# Deep Learning

# Session-5

# Table of Contents

# What is CNN?

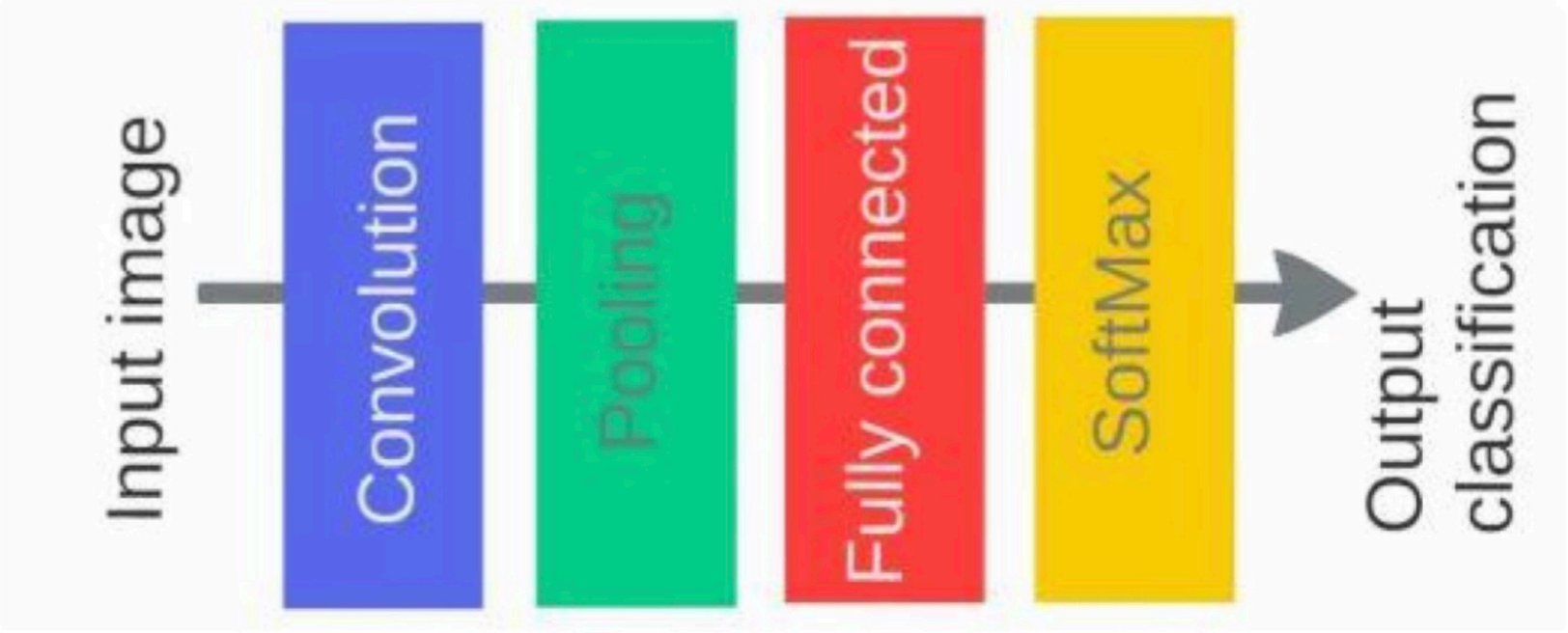
- A convolutional neural network (CNN, or ConvNet) is a class of artificial neural network (ANN), most commonly applied to analyze **visual imagery.**
- A CNN can take in an **input image**, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other.
- The architecture of a CNN is very loosely analogous to that of the **connectivity pattern of Neurons in the Human Brain** and was inspired by the **organization of the Visual Cortex**.

# What is CNN?

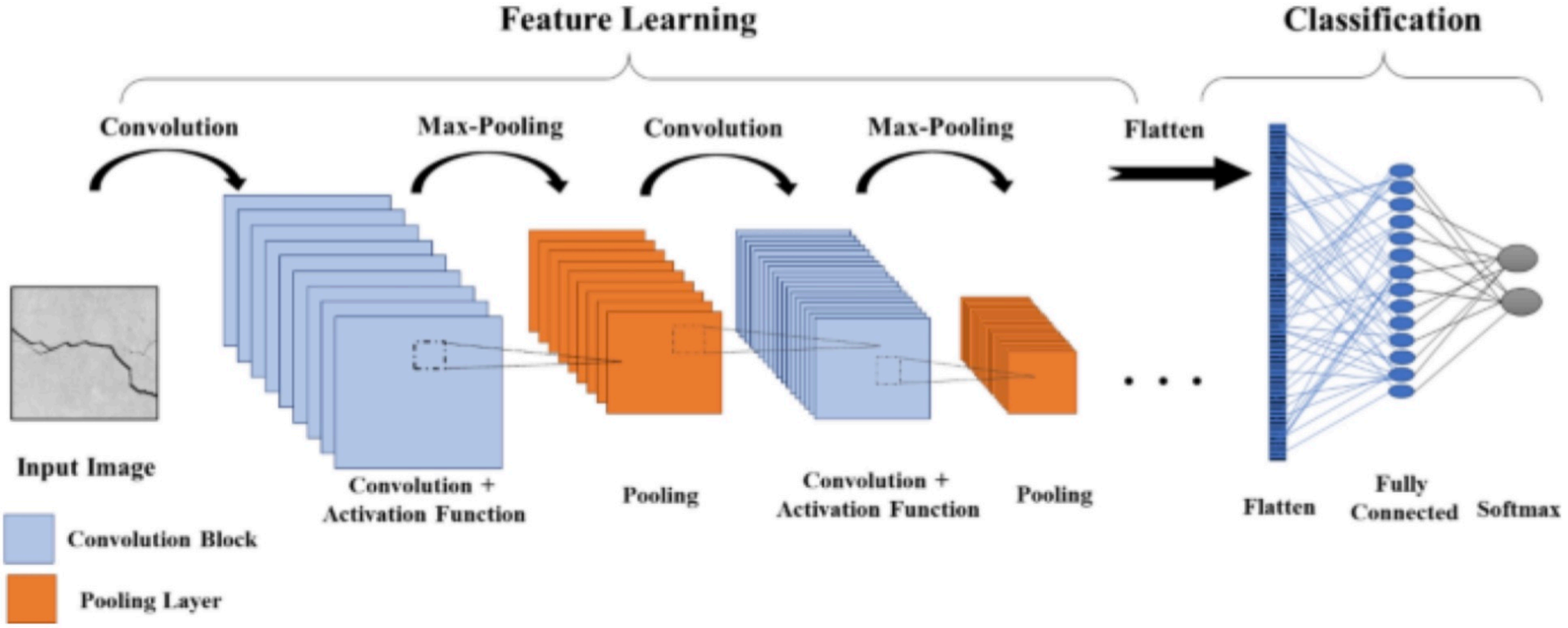
## Application Examples of CNN

- Face Detection
- Facial emotion recognition
- Object detection
- Self-driving or autonomous cars
- Auto translation
- Next word prediction in sentence
- Handwritten character recognition
- X-ray image analysis
- Cancer detection
- Visual question answering
- Image captioning
- Biometric authentication
- Document classification
- 3D medical image segmentation

# CNN Theory and Architecture



# CNN Theory and Architecture



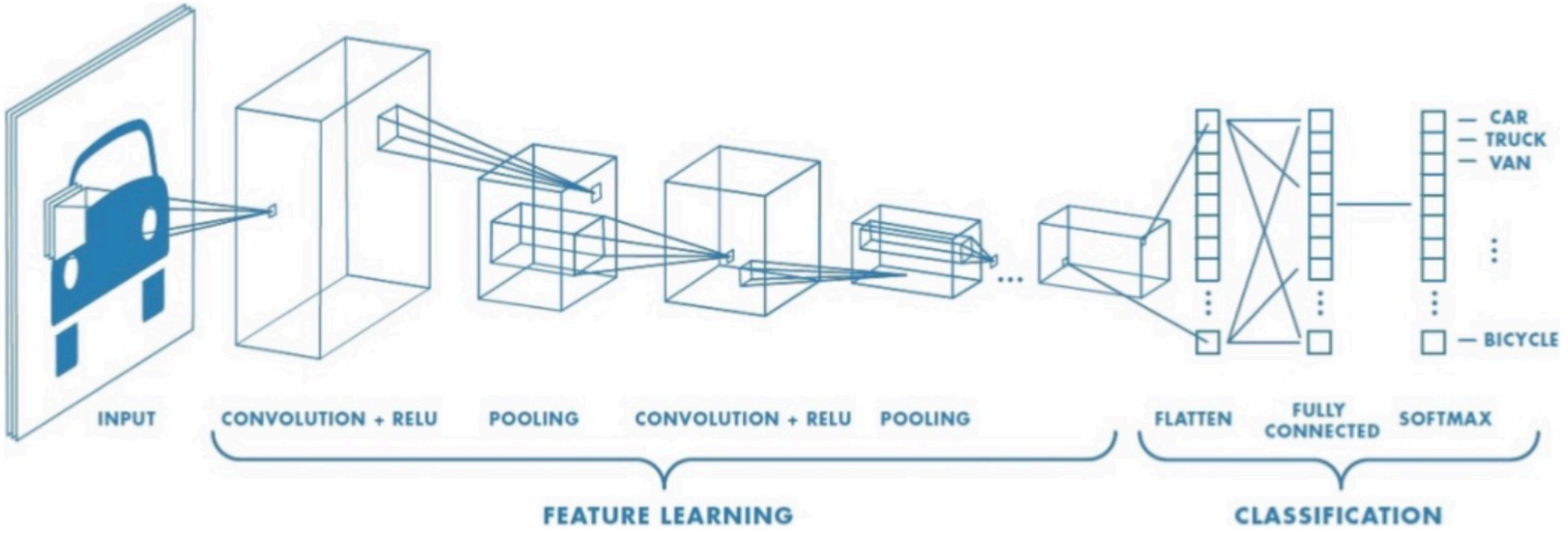
# CNN Theory and Architecture

## 1. The Hidden Layers/Feature Extraction Part

In this part, the network will perform a series of operations during which the features are detected. If you had a picture of a zebra, this is the part where the network would recognise its stripes, two ears, and four legs.

## 2. The Classification Part

This part assigns a probability of the object being in that object.

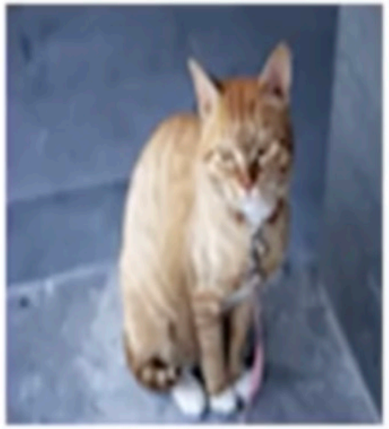



**CNN have two components**

# CNN Theory and Architecture

## Intuition Behind CNN

- Why do we need convolutional layers, anyway?
- Why can't we just use MLP (Multilayer Perceptron) like we have so far?



64x64


Imagine you're training a cat-classifier, where the input is a 64x64 image.

# CNN Theory and Architecture

## Intuition Behind CNN

- **64x64** is a very small image nowadays. Nonetheless, since we have to flatten the image before feeding it into a neural network, and considering the image has 3 colour channels, we end up with **12288** features for every data point (image) in our dataset.

**64 x 64 x 3 = 12.288**

- If, however, your image is relatively high res. (**1024x1024**), you end up with **3.145.728** features for every data point. Wow. That would be incredibly expensive to train!

**1024 x 1024 x 3 = 3.145.728**

# CNN Theory and Architecture

## Intuition Behind CNN

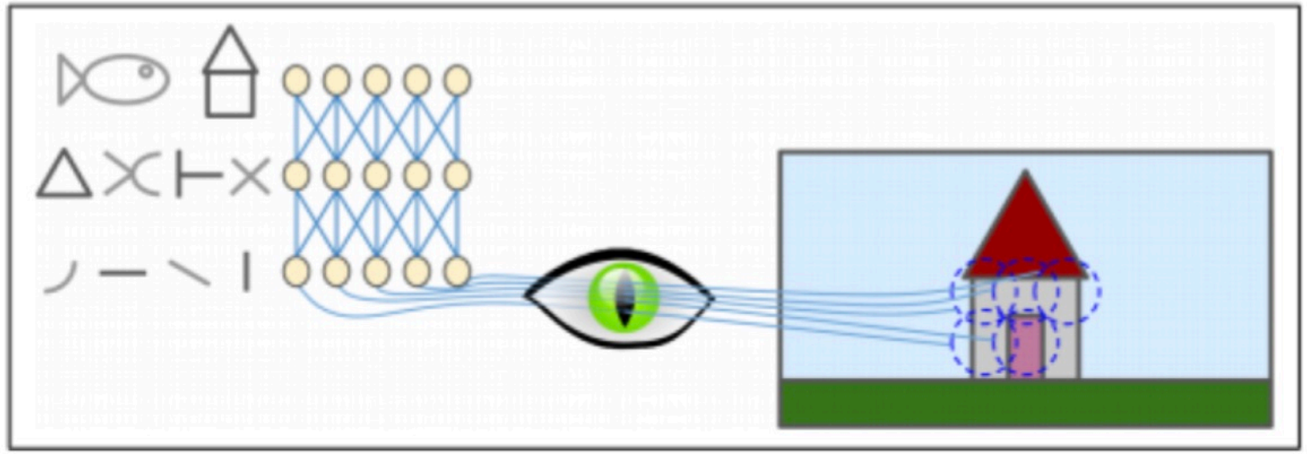
- Not only it would be extremely expensive to train, it would be very **difficult to find enough images** to prevent your neural network from **overfitting**.

- Also, when we flatten the image, we **lose a lot of the information** that we would otherwise gain if we looked at the image in 2D, instead of 1D.

**This is why we use convolutional layers.**

# CNN Theory and Architecture

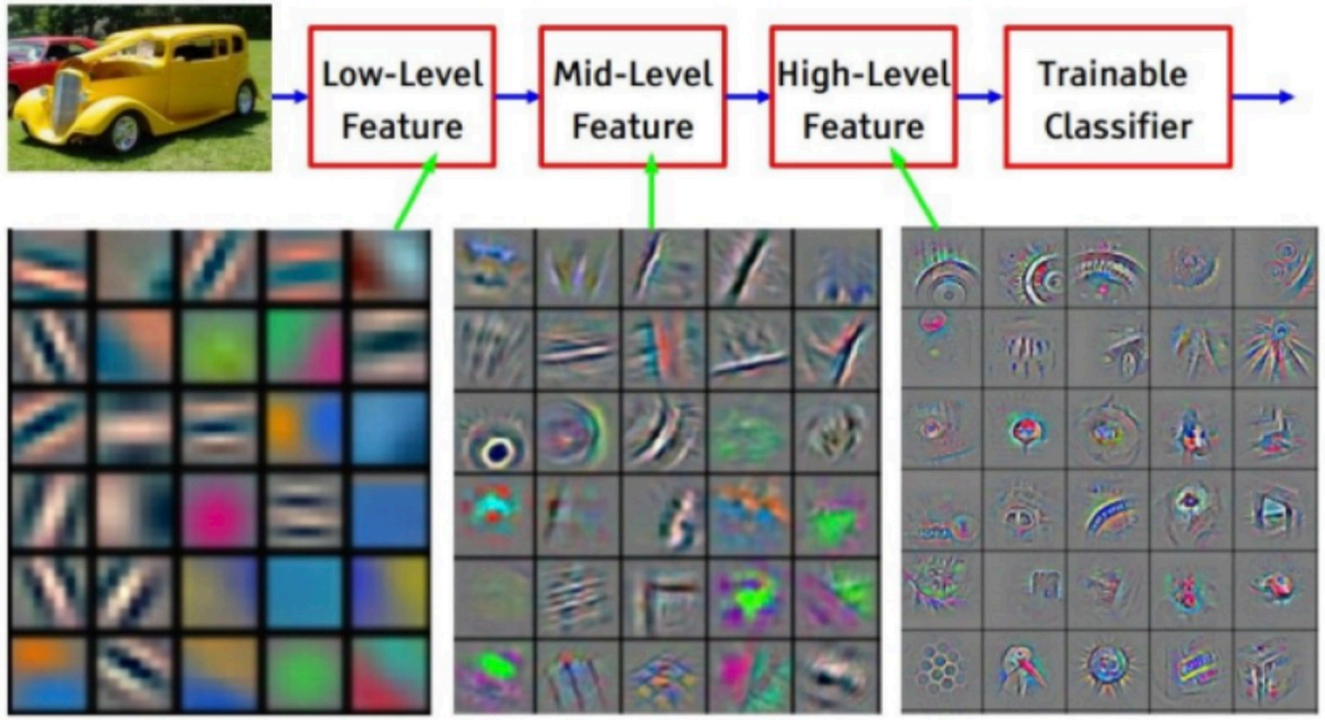
## Organization of the Visual Cortex



- Many neurons in the brain's visual cortex have a **tiny local receptive fields**.
- They only respond to visual stimuli in a **specific portion of the visual field**.
- Different neurons' receptive fields may **overlap**, so they tile the entire visual field.
- Some neurons only respond to images of **horizontal lines**, while others only respond to **lines of varying orientation**s.
- Some neurons have **bigger receptive fields** and respond to **more complicated patterns** that are a mixture of lower-level patterns.

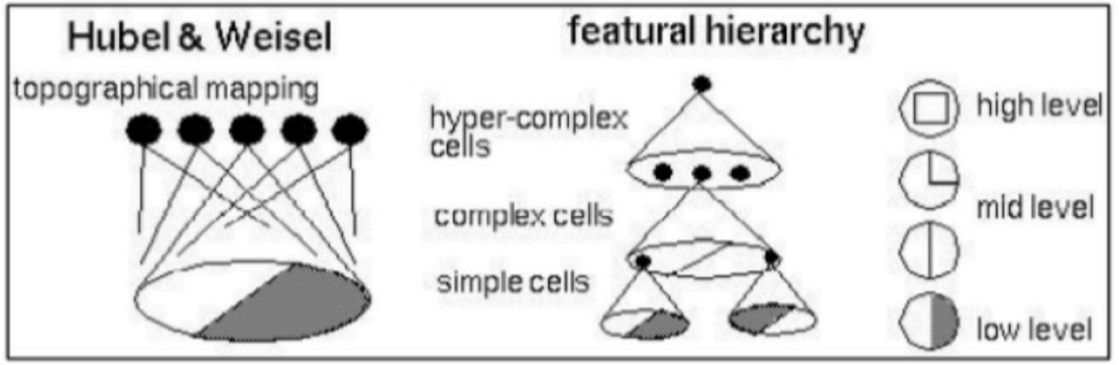
# CNN Theory and Architecture

**Preview**

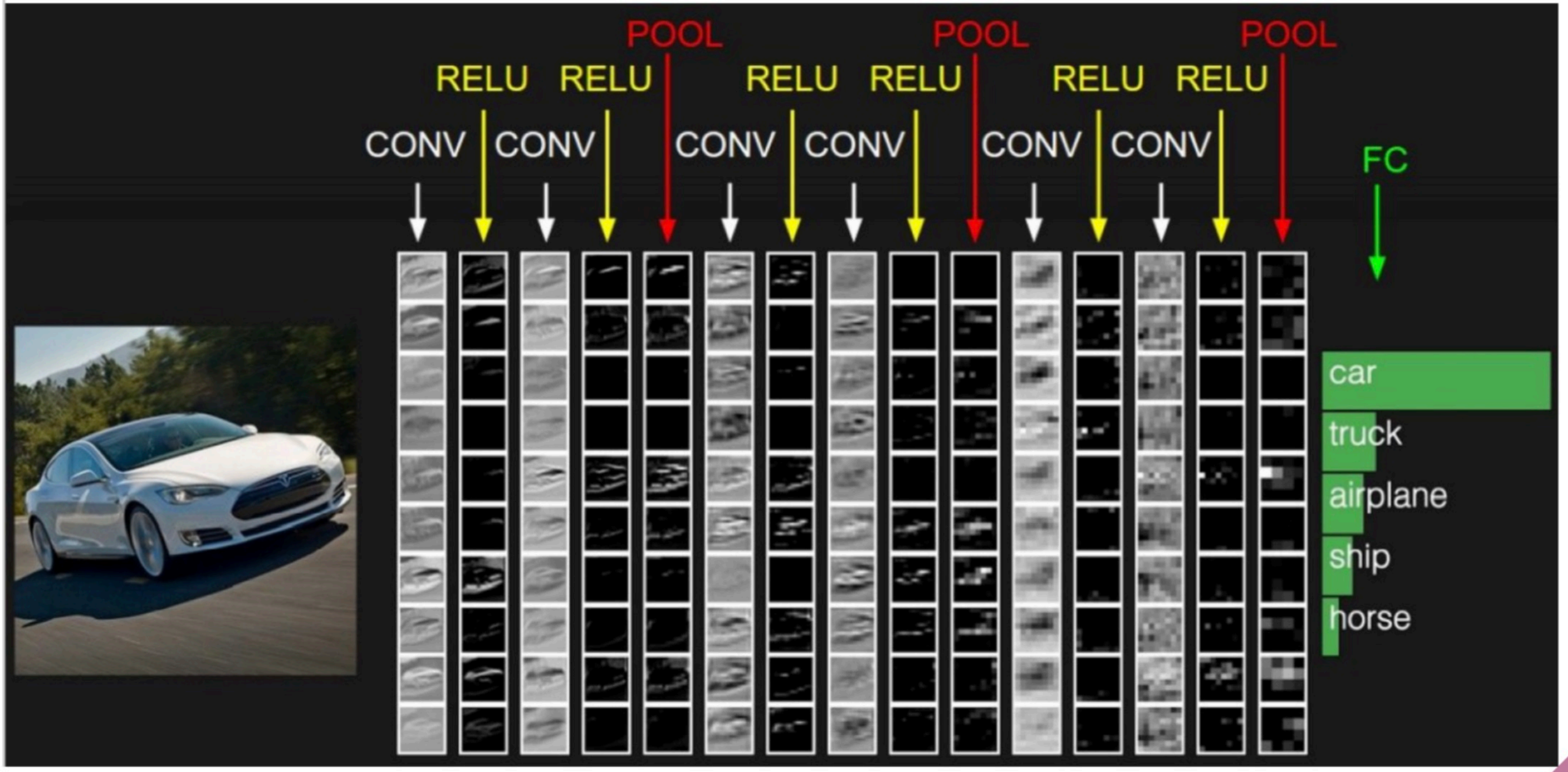



*[From recent Yann LeCun slides]*

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]



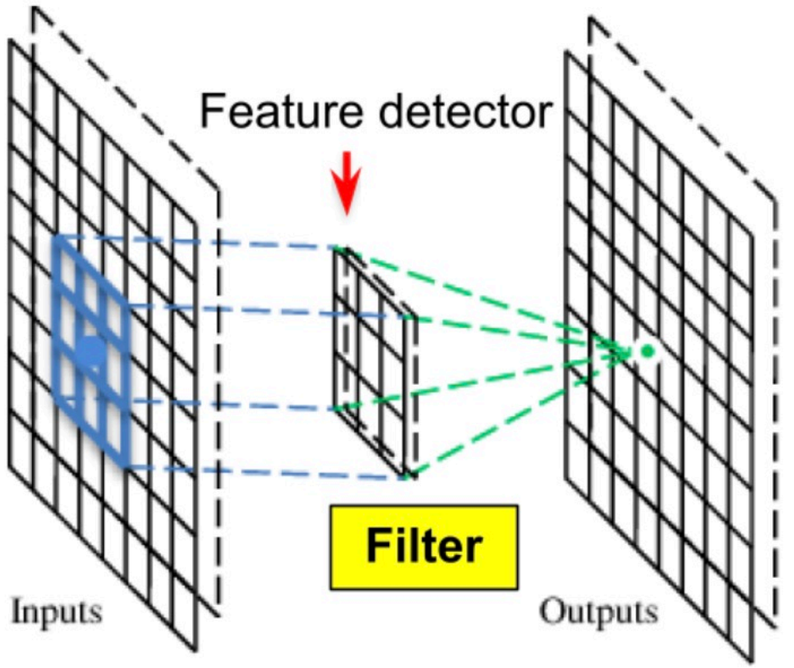
# CNN Theory and Architecture



# Convolution Layers

## What is Convolution?

- The word convolution refers to a mathematical operation that makes filtering possible.
- A convolutional layer has a number of filters that does convolutional operation.



# Convolution Layers

## Types of Convolutional Layers

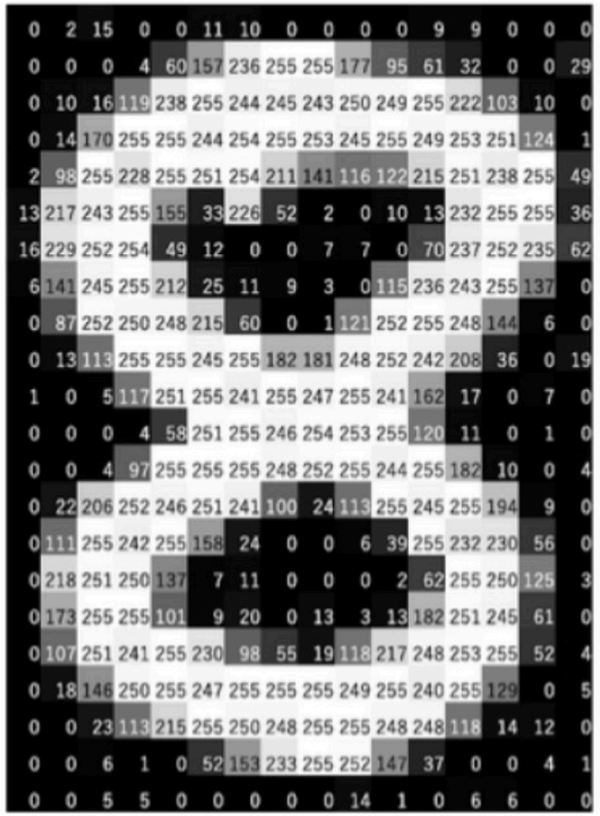
- **1D Conv. Layer:** Kernel (filter) moves in only one direction with them. These kinds of CNNs are typically applied to time-series data.

- **2D Conv. Layer:** Kernels moves in two directions. These are commonly used in picture labeling and processing.

- **3D Conv. Layer:** Kernel travels in three directions. This form of CNN commonly used on 3D images such as CT scans and MRIs.

# Convolution Layers

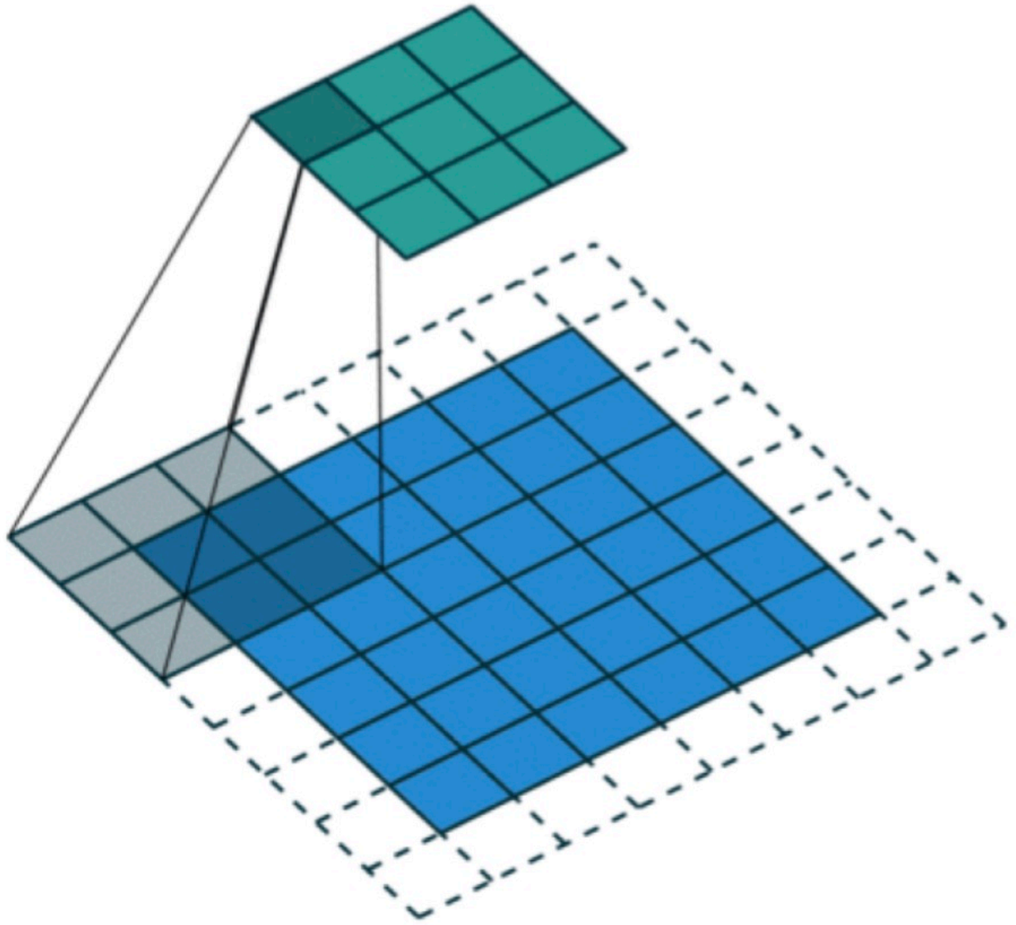
## Numerical view of an image



| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2 | 15 | 0 | 0 | 11 | 10 | 0 | 0 | 0 | 0 | 9 | 9 | 0 | 0 | 0 |
| 0 | 0 | 0 | 4 | 60 | 157 | 236 | 255 | 255 | 177 | 95 | 61 | 32 | 0 | 0 | 29 |
| 0 | 10 | 16 | 119 | 238 | 255 | 244 | 245 | 243 | 250 | 249 | 255 | 222 | 103 | 10 | 0 |
| 0 | 14 | 170 | 255 | 255 | 244 | 254 | 255 | 253 | 245 | 255 | 249 | 253 | 251 | 124 | 1 |
| 2 | 98 | 255 | 228 | 255 | 251 | 254 | 211 | 141 | 116 | 122 | 215 | 251 | 238 | 255 | 49 |
| 13 | 217 | 243 | 255 | 155 | 33 | 226 | 52 | 2 | 0 | 10 | 13 | 232 | 255 | 255 | 36 |
| 16 | 229 | 252 | 254 | 49 | 12 | 0 | 0 | 7 | 7 | 0 | 70 | 237 | 252 | 235 | 62 |
| 6 | 141 | 245 | 255 | 212 | 25 | 11 | 9 | 3 | 0 | 115 | 236 | 243 | 255 | 137 | 0 |
| 0 | 87 | 252 | 250 | 248 | 215 | 60 | 0 | 1 | 121 | 252 | 255 | 248 | 144 | 6 | 0 |
| 0 | 13 | 113 | 255 | 255 | 245 | 255 | 182 | 181 | 248 | 252 | 242 | 208 | 36 | 0 | 19 |
| 1 | 0 | 5 | 117 | 251 | 255 | 241 | 255 | 247 | 255 | 241 | 162 | 17 | 0 | 7 | 0 |
| 0 | 0 | 0 | 4 | 58 | 251 | 255 | 246 | 254 | 253 | 255 | 120 | 11 | 0 | 1 | 0 |
| 0 | 0 | 4 | 97 | 255 | 255 | 255 | 248 | 252 | 255 | 244 | 255 | 182 | 10 | 0 | 4 |
| 0 | 22 | 206 | 252 | 246 | 251 | 241 | 100 | 24 | 113 | 255 | 245 | 255 | 194 | 9 | 0 |
| 0 | 111 | 255 | 242 | 255 | 158 | 24 | 0 | 0 | 6 | 39 | 255 | 232 | 230 | 56 | 0 |
| 0 | 218 | 251 | 250 | 137 | 7 | 11 | 0 | 0 | 0 | 2 | 62 | 255 | 250 | 125 | 3 |
| 0 | 173 | 255 | 255 | 101 | 9 | 20 | 0 | 13 | 3 | 13 | 182 | 251 | 245 | 61 | 0 |
| 0 | 107 | 251 | 241 | 255 | 230 | 98 | 55 | 19 | 118 | 217 | 248 | 253 | 255 | 52 | 4 |
| 0 | 18 | 146 | 250 | 255 | 247 | 255 | 255 | 255 | 249 | 255 | 240 | 255 | 129 | 0 | 5 |
| 0 | 0 | 23 | 113 | 215 | 255 | 250 | 248 | 255 | 255 | 248 | 248 | 118 | 14 | 12 | 0 |
| 0 | 0 | 6 | 1 | 0 | 52 | 153 | 233 | 255 | 252 | 147 | 37 | 0 | 0 | 4 | 1 |
| 0 | 0 | 5 | 5 | 0 | 0 | 0 | 0 | 0 | 14 | 1 | 0 | 6 | 6 | 0 | 0 |

# Convolution Layers

**What is Filter?**



# Convolution Layers

**What is Filter?**

| | | | | | |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |

6 x 6 image

**These are the network parameters to be learned.**

| | | |
|---|---|---|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

Filter 1

| | | |
|---|---|---|
| -1 | 1 | -1 |
| -1 | 1 | -1 |
| -1 | 1 | -1 |

Filter 2

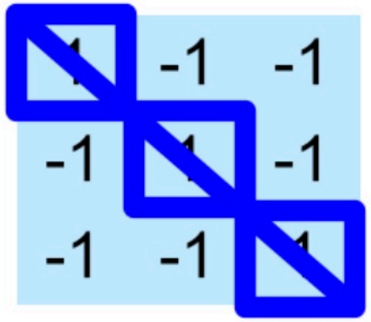
⋮ ⋮

Each filter detects a small pattern (3 x 3).

Slide from Uni. Waterloo

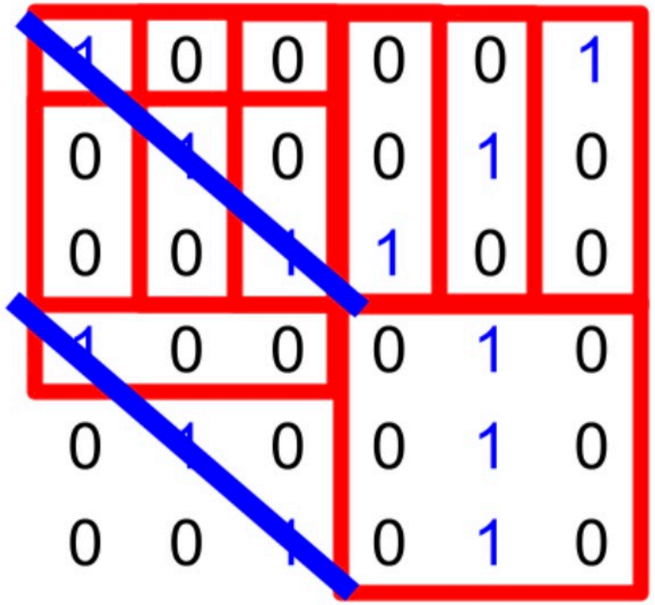
# Convolution Layers

**How to work Filters?**

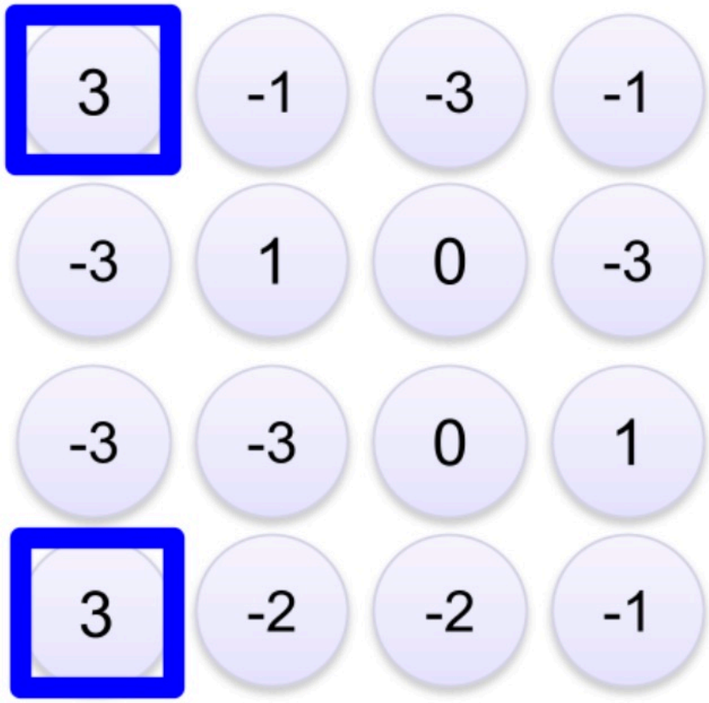



Filter 1

stride=1




6 x 6 image




Slide from Uni. Waterloo

# Convolution Layers

**How to work Filters?**

| | | |
|---|---|---|
| -1 | 1 | -1 |
| -1 | 1 | -1 |
| -1 | 1 | -1 |

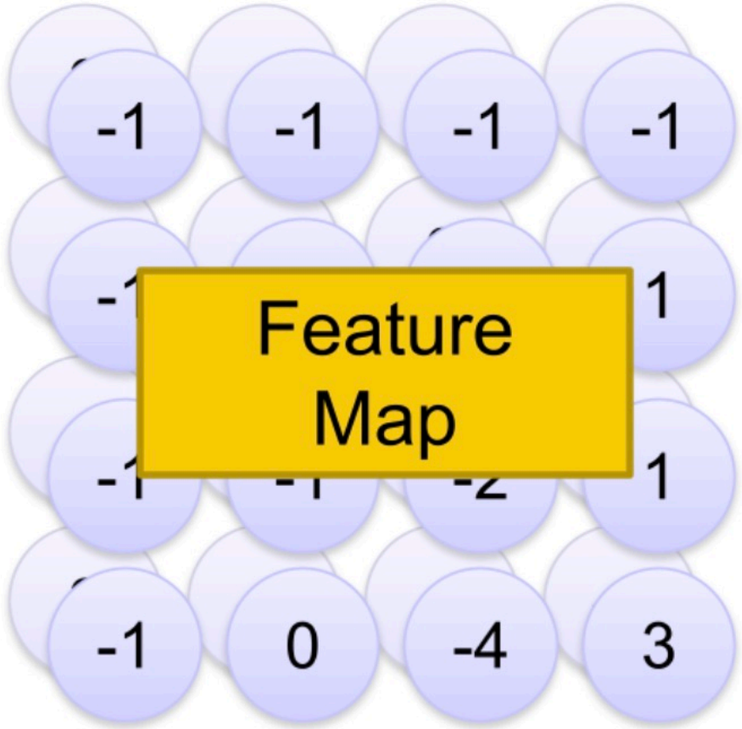
Filter 2

stride=1

| | | | | | |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |

6 x 6 image

Repeat this for each filter

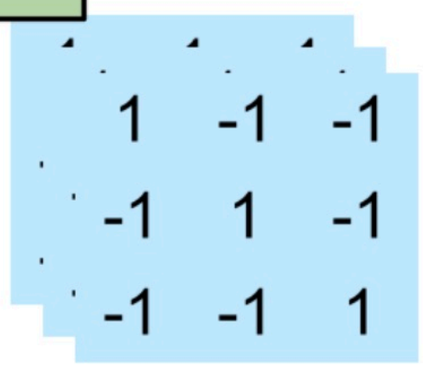



Two 4 x 4 images
Forming 2 x 4 x 4 matrix

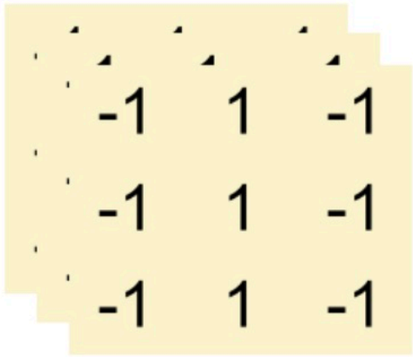
Slide from Uni. Waterloo

# Convolution Layers

**How to work Filters?**

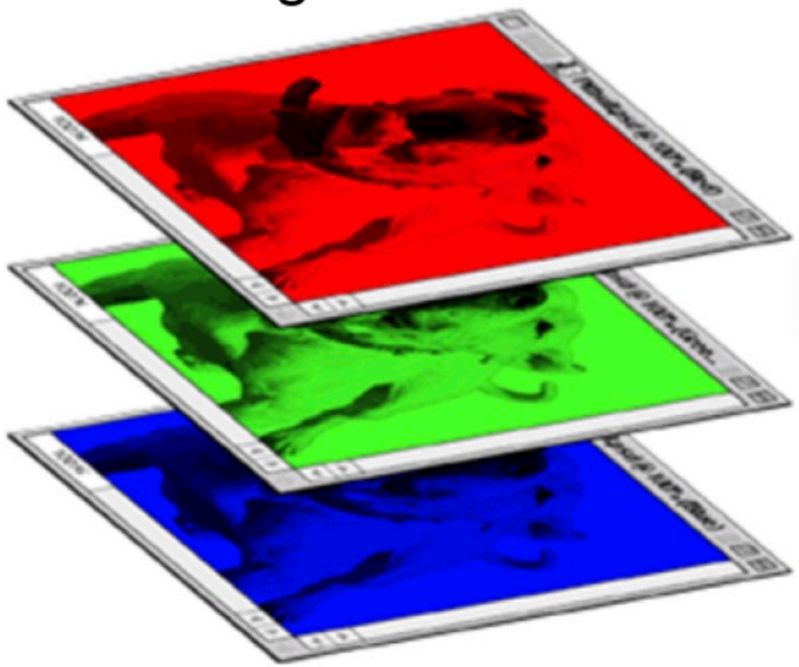


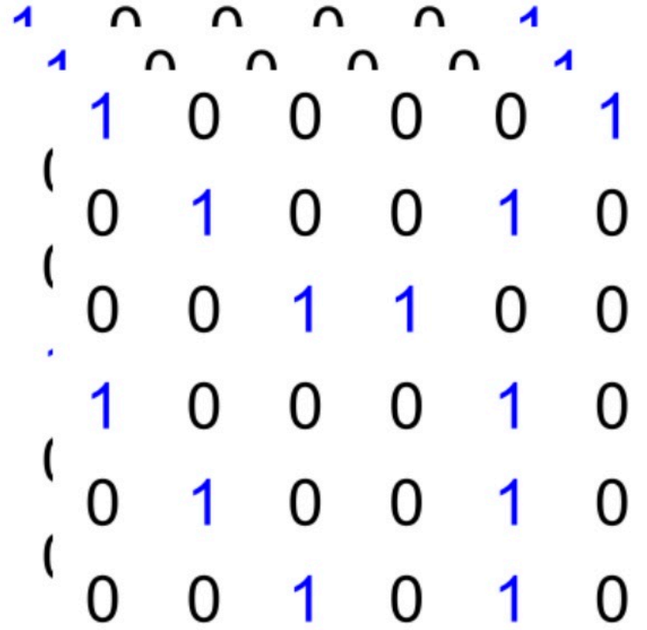

Filter 1




Filter 2

Color image






Slide from Uni. Waterloo

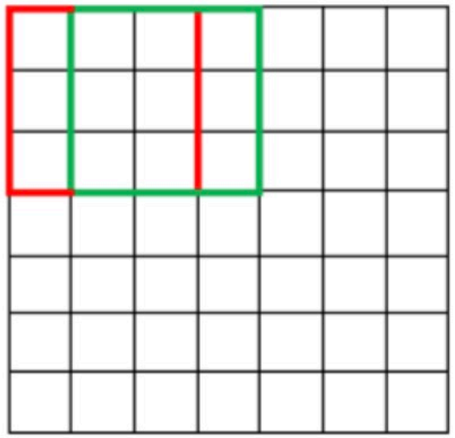
# Convolution Layers

## Stride

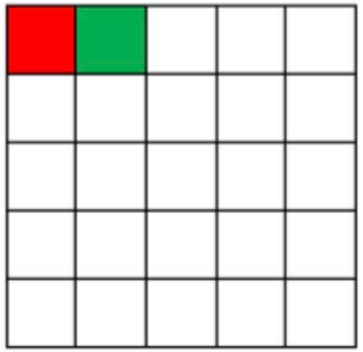
- **Stride** determines how the filter convolves around the input volume.

**with stride 1**

7 x 7 Input Volume

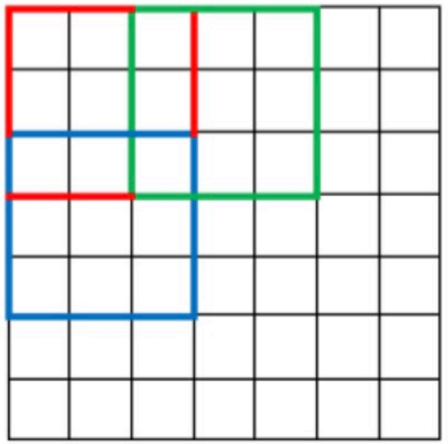


5 x 5 Output Volume

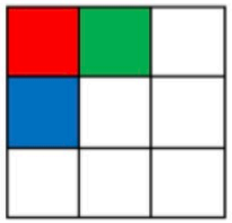


**with stride 2**

7 x 7 Input Volume



3 x 3 Output Volume

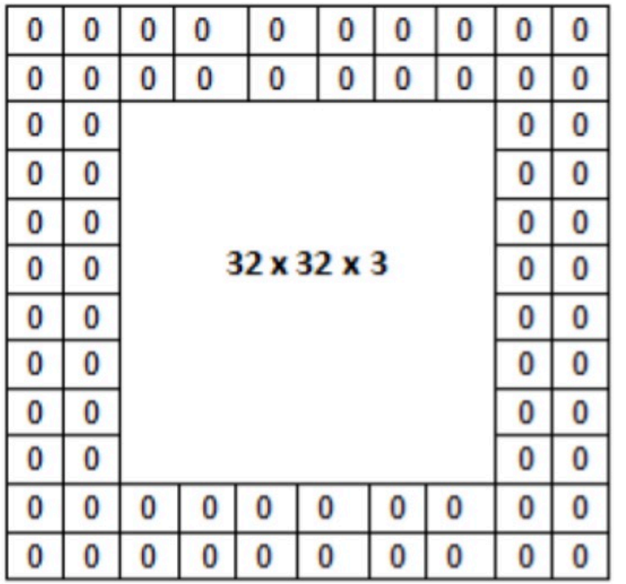


# Convolution Layers

## Padding

- **Padding**, which is a feature that adds blank or empty pixels to the image frame works in conjunction with stride, to allow for a minimal reduction in size in the output layer.

- It is a method of enlarging the size of a picture to compensate for the fact that stride lowers the size.



# Convolution Layers

## Types of Padding

### Same Padding

- The padding layers **append zero values** in the outer frame of the images or data so the filter we are using can **cover the edge of the matrix** and make the inference with them too.
- **The size of output is remaining the same**.

### Valid Padding

- This type of padding can be considered as no padding.
- In VALID (i.e. no padding mode), Tensorflow will **drop right and/or bottom cells** if your filter and stride don't fully cover the input image.

# Convolution Layers

**How to calculate shape of outcome after filtering?**

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

$n_{in}$: number of input features
$n_{out}$: number of output features
$k$: convolution kernel size
$p$: convolution padding size
$s$: convolution stride size

**What is the outcome shape?**
input 7x9
filter 3x3
stride =1
valid padding

?

# Convolution Layers

**How to calculate shape of outcome after filtering?**

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

$n_{in}$: number of input features
$n_{out}$: number of output features
$k$: convolution kernel size
$p$: convolution padding size
$s$: convolution stride size

**What is the outcome shape?**
input 7x9
filter 3x3
stride =1
valid padding

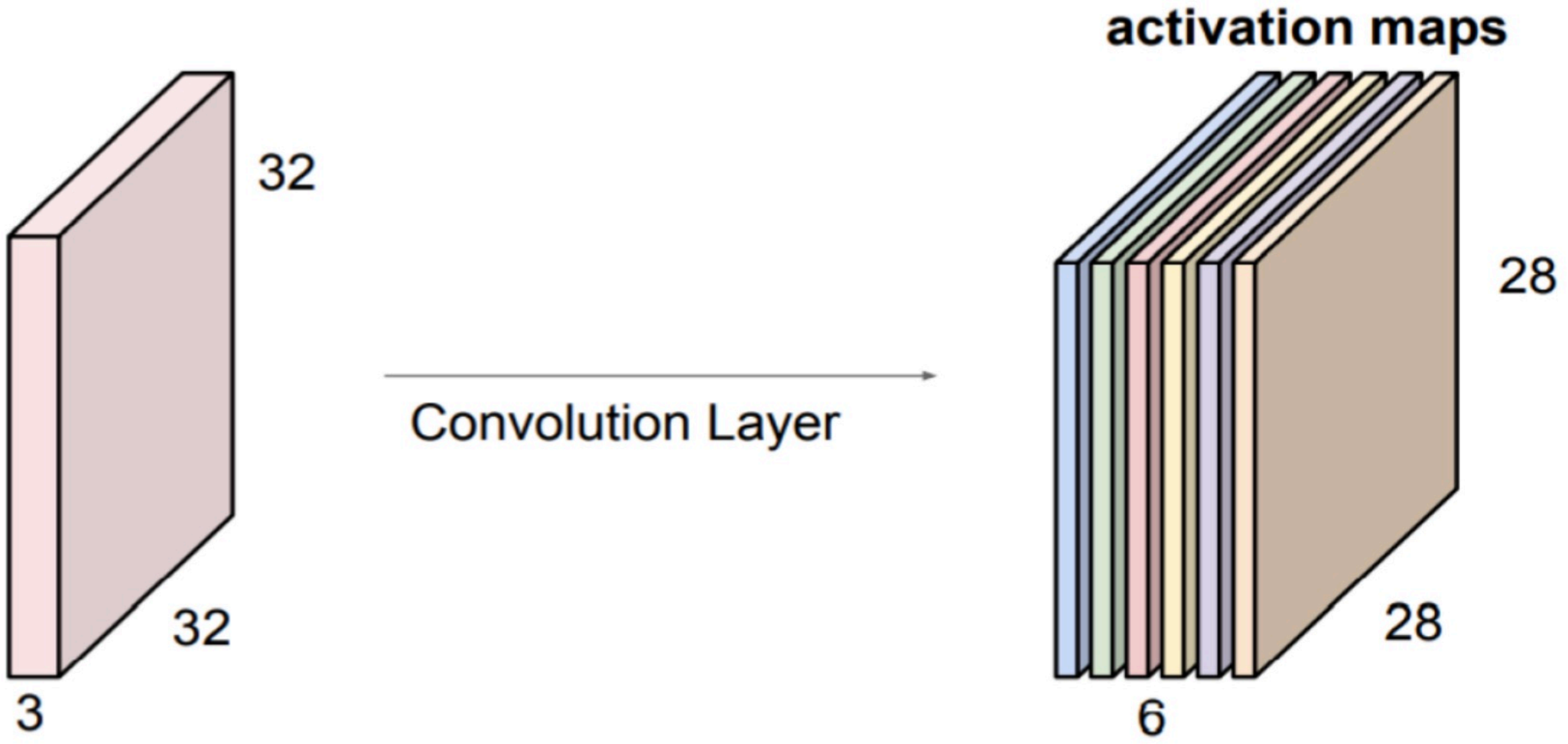
**output height**
((7 + (2 x 0) - 3) / 1) + 1 = 5

**output width**
((9 + (2 x 0) - 3) / 1) + 1 = 7

**output shape (5 x 7)**

# Convolution Layers

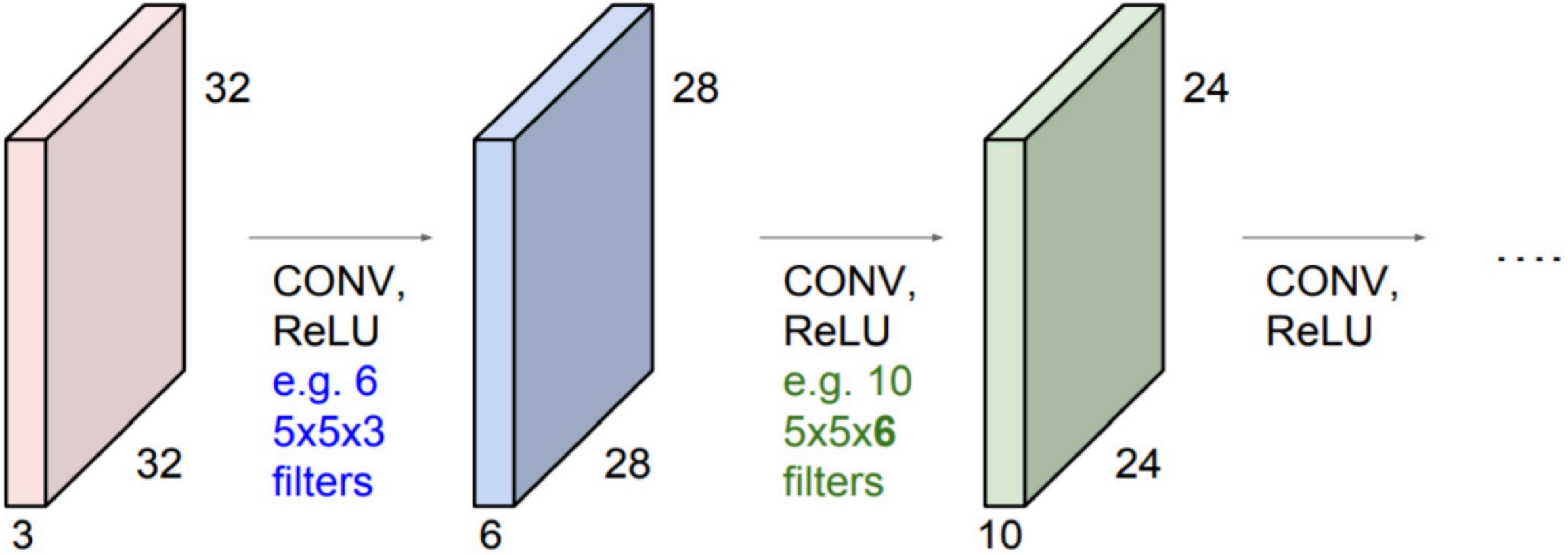
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:




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

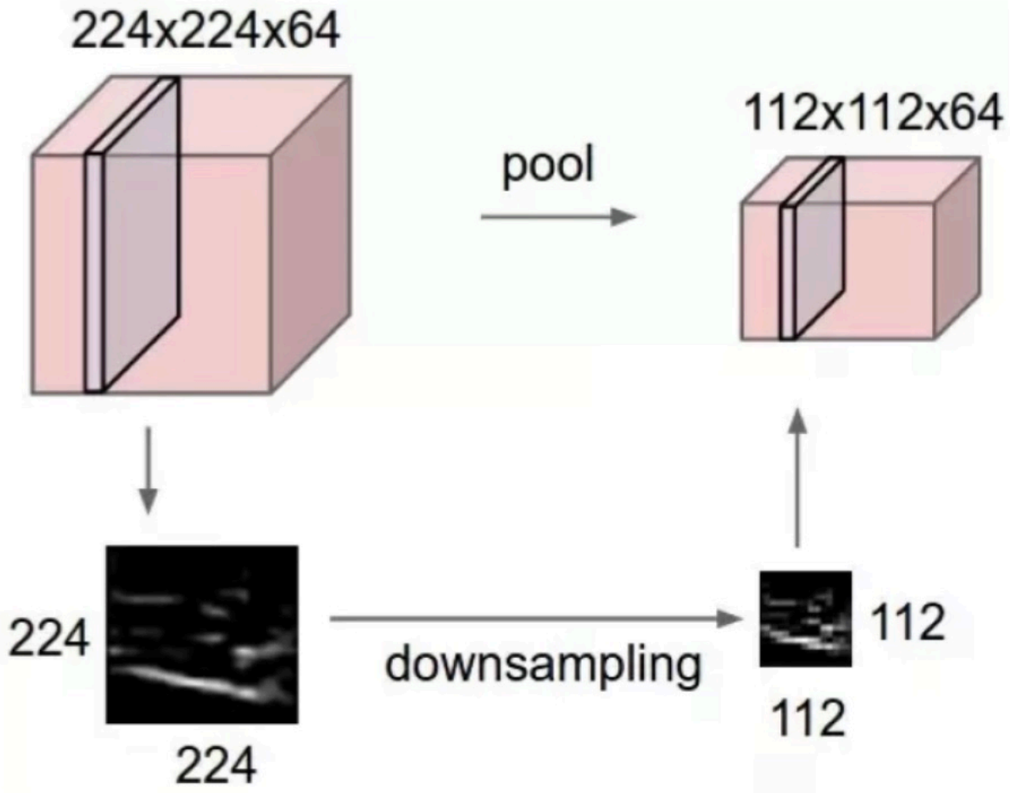
# Convolution Layers

**Preview:** ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



# Pooling Layer

- Pooling layer is to progressively **reduce the spatial size** of the representation to **reduce the network complexity and computational cost**.



# Pooling Layer

**Why pooling?**

Subsampling pixels will not change the object

bird

Subsampling

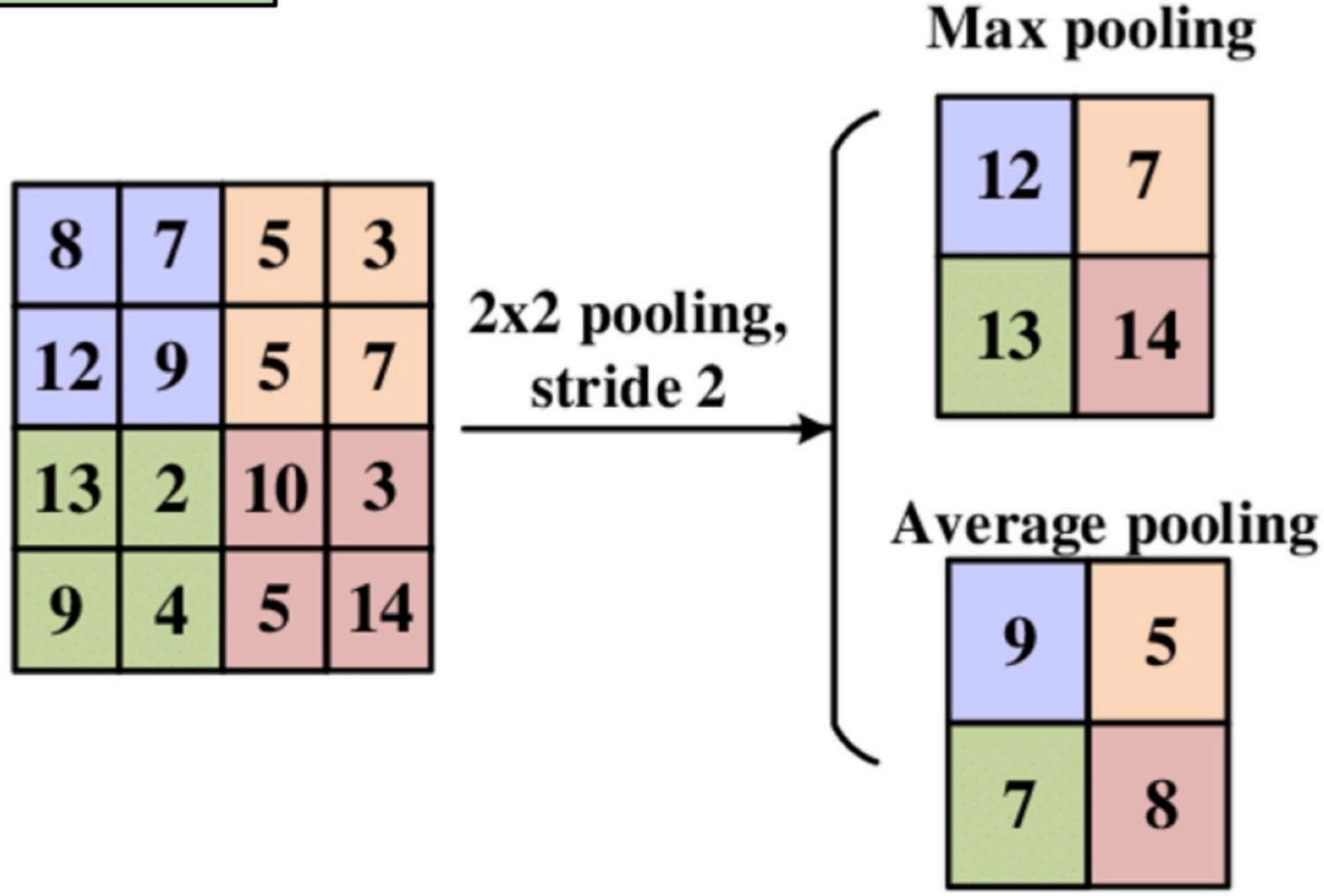
bird

We can subsample the pixels to make image smaller, fewer parameters to characterize the image

Slide from Uni. Waterloo

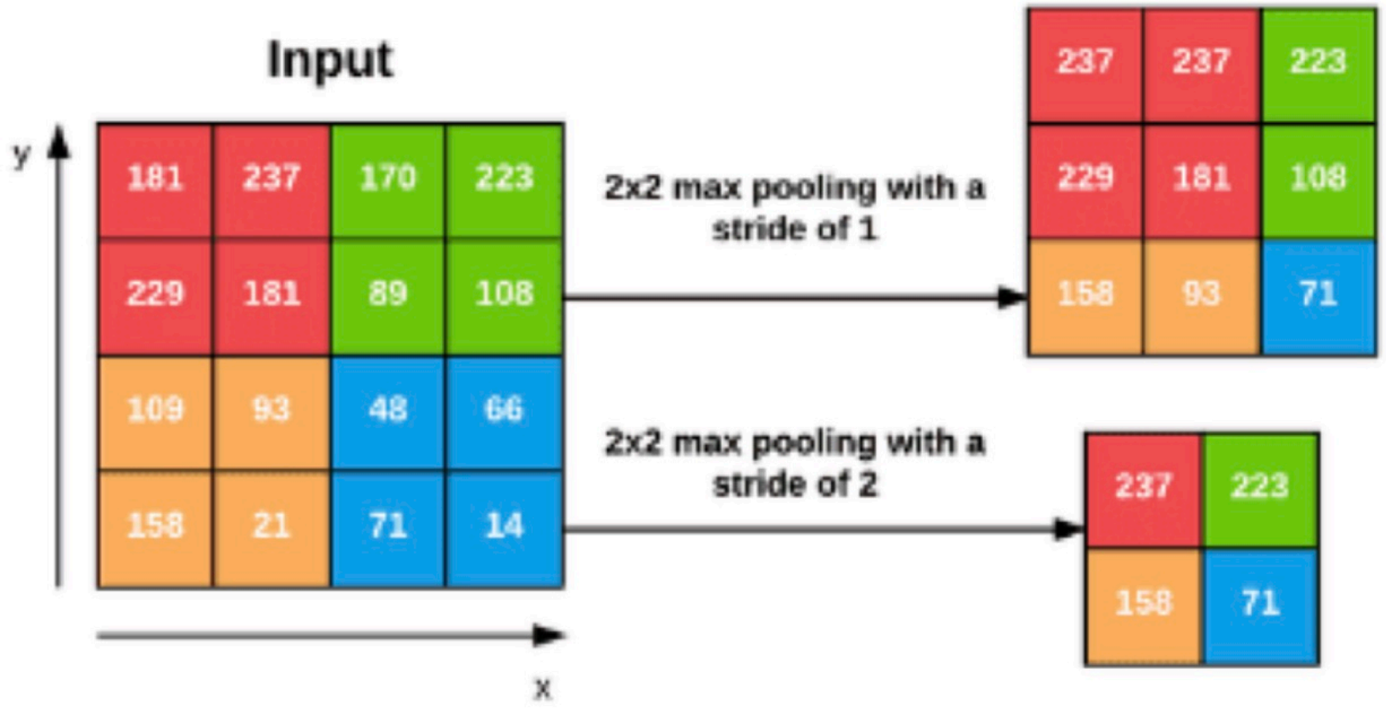
# Pooling Layer

**Types of pooling**



# Pooling Layer

**Pooling example with different stride value**



# Pooling Layer

**Let's see pooling operation with animation**

| | | | | | |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |

6 x 6 image

| | | |
|---|---|---|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

Filter 1

| | | |
|---|---|---|
| -1 | 1 | -1 |
| -1 | 1 | -1 |
| -1 | 1 | -1 |

Filter 2

| | |
|---|---|
| 3 | -1 |
| -3 | 1 |

| | |
|---|---|
| -3 | -1 |
| 0 | -3 |

| | |
|---|---|
| -3 | -3 |
| 3 | -2 |

| | |
|---|---|
| 0 | 1 |
| -2 | -1 |

| | |
|---|---|
| -1 | -1 |
| -1 | -1 |

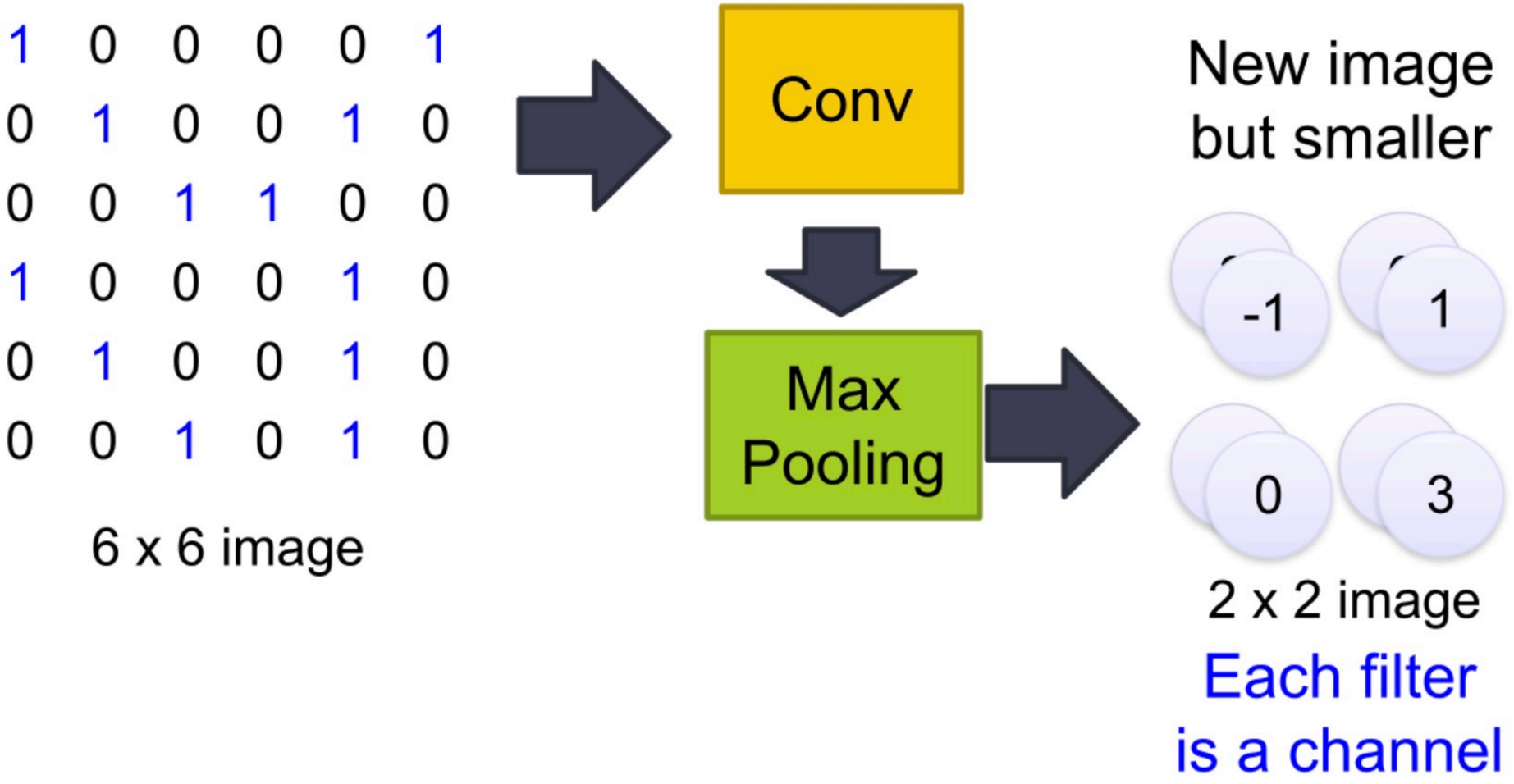
| | |
|---|---|
| -1 | -1 |
| -2 | 1 |

| | |
|---|---|
| -1 | -1 |
| -1 | 0 |

| | |
|---|---|
| -2 | 1 |
| -4 | 3 |

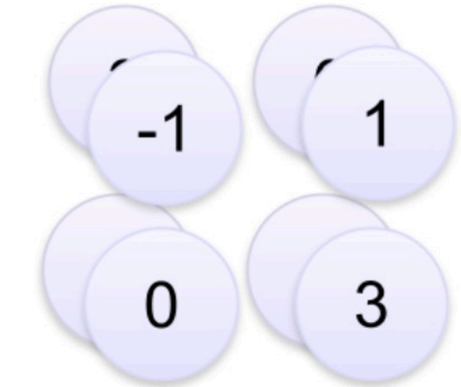
# Pooling Layer

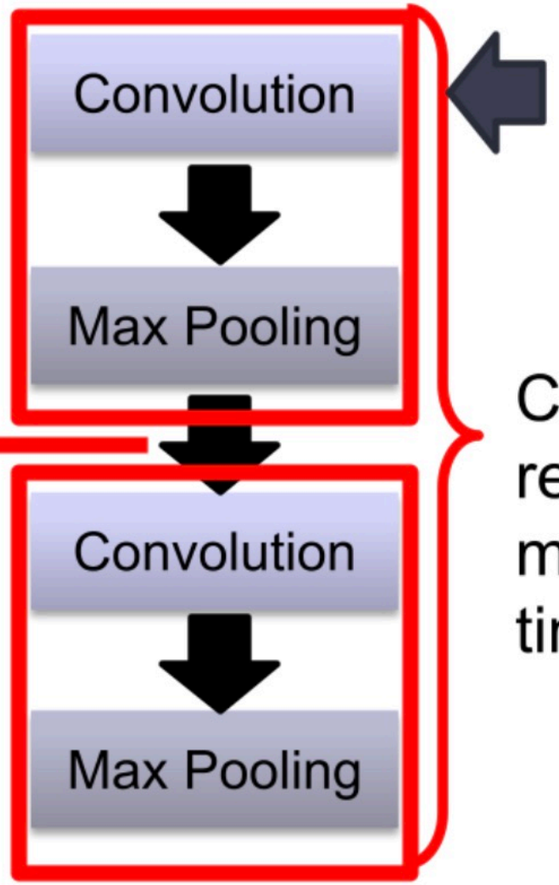
**Let's see pooling operation with animation**




Slide from Uni. Waterloo

# Convolution + Pooling

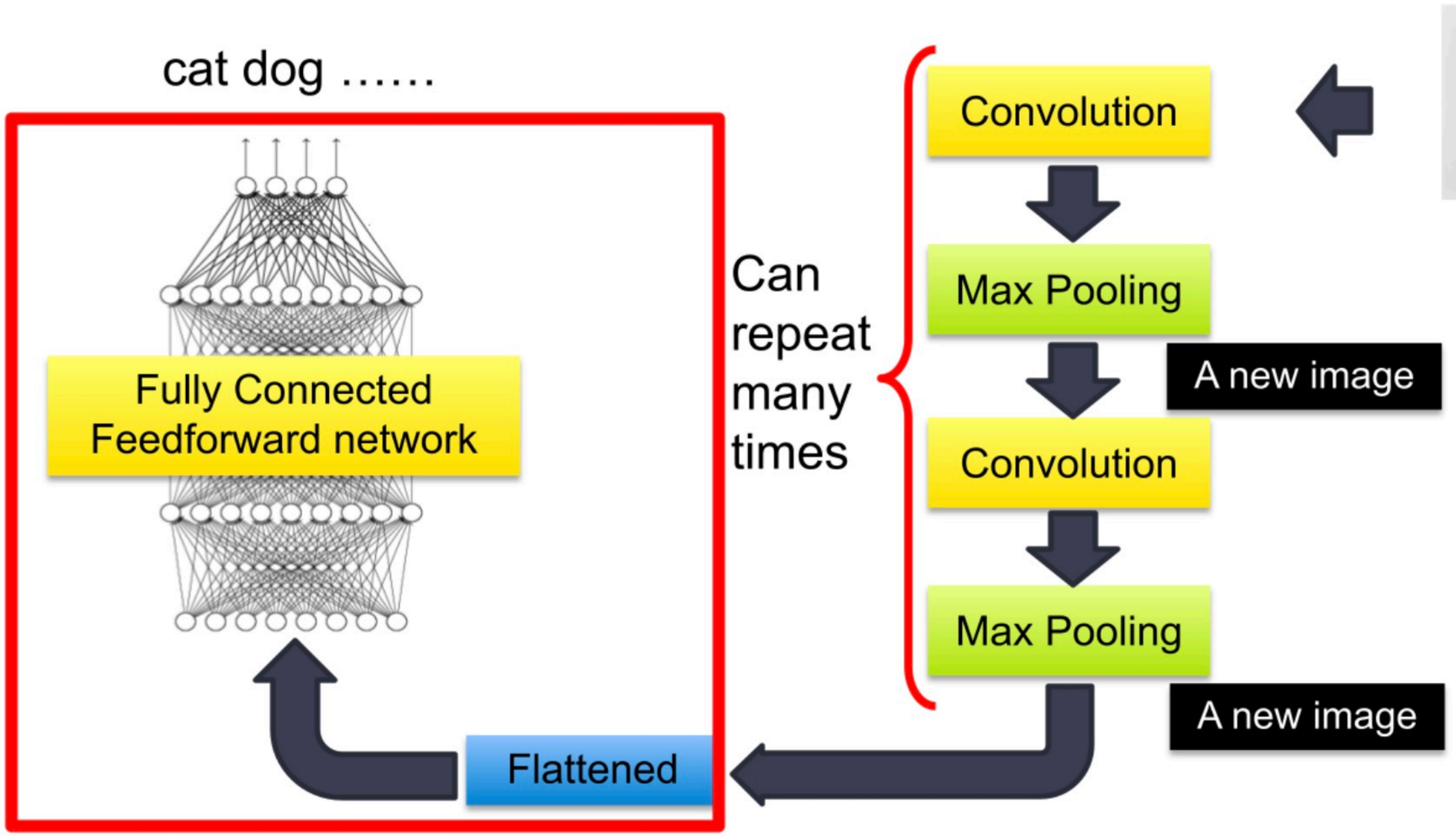



- Smaller than the original image
- The number of channels is the number of filters

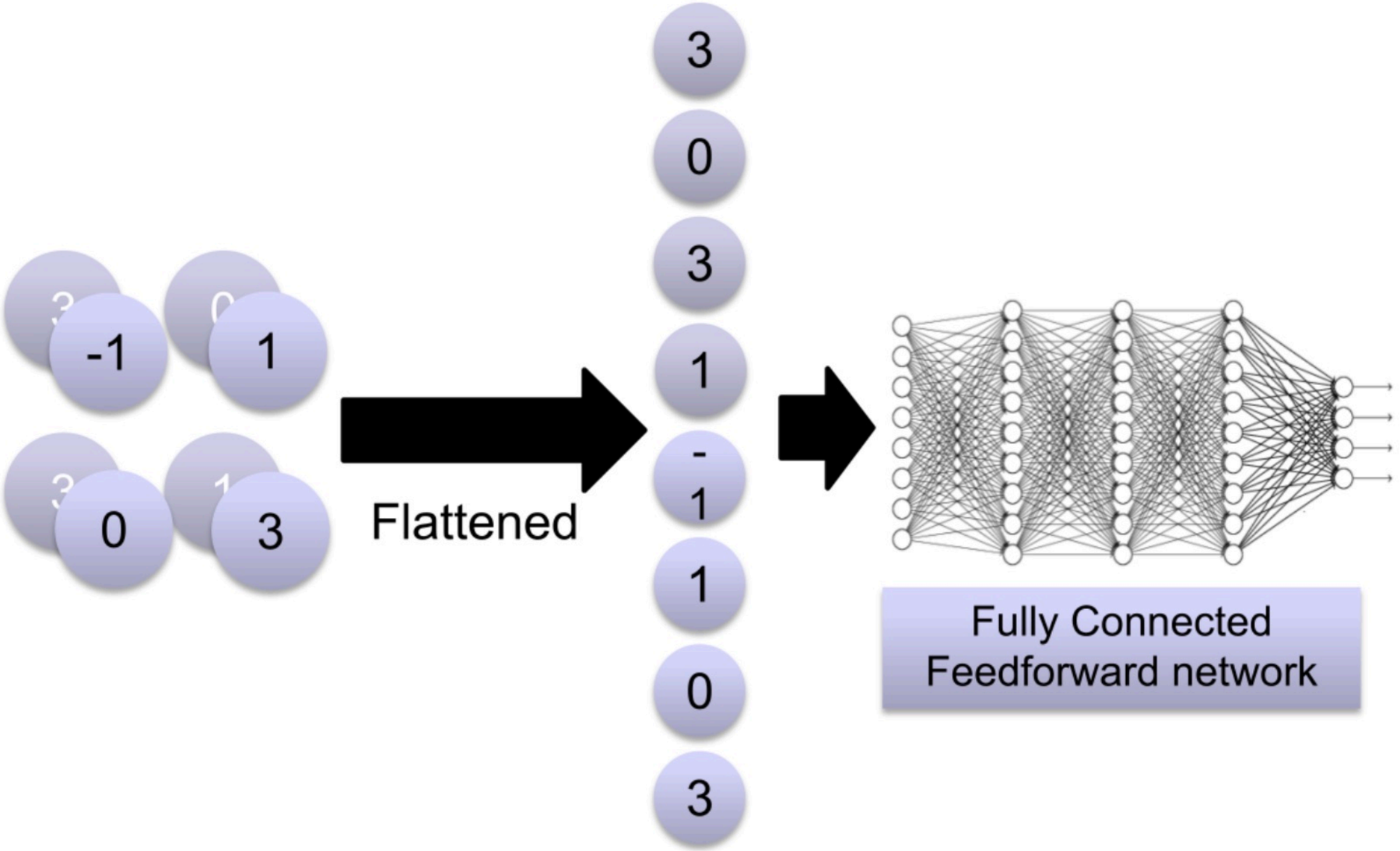
Slide from Uni. Waterloo

# Flattened and FC Layer



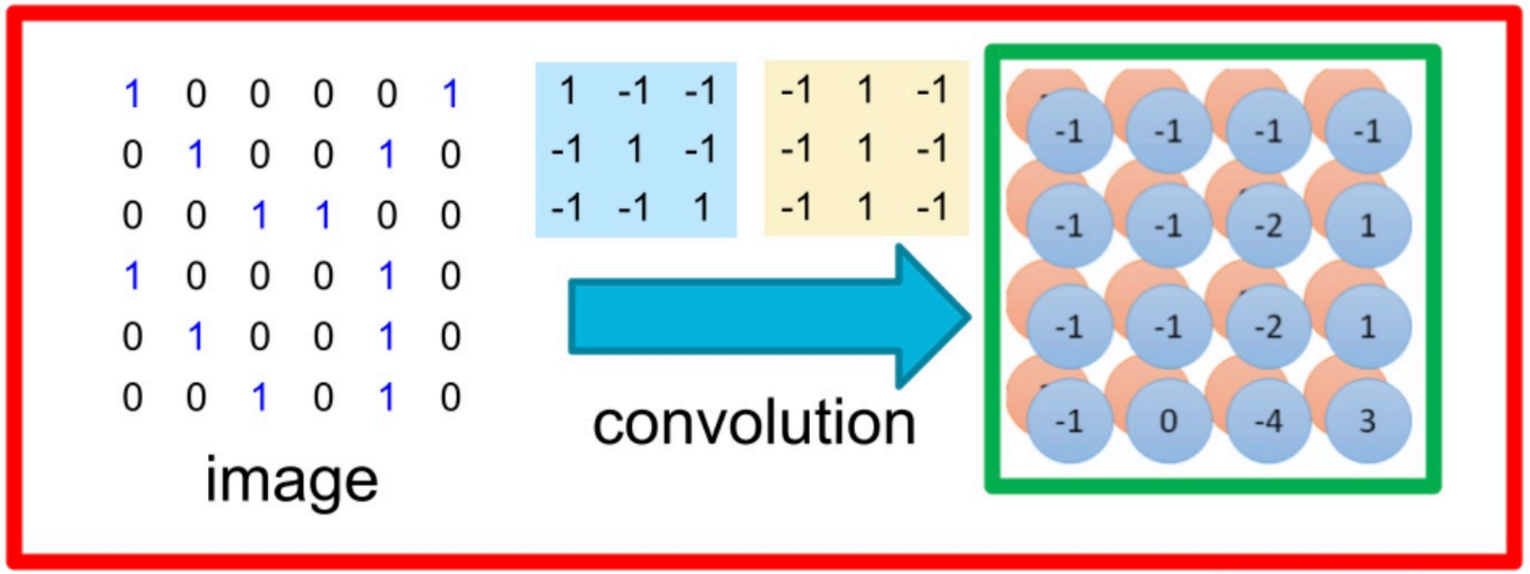

Slide from Uni. Waterloo

# Flattened and FC Layer

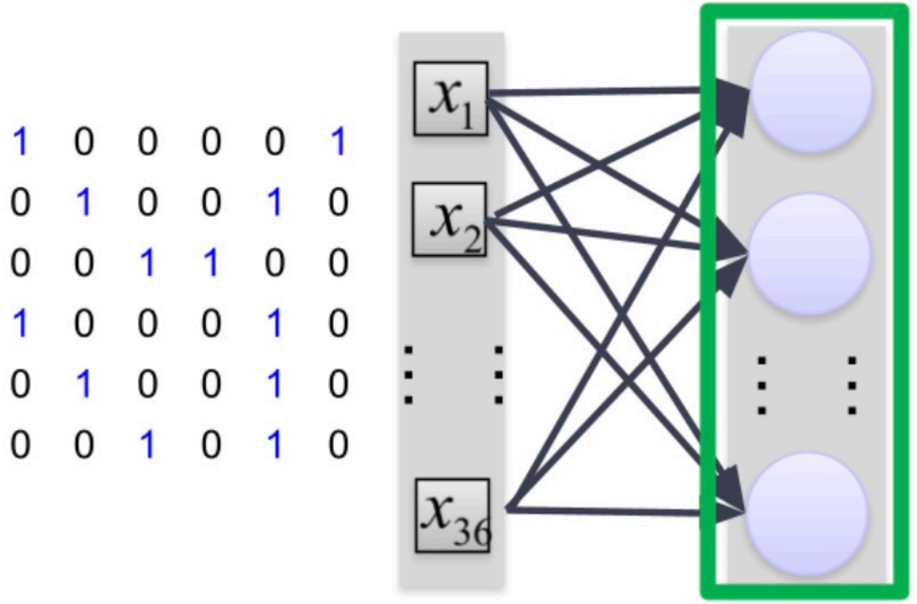



Slide from Uni. Waterloo

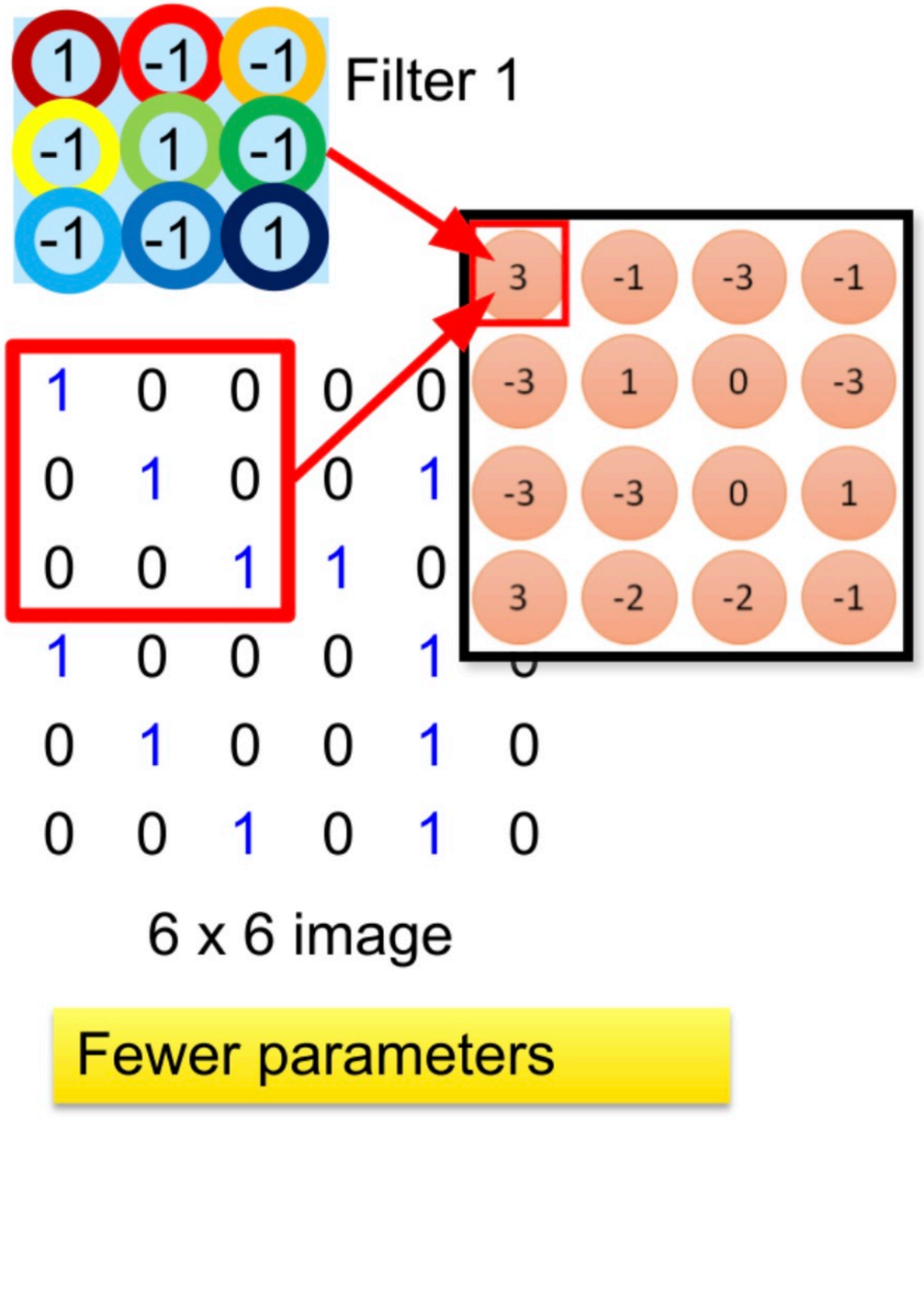
# Convolution v.s. Fully Connected



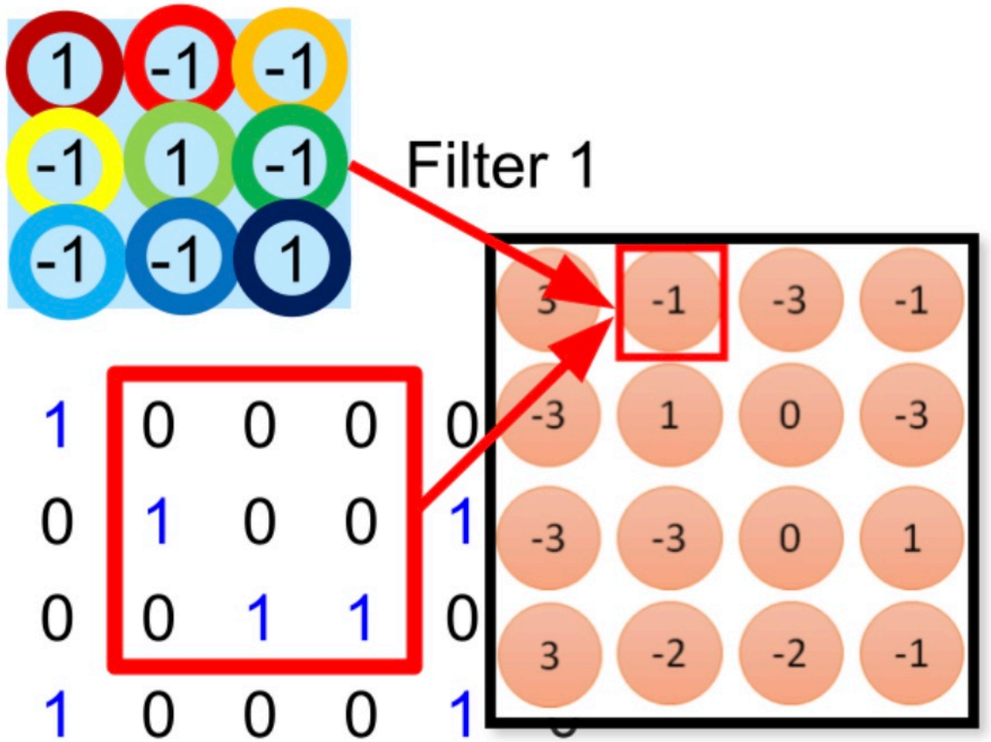

Fully-connected




Slide from Uni. Waterloo

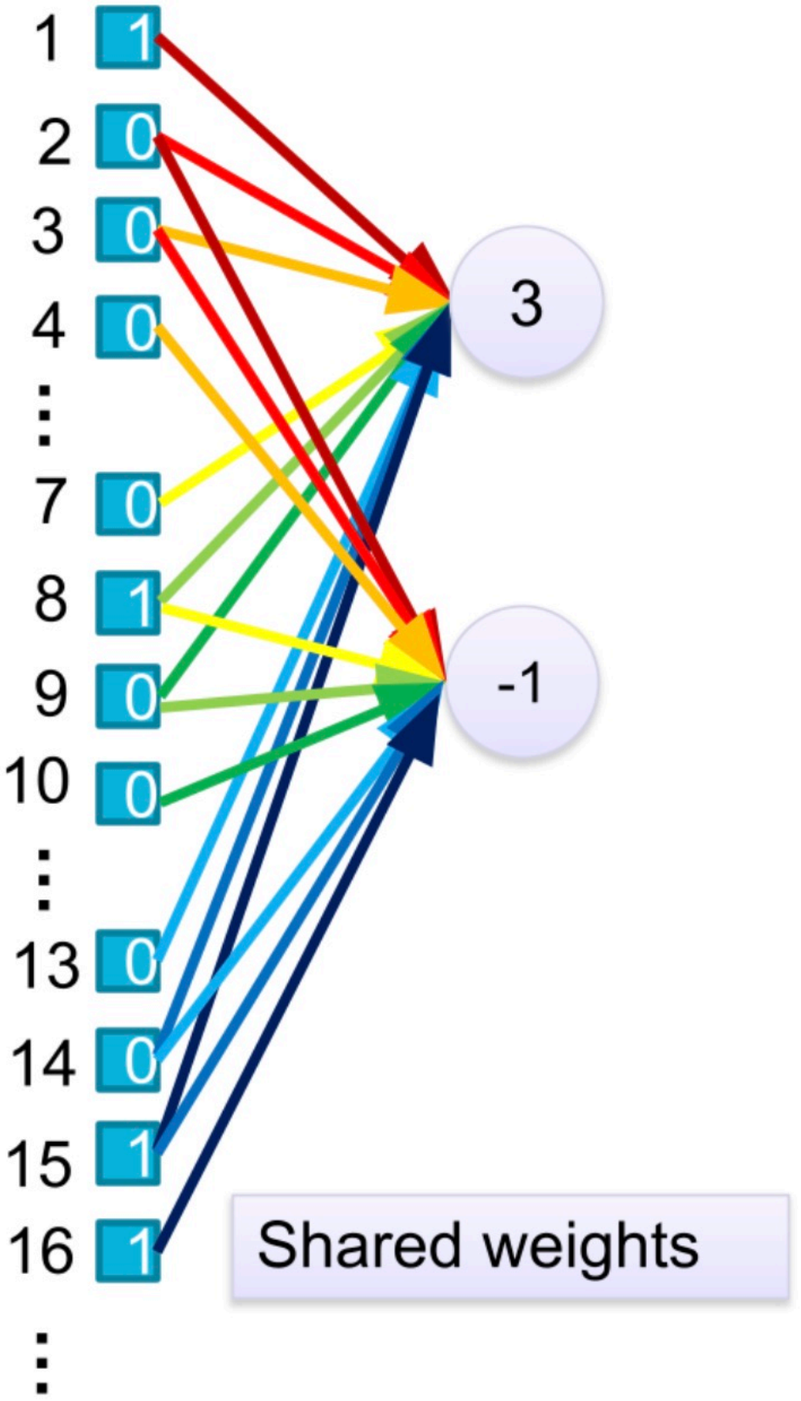



Slide from Uni. Waterloo




Fewer parameters

Even fewer parameters

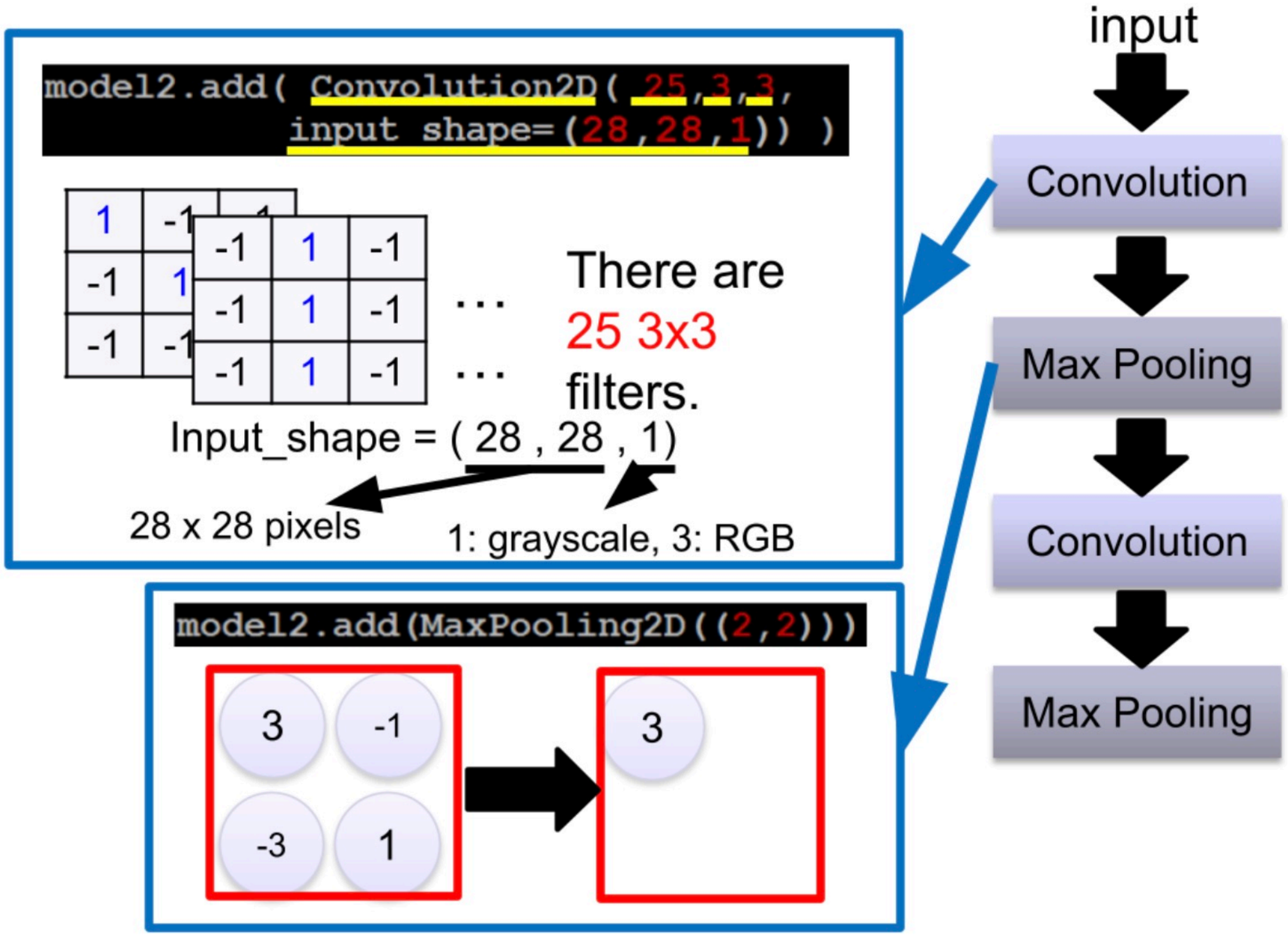



Shared weights

Slide from Uni. Waterloo

# Whole Process

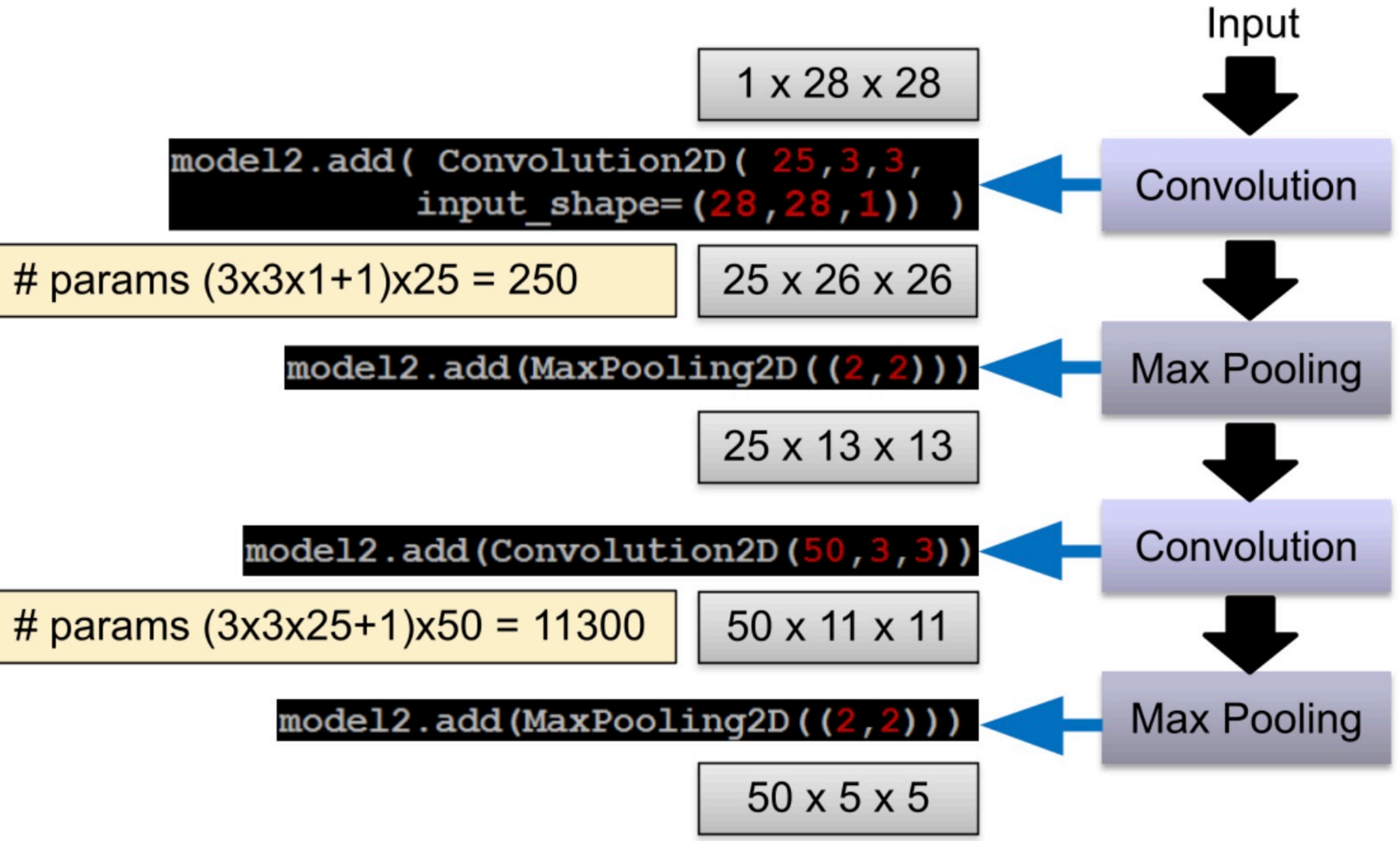
Only modified the ***network structure*** and ***input format (vector -> 3-D tensor)***




Slide from Uni. Waterloo

# Whole Process

Only modified the ***network structure*** and ***input format (vector -> 3-D array)***




**Number of parameter in CL**

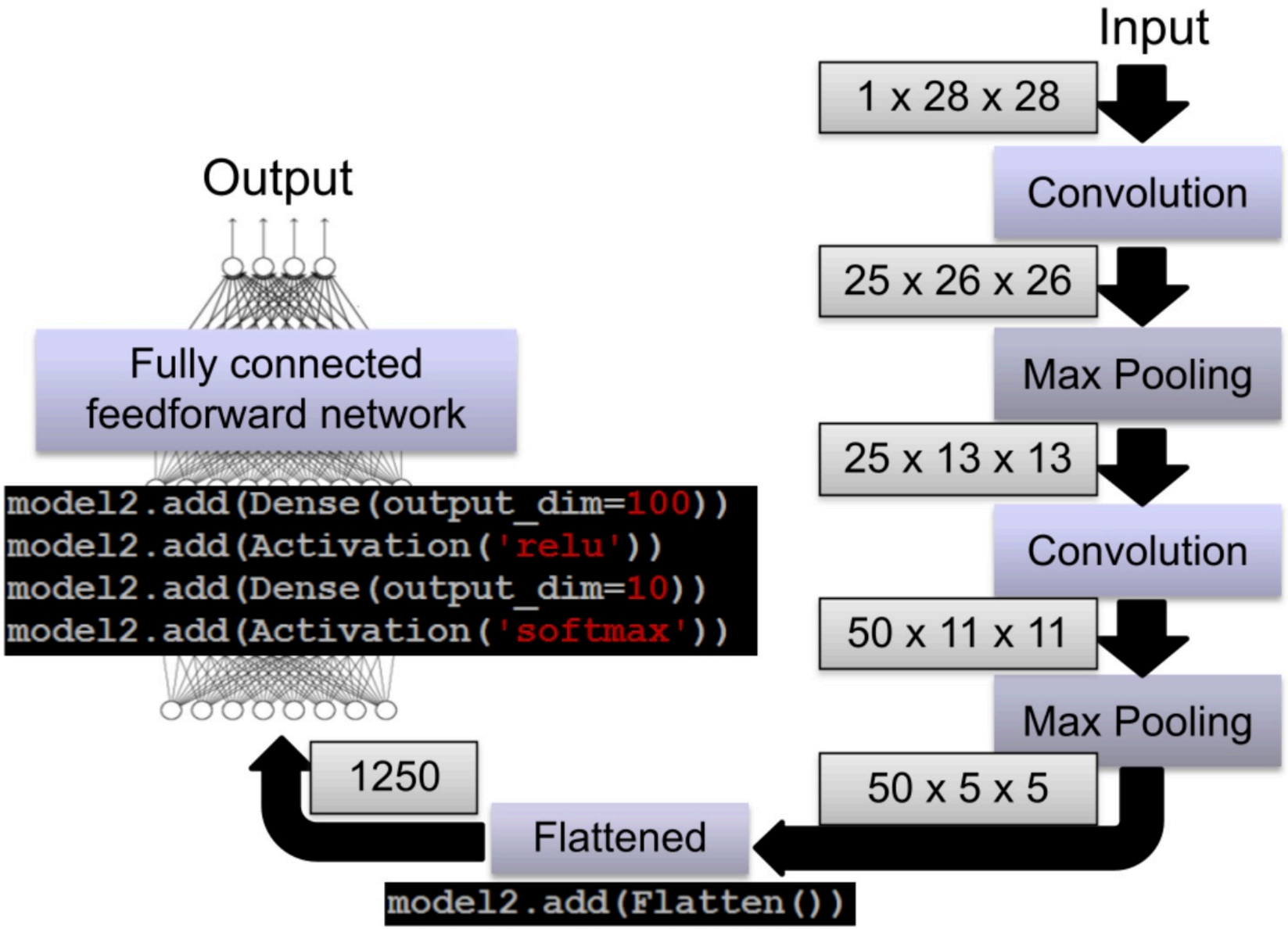
**((n x m x l) + 1) x k**

**n x m = filter shape**
**l = number of input channel** (feature maps as input)
**k = number of output channel** (feature maps as output)

Slide from Uni. Waterloo

# Whole Process

Only modified the ***network structure*** and ***input format (vector -> 3-D array)***




Slide from Uni. Waterloo

# Overview

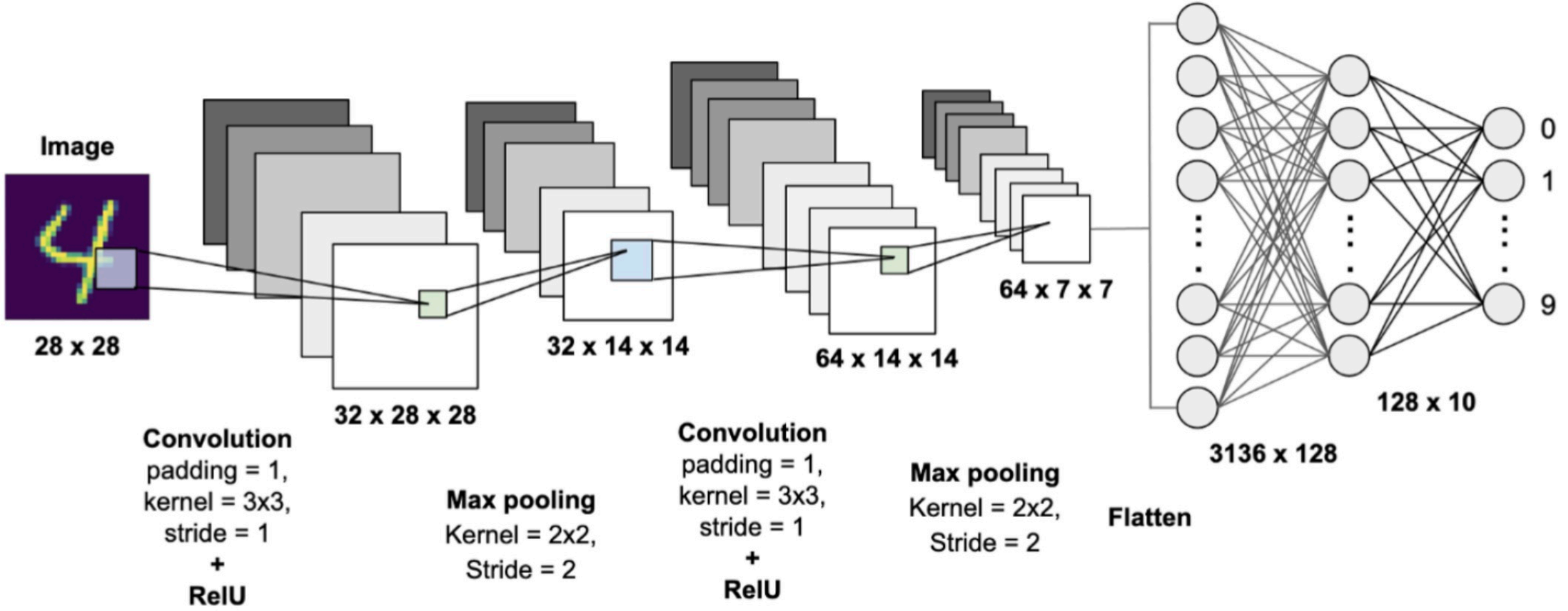


Image
28 x 28
32 x 28 x 28
32 x 14 x 14
64 x 14 x 14
64 x 7 x 7
3136 x 128
128 x 10
0
1
9
Convolution
padding = 1,
kernel = 3x3,
stride = 1
+
RelU
Max pooling
Kernel = 2x2,
Stride = 2
Convolution
padding = 1,
kernel = 3x3,
stride = 1
+
RelU
Max pooling
Kernel = 2x2,
Stride = 2
Flatten