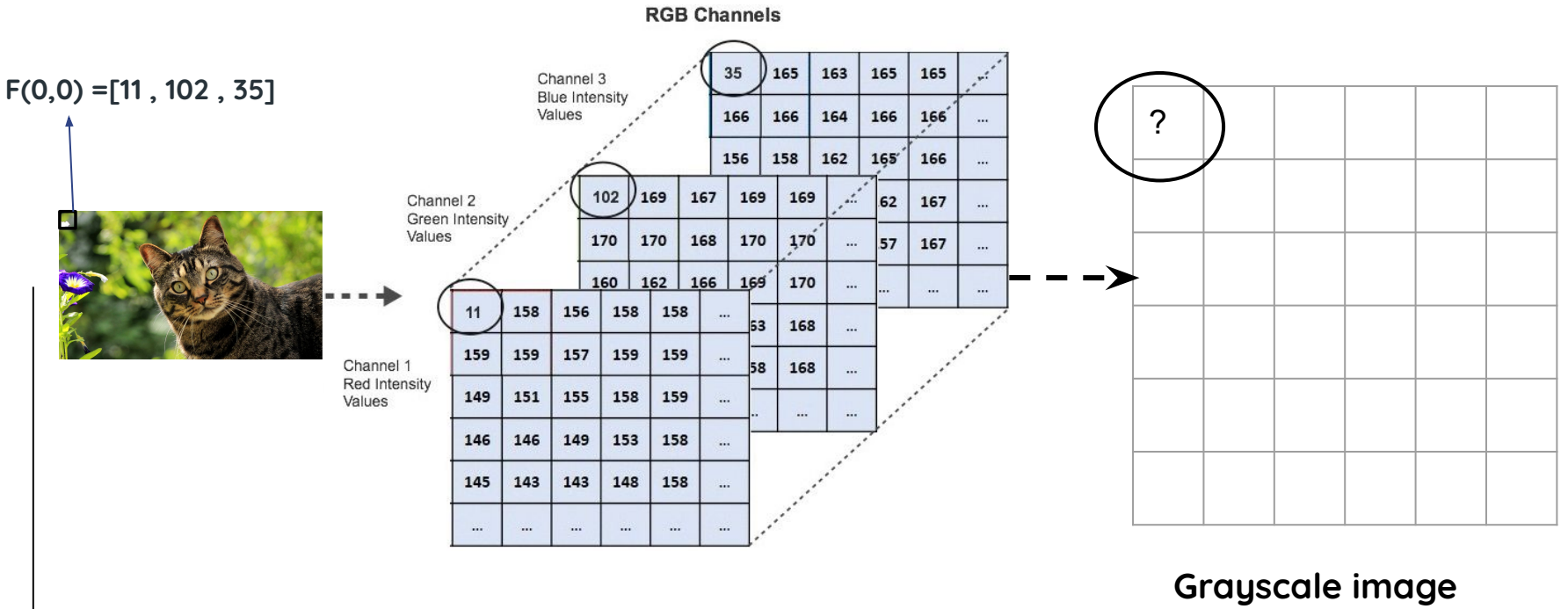


What is an image?



[source: https://freecontent.manning.com/the-computer-vision-pipeline-part-2-input-images/]

What is an image?



[source: https://freecontent.manning.com/the-computer-vision-pipeline-part-2-input-images/]

What is an image?

F(0,0) = R(0,0) x G(0,0) x B(0,0) ?

F(0,0) =[11 , 102 , 35]

RGB Channels

Channel 3
Blue Intensity
Values

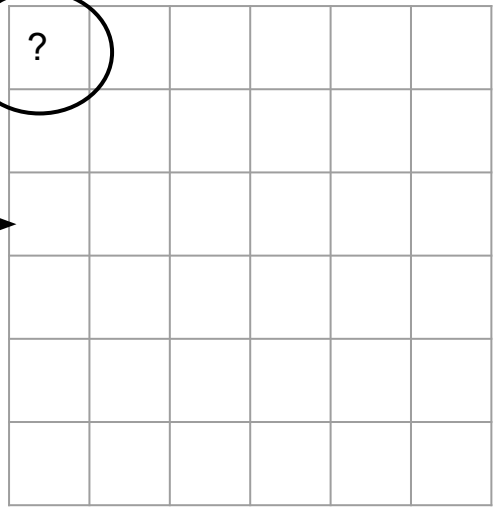
35	165	163	165	165	...
166	166	164	166	166	...
156	158	162	165	166	...

Channel 2
Green Intensity
Values

102	169	167	169	169	...
170	170	168	170	170	...
160	162	166	169	170	...

Channel 1
Red Intensity
Values

11	158	156	158	158	...
159	159	157	159	159	...
149	151	155	158	159	...
146	146	149	153	158	...
145	143	143	148	158	...
...	...	...	...	...	...



Grayscale image

[source: https://freecontent.manning.com/the-computer-vision-pipeline-part-2-input-images/]

What is an image?

Grayscale formula

$$Gray = 0.299.R + 0.587.G + 0.114.B$$

[source: https://freecontent.manning.com/the-computer-vision-pipeline-part-2-input-images/]

What is an image?

F(0,0) = 0.299 x 11 +0.587 x 102

+ 0.114 x 35 = 67.15

F(0,0) =[11 , 102 , 35]

RGB Channels

Channel 3
Blue Intensity
Values

35	165	163	165	165	...
166	166	164	166	166	...
156	158	162	165	166	...
...	...	...	...	...	...

Channel 2
Green Intensity
Values

102	169	167	169	169	...
170	170	168	170	170	...
160	162	166	169	170	...
...	...	...	...	...	...

Channel 1
Red Intensity
Values

11	158	156	158	158	...
159	159	157	159	159	...
149	151	155	158	159	...
146	146	149	153	158	...
145	143	143	148	158	...
...	...	...	...	...	...

67					

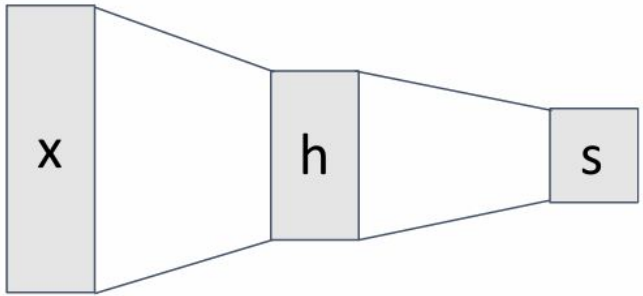
Grayscale image

[source: https://freecontent.manning.com/the-computer-vision-pipeline-part-2-input-images/]

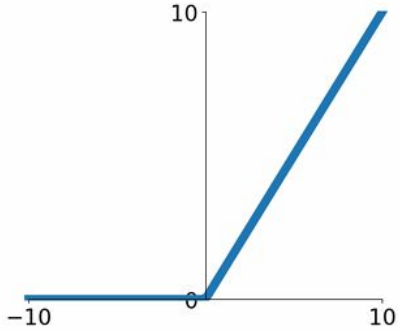
Last Time

Fully-Connected Neural Network

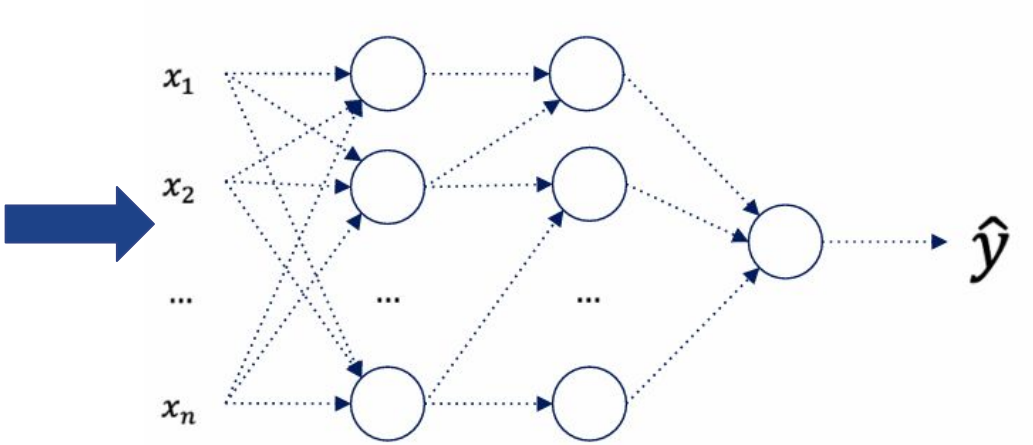
Fully-Connected Layers



Activation Function



Fully-Connected Neural Network



Input image dimension:

2000 x 1000 x 3

(Height × Width × Channels)

Input layer

6 Millions

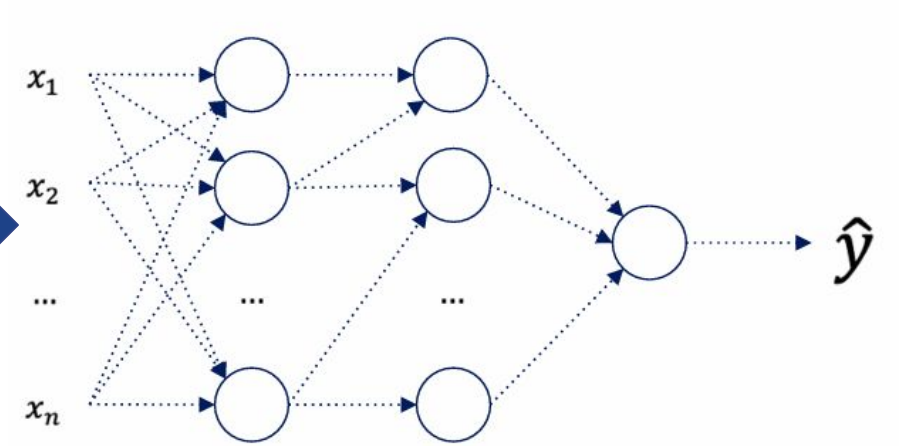
(2000 x 1000 x 3)

1 hidden layer

1000 neurons

Fully-Connected Neural Network

The flattening concept



Input image dimension:

2000 x 1000 x 3

(Height × Width × Channels)

Input layer

6 Millions

(2000 x 1000 x 3)

+

1 hidden layer

1000 neurons

=

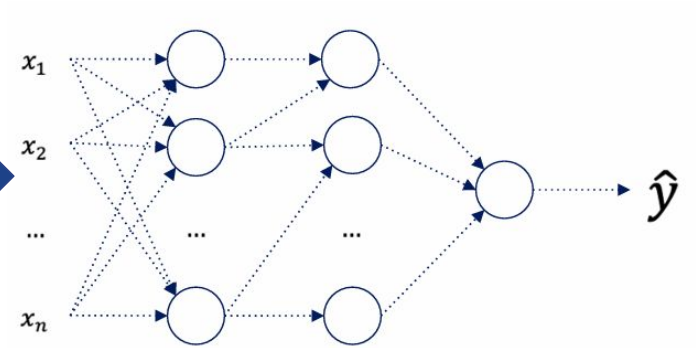
6,000,000 × 1000=

6,000,000,000

Fully-Connected Neural Network

6 BILLION weights (+ 1000 biases)

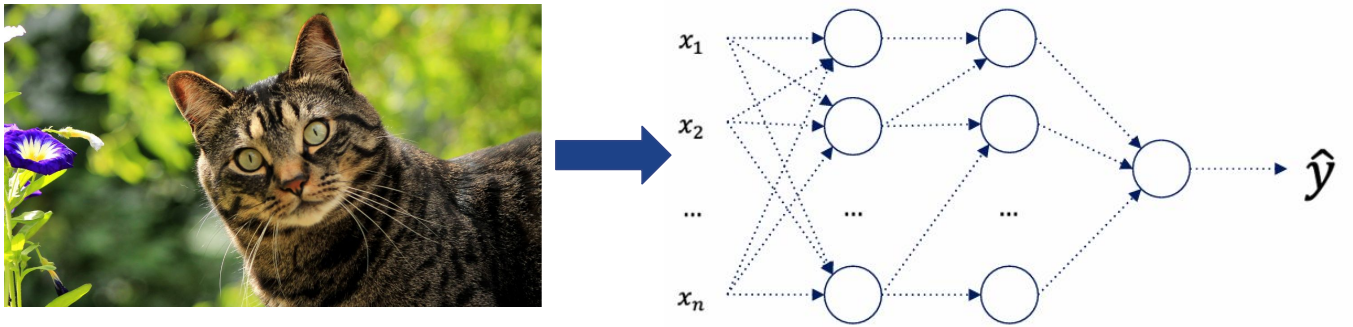
Just for the first layer



Fully-Connected Neural Network

Storing **6B** parameters in float32

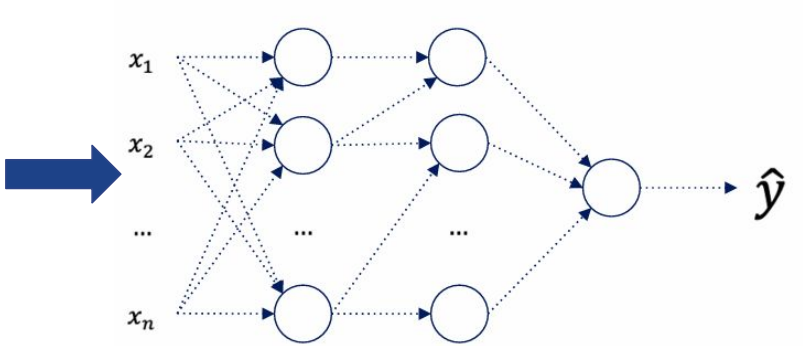
requires about **24 GB of memory.**



Fully-Connected Neural Network

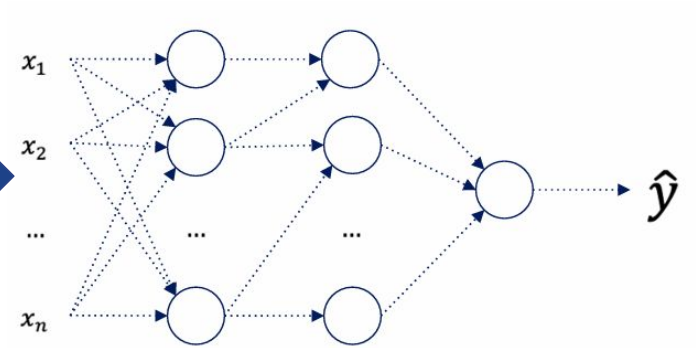
Fully connected neural networks are not adapted to image data

- Connect neurons in hidden layer to all neurons in input layer
- Too many parameters
- No spatial information



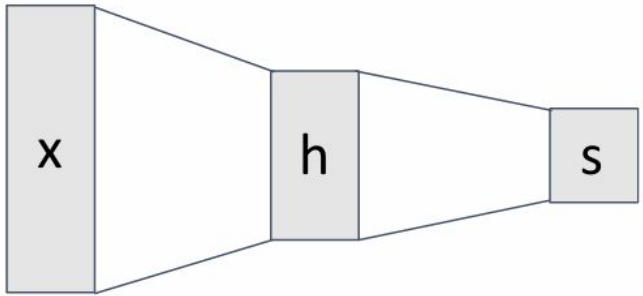
Fully-Connected Neural Network

What if Convolutional Network can resolve this ?

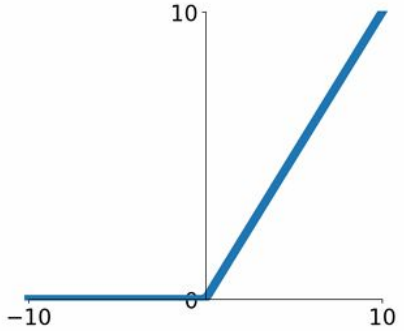


Convolutional Network

Fully-Connected Layers

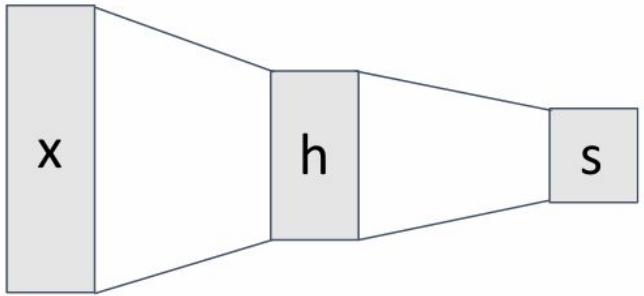


Activation Function

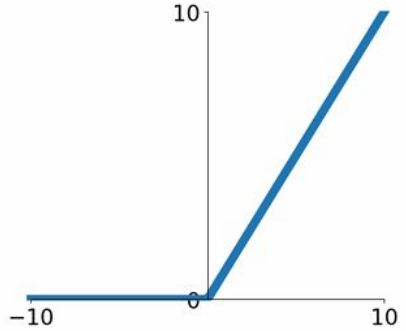


Components of a Convolutional Network

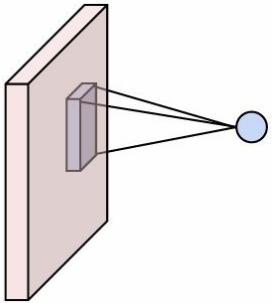
Fully-Connected Layers



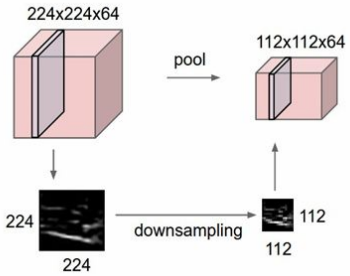
Activation Function



Convolution Layers



Pooling Layers

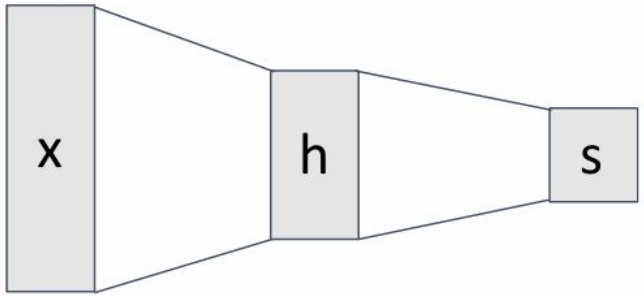


Normalization

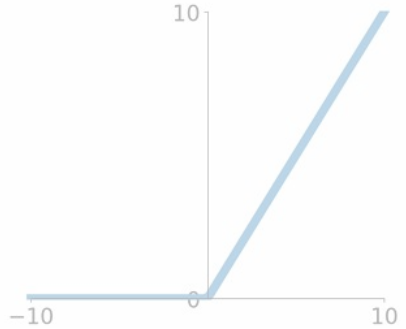
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Components of a Convolutional Network

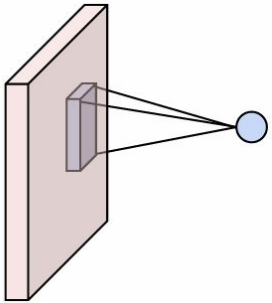
Fully-Connected Layers



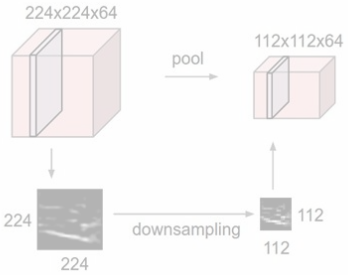
Activation Function



Convolution Layers



Pooling Layers

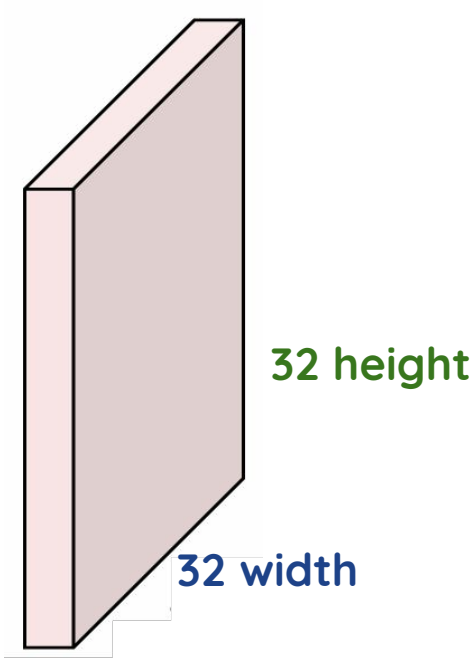


Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Convolution Layer

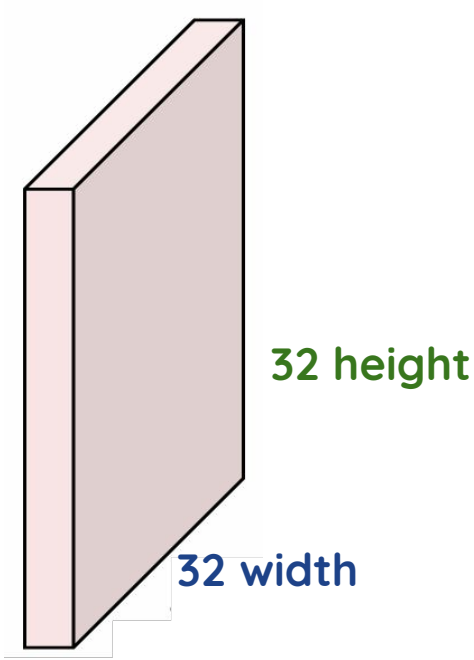
3 x 32 x 32 image



We want to preserve spatial structure because nearby pixels are related

Convolution Layer

3 x 32 x 32 image

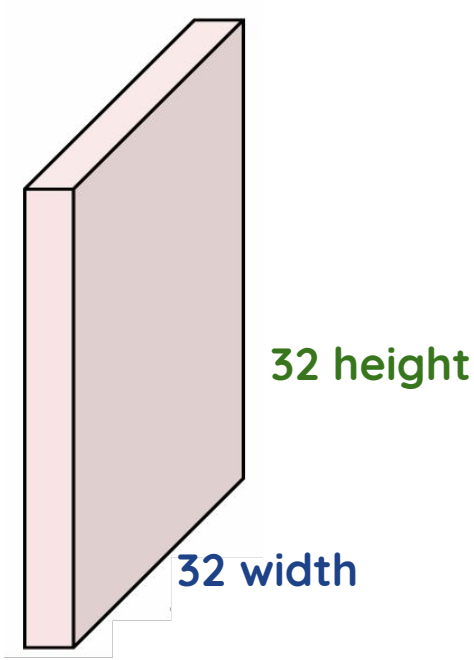


We want to preserve spatial structure because nearby pixels are related

How do we compute features without flattening?

Convolution Layer

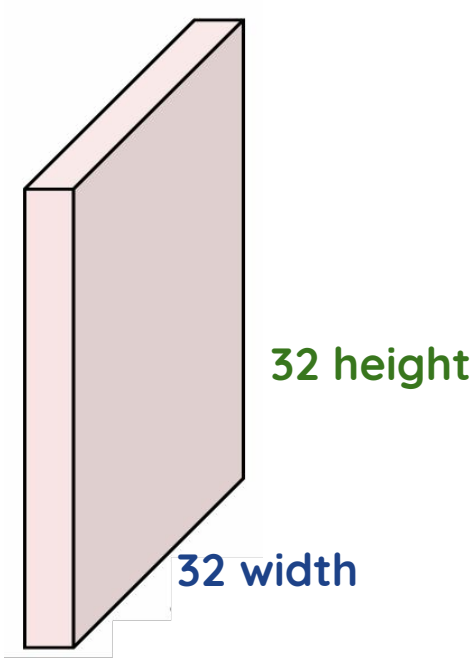
3 x 32 x 32 image



- Images are made of local patterns: edges, corners, textures (like we saw in the last chapter)
- We should look at small neighborhoods, not the whole image

Convolution Layer

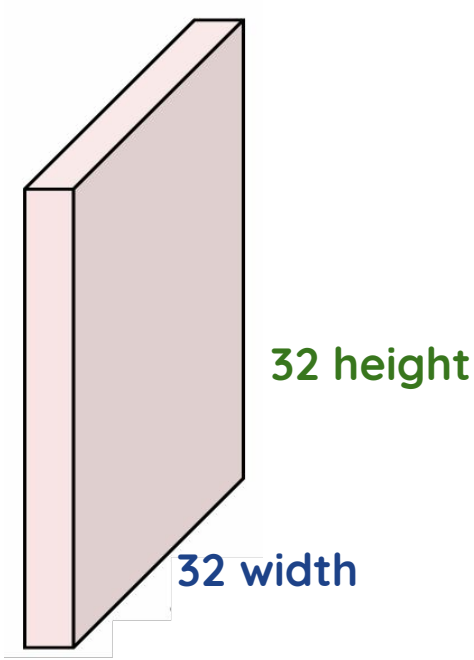
3 x 32 x 32 image



To do that, we introduce a small learnable block called a filter (kernel)

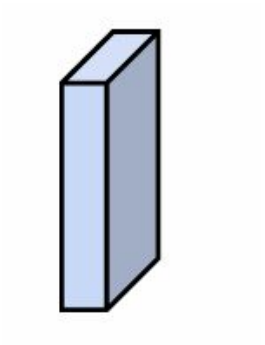
Convolution Layer

3 x 32 x 32 image



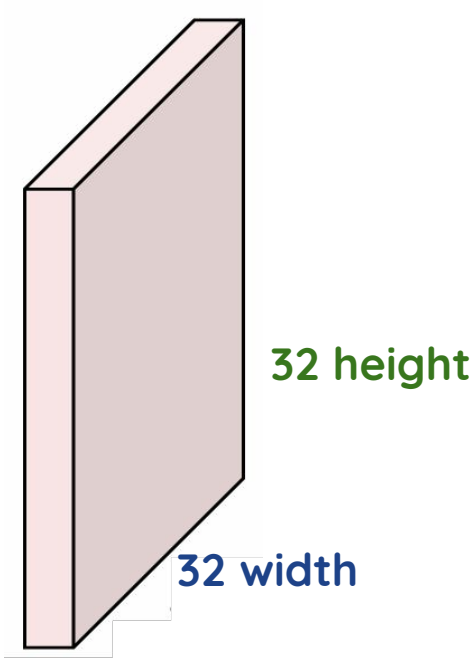
A filter = small block of learnable weights

Example filter size: 3 × 5 × 5



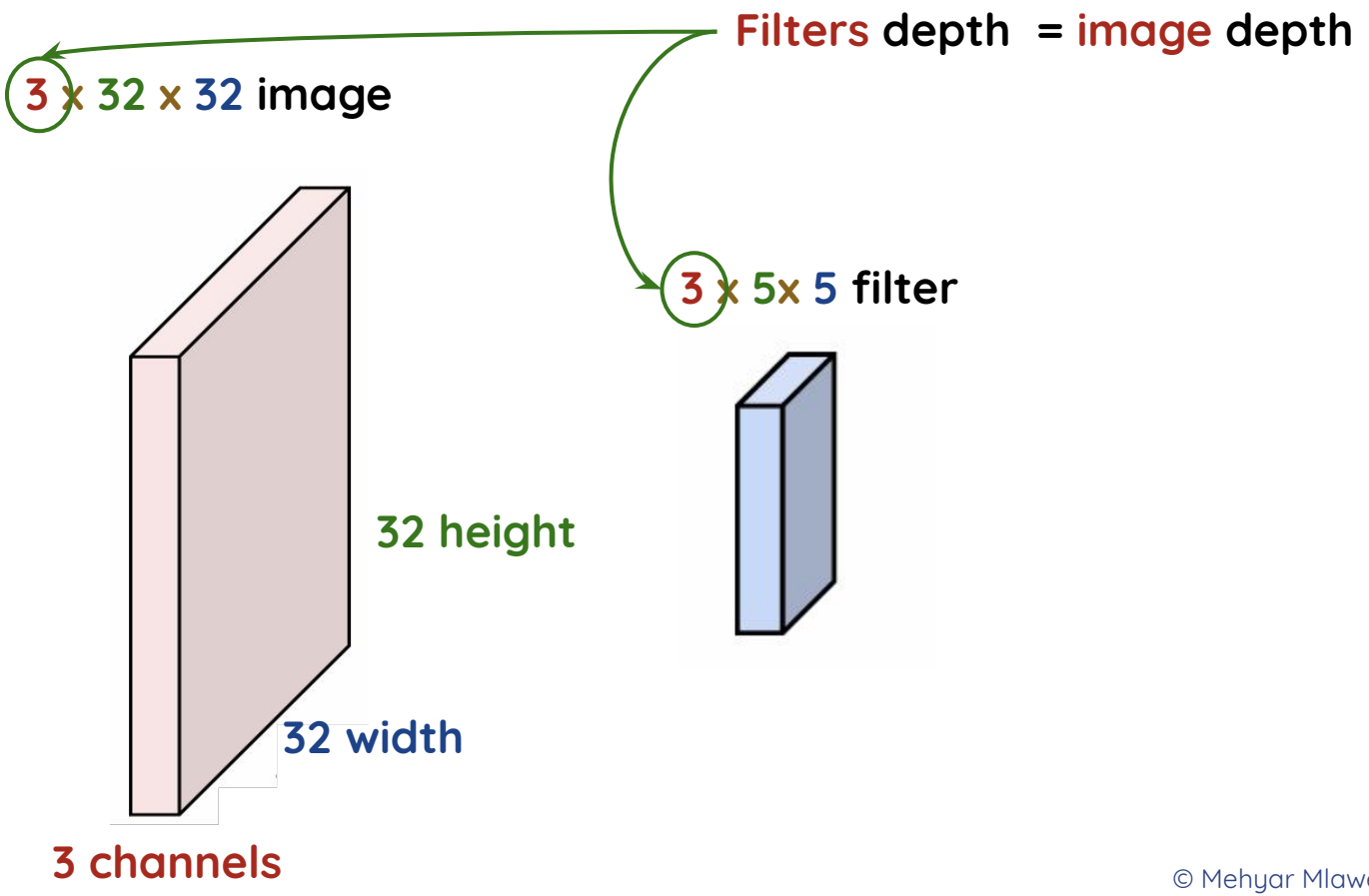
Convolution Layer

3 x 32 x 32 image

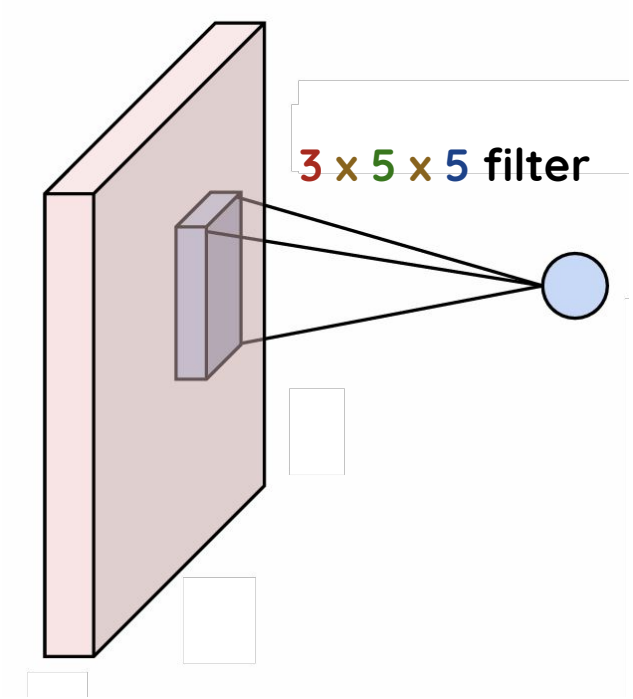


3 x 5x 5 filter

Convolution Layer



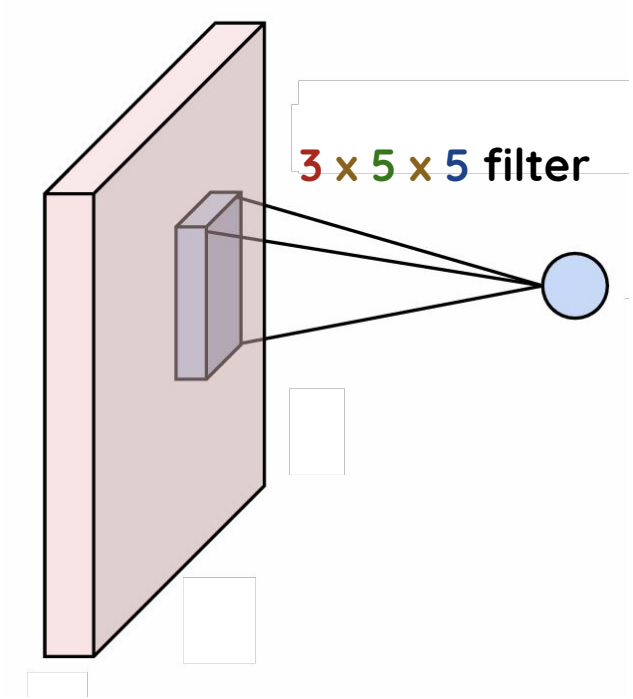
Convolution Layer



Produces 1 number:

the result of taking a dot product between the filter and a small 3x5x5 chunk of the image

Convolution Layer



1 Filter + 1 spatial position produces 1 number

W: input

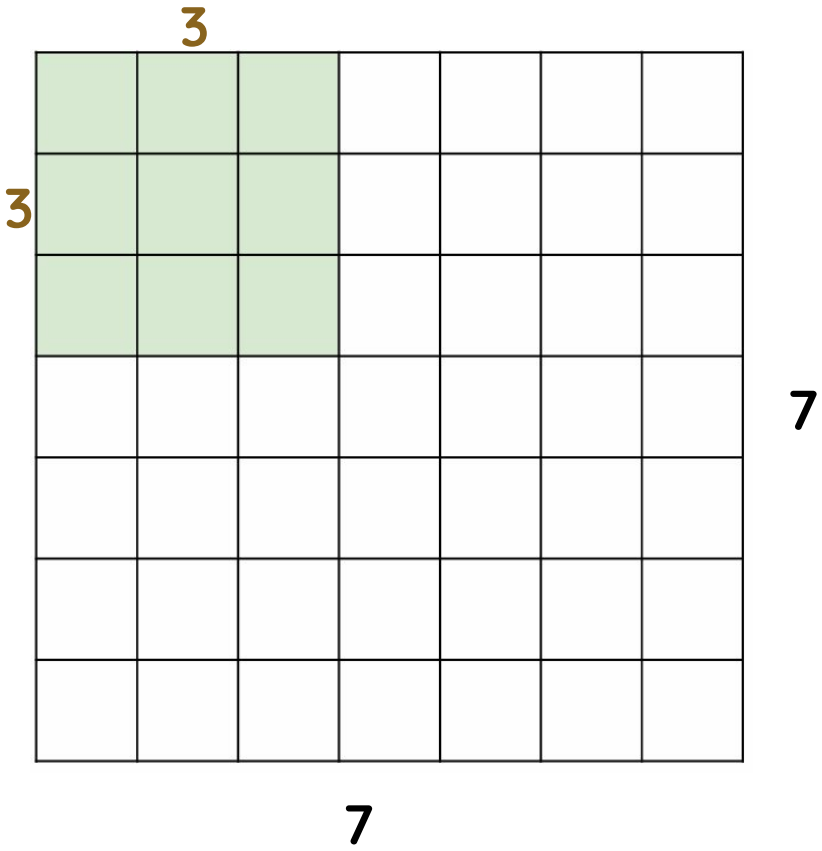
K: Filter (kernel)

Output = W- K + 1

= 32 - 5 +1 = 28

Convolution Layer

A closer look!



3 x 3 x 3 filter

3 x 7 x 7 Image

Convolution Layer

$$= 3 \cdot 1 + 0 \cdot 3 + 1 \cdot 2 + 1 \cdot 3 + 5 \cdot 2 + 8 \cdot 1 + 2 \cdot 2 + 7 \cdot 1 + 2 \cdot 1$$

3 x 3 x 3 filter

3 x 7 x 7 Image

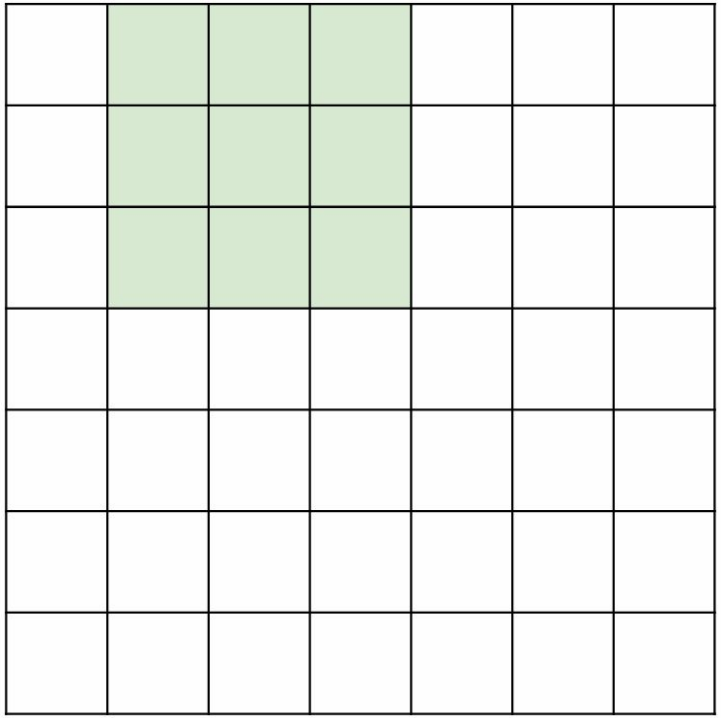
3	0	1	2	7	4	5
1	5	8	9	3	1	0
2	7	2	5	1	3	2
0	1	3	7	8	5	6
9	8	5	7	5	0	1
7	8	9	7	8	6	0
7	8	0	5	2	3	1

1	3	2
3	2	1
2	1	1

39				

Convolution Layer

Sliding the kernel

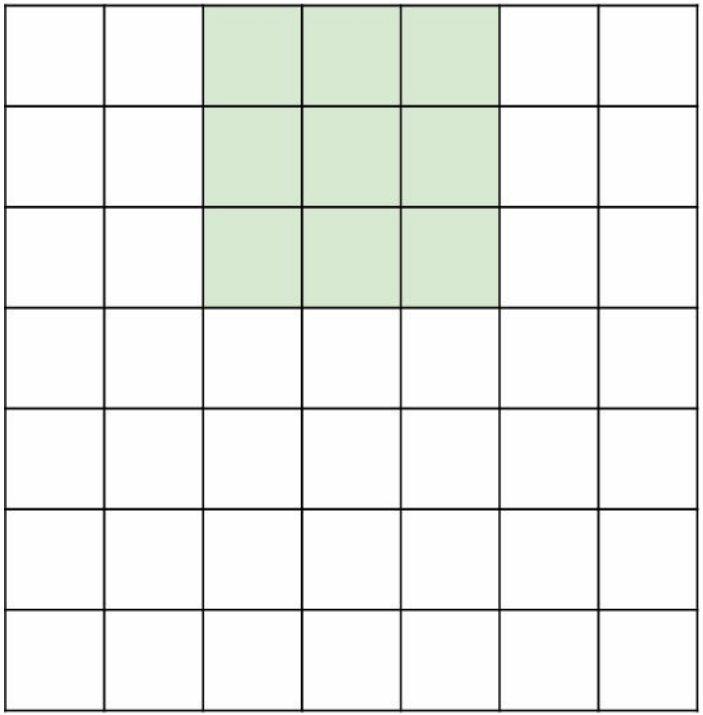


3 x 3 x 3 filter

3 x 7 x 7 Image

Convolution Layer

Sliding the kernel



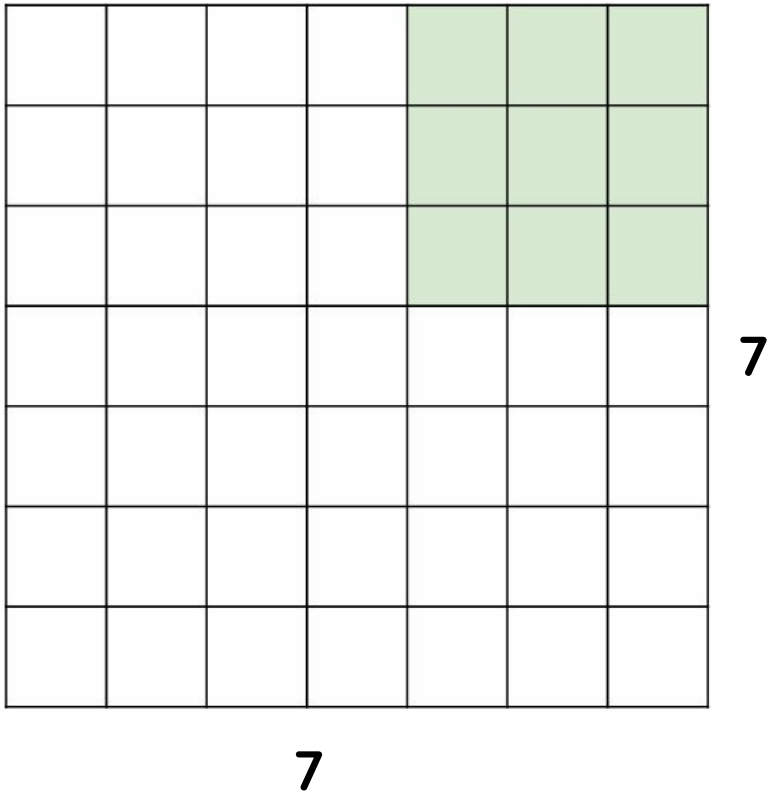
7

7

3 x 3 x 3 filter

3 x 7 x 7 Image

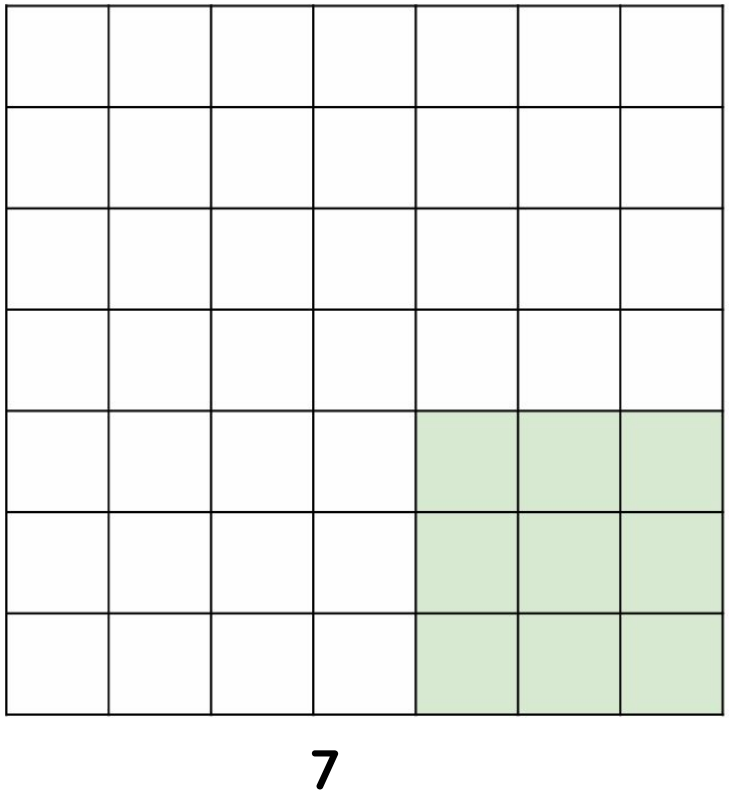
Convolution Layer



3 x 3 x 3 filter

3 x 7 x 7 Image

Convolution Layer

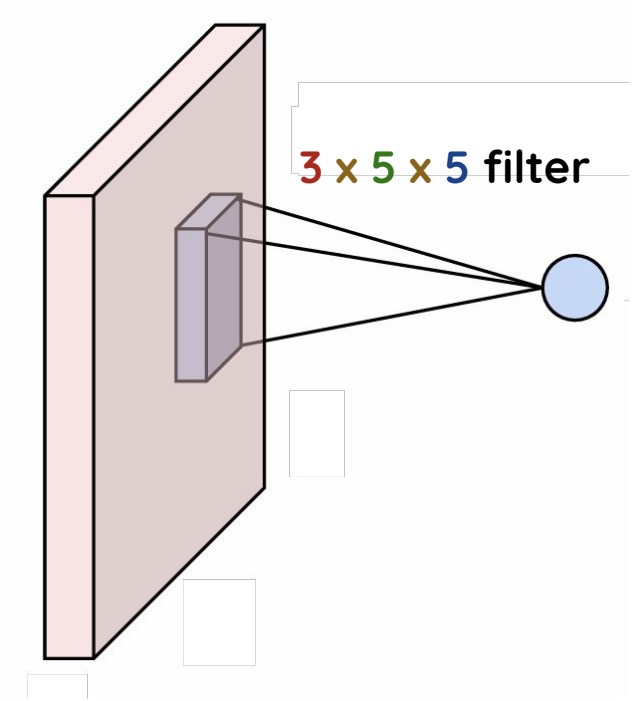


3 x 3 x 3 filter

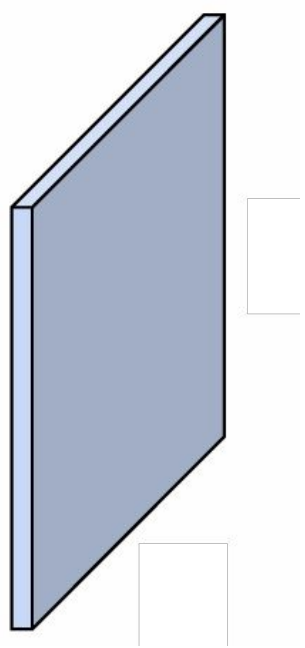
3 x 7 x 7 Image

Output : 5 x 5

Convolution Layer

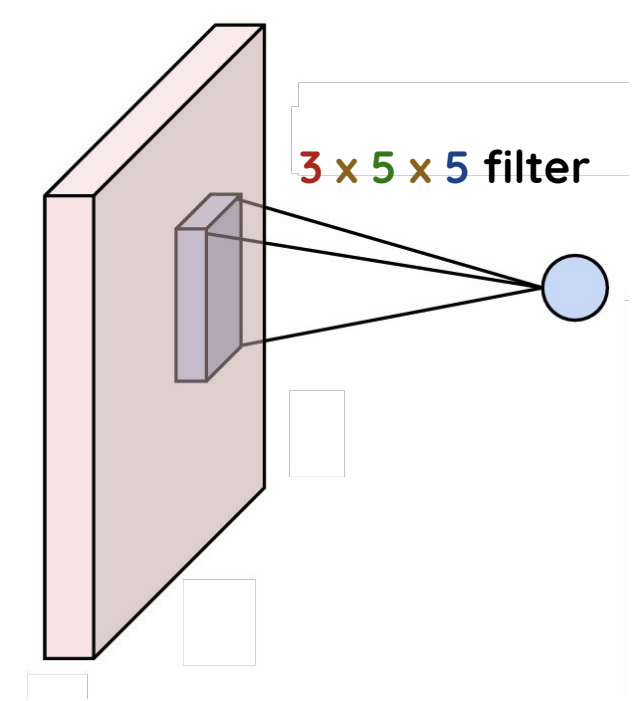


After convolving (sliding) the filter over all spatial locations

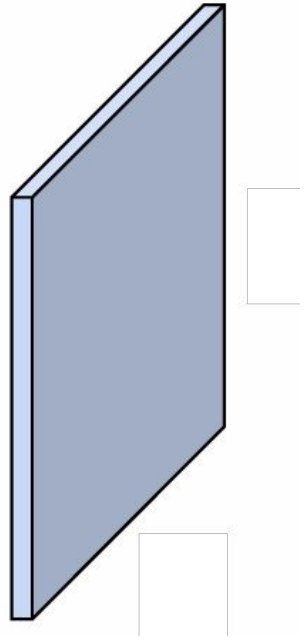


Convolution Layer

Activation map 1 x 28 x 28

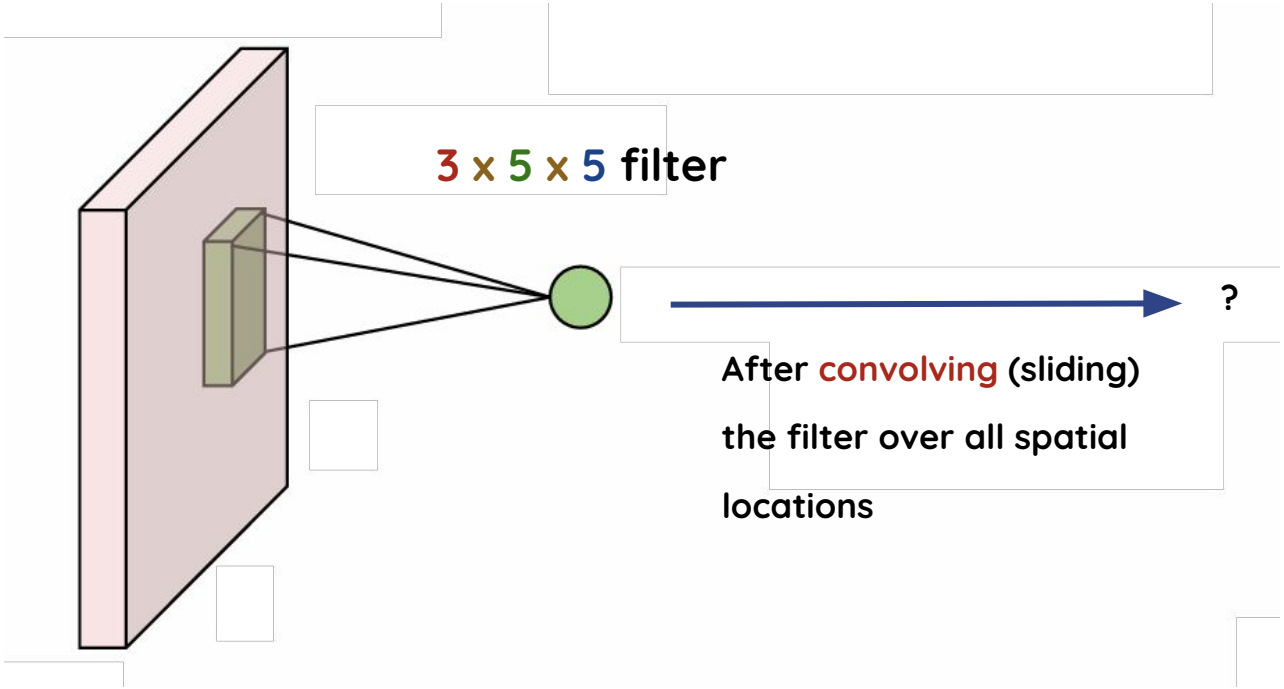


After convolving (sliding) the filter over all spatial locations



Convolution Layer

Consider repeating with a second filter

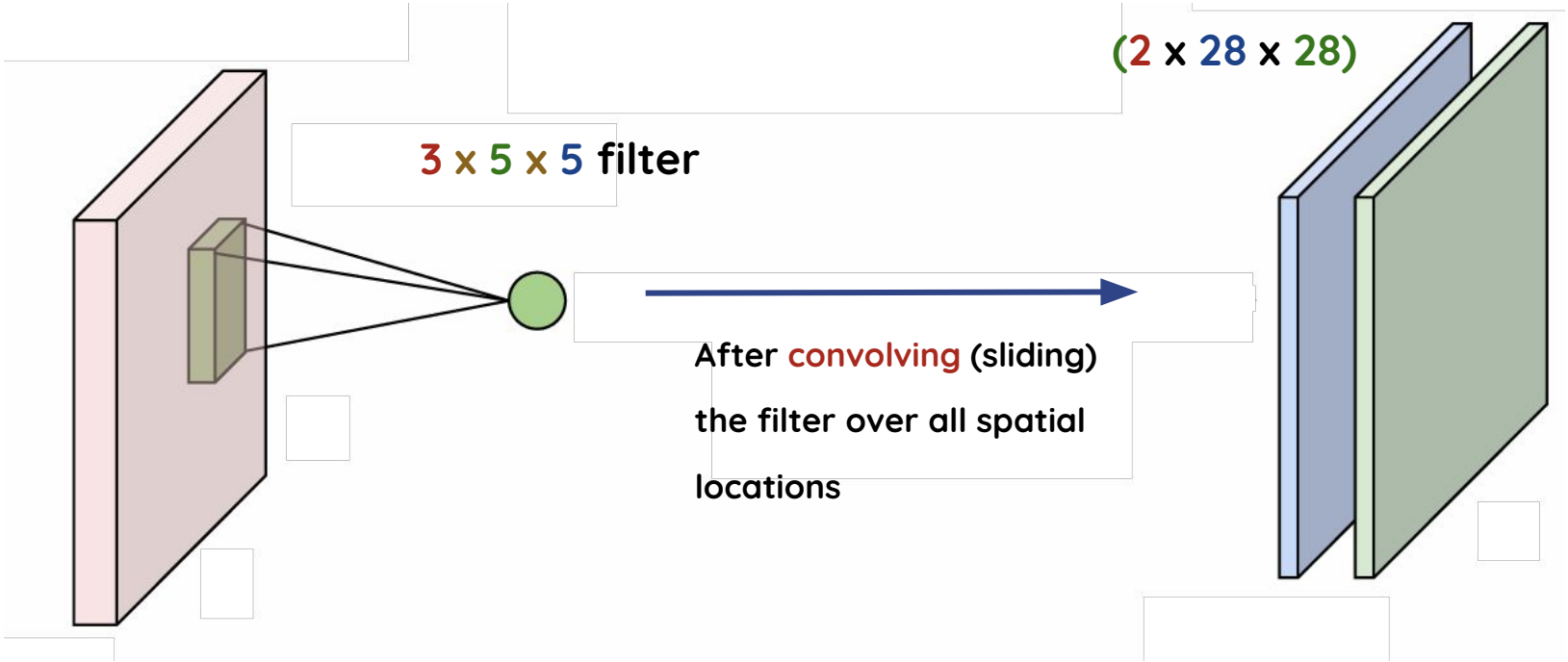


Convolution Layer

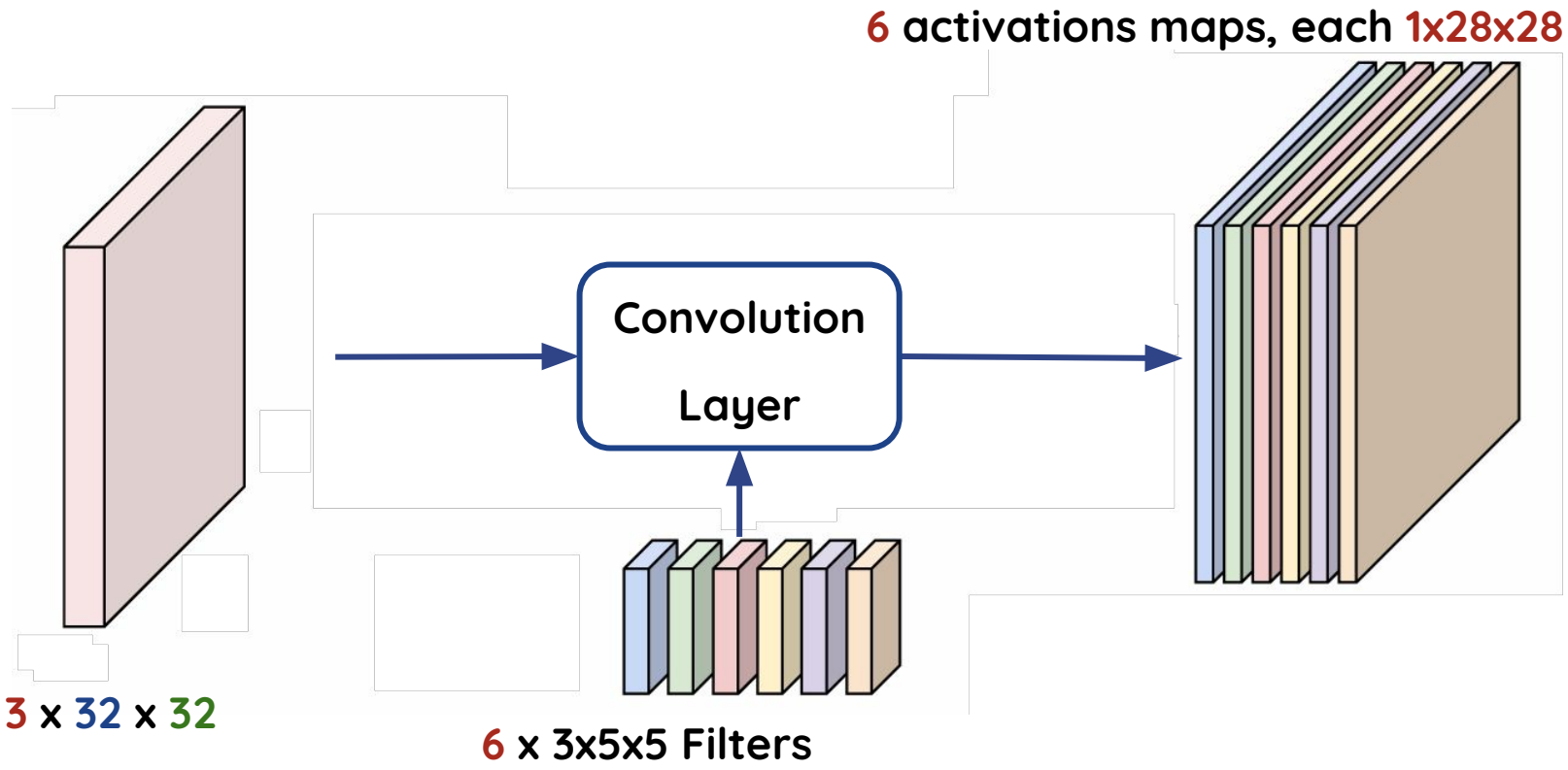
Consider repeating with a second filter

Two activation maps 1 x 28 x 28

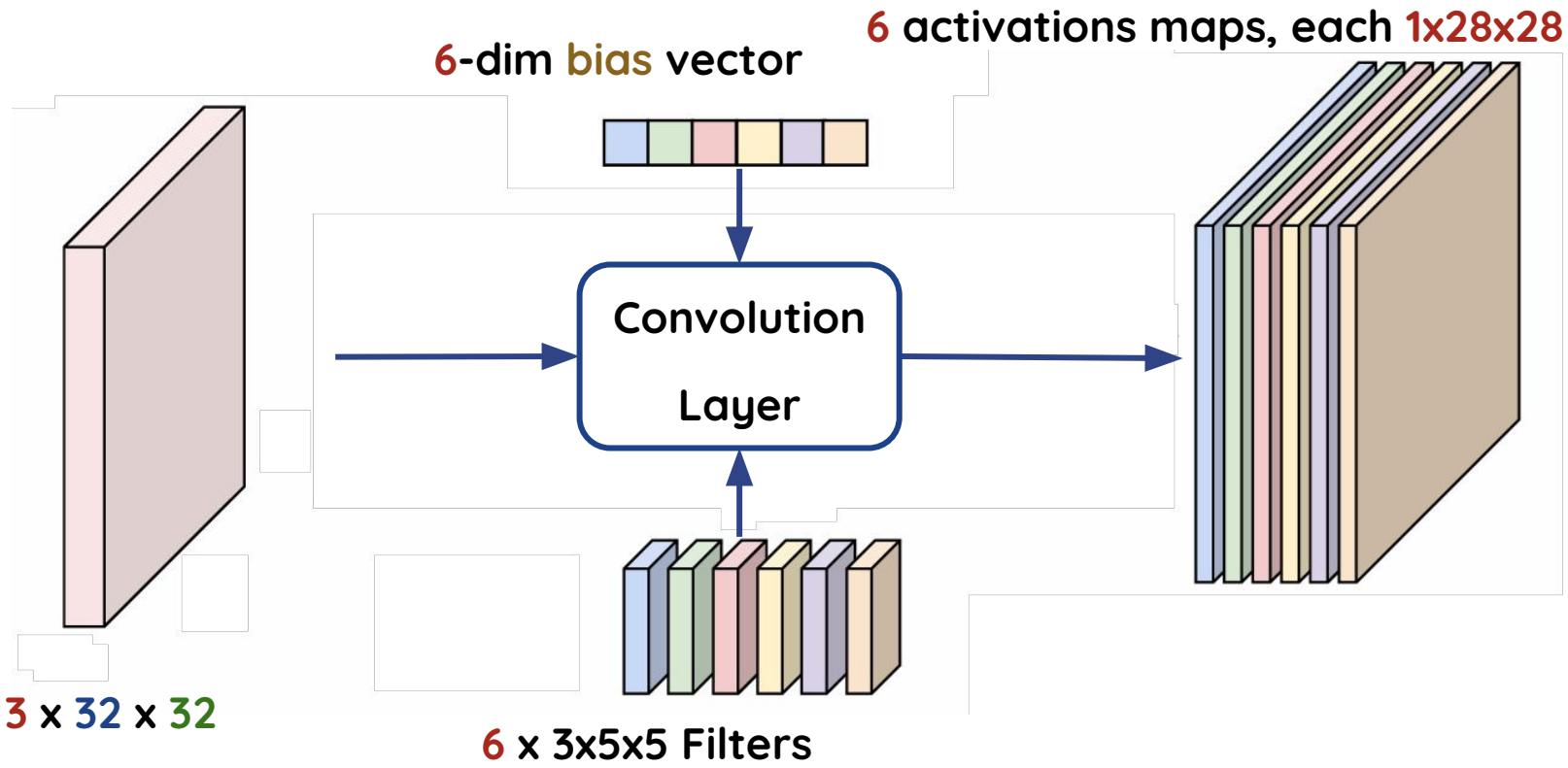
(2 x 28 x 28)



Convolution Layer



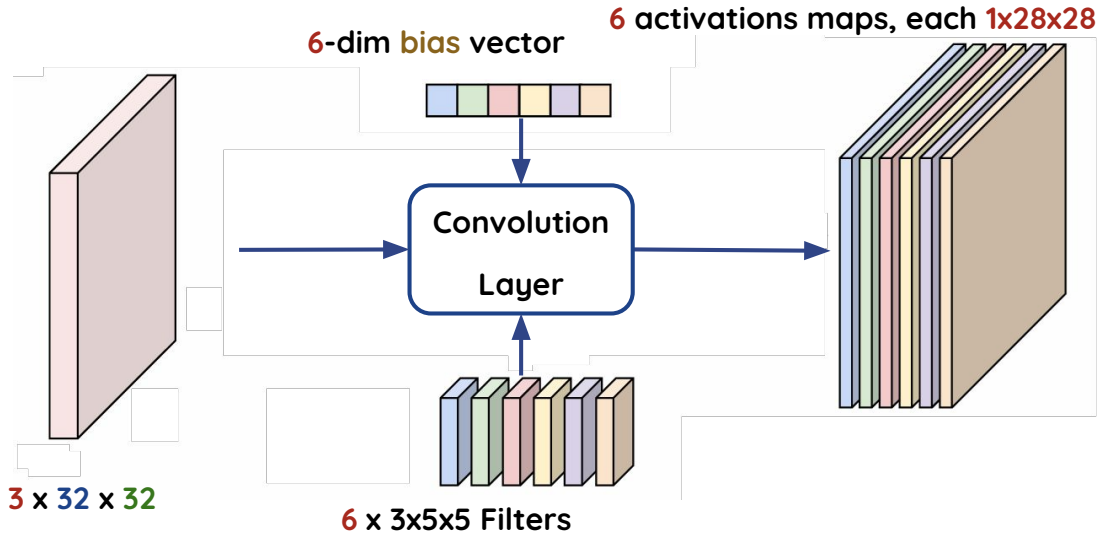
Convolution Layer



Convolution Layer

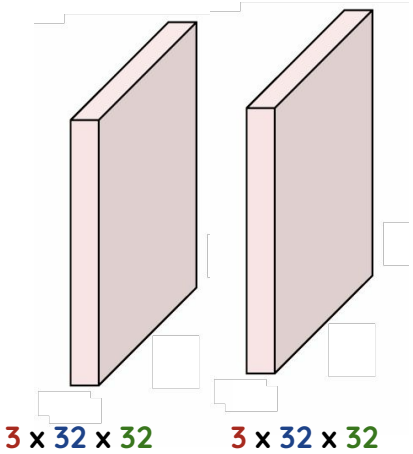
For each filter k :

$$z_{i,j}^{(k)} = (W_k * X)_{i,j} + b_k$$

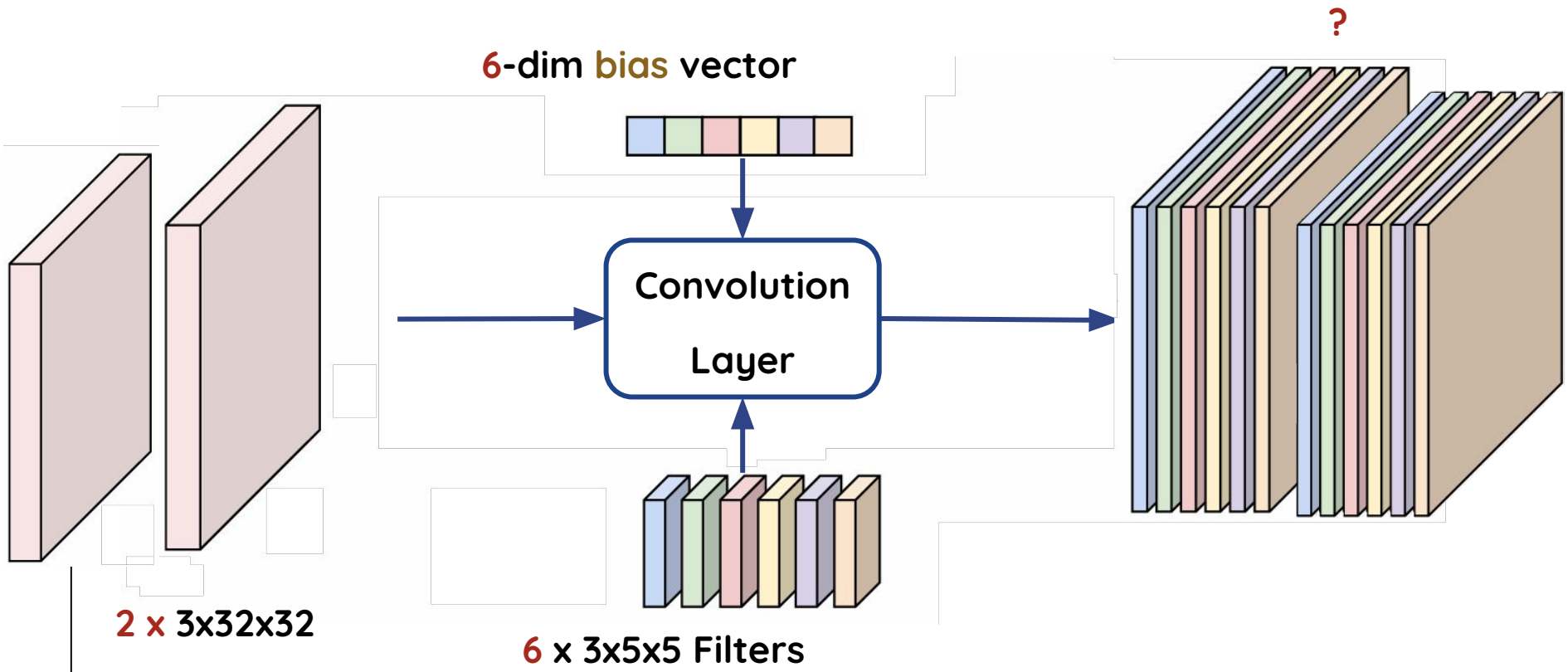


Convolution Layer

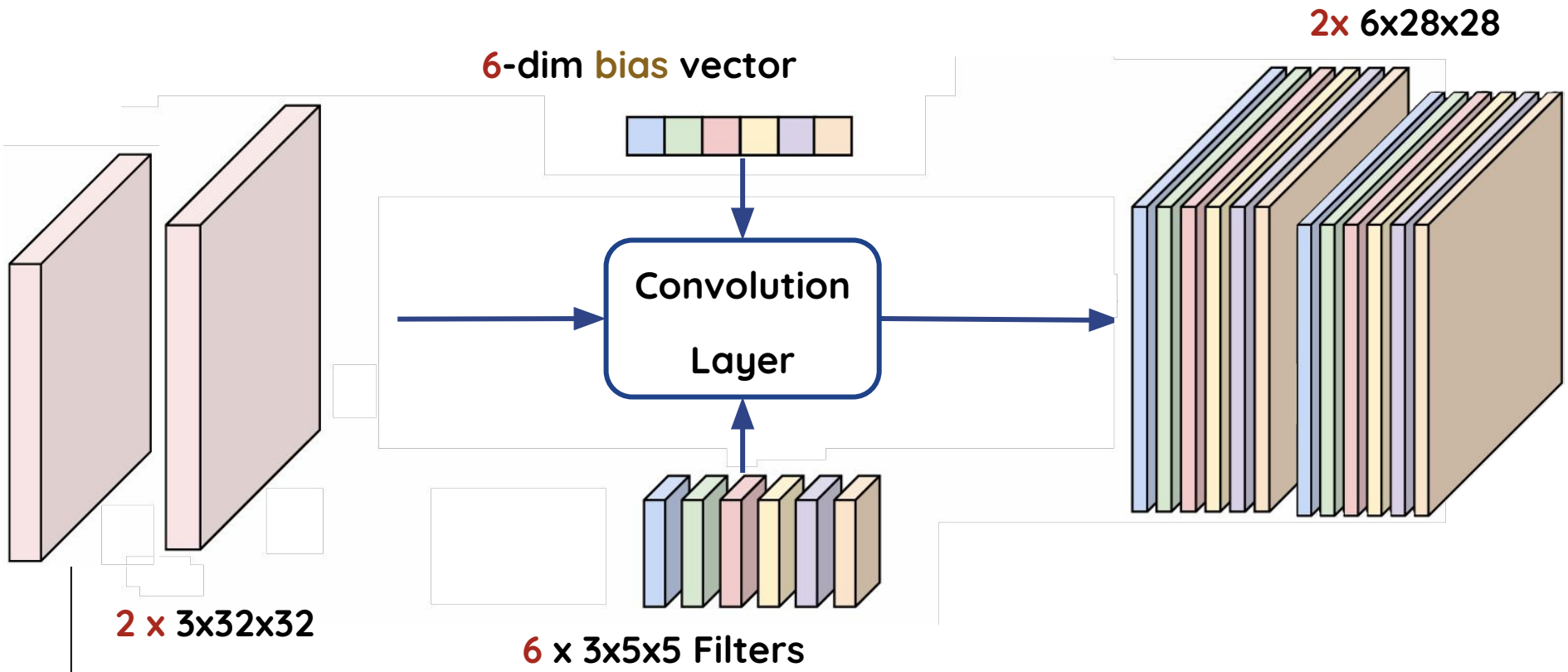
What if we have batch of images ?



Convolution Layer

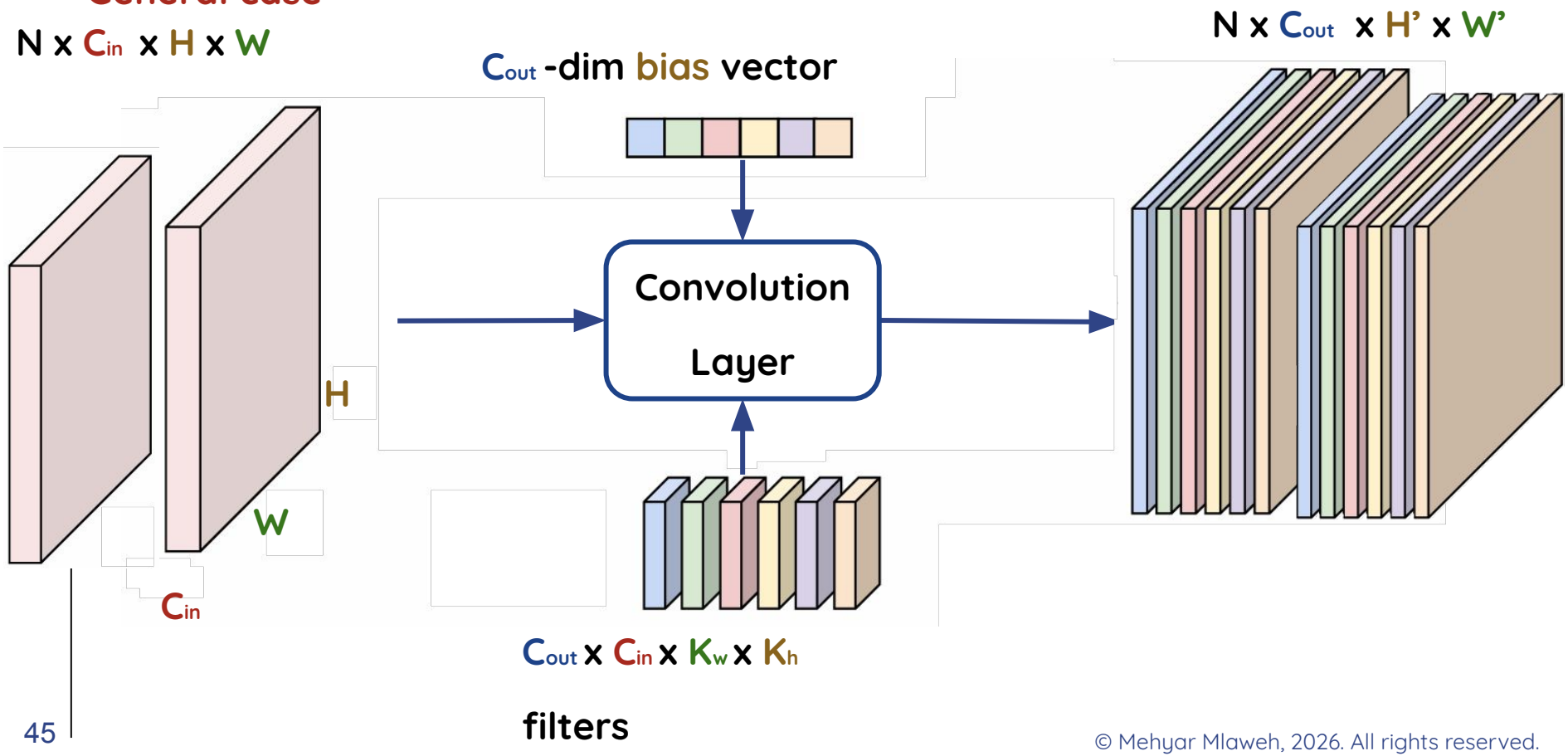


Convolution Layer

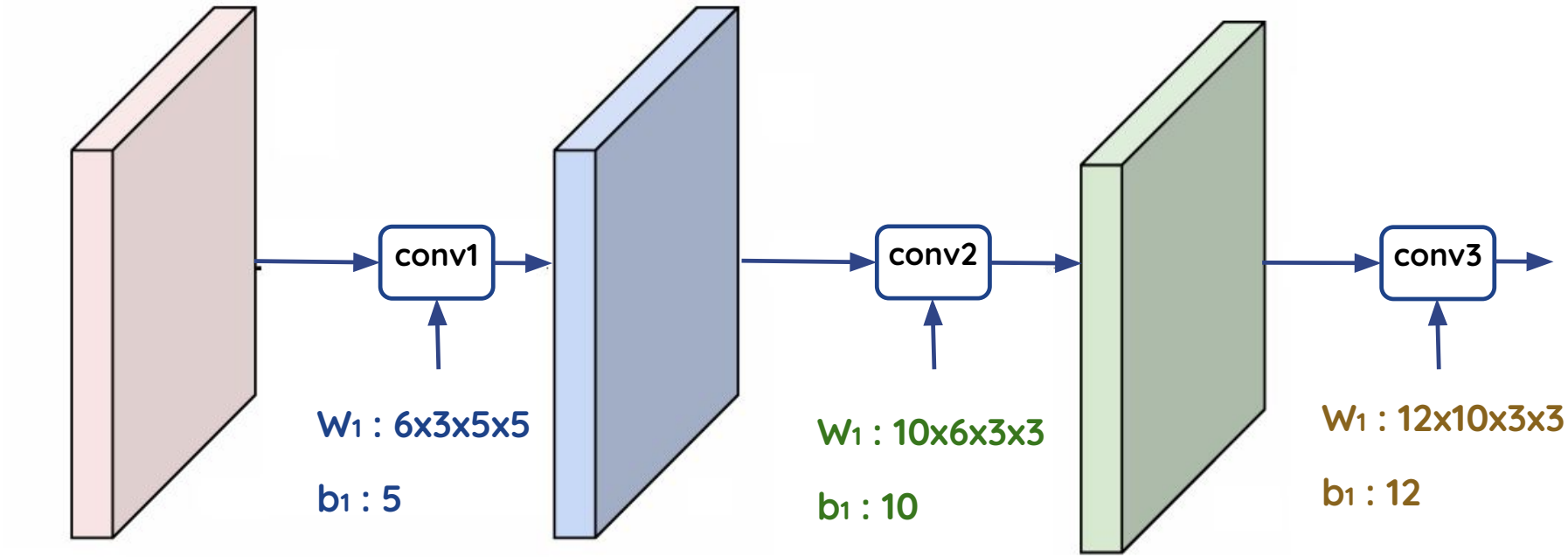


Convolution Layer

General case

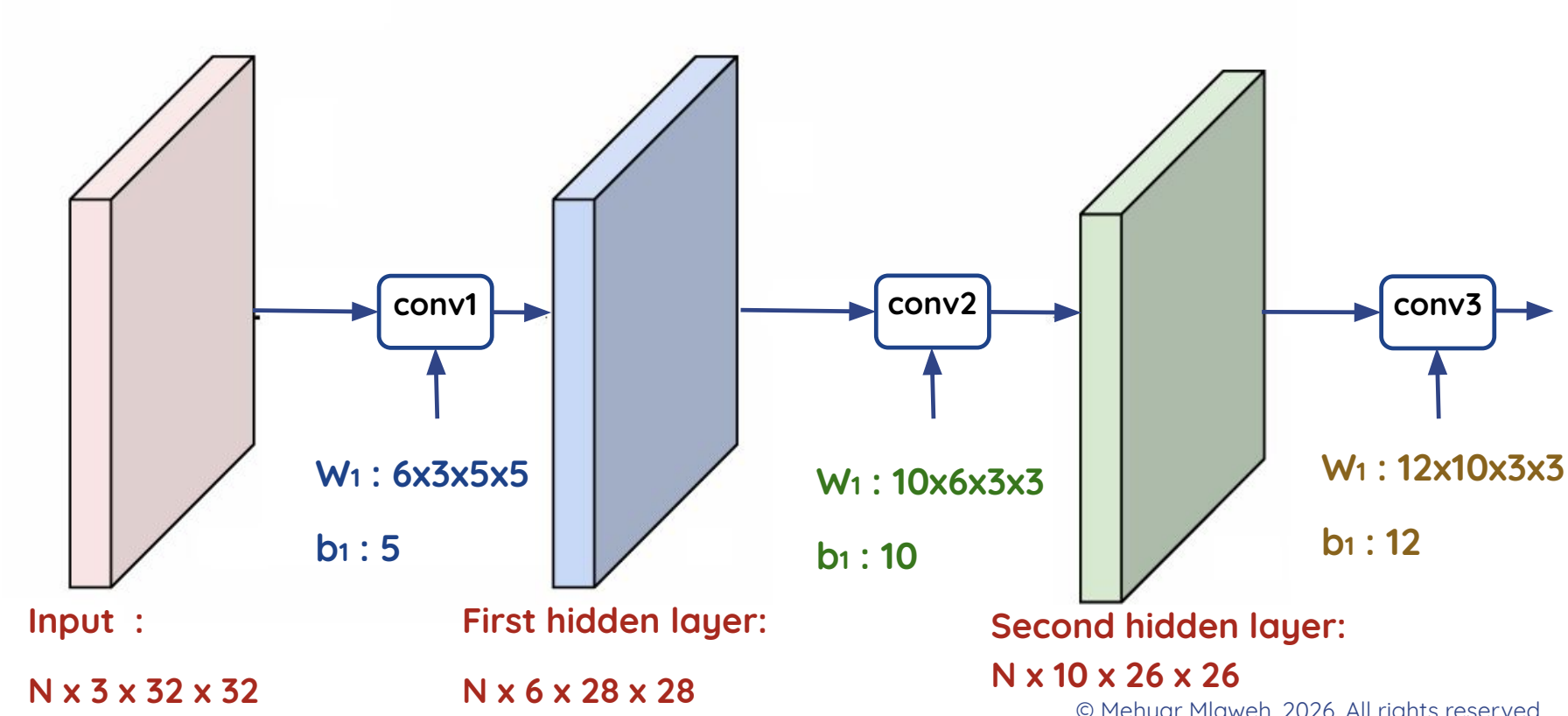


Stacking Convolutions



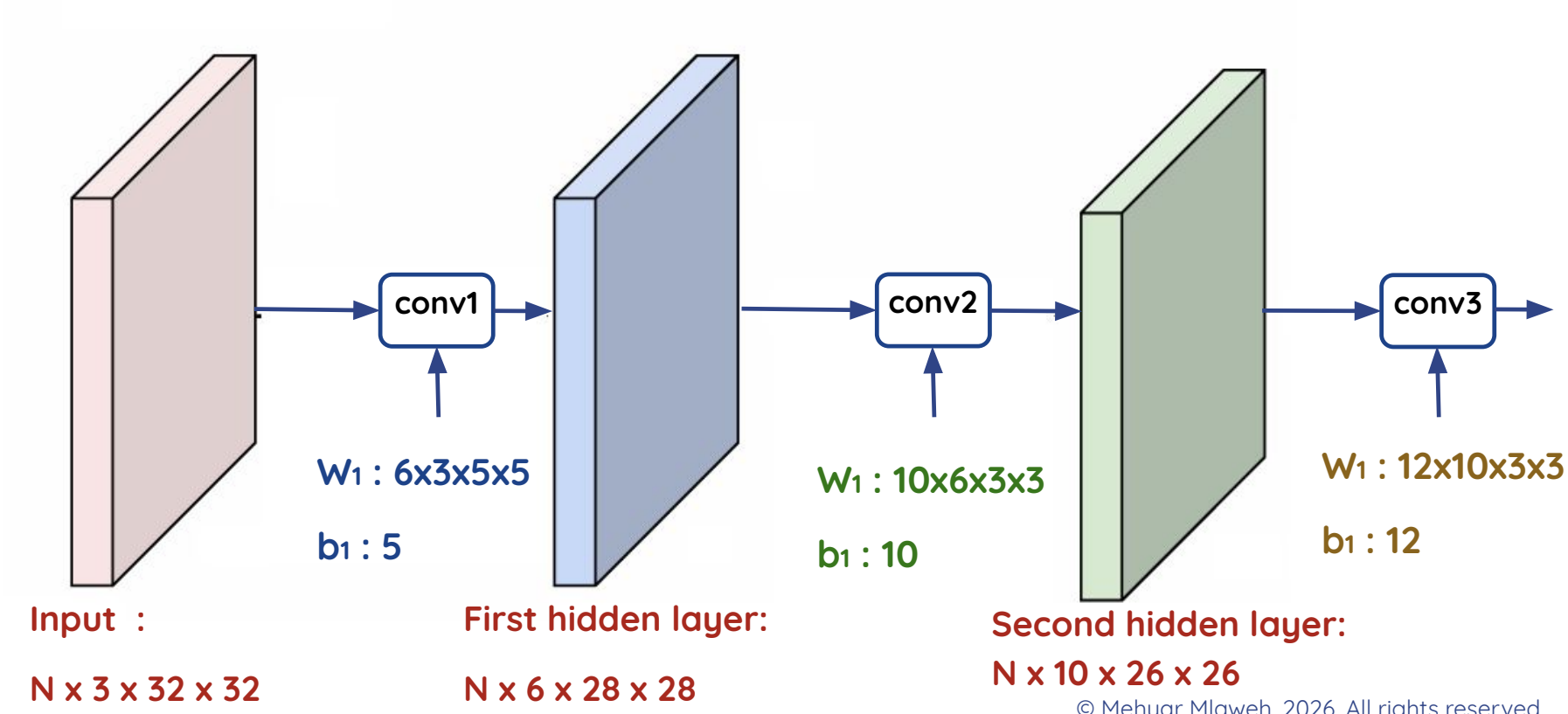
Convolution neural network

So convolution is just linear filtering



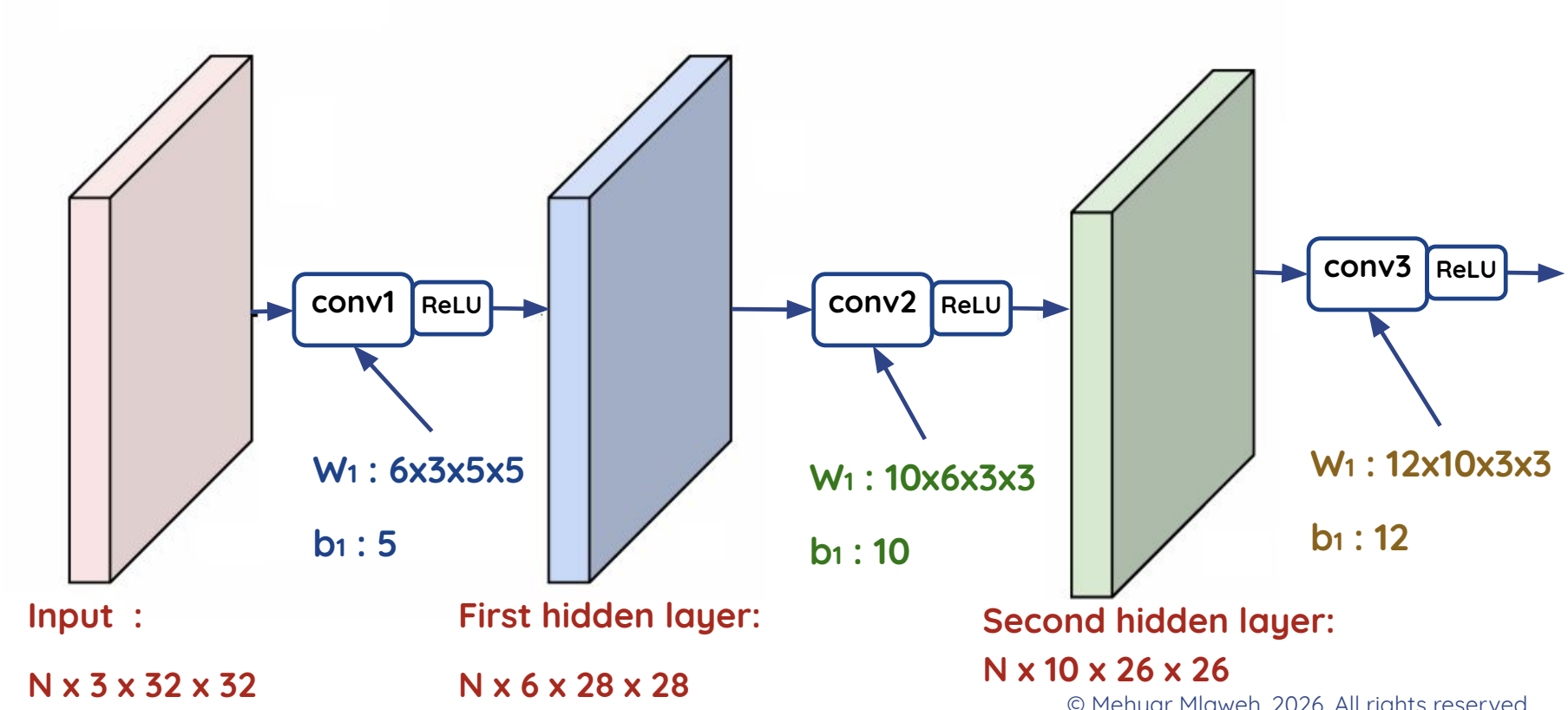
Convolution neural network

We are missing non-linearity



Convolution neural network

Let's try to add some non-linearity



Convolution neural network

Let's try to add some non-linearity

Conv → ReLU → Conv → ReLU → Conv → ReLU

Convolution neural network

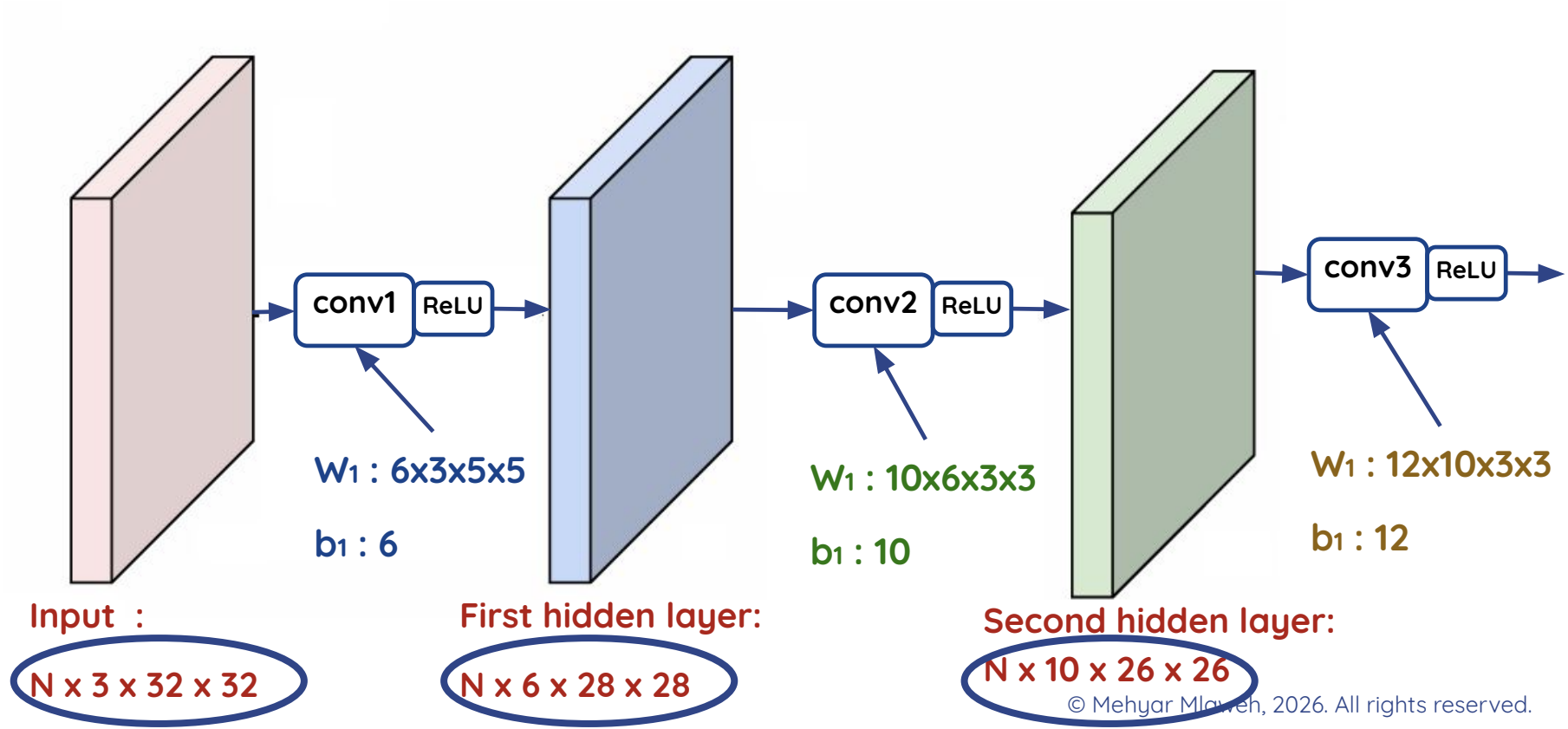
Let's try to add some non-linearity

Conv → ReLU → Conv → ReLU → Conv → ReLU

ReLU(x)= max(0,x)

Convolution neural network

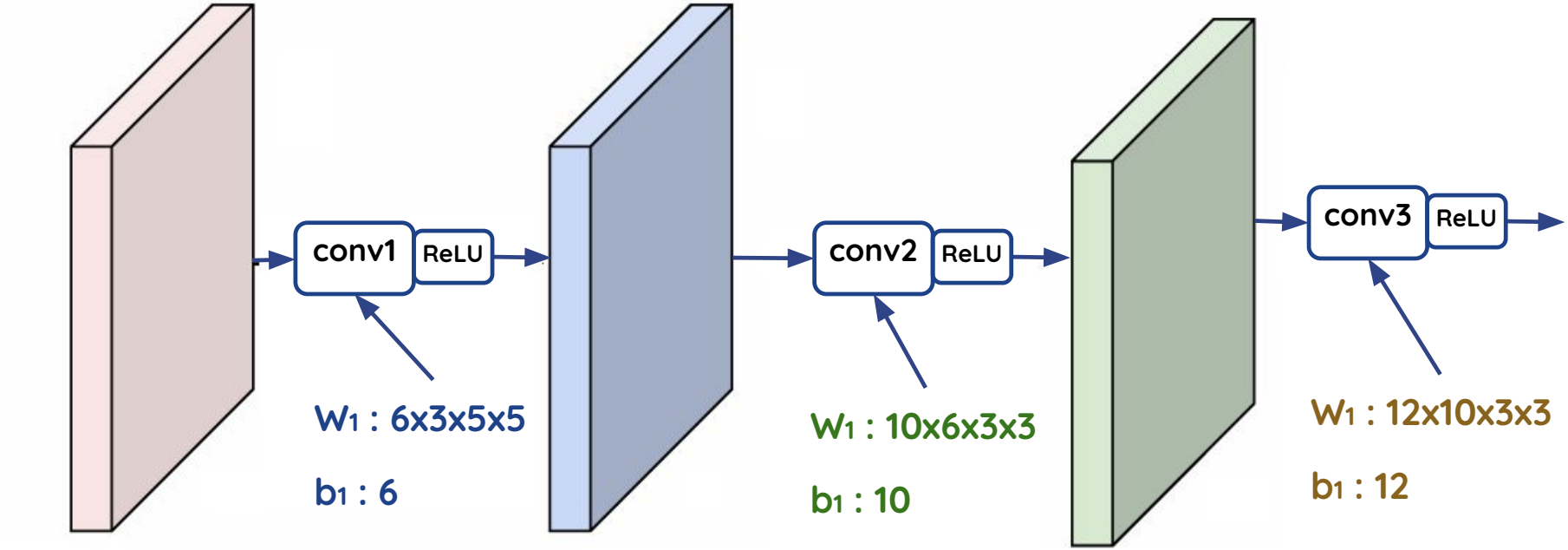
Input: 32×32 After conv1: 28×28 After conv2: 26×26



Convolution neural network

Input: 32×32 After conv1: 28×28 After conv2: 26×26

Every convolution reduces spatial resolution



Convolution neural network

32 → 28 → 26 → 24 → …

- **Deep networks shrink too fast**
- **We may lose spatial information**

Convolution neural network

32 → 28 → 26 → 24 → …

- Deep networks shrink too fast
- We may lose spatial information

Solution : padding

Add zeros around the input

Convolution neural network

32 → 28 → 26 → 24 → …

- Deep networks shrink too fast
- We may lose spatial information

Solution : padding

Add zeros around the input

Convolution neural network

Solution : padding

Add zeros around the input

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

Convolution neural network

Dauphine | PSL UNIVERSITÉ PARIS

Solution : padding

Add zeros around the input

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

Input : W

Filter : K

Padding : P

Output : W - K + 1 +2.P

Convolution neural network

Solution : padding

Add zeros around the input

Input : W

Filter : K

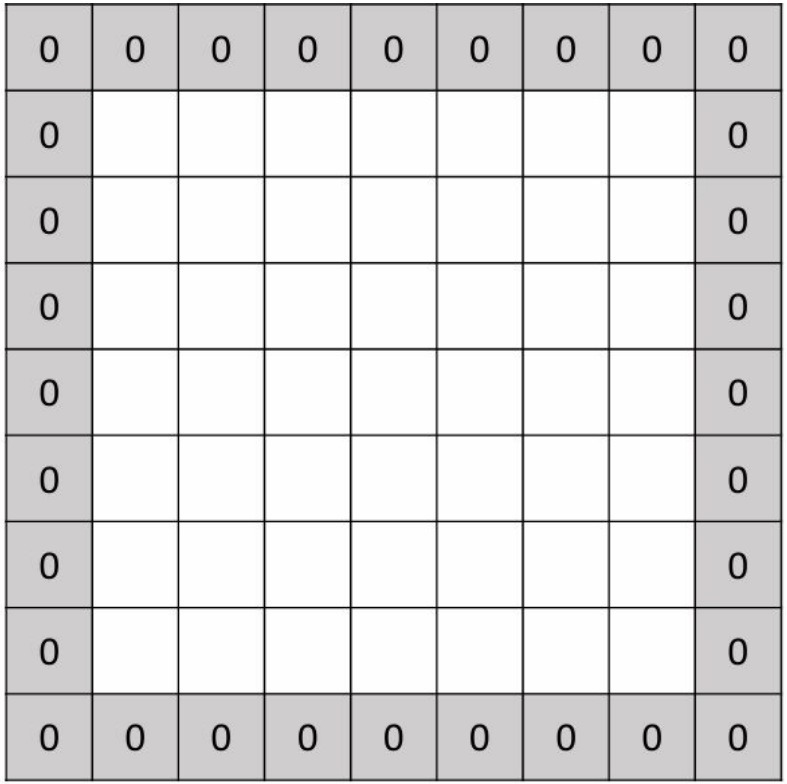
Padding : P

Output : $W - K + 1 + 2.P$

Very common :

Set P = (K – 1) / 2

To make output have size as input!



Convolution neural network

Dauphine | PSL UNIVERSITÉ PARIS

Example

Input : 3 x 32 x 32

Filter : 3 x 5 x 5

Output with P = 0:

3 × 28 × 28

Output with P = 2:

3 × 32 x 32

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

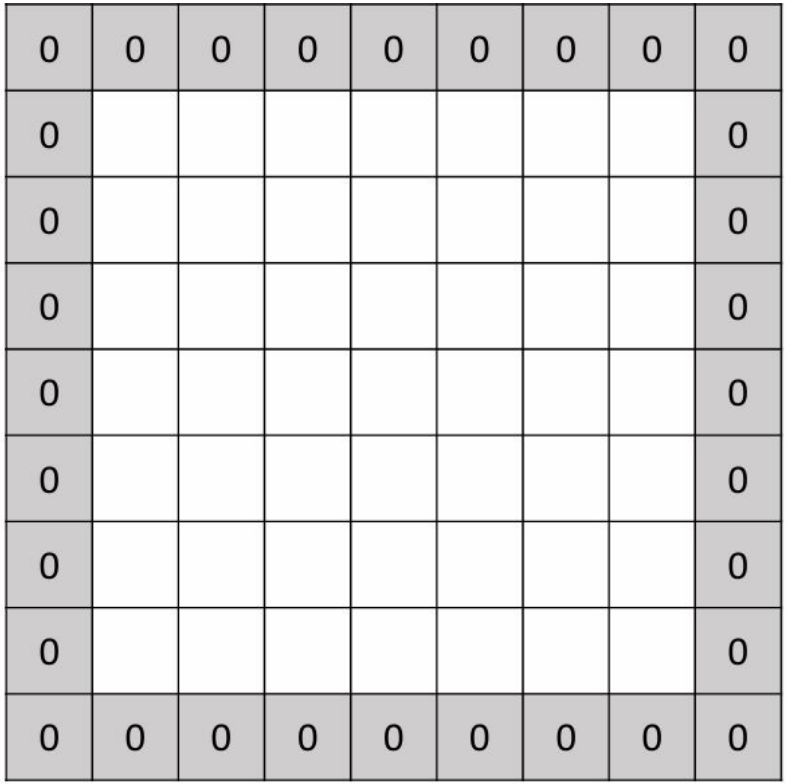
Convolution neural network

But doesn't adding zeros introduce noise and corrupt the image?

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

Convolution neural network

But doesn't adding zeros introduce noise and corrupt the image?



No!

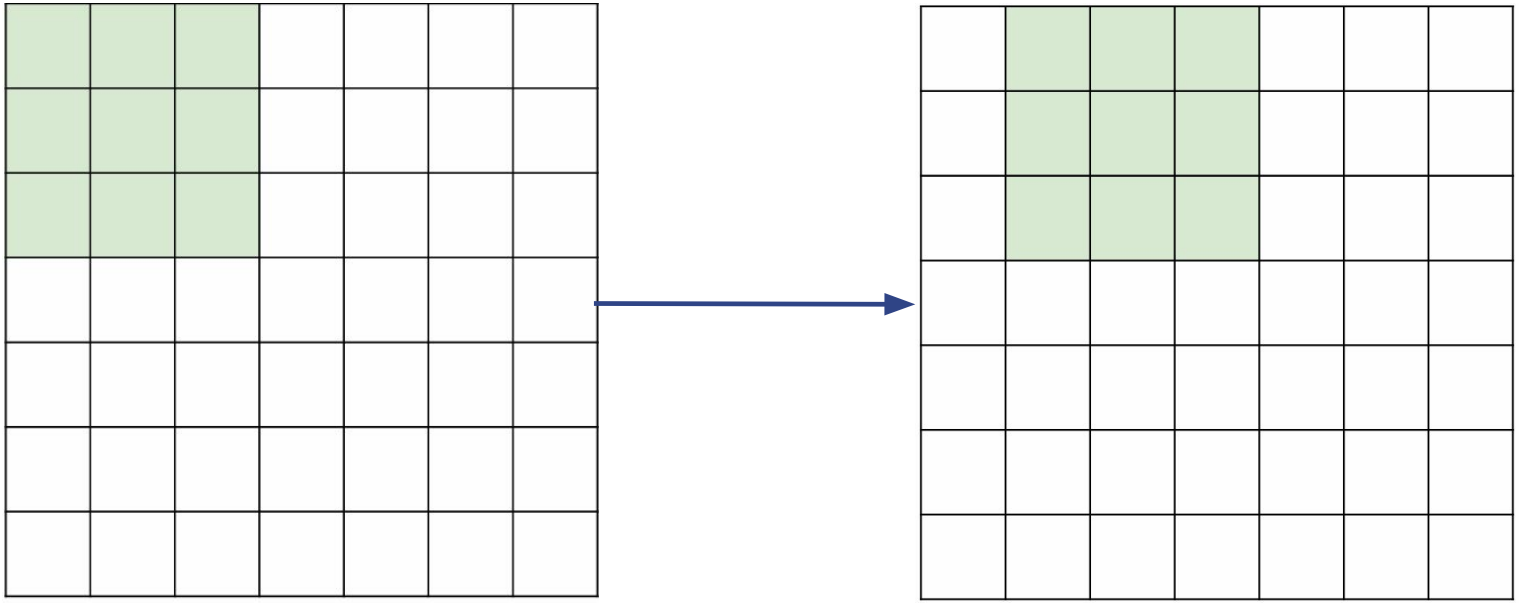
1 - All real pixels remain unchanged.

2 - It prevents losing border information

Convolution neural network

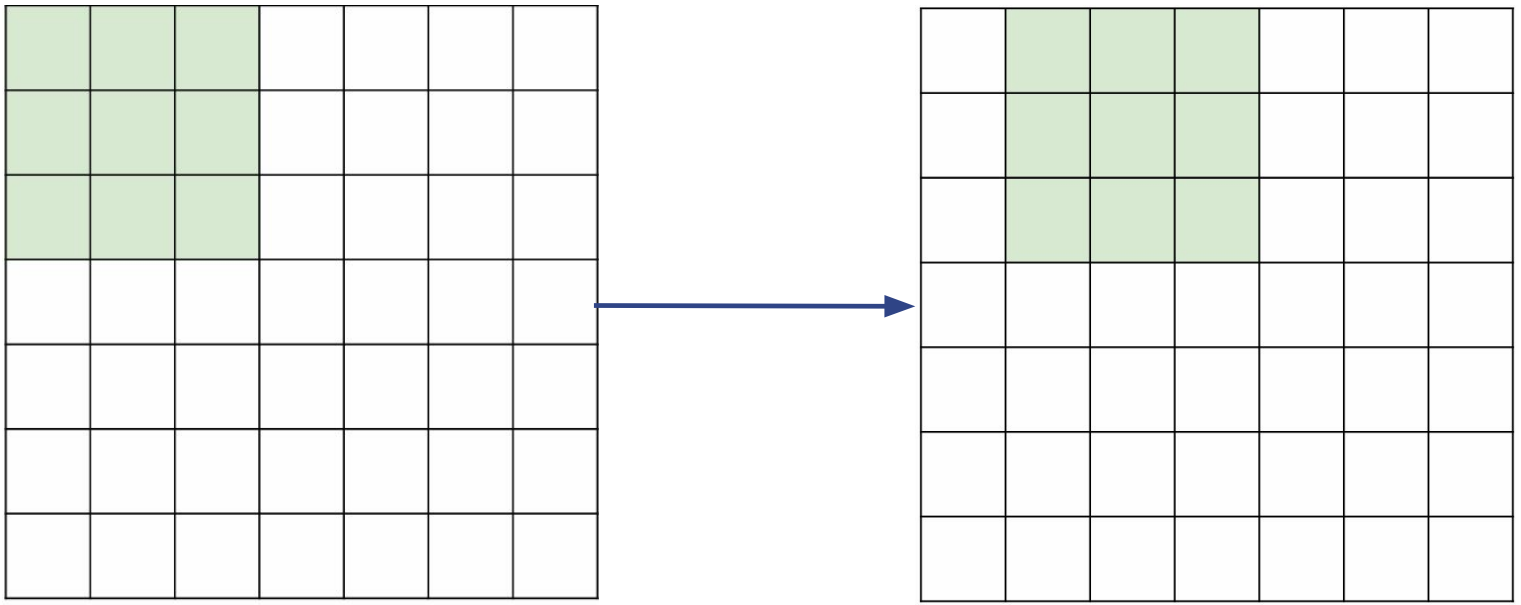
So far, our filter moves one pixel at a time.

But do we really need to compute every position?



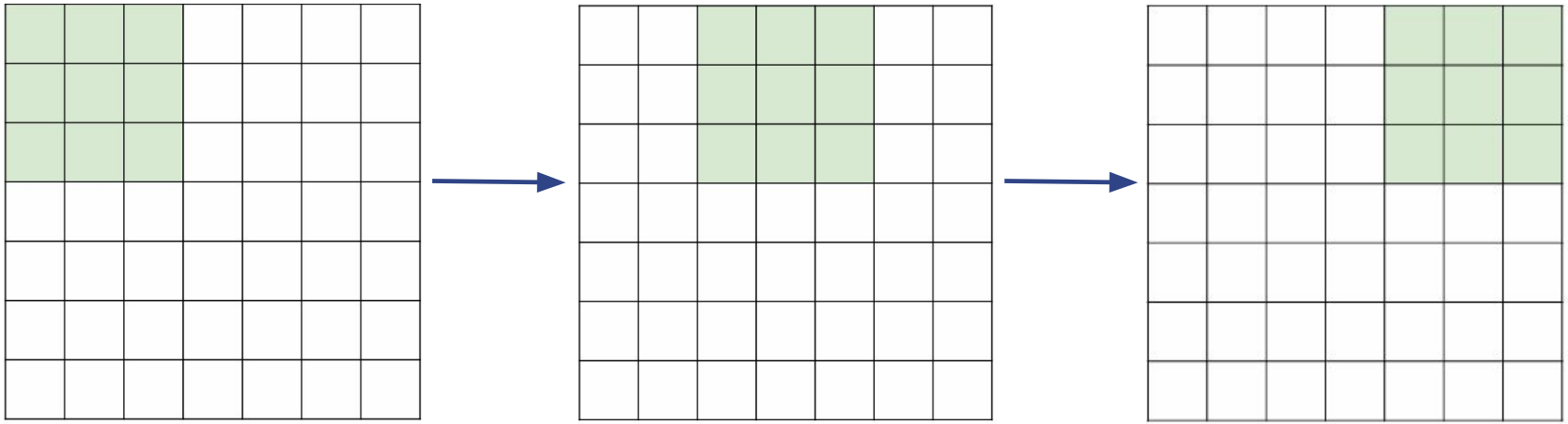
Convolution neural network

Stride = 1 → move 1 pixel (default)



Convolution neural network

Stride = 2 → move 2 pixels (skip one position)



General Formula

$$W_{out} = \frac{W - K + 2P}{S} + 1$$

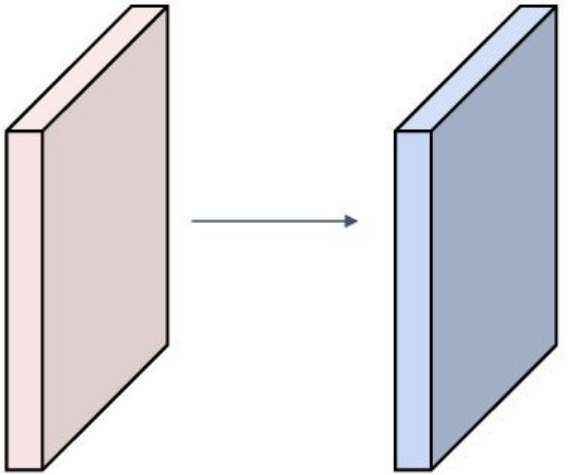
Convolution Example

$$W_{out} = \frac{W - K + 2P}{S} + 1$$

What If we have input 3 x 32 x 32

10 3x5x5 filters with stride 1 and pad 2

Output volume size: ?



Convolution Example

$$W_{out} = \frac{W - K + 2P}{S} + 1$$

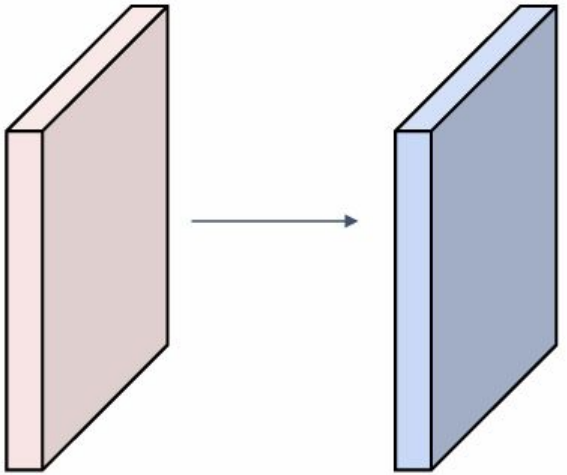
What If we have input 3 x 32 x 32

10 3x5x5 filters with stride 1 and pad 2

Output volume size:

(32+2*2-5)/1 + 1 = 32 spatially,

so 10 x 32 x 32



Convolution Example

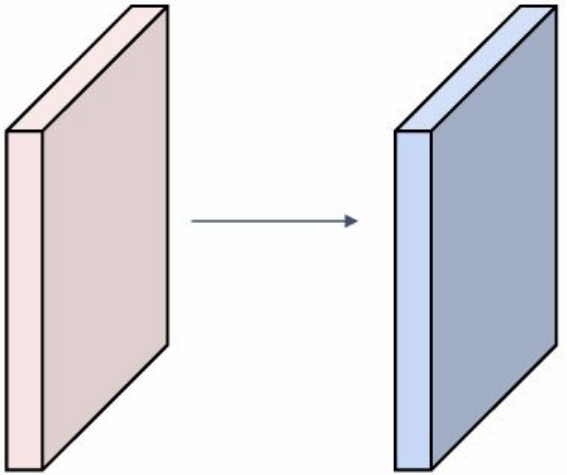
$$W_{out} = \frac{W - K + 2P}{S} + 1$$

What If we have input 3 x 32 x 32

10 3x5x5 filters with stride 1 and pad 2

Output volume size: 10 x 32 x 32

Number of learnable parameters: ?



Convolution Example

$$W_{out} = \frac{W - K + 2P}{S} + 1$$

What If we have input 3 x 32 x 32

10 3x5x5 filters with stride 1 and pad 2

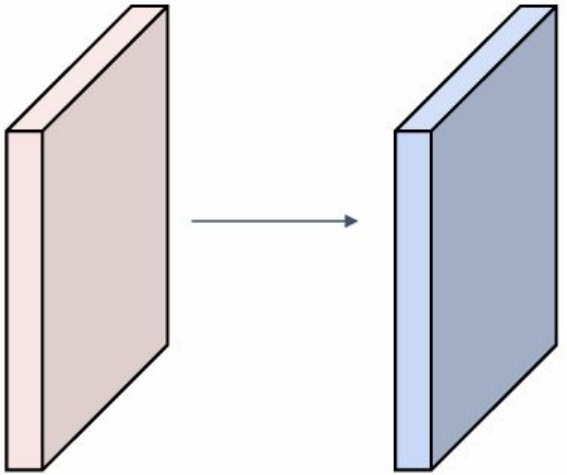
Output volume size: 10 x 32 x 32

Number of learnable parameters:

Parameters per filter : 3 * 5 * 5 +1 (bias)

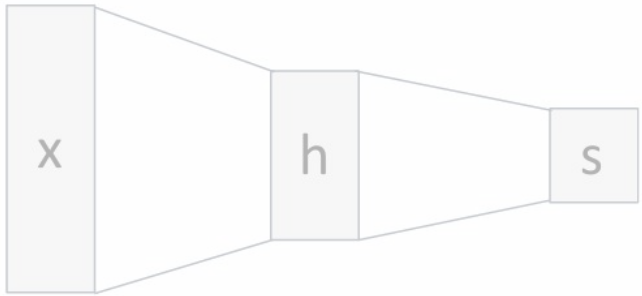
= 76

10 Filters so total is 760

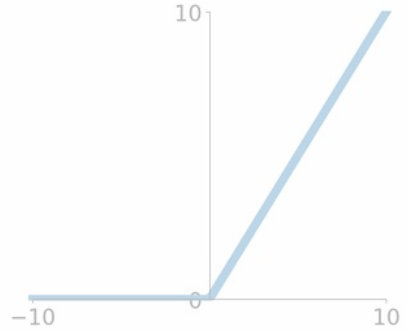


Components of a Convolutional Network

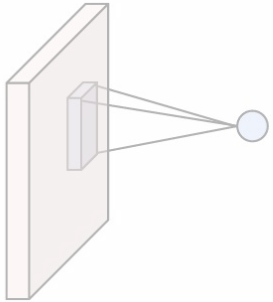
Fully-Connected Layers



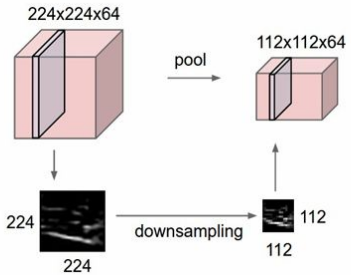
Activation Function



Convolution Layers



Pooling Layers



Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Pooling Layers

After several convolution layers, what happens to spatial size?

Pooling Layers

After several convolution layers, what happens to spatial size?

Even with padding, feature maps remain large:

Example:
10 × 32 × 32
→ next layer still 32 × 32

Pooling Layers

After several convolution layers, what happens to spatial size?

Even with padding, feature maps remain large:

Example:
10 × 32 × 32
→ next layer still 32 × 32

This is computationally expensive

Solution : Pooling layers

Pooling is a down-sampling operation that reduces the spatial dimensions (width and height) of the input feature map while retaining important features.

The pooling operator is typically applied after a convolutional layer

Pooling operation does not have learnable parameters

Different parameters define the pooling operation :

- Filter size
- Stride parameter
- Pooling operation : Max Pooling, Average Pooling, Min Pooling, etc

Example of Max and Average Pooling

3	0	7	4
1	5	3	1
4	2	2	8
2	4	3	9

$\Longrightarrow$

5	7
4	9

Max Pooling

3	0	7	4
1	5	3	1
4	2	2	8
2	4	3	9

$\Longrightarrow$

2.25	3.75
3	5.5

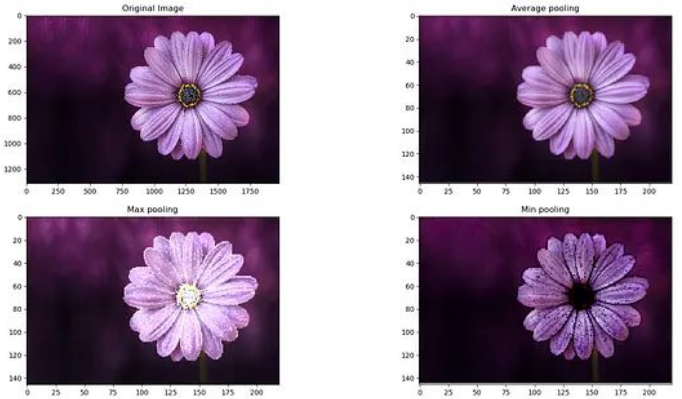
Average Pooling

Pooling Layers

The main advantage of pooling is :

- Reduces the number of parameters in the network
- which helps prevent overfitting and improves the computational efficiency of the mode

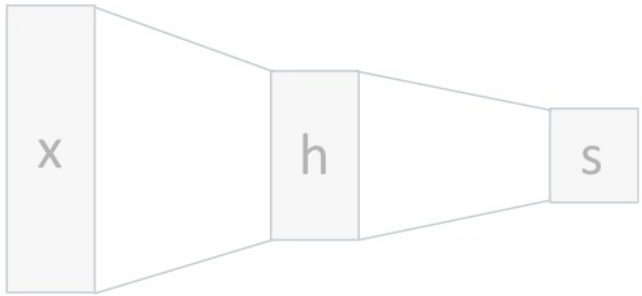
Pooling also helps reducing noise Especially Max Pooling that only extract higher activation



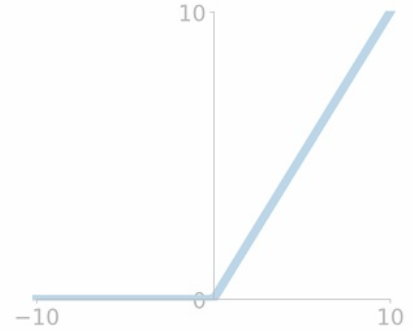
[source : medium.com/@bdhuma/]

Components of a Convolutional Network

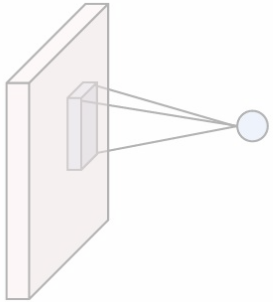
Fully-Connected Layers



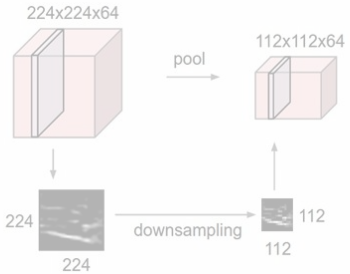
Activation Function



Convolution Layers



Pooling Layers

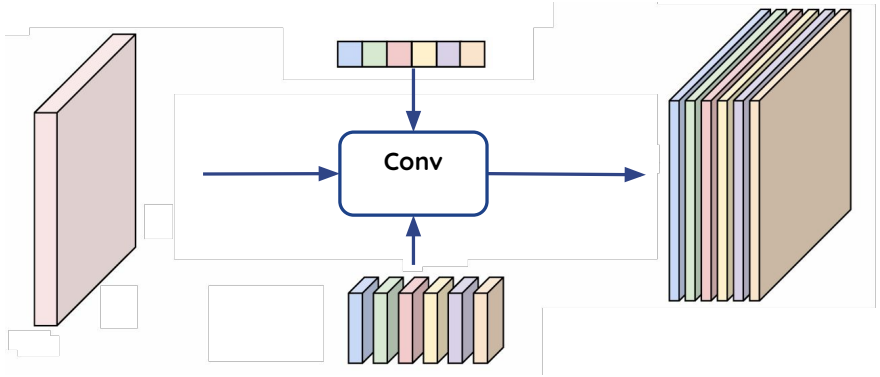


Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalization

After convolution and ReLU, what are the values inside the feature maps?



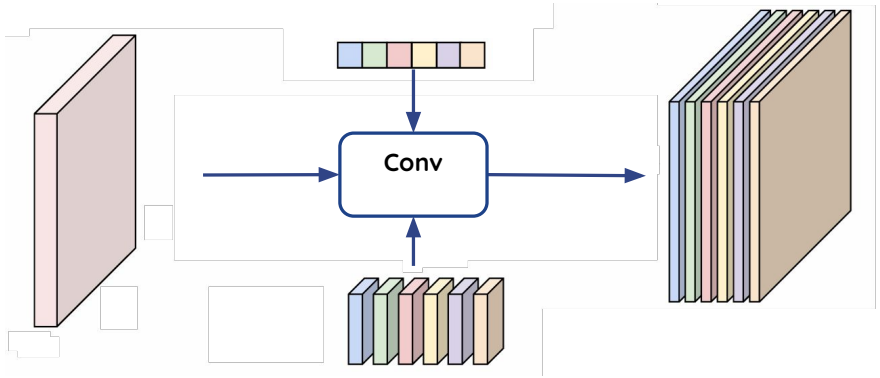
Normalization

After convolution and ReLU, what are the values inside the feature maps?

They can be:

- **Very large**
- **Very small**
- **Different distributions across channels**
- **Different distributions across batches**

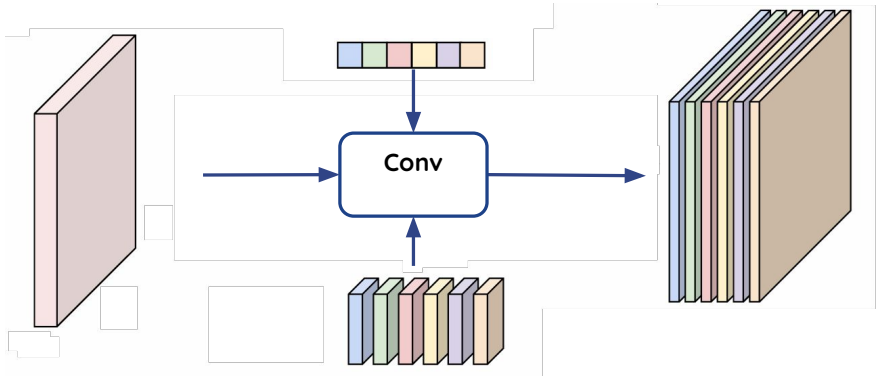
Training can become unstable.



Normalization

Idea: “Normalize” the outputs of a layer so they have zero mean and unit variance ?

Improves optimization.



Normalization

In CNNs we usually use Batch Normalization

For each channel:

$$\hat{x}^{(k)} = \frac{x^{(k)} - E\left[x^{(k)}\right]}{\sqrt{\mathrm{Var}\left[x^{(k)}\right] + \varepsilon}}$$

Then:

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$

$\mu^{(k)}$: mean of channel k over the mini-batch

$\sigma^{2\,(k)}$: variance of channel k

ε : small constant for numerical stability

$\gamma^{(k)}$: learnable scale parameter

$\beta^{(k)}$: learnable shift parameter

[Improved Texture Networks: Maximizing Quality and Diversity in Feed-forward Stylization and Texture Synthesis, 2017]

Normalization

Usually applied after convolution and before activations.

$$\hat{x}^{(k)} = \frac{x^{(k)} - E\left[x^{(k)}\right]}{\sqrt{\mathrm{Var}\left[x^{(k)}\right] + \varepsilon}}$$

Normalization

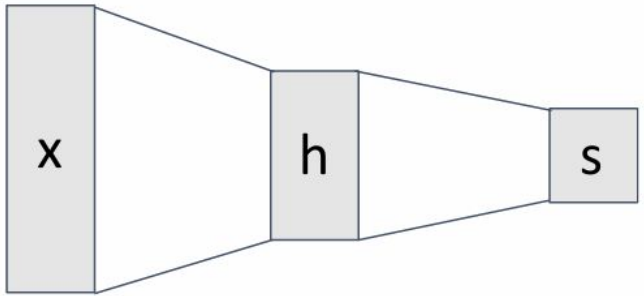
Usually applied after convolution and before activations.

Because we want to normalize the linear activations before applying the non-linearity.

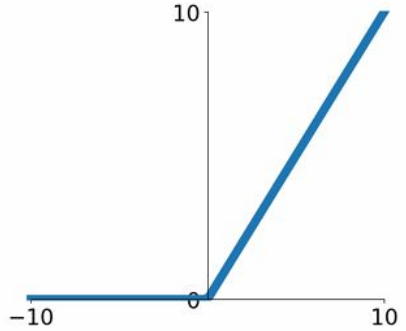
$$\hat{x}^{(k)} = \frac{x^{(k)} - E\left[x^{(k)}\right]}{\sqrt{\mathrm{Var}\left[x^{(k)}\right] + \varepsilon}}$$

Summary

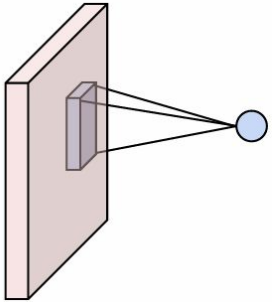
Fully-Connected Layers



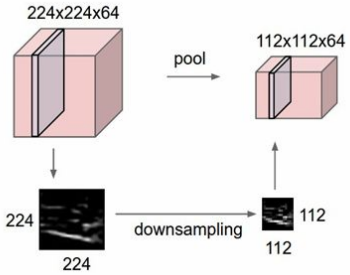
Activation Function



Convolution Layers



Pooling Layers

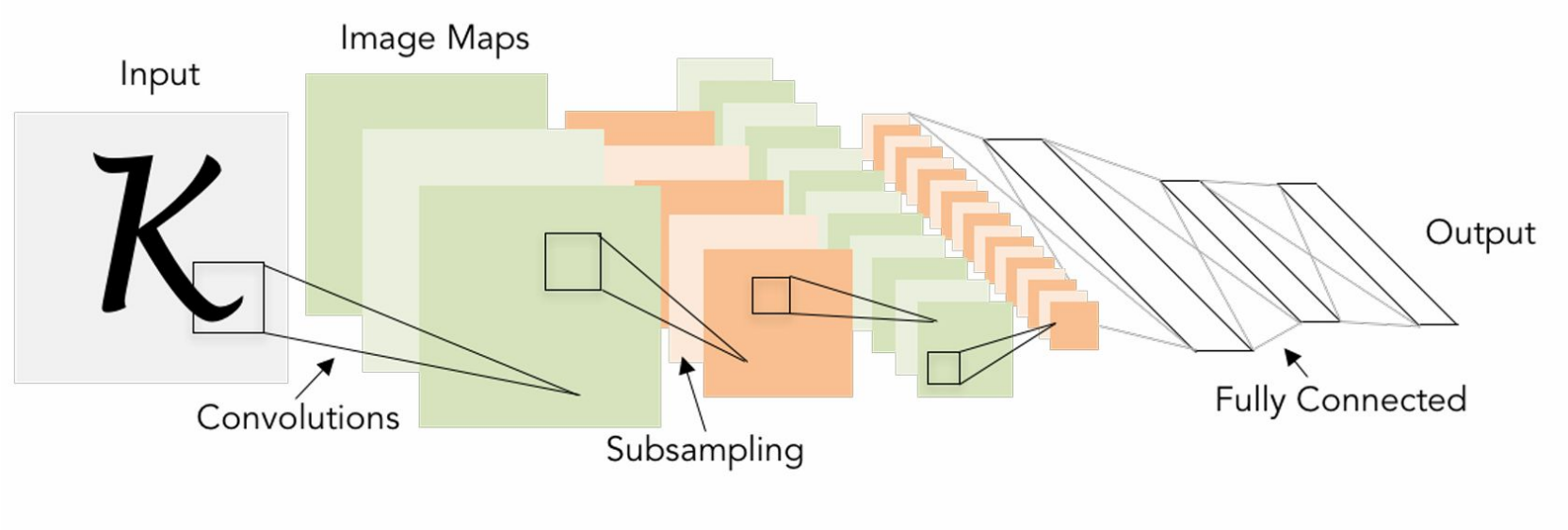


Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Summary

Problem : **What is the right way to combine all these components ?**



CNNs Architectures

References

Books

• Computer Vision: Models, Learning, and Inference (2012). https://amzn.to/2rxrdOF
• Programming Computer Vision with Python (2012). https://amzn.to/2QKTaAL
• Multiple View Geometry in Computer Vision (2004). https://amzn.to/2LfHLE8
• Computer Vision: Algorithms and Applications (2010). https://amzn.to/2LcIt4J
• Computer Vision: A Modern Approach (2002). https://amzn.to/2rv7AqJ
• Deep Learning for Computer Vision : Image Classification, Object Detection and Face Recognition in Python(2019). https://machinelearningmastery.com/deep-learning-for-computer-vision/
• Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow — Aurélien Géron (2019) https://amzn.to/3uZbN6Z

Articles & Online Resources

• Krizhevsky, Sutskever, Hinton (2012) — ImageNet Classification with Deep CNNs (AlexNet — deep learning breakthrough)
• Improved Texture Networks: Maximizing Quality and Diversity in Feed-forward Stylization and Texture Synthesis, 2017
• "Computer vision," Wikipedia. https://en.wikipedia.org/wiki/Computer_vision
• "Machine vision," Wikipedia. https://en.wikipedia.org/wiki/Machine_vision
• "Digital image processing," Wikipedia. https://en.wikipedia.org/wiki/Digital_image_processing
• "Training Convolutional Neural Networks: What Is Machine Learning" Analog Devices. https://www.analog.com/en/resources/analog-dialogue/articles/training-convolutional-neural-networks-what-is-machine-learning-part-2.html