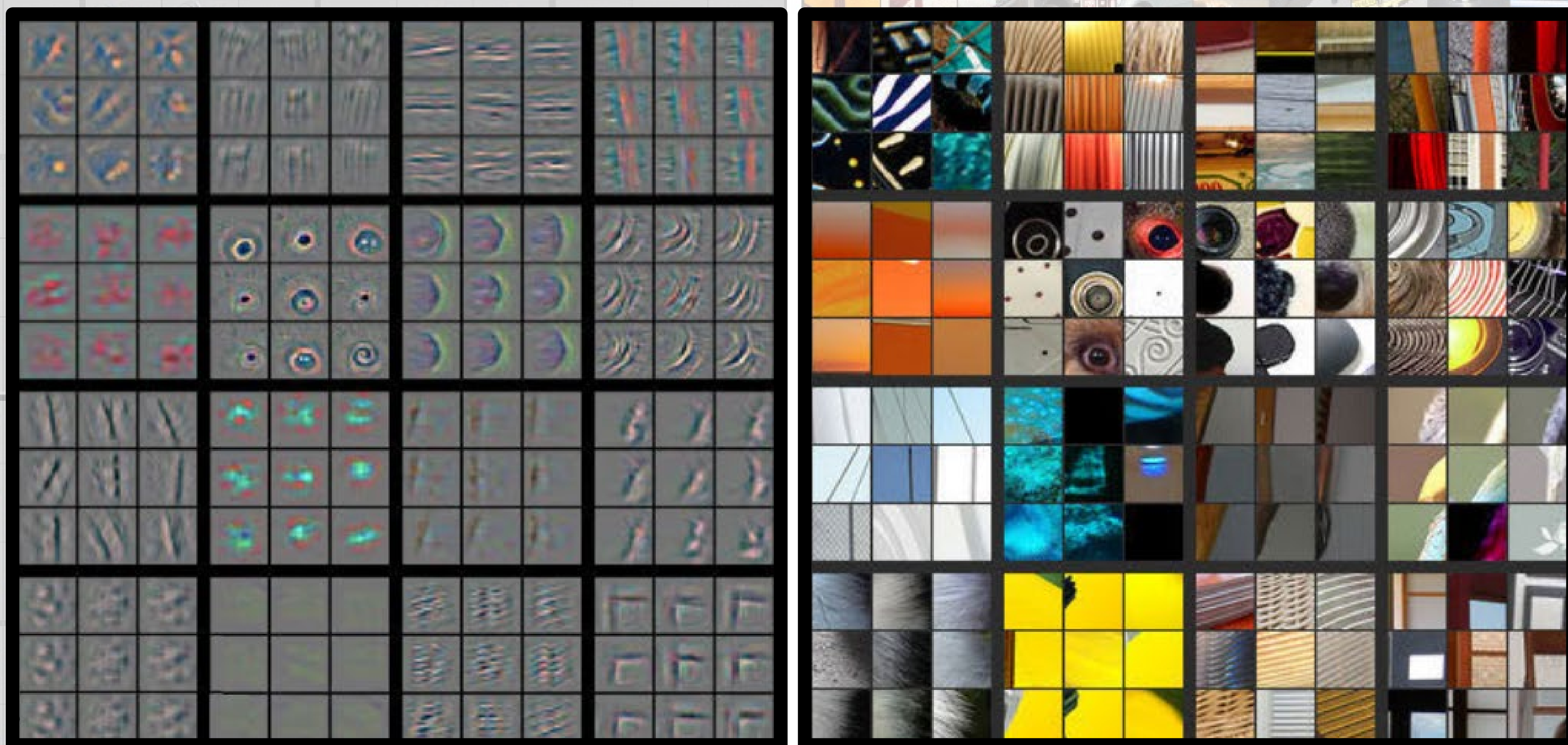


Convolutional Neural Networks

Eren Erdal Aksoy

HALMSTAD UNIVERSITY

Introduction: What is an Image?

- An image is a 2D function: $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y})$, where ***x*** and ***y*** are spatial coordinates, and the ***amplitude*** of the function is called ***intensity*** or ***grey level***.
- An images is just a matrix of numbers, e.g. [0, 255].
- Images are composed of picture elements: ***pixels***

Tasks in Computer Vision

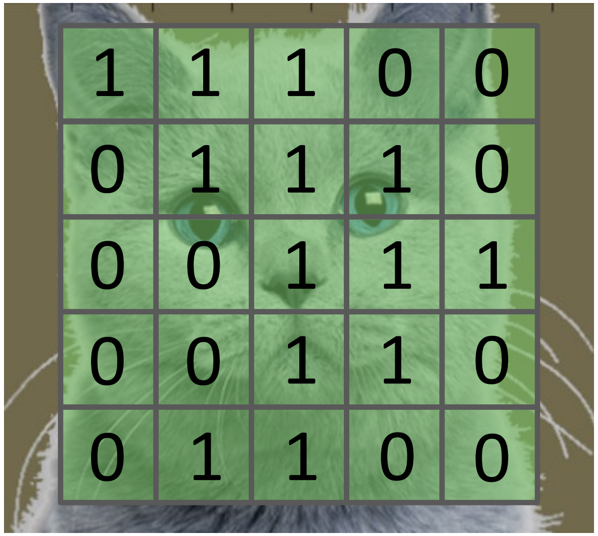
- **Classification**: output variable takes class label. Can produce probability of belonging to a particular class.
- **Regression**: output variable takes continuous value

What we see!

What computers see!

Introduction: What is an Image Processing?

- **Image processing** is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it.
- For instance, we slide the **3x3 filter** over the image, element-wise multiply, and add the outputs



Image

Result

1	0	1
0	1	0
1	0	1

Filter (kernel)

$$I(x,y) * h = \sum_{i=-a}^{a} \sum_{j=-b}^{b} \underbrace{I(x-i, y-j) . h(i,j)}_{\text{Inner Product}}$$

Image Convolution

a process of combining image **pixels** with a certain matrix weight to identify specific features of the image, such as edge detection, sharpening, blurring,

Introduction: Image Convolution

- **Image processing** is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it.
- For instance, we slide the **3x3 filter** over the image, element-wise multiply, and add the outputs

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

$$I(x,y) * h = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I(x-i, y-j).\, h(i,j)$$

Result

1	0	1
0	1	0
1	0	1

Filter (kernel)

Introduction: Image Convolution

- **Image processing** is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it.
- For instance, we slide the **3x3 filter** over the image, element-wise multiply, and add the outputs

$$I(x,y) * h = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I(x-i, y-j).\, h(i,j)$$

$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0	0
$0_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$	1	0
$0_{\times 1}$	$0_{\times 0}$	$1_{\times 1}$	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Result

(1x1)+(1x0)+(1x1)+
(0x0)+(1x1)+(1x0)+
(0x1)+(0x0)+(1x1)= 4

1	0	1
0	1	0
1	0	1

Filter (kernel)

Introduction: Image Convolution

- **Image processing** is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it.
- For instance, we slide the **3x3 filter** over the image, element-wise multiply, and add the outputs

1	1 ×1	1 ×0	0 ×1	0
0	1 ×0	1 ×1	1 ×0	0
0	0 ×1	1 ×0	1 ×1	1
0	0	1	1	0
0	1	1	0	0

Image

1	0	1
0	1	0
1	0	1

Filter (kernel)

$$I(x,y) * h = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I(x-i, y-j).\, h(i,j)$$

4	3	

Result

(1x1)+(1x0)+(0x1)+
(1x0)+(1x1)+(1x0)+
(0x1)+(1x0)+(1x1)= 3

Introduction: Image Convolution

- **Image processing** is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it.
- For instance, we slide the **3x3 filter** over the image, element-wise multiply, and add the outputs

$$I(x, y) * h = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I(x-i, y-j) . h(i, j)$$

1	1	1 ×1	0 ×0	0 ×1
0	1	1 ×0	1 ×1	0 ×0
0	0	1 ×1	1 ×0	1 ×1
0	0	1	1	0
0	1	1	0	0

Image

4	3	4

Result

(1x1)+(0x0)+(0x1)+
(1x0)+(1x1)+(0x0)+
(1x1)+(1x0)+(1x1)= 4

1	0	1
0	1	0
1	0	1

Filter (kernel)

Introduction: Image Convolution

- **Image processing** is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it.
- For instance, we slide the **3x3 filter** over the image, element-wise multiply, and add the outputs

1	1	1	0	0
0 ×1	1 ×0	1 ×1	1	0
0 ×0	0 ×1	1 ×0	1	1
0 ×1	0 ×0	1 ×1	1	0
0	1	1	0	0

Image

1	0	1
0	1	0
1	0	1

Filter (kernel)

$$I(x,y) * h = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I(x-i, y-j).\, h(i,j)$$

4	3	4
2		

Result

(0x1)+(1x0)+(1x1)+
(0x0)+(0x1)+(1x0)+
(0x1)+(0x0)+(1x1)= 2

Introduction: Image Convolution

- **Image processing** is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it.
- For instance, we slide the **3x3 filter** over the image, element-wise multiply, and add the outputs

1	1	1	0	0
0	1 ×1	1 ×0	1 ×1	0
0	0 ×0	1 ×1	1 ×0	1
0	0 ×1	1 ×0	1 ×1	0
0	1	1	0	0

Image

$$I(x,y) * h = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I(x-i, y-j) . h(i,j)$$

4	3	4
2	4	

Result

(1x1)+(1x0)+(1x1)+
(0x0)+(1x1)+(1x0)+
(0x1)+(1x0)+(1x1)= 4

1	0	1
0	1	0
1	0	1

Filter (kernel)

Introduction: Image Convolution

- **Image processing** is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it.
- For instance, we slide the **3x3 filter** over the image, element-wise multiply, and add the outputs

1	1	1	0	0
0	1	$1_{\times 1}$	$1_{\times 0}$	$0_{\times 1}$
0	0	$1_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$
0	0	$1_{\times 1}$	$1_{\times 0}$	$0_{\times 1}$
0	1	1	0	0

Image

1	0	1
0	1	0
1	0	1

Filter (kernel)

$$I(x,y) * h = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I(x-i, y-j).\, h(i,j)$$

4	3	4
2	4	3

Result

(1x1)+(1x0)+(0x1)+
(1x0)+(1x1)+(1x0)+
(1x1)+(1x0)+(0x1)= 3

Introduction: Image Convolution

- **Image processing** is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it.
- For instance, we slide the **3x3 filter** over the image, element-wise multiply, and add the outputs

1	1	1	0	0
0	1	1	1	0
0 ×1	0 ×0	1 ×1	1	1
0 ×0	0 ×1	1 ×0	1	0
0 ×1	1 ×0	1 ×1	0	0

Image

1	0	1
0	1	0
1	0	1

Filter (kernel)

$$I(x,y) * h = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I(x-i, y-j) . h(i,j)$$

4	3	4
2	4	3
2		

Result

(0x1)+(0x0)+(1x1)+
(0x0)+(0x1)+(1x0)+
(0x1)+(1x0)+(1x1)= 2

Introduction: Image Convolution

- **Image processing** is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it.
- For instance, we slide the **3x3 filter** over the image, element-wise multiply, and add the outputs

1	1	1	0	0
0	1	1	1	0
0	0 ×1	1 ×0	1 ×1	1
0	0 ×0	1 ×1	1 ×0	0
0	1 ×1	1 ×0	0 ×1	0

Image

1	0	1
0	1	0
1	0	1

Filter (kernel)

$$I(x,y) * h = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I(x-i, y-j).\, h(i,j)$$

4	3	4
2	4	3
2	3	

Result

(0x1)+(1x0)+(1x1)+
(0x0)+(1x1)+(1x0)+
(1x1)+(1x0)+(0x1)= 3

Introduction: Image Convolution

- **Image processing** is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it.
- For instance, we slide the **3x3 filter** over the image, element-wise multiply, and add the outputs

1	1	1	0	0
0	1	1	1	0
0	0	$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$
0	0	$1_{\times 0}$	$1_{\times 1}$	$0_{\times 0}$
0	1	$1_{\times 1}$	$0_{\times 0}$	$0_{\times 1}$

Image

$$I(x,y) * h = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I(x-i, y-j).\, h(i,j)$$

4	3	4
2	4	3
2	3	4

Result

(1x1)+(1x0)+(1x1)+
(1x0)+(1x1)+(0x0)+
(1x1)+(0x0)+(0x1)= 4

1	0	1
0	1	0
1	0	1

Filter (kernel)

Introduction: Feature Extraction

- **Each filter** identifies a **specific set of features** of the image

Original

Sharpen

Edge Detect

Strong Edge Detect

How to define mask coefficients?

Depends on what the filter is supposed to do!

Original

0	0	0
0	0	0
0	0	0

Filter

Filtered (Deleting)

Introduction: Feature Extraction

- **Each filter** identifies a **specific set of features** of the image

Original

Sharpen

Edge Detect

Strong Edge Detect

How to define mask coefficients?

Depends on what the filter is supposed to do!

Original

0	0	0
0	1	0
0	0	0

Filter

Filtered
(No change)

Introduction: Feature Extraction

- **Each filter** identifies a **specific set of features** of the image

Original

Sharpen

Edge Detect

Strong Edge Detect

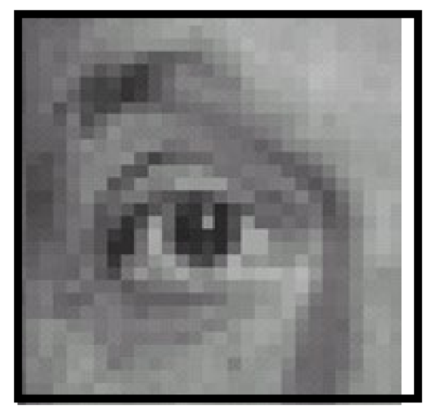
How to define mask coefficients?

Depends on what the filter is supposed to do!

Original

0	0	0
0	0	1
0	0	0

Filter

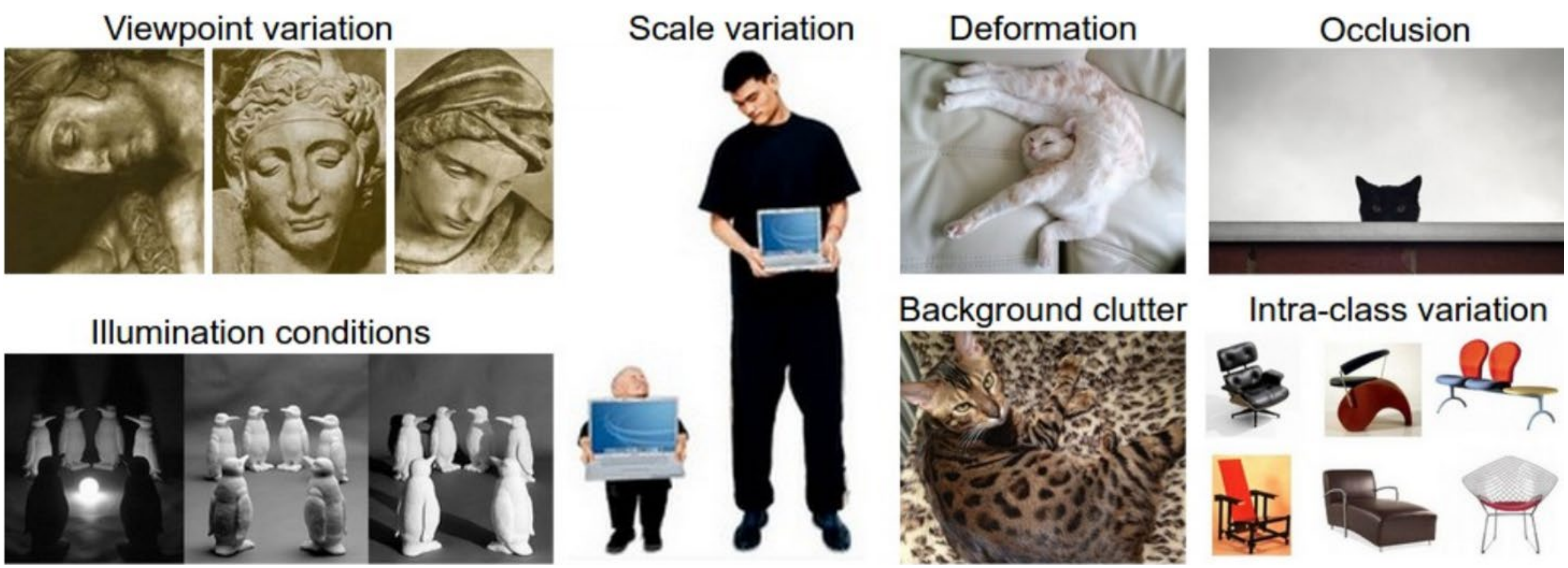


Shifted *left* by 1 pixel

Manual Feature Extraction

- Features should be **robust to image variations**

Domain Knowledge → **Define Features** → **Detect Features to Classify**

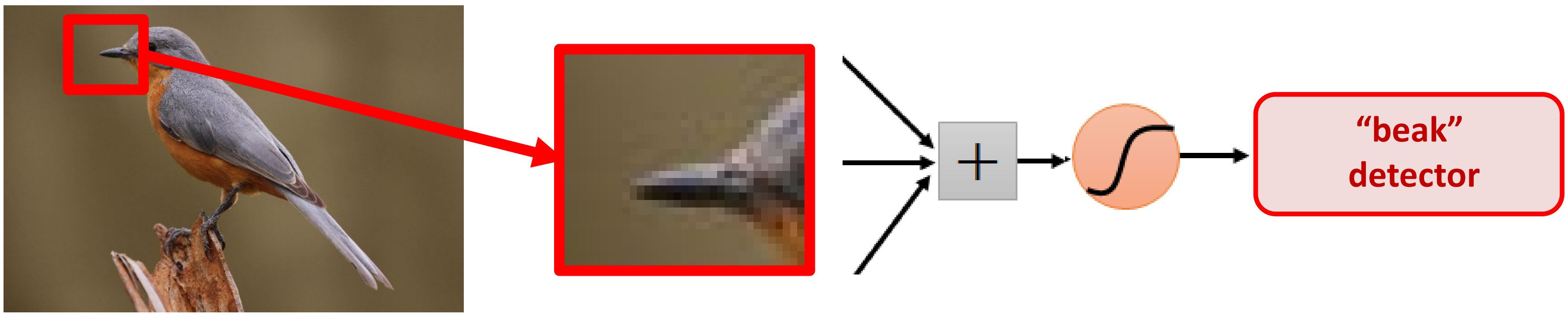


Can we learn a *hierarchy of features* *robust to* *image variations* directly from the data instead of hand engineering?

Solution: Convolutional Neural Networks

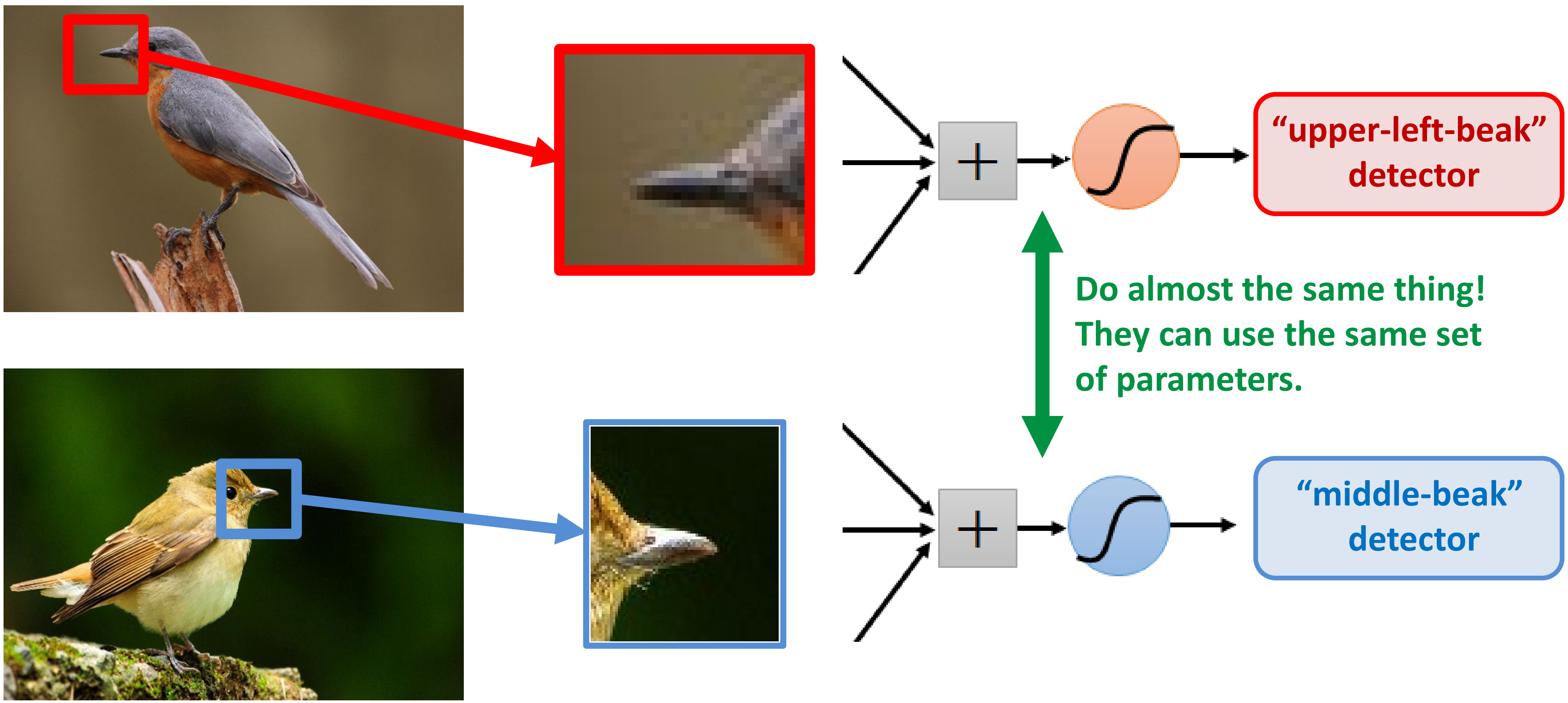
Why CNN for Image?

1. **Some patterns are much smaller than the whole image**
   - A neuron **does not have to see the whole image** to discover the pattern
   - Connecting to a small region with **fewer parameters**



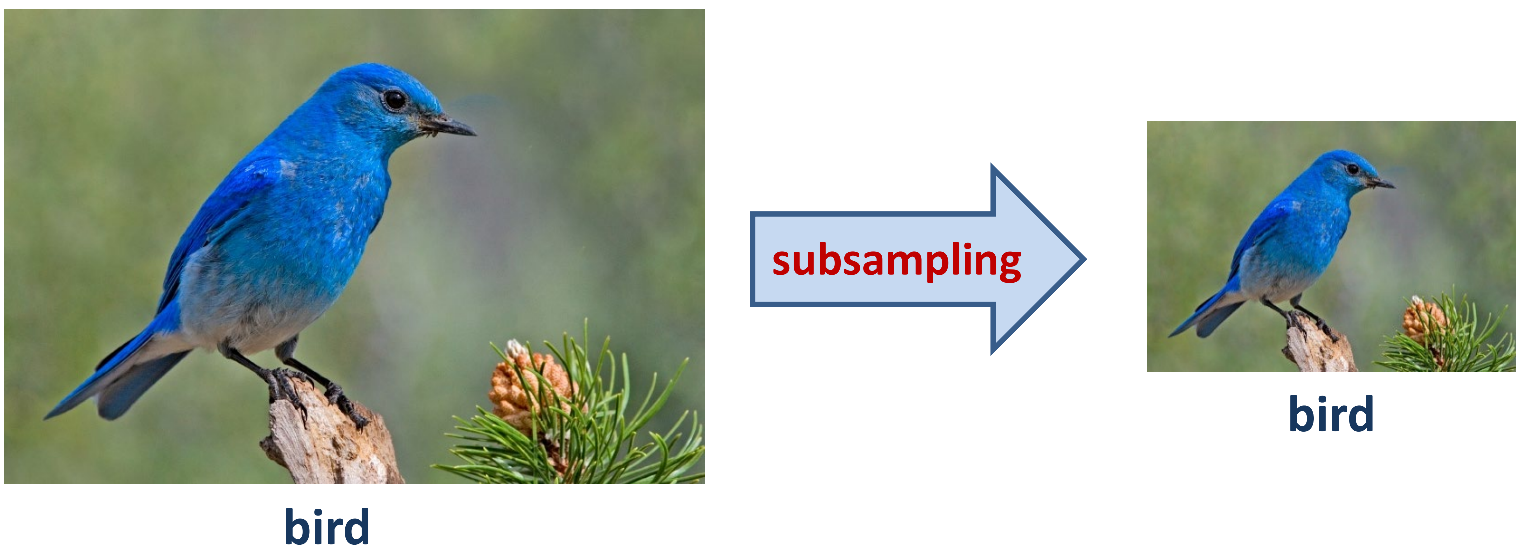
Why CNN for Image?

1. **Some patterns are much smaller than the whole image**
   - A neuron **does not have to see the whole image** to discover the pattern
   - Connecting to a small region with **fewer parameters**
2. **The very same patterns appear in different regions**

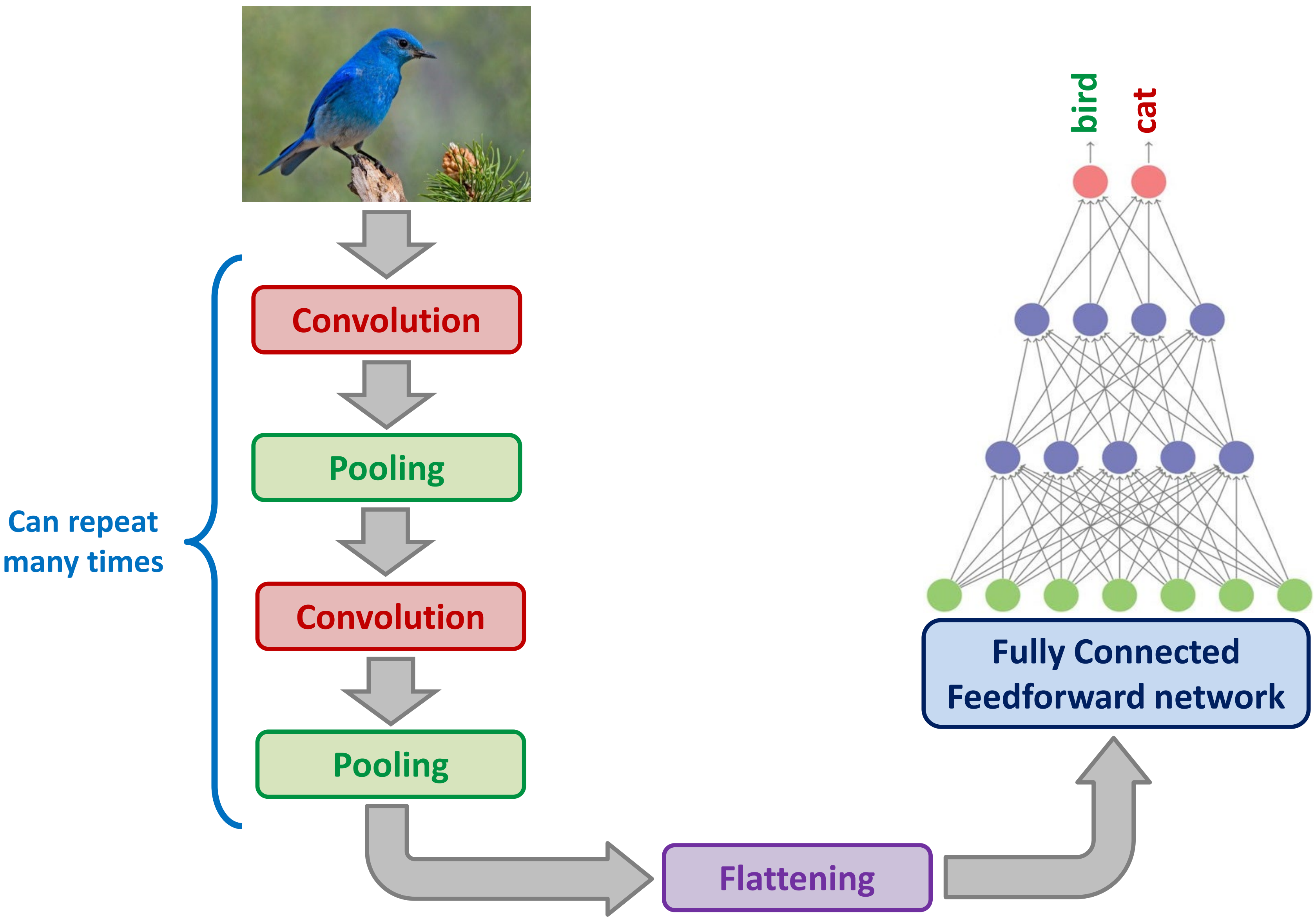


Why CNN for Image?

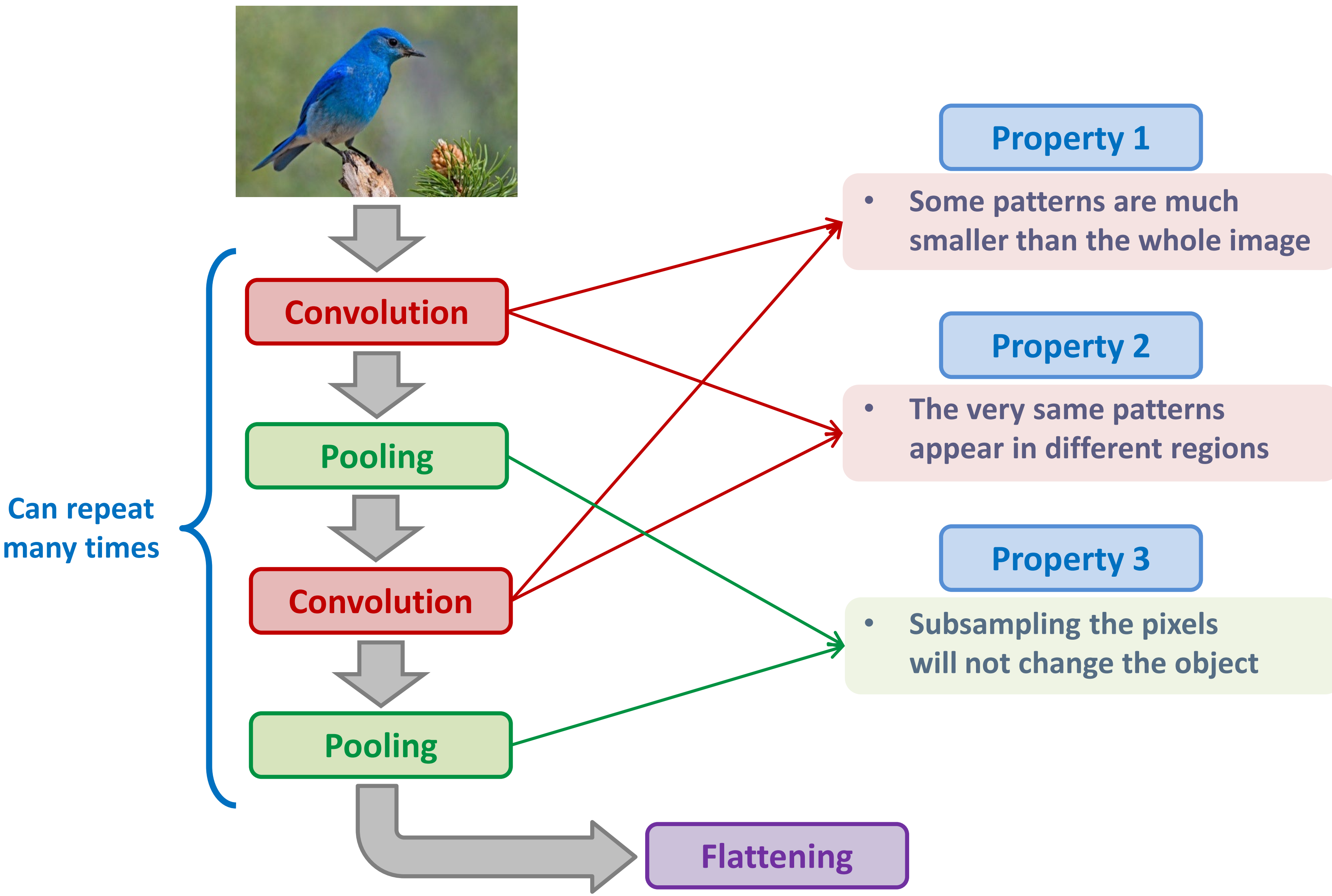
1. **Some patterns are much smaller than the whole image**
   - A neuron **does not have to see the whole image** to discover the pattern
   - Connecting to a small region with **fewer parameters**
2. **The very same patterns appear in different regions**
3. **Subsampling the pixels will not change the object**
   - We can **subsample** the pixels to make the image **smaller**
   - **Fewer parameters** for the network to process the image



The Whole CNN



The Whole CNN



CNN: Convolution

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

⋮

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

Property 1

Note that these filter indices are the network parameters to be learned. We do not predefine them like in conventional image processing methods.

CNN: Convolution

Stride is how far the filter moves in every step along one direction.

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

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

3 -1

CNN: Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

What if Stride is 2?

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

3   -3

Unless otherwise stated, **stride is by default 1**

CNN: Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 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

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

Feature Map

A feature map, a.k.a. Activation Map**,** is the result of applying a convolution across an image

CNN: Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 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

Why do we have two highest scores there?

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

Feature Map

CNN: Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

This filter detects **left diagonal edges**

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

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

Property 2

The very same patterns appear in different regions

CNN: Convolution

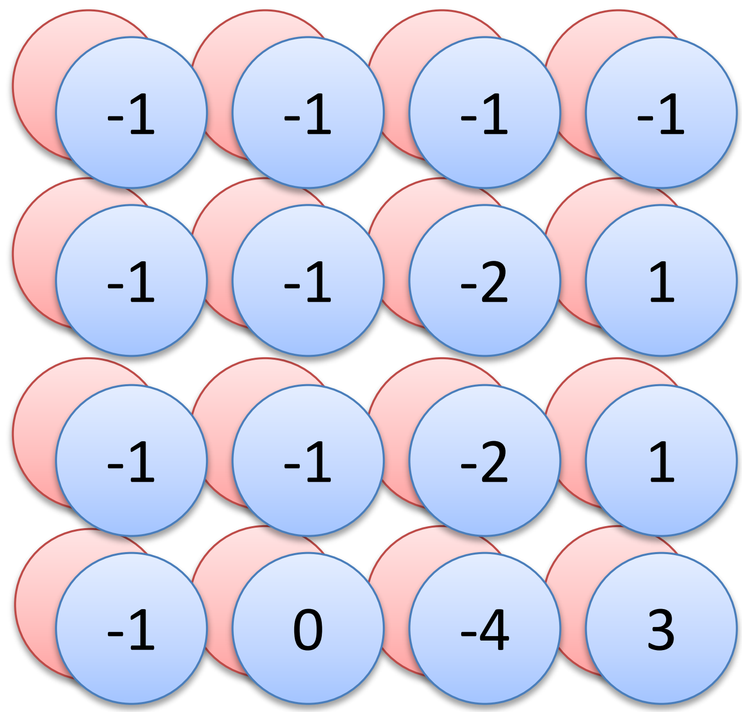
-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

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

Follow the same process for **every filter**



Feature Maps

CNN: Convolution

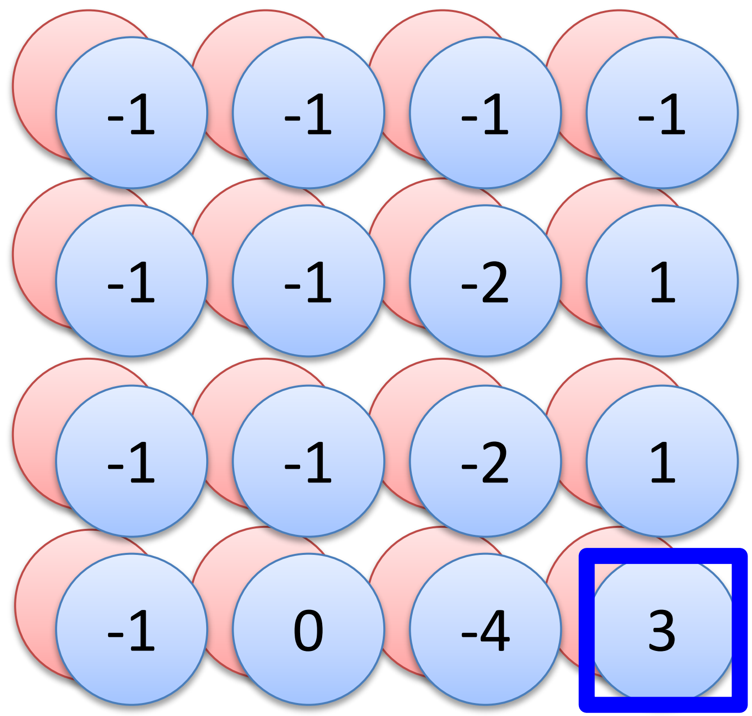
-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

This filter detects **vertical edges**

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



Feature Maps are 4 x 4 images

For example,
if we had **6 3x3 filters**, we'll get **6 separate feature maps**.
We stack these up to get a "**new image**" of size **4x4x6**

CNN: Spatial Dimensions

- A closer look at spatial dimensions

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

Input size: **6x6** (spatially)
Filter size: **3x3**
Stride: **1**
Output image size: **4x4**

- **Note that** applying convolution repeatedly will **shrink feature maps spatially**!
- **Shrinking** too **fast** is **not good**, doesn't work well!
- To **preserve size spatially**, it is common to **zero-pad** the border

0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	0
0	0	1	0	0	1	0	0
0	0	0	1	1	0	0	0
0	1	0	0	0	1	0	0
0	0	1	0	0	1	0	0
0	0	0	1	0	1	0	0
0	0	0	0	0	0	0	0

Padding: 1

Output Image Size

$$Width_{out} = \frac{Width_{in} - Width_{filter} + 2 * Padding_{Size}}{Stride} + 1$$

$$Height_{out} = \frac{Height_{in} - Height_{filter} + 2 * Padding_{Size}}{Stride} + 1$$

$$Dimension_{out} = Number_{filter}$$

CNN: Spatial Dimensions

- **For example;**

  Input Image: 32x32x3
  Filter size: 5x5
  Filter number: 10
  Stride: 1
  Padding: 2

Output Image Size

$$Width_{out} = \frac{Width_{in} - Width_{filter} + 2 * Padding_{Size}}{Stride} + 1 = \frac{32 - 5 + 2 * 2}{1} + 1 = \mathbf{32}$$

$$Height_{out} = \frac{Height_{in} - Height_{filter} + 2 * Padding_{Size}}{Stride} + 1 = \frac{32 - 5 + 2 * 2}{1} + 1 = \mathbf{32}$$

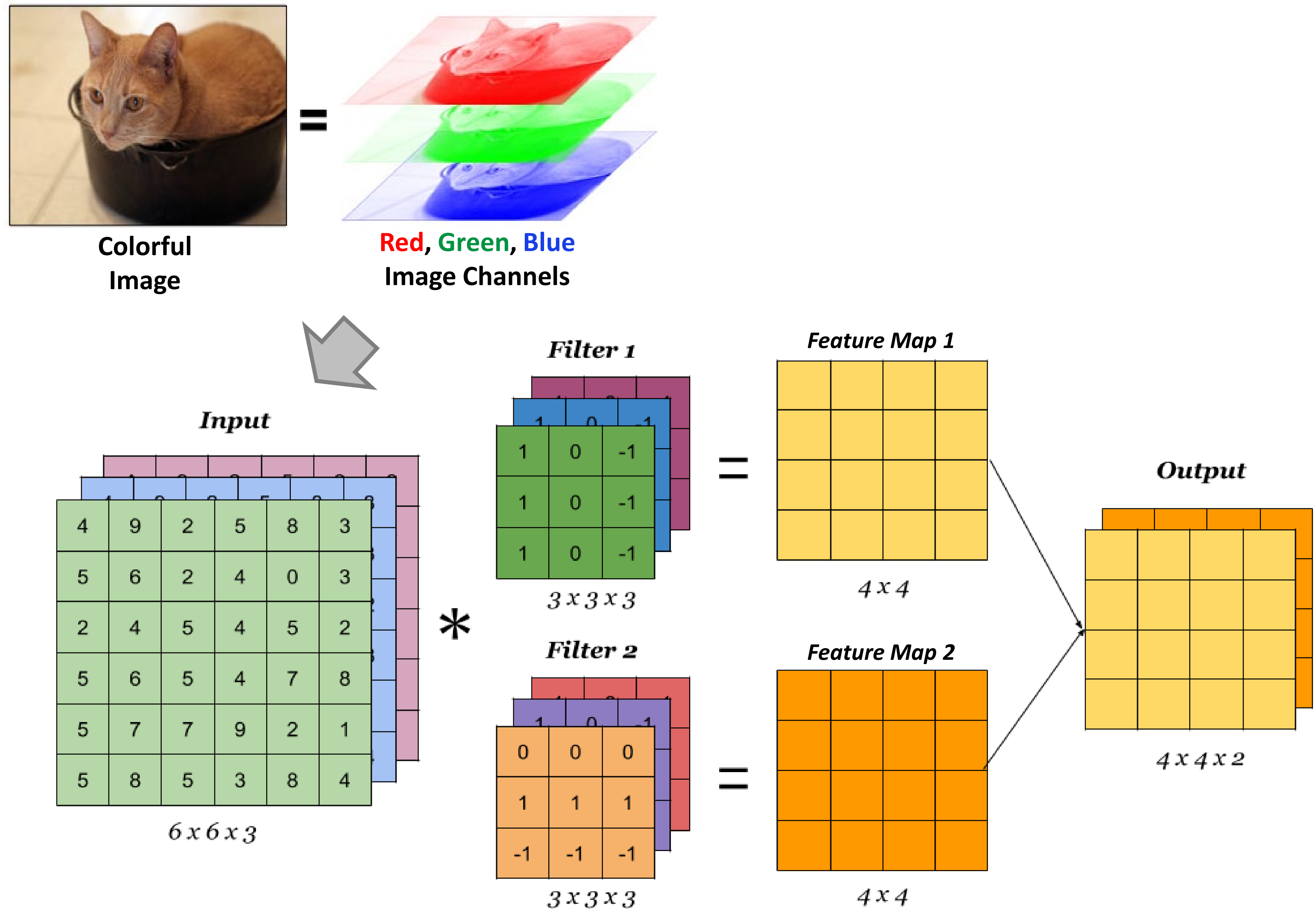
$$Dimension_{out} = Number_{filter} = \mathbf{10}$$

=> Output Image Size: 32x32x10

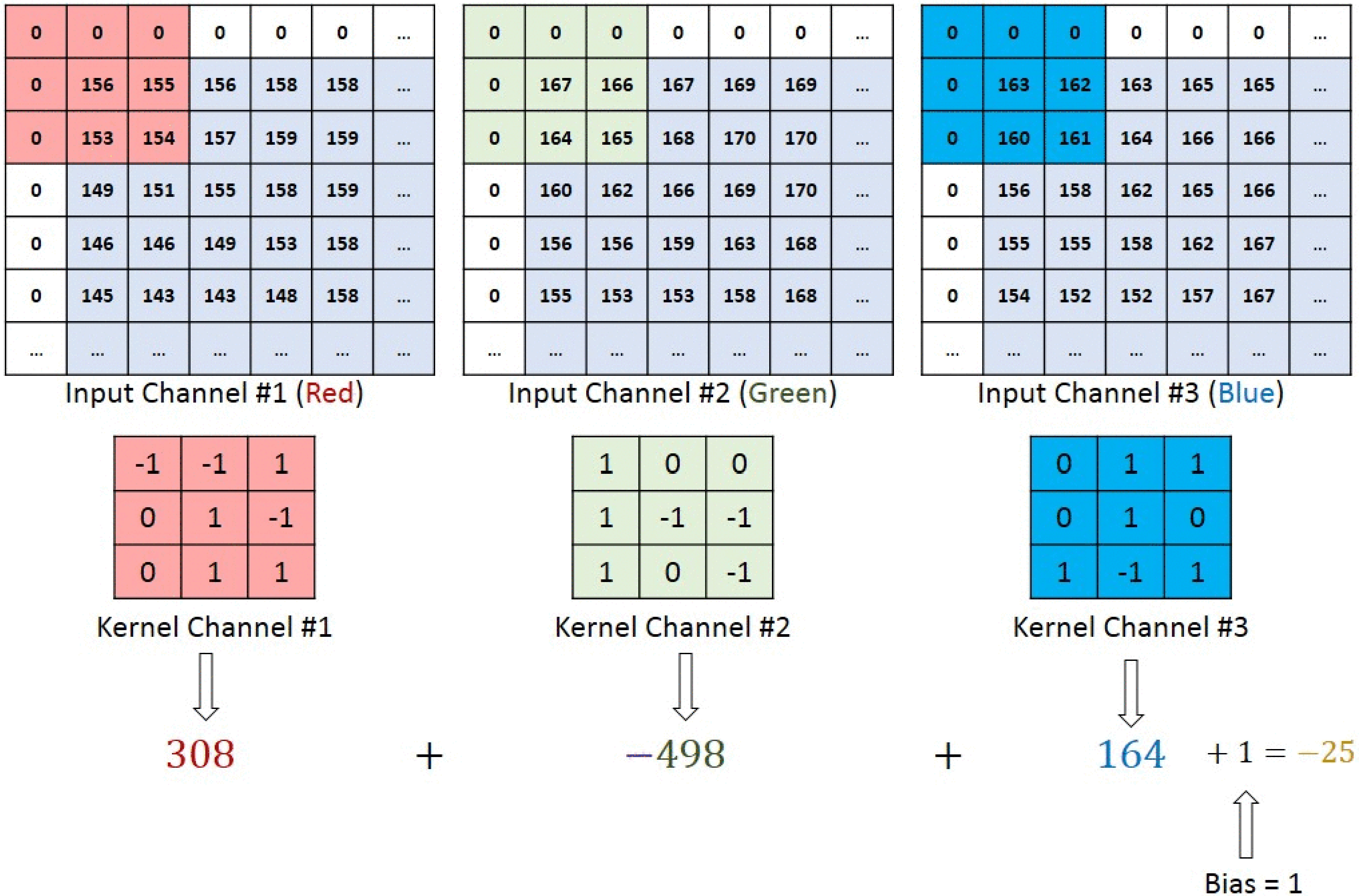
Number of parameters in this layer?

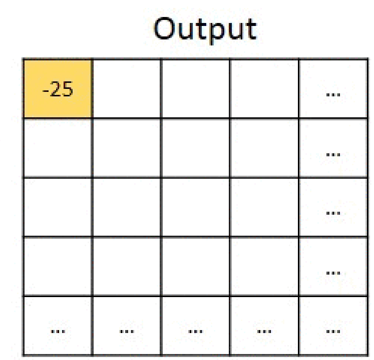
Each filter has 5x5x3 +1 = 76 parameters (+1 for bias)
Since there are 10 filters, in total 76*10 = 760 parameters

CNN: Convolution for Multi-channel Images



CNN: Convolution for Multi-channel Images

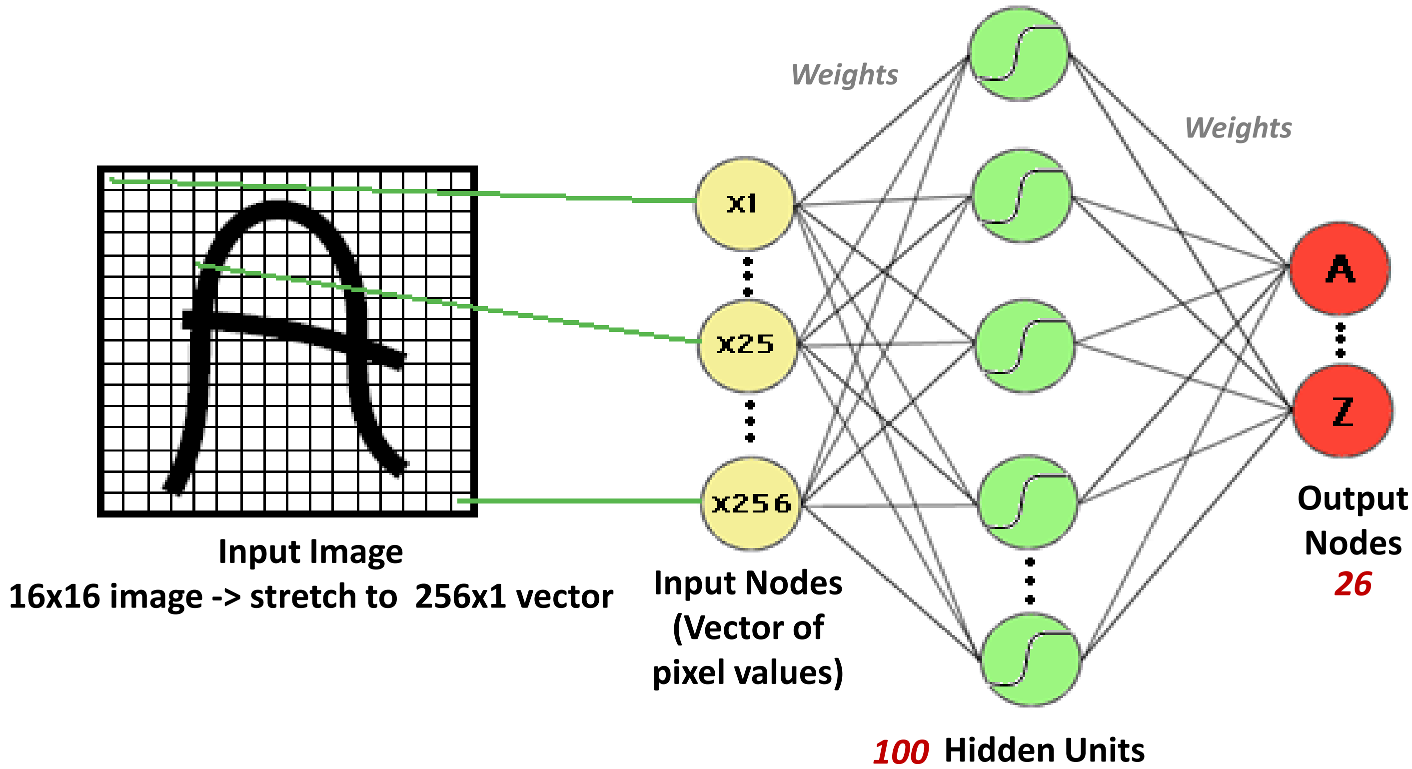




Click here to see the gif animation!

Convolution versus Fully Connected

- In a **fully-connected** mode,
  - a neuron in the hidden layer is connected to all neurons in the input layer
  - **no spatial information** is preserved (i.e. if we shuffle the image pixels, there is no difference!)
  - the number of **trainable parameters** becomes extremely **large**

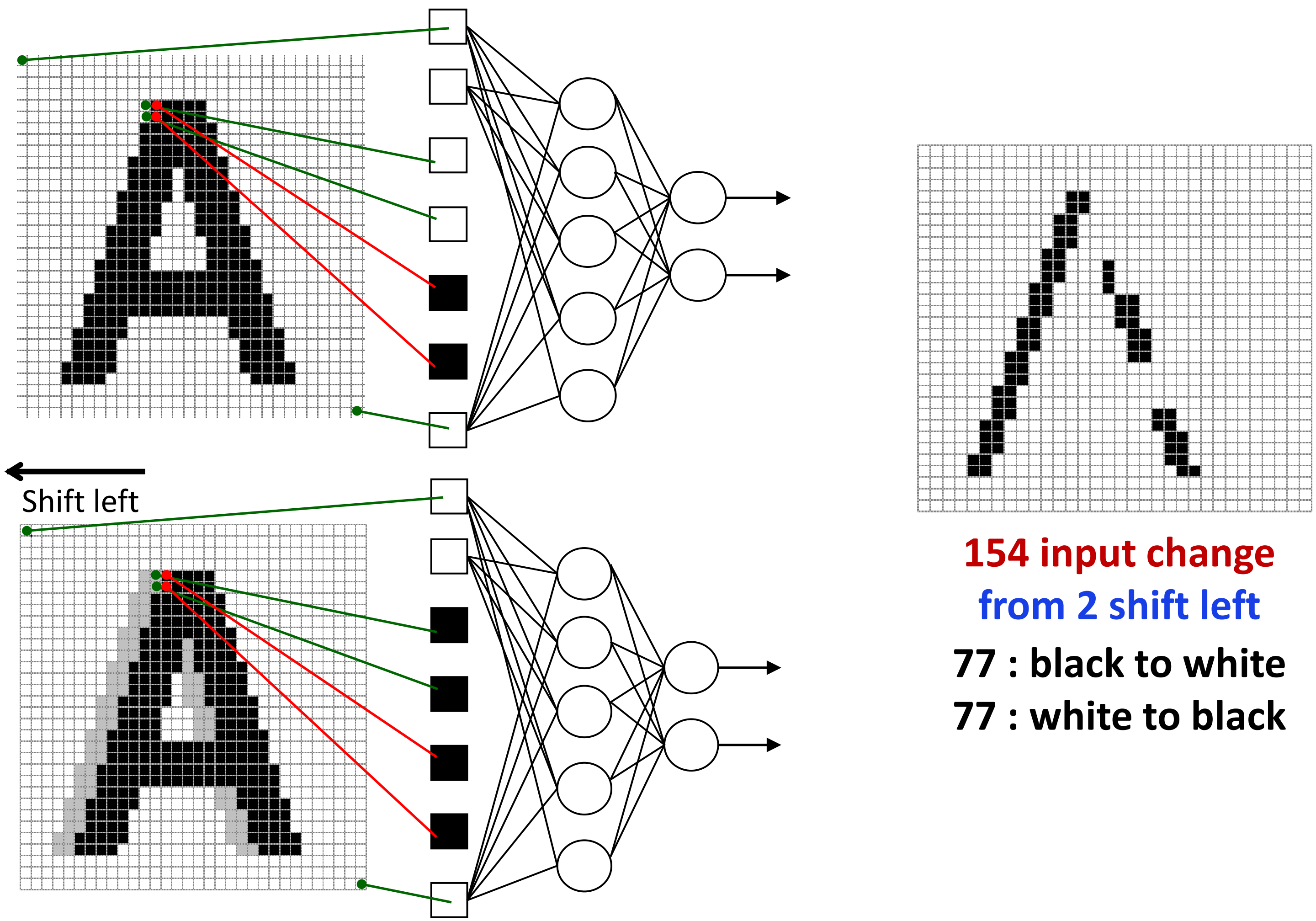


Total trainable parameter number:

(256x100)+ 100 +(26x100) + 26 = 28326

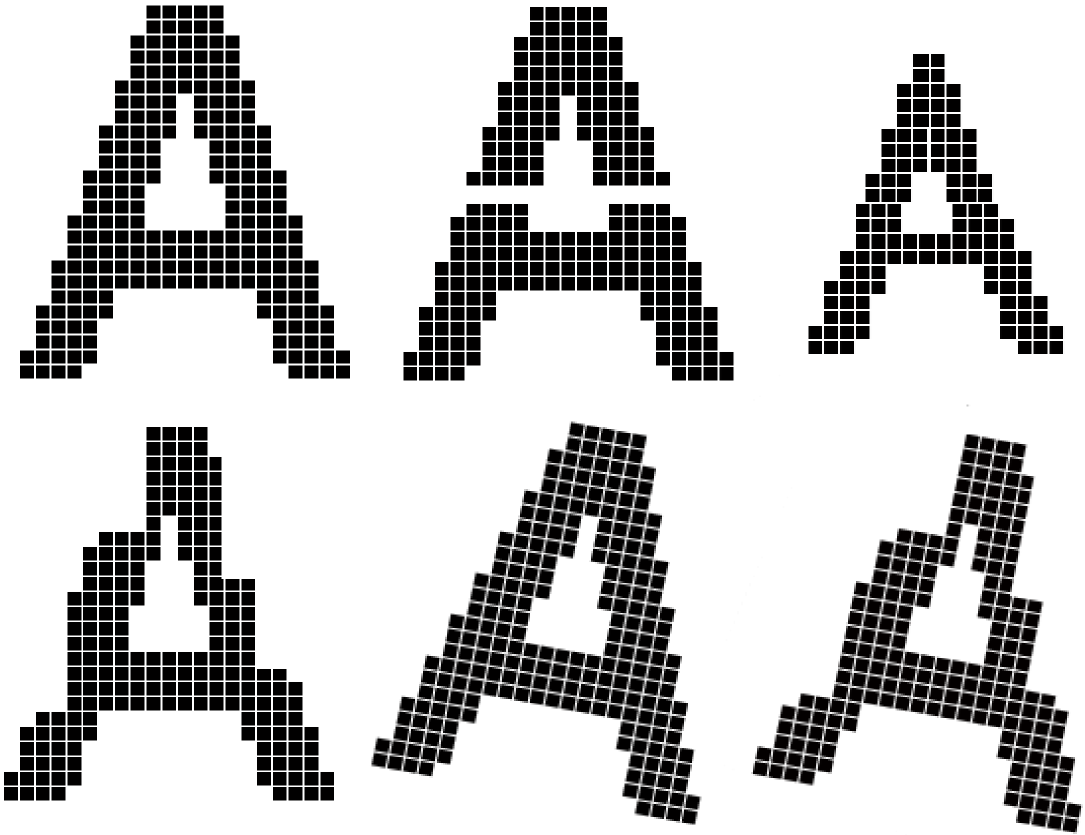
Convolution versus Fully Connected

- In a **fully-connected** mode,
  - the number of **trainable parameters** becomes extremely **large**
  - there is ***little or no invariance*** to shifting, scaling, and other forms of distortion



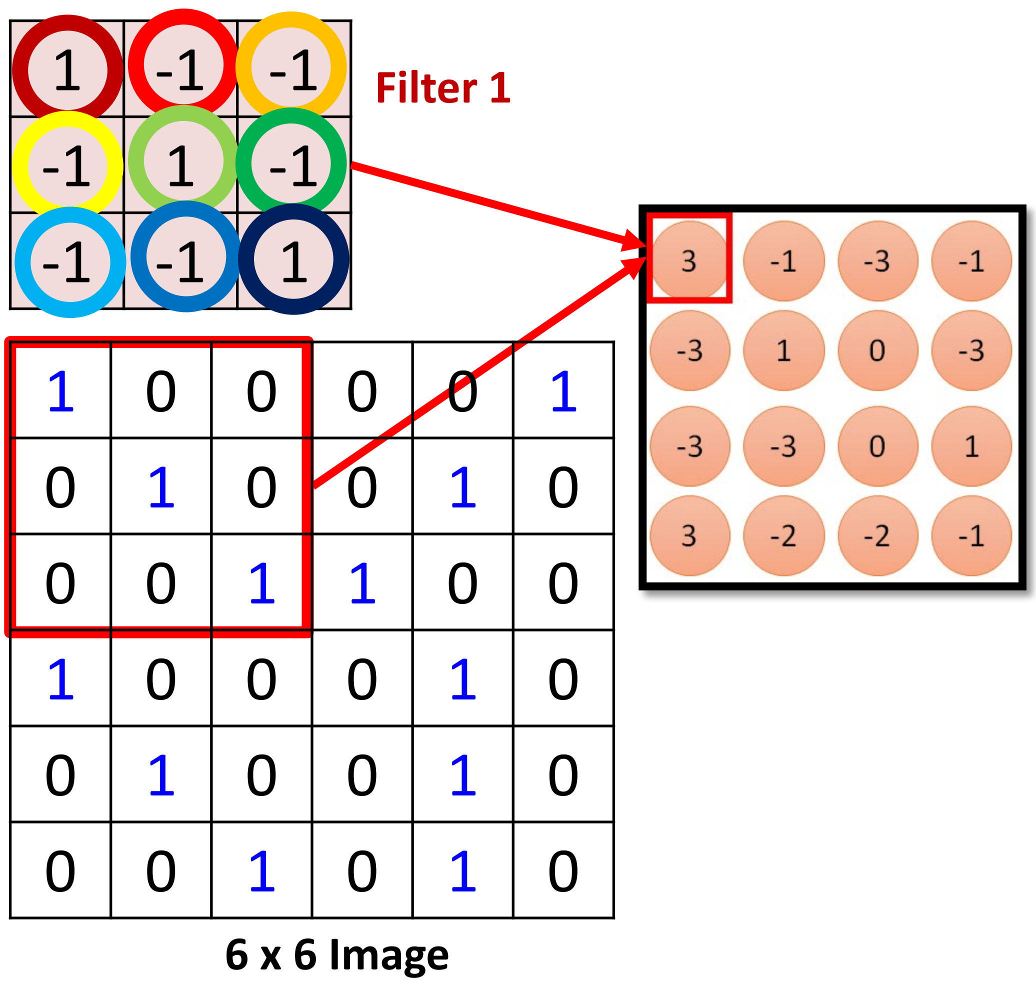
Convolution versus Fully Connected

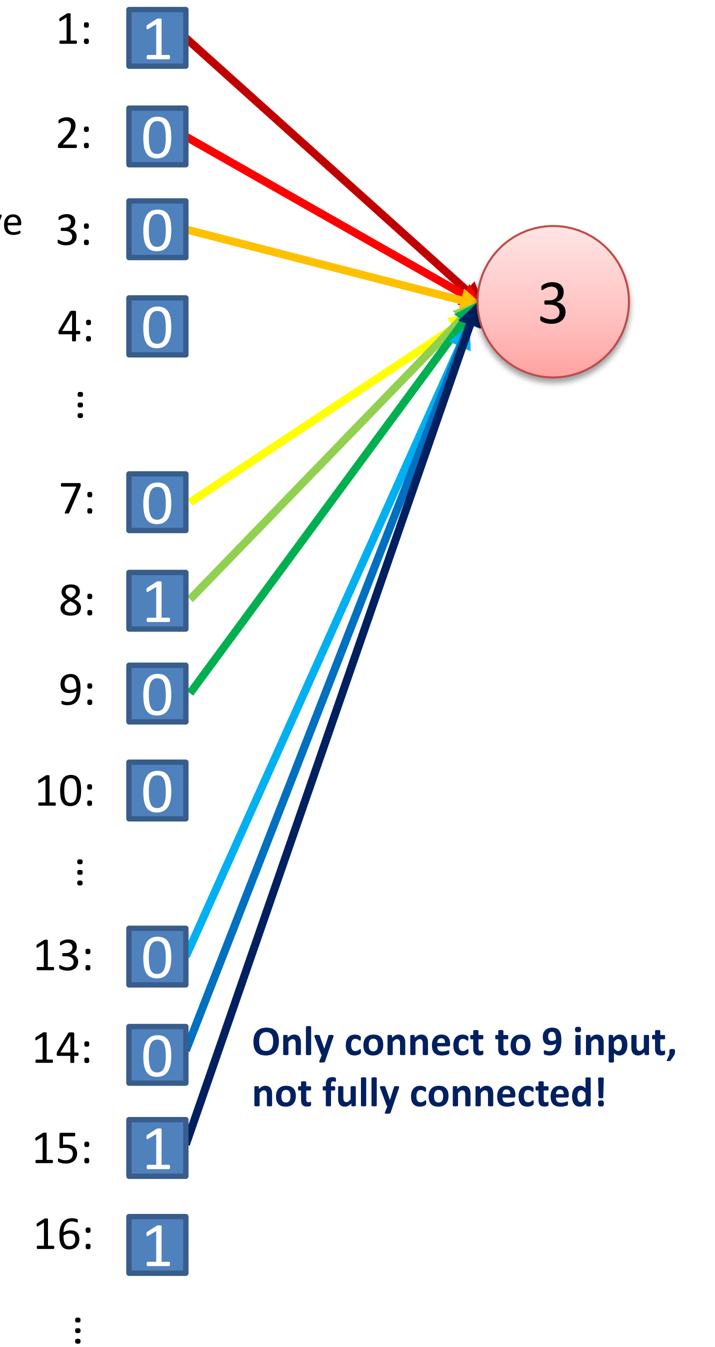
- In a **fully-connected** mode,
  - the number of **trainable parameters** becomes extremely **large**
  - there is ***little or no invariance*** to shifting, scaling, and other forms of distortion
  - the **topology** of the input data is completely **ignored**



Convolution versus Fully Connected

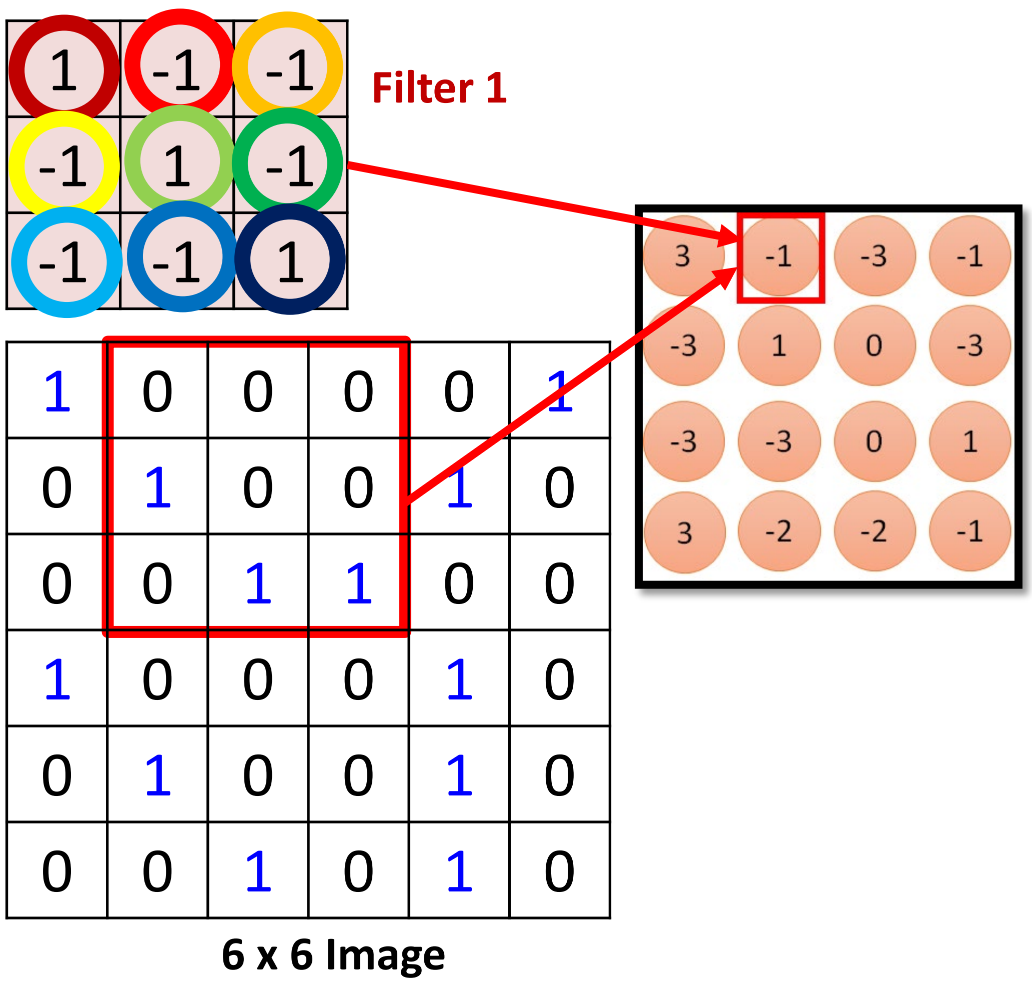
- In a **convolution** mode, spatial structure is preserved. We have
  - **fewer parameters!**
  - **shared weights!**

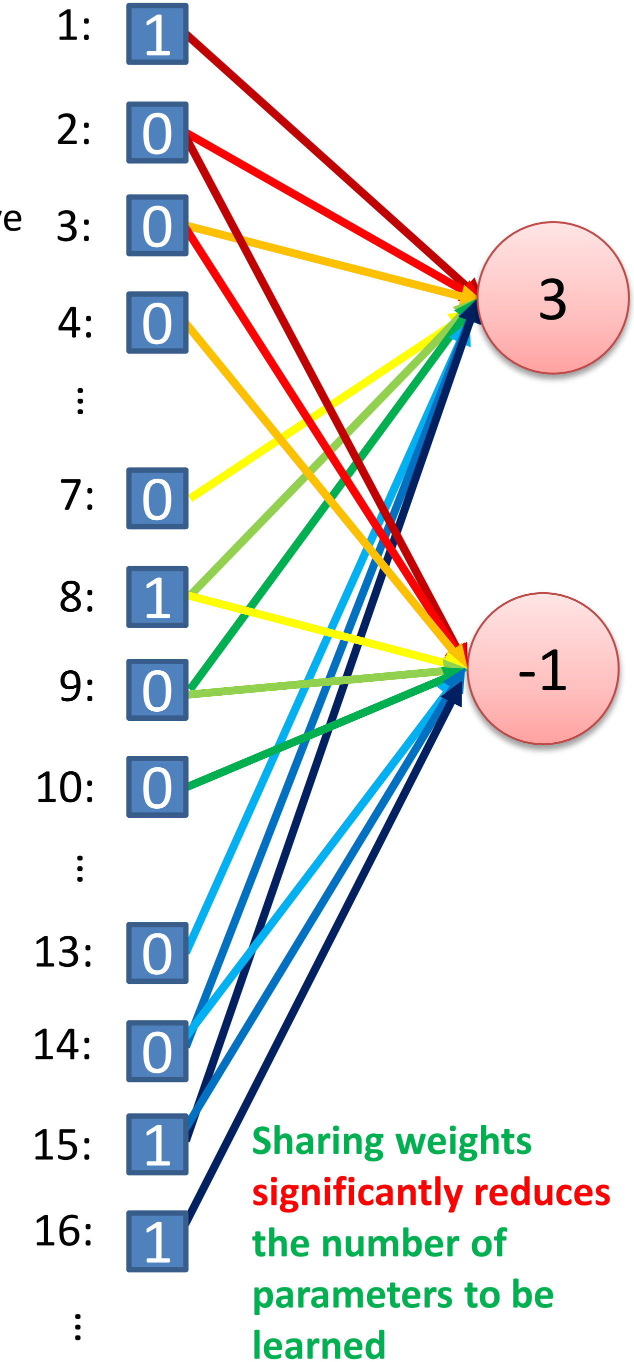




Convolution versus Fully Connected

- In a **convolution** mode, spatial structure is preserved. We have
  - **fewer parameters!**
  - **shared weights!**





CNN: Pooling

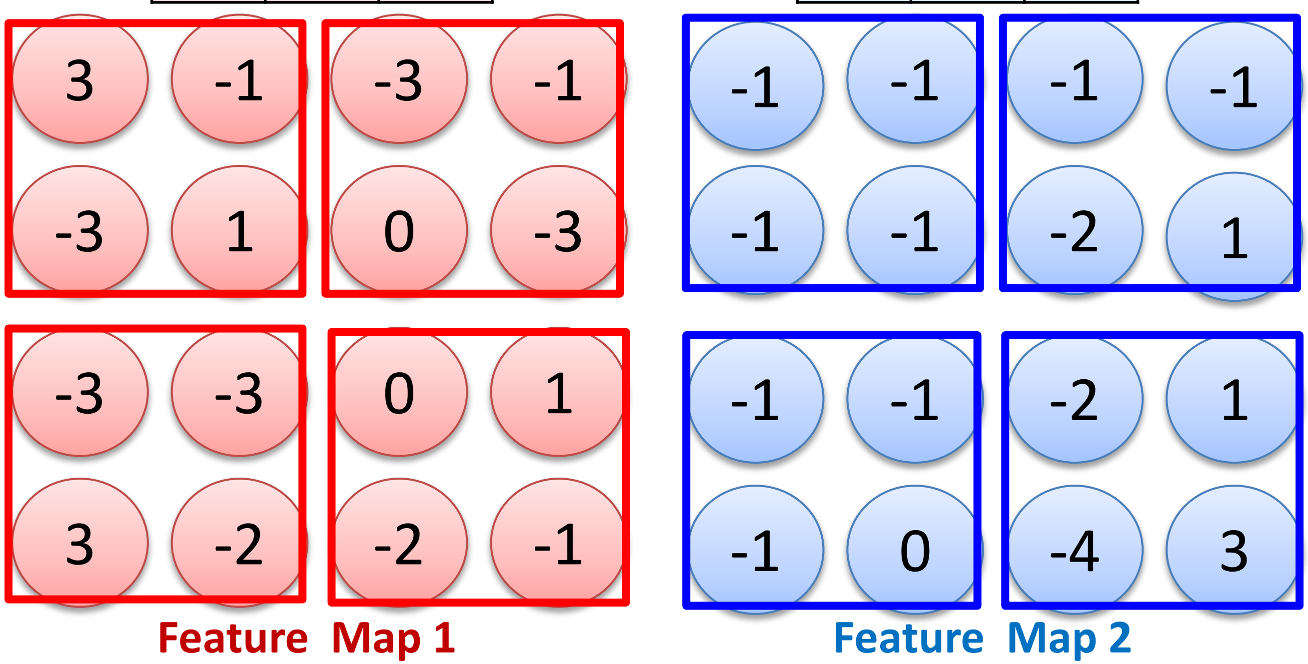
- Pooling layer makes the representation **smaller**, more **manageable,** and **robust to position variations**
- Pooling operates over each feature map (a.k.a. activation map) **independently**
  - **Max-pooling:** takes a max response over spatial locations
  - **Average-pooling:** takes an average response over spatial locations

Filter 1

1	-1	-1
-1	1	-1
-1	-1	1

Filter 2

-1	1	-1
-1	1	-1
-1	1	-1

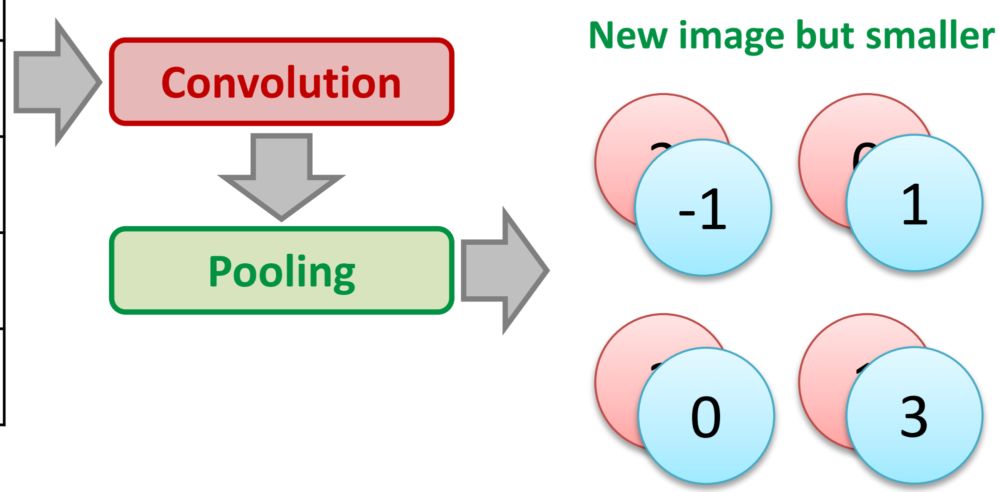


CNN: Pooling

- Pooling layer makes the representation **smaller**, more **manageable,** and **robust to position variations**
- Pooling operates over each feature map (a.k.a. activation map) **independently**
  - **Max-pooling:** takes a max response over spatial locations
  - **Average-pooling:** takes an average response over spatial locations

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



2 x 2 Image

Each filter is a channel

CNN: Flattening

Can repeat many times

Convolution

Pooling

New image but smaller

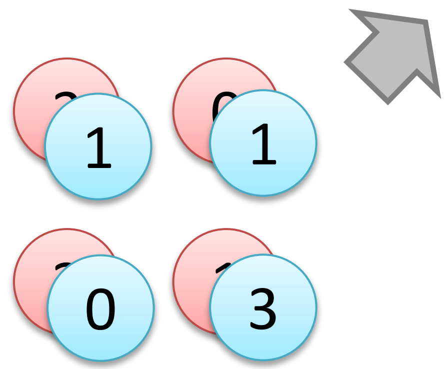
Convolution

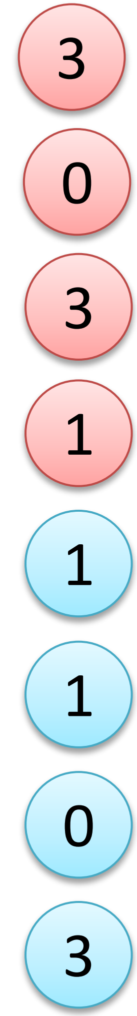
Pooling

New image but smaller

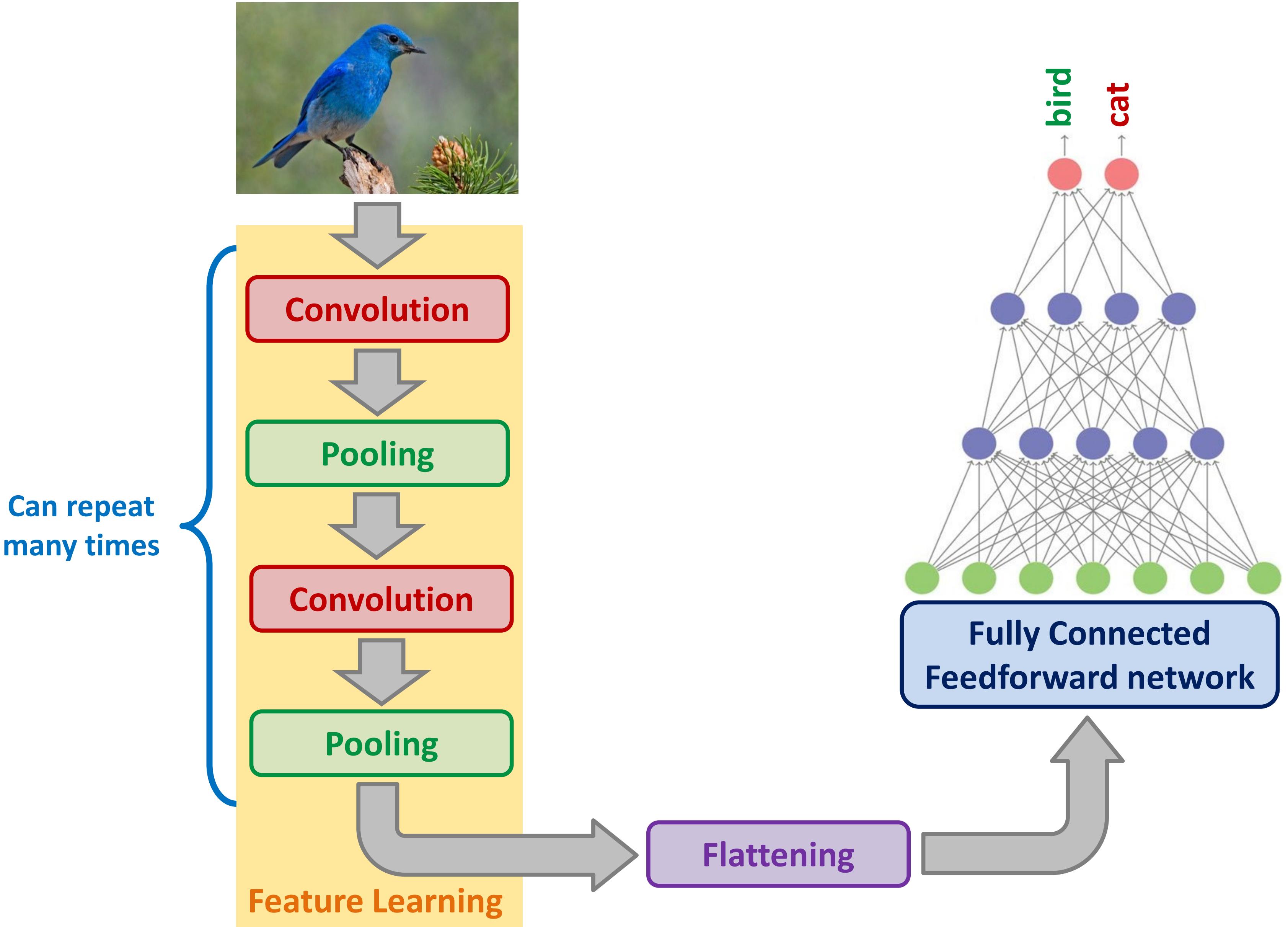
Flattening

Flattening is the process of converting all feature maps into a single long continuous linear vector.



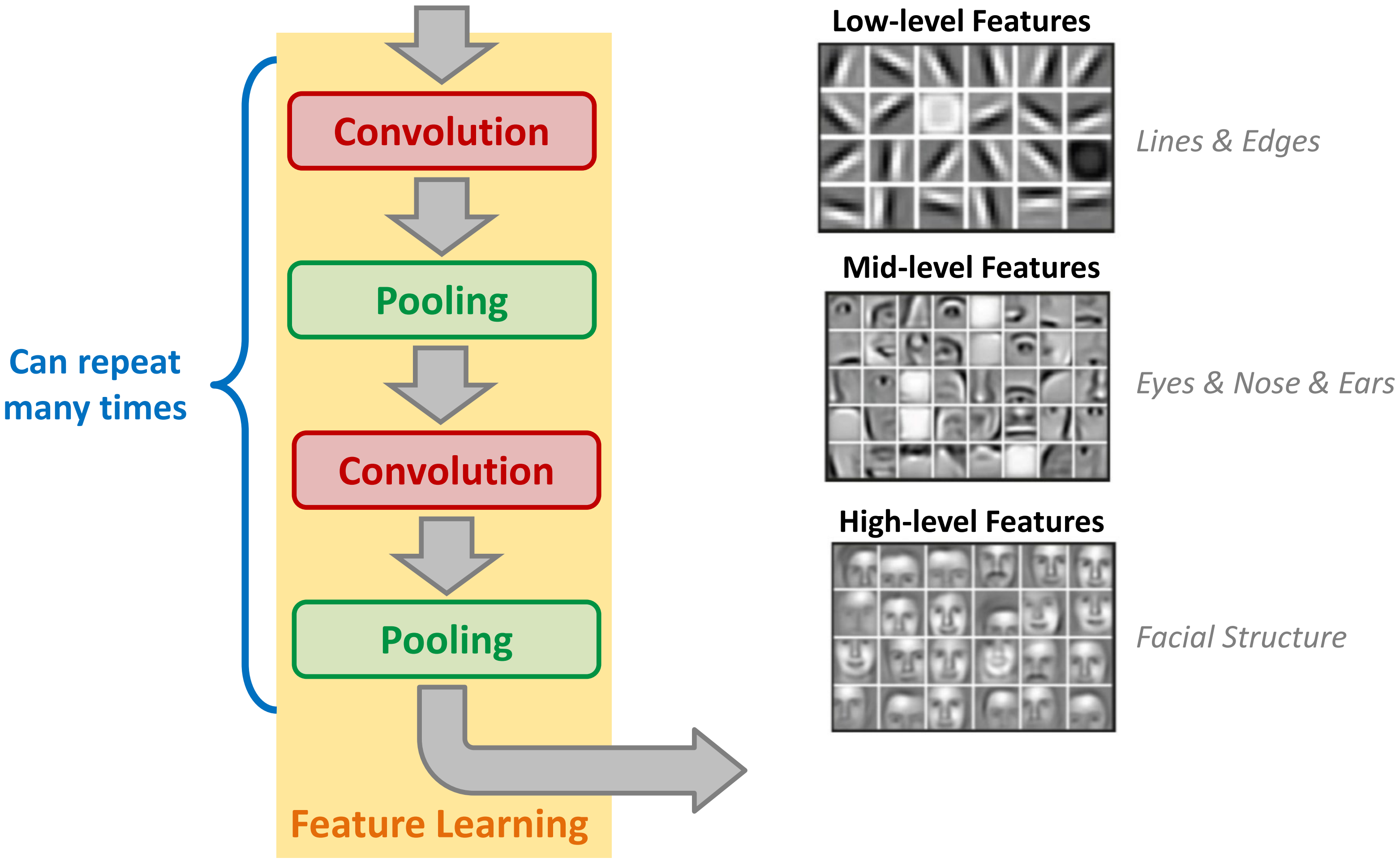


The Whole CNN



The Whole CNN

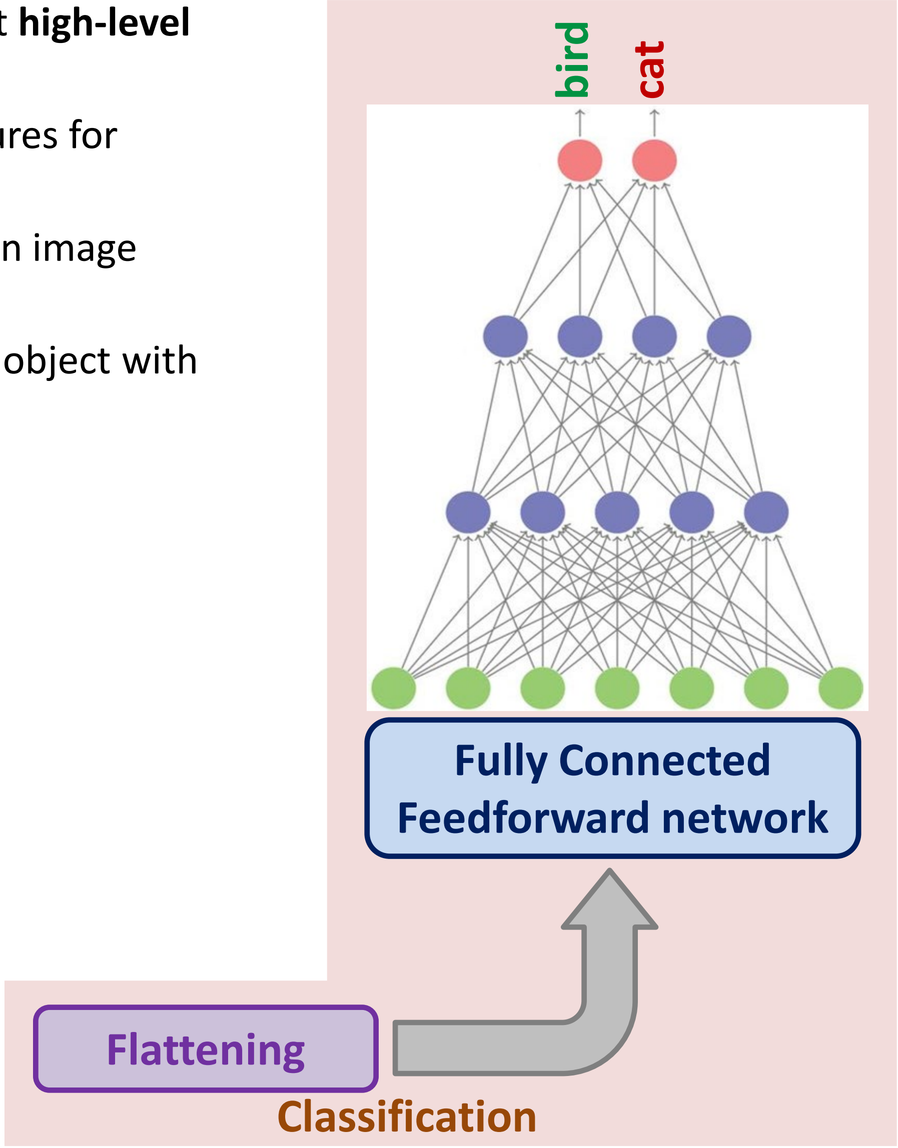
- **Feature Learning**
  - Learn a hierarchy of **features** in input image through **convolution**
  - Introduce **non-linearity** through activation function (**Conv + ReLU**)
  - Reduce dimensionality and preserve spatial invariance with **pooling**



The Whole CNN

- **Classification**
  - Convolution and Pooling layers output **high-level features** of the input
  - Fully connected layer uses these features for **classifying** input image
  - Express output as the **probability** of an image belonging to a particular class
  - Apply **Softmax function** to classify an object with probabilistic values between **0 and 1**

$$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)}$$



The Whole CNN in PyTorch

1: Black & White
3: RGB

Input Image = 28 x 28 x 3
(Width x Height x Channel)

Convolution
↓
Pooling
↓
Convolution
↓
Pooling

```
layers = []
```

```
layers.append(Conv2d(3,25, kernel_size=(3,3), padding=0)))
layers.append(Relu())
```

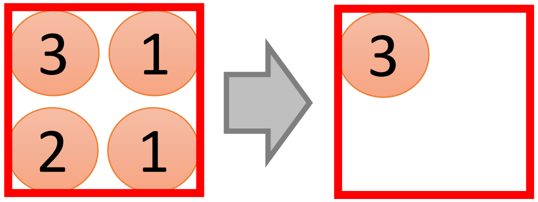
1	-1	-1
-1		
-1		

-1	1	-1
-1	1	-1
-1	1	-1

……

There are 25 3x3 filters.

```
layers.add(MaxPool2d(kernel_size=(2,2)))
```



The Whole CNN in PyTorch

28 x 28 x 3

```
layers = []
```

Convolution

```
layers.append(Conv2d(3,25, kernel_size=(3,3), padding=0)))
layers.append(Relu())
```

26 x 26 x 25

How many parameters for each filter? **28** (3x3x3+1)

Pooling

```
layers.add(MaxPool2d(kernel_size=(2,2)))
```

13 x 13 x 25

How many parameters for each pooling? **None!**

Convolution

```
layers.append(Conv2d(25,50, kernel_size=(3,3), padding=0)))
layers.append(Relu())
```

11 x 11 x 50

How many parameters for each filter? **226** (3x3x25+1)

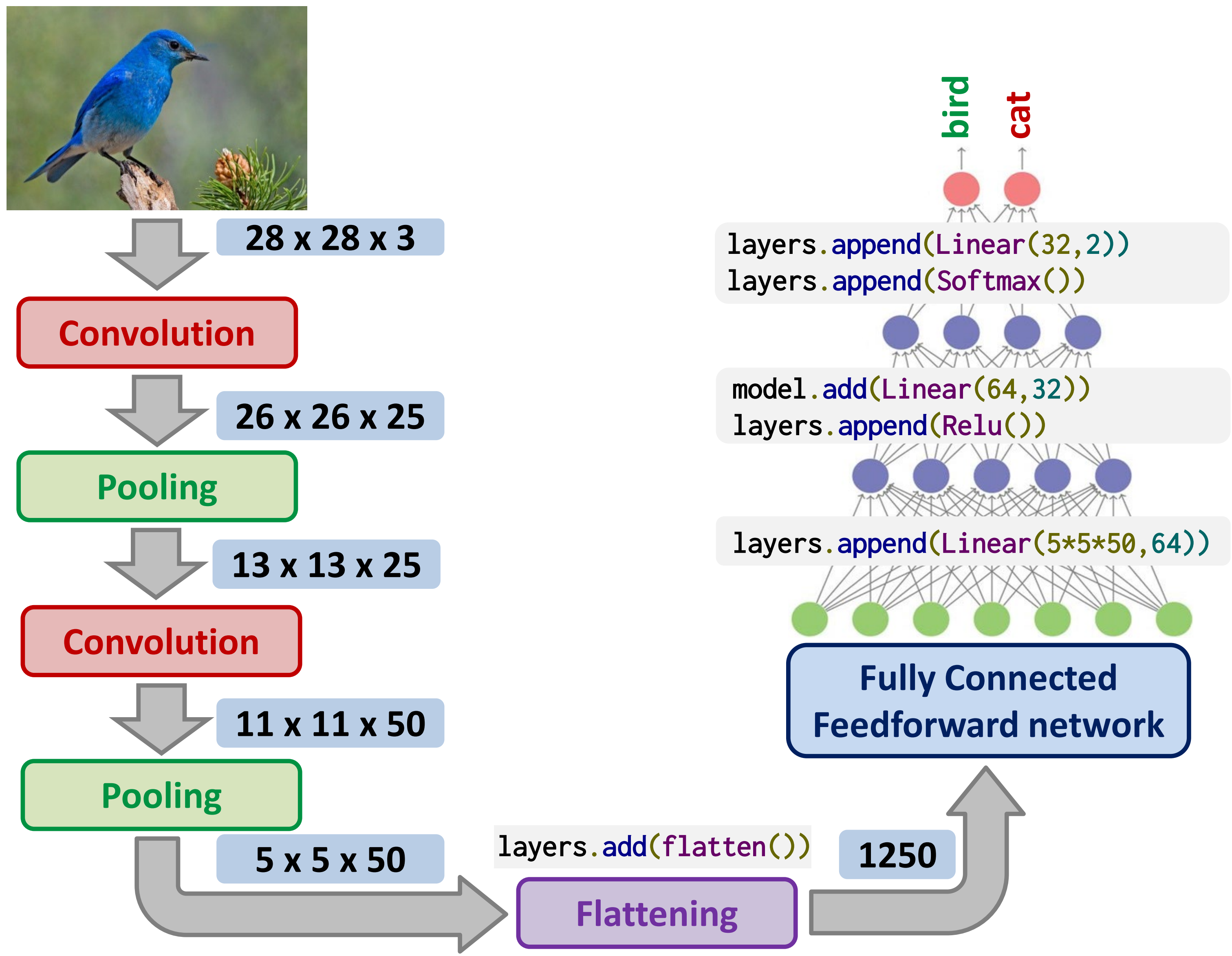
Pooling

```
layers.add(MaxPool2d(kernel_size=(2,2)))
```

5 x 5 x 50

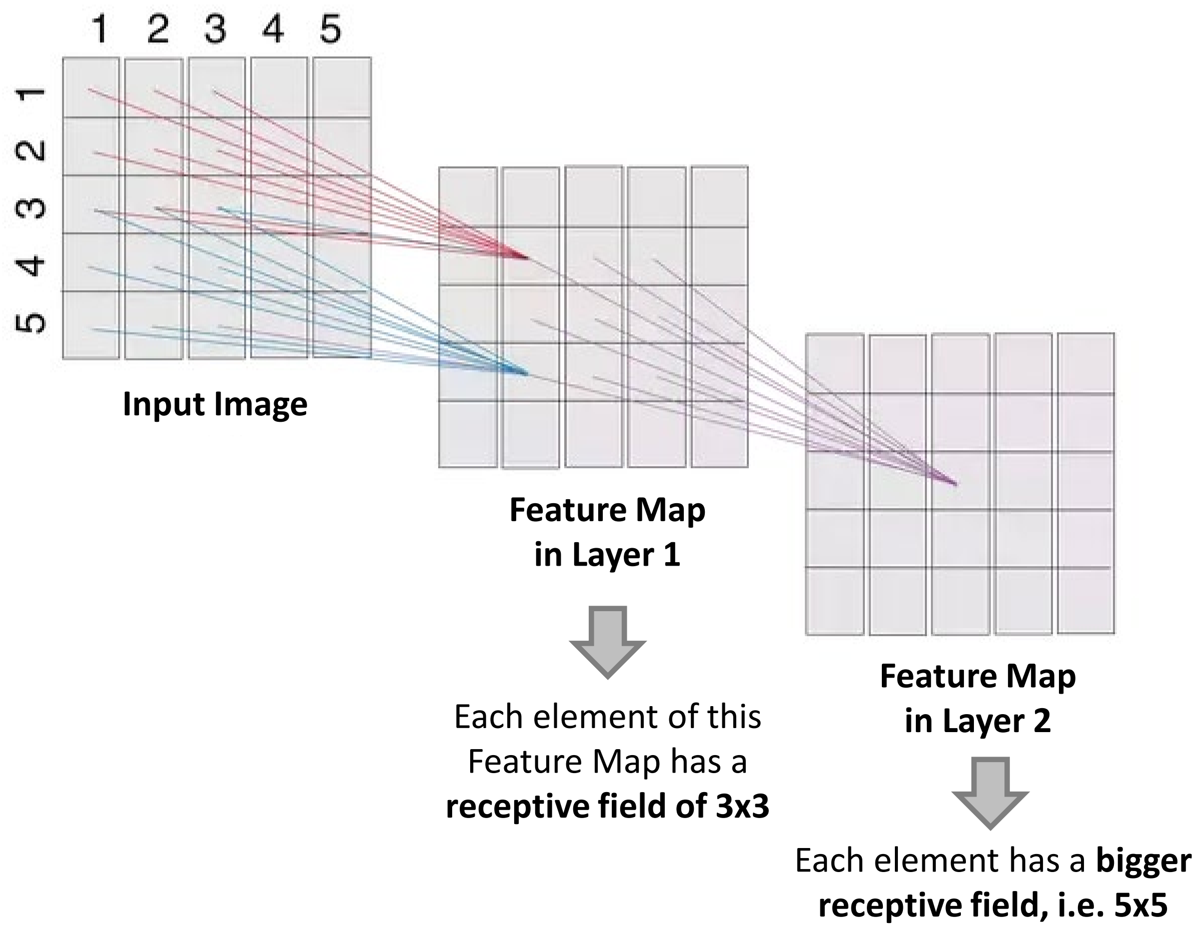
How many parameters for each pooling? **None!**

The Whole CNN in PyTorch



CNN: Receptive Fields

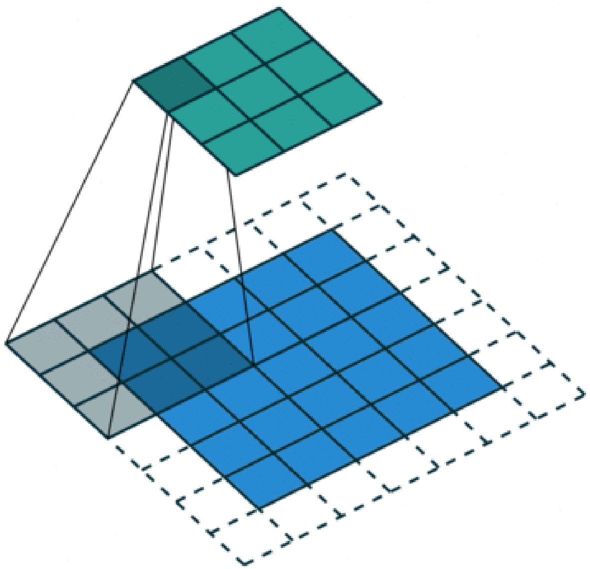
- The **receptive field** is defined as the region in the input space that a particular CNN's feature is looking at (i.e. be affected by)
- With the network going **deeper**, the size of the **receptive field** is becoming **larger**.



The **receptive field** of vanilla convolutions **grows linearly** with the number of layers (**3×3, 5×5, 7×7**, etc.).

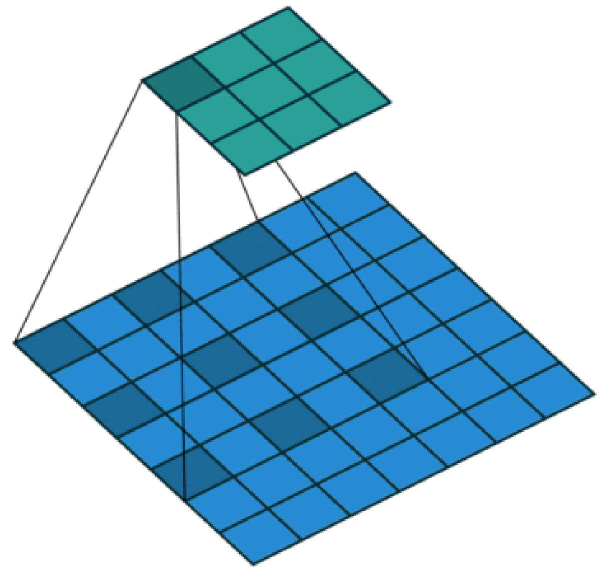
CNN: Dilated Convolutions

- Vanilla convolutions struggle to integrate global context since the effective receptive field of units can only grow linearly with layers.
- The dilation operation supports the exponential expansion of the receptive field without loss of resolution or coverage.
- The receptive field **grows exponentially** while the number of parameters **grows linearly**.
- Dilated convolutions have **gridding artifacts**
  - Neighboring output units are computed from completely separated sets of input units.



Standard convolution

(the receptive field is 5x5)

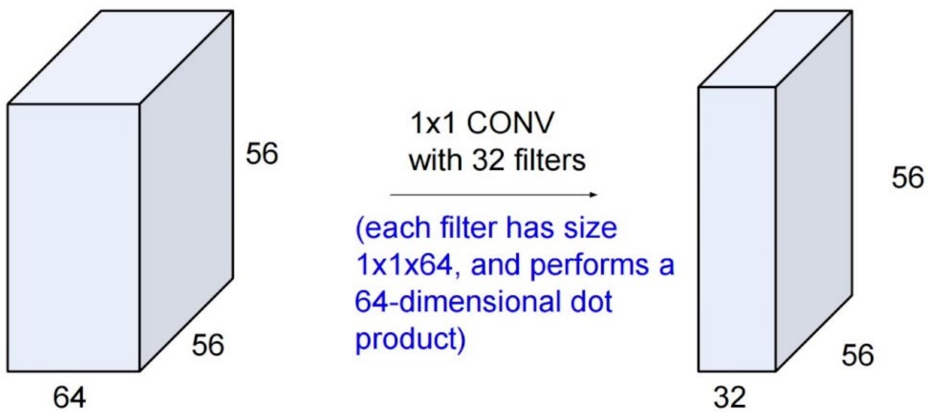


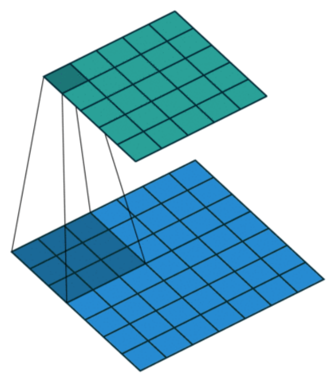
Dilated convolution

(the receptive field is larger, i.e. 7x7)

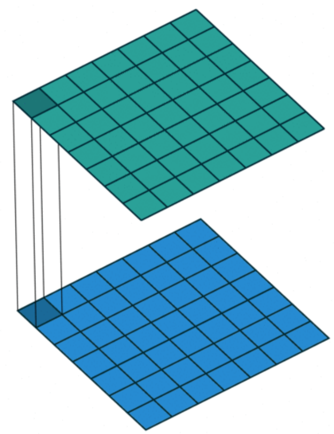
CNN: 1x1 Convolution

- 1x1 convolution leads to an **increase/decrease in the dimension** of the number of feature maps
- Useful in semantic segmentation to merge different layers



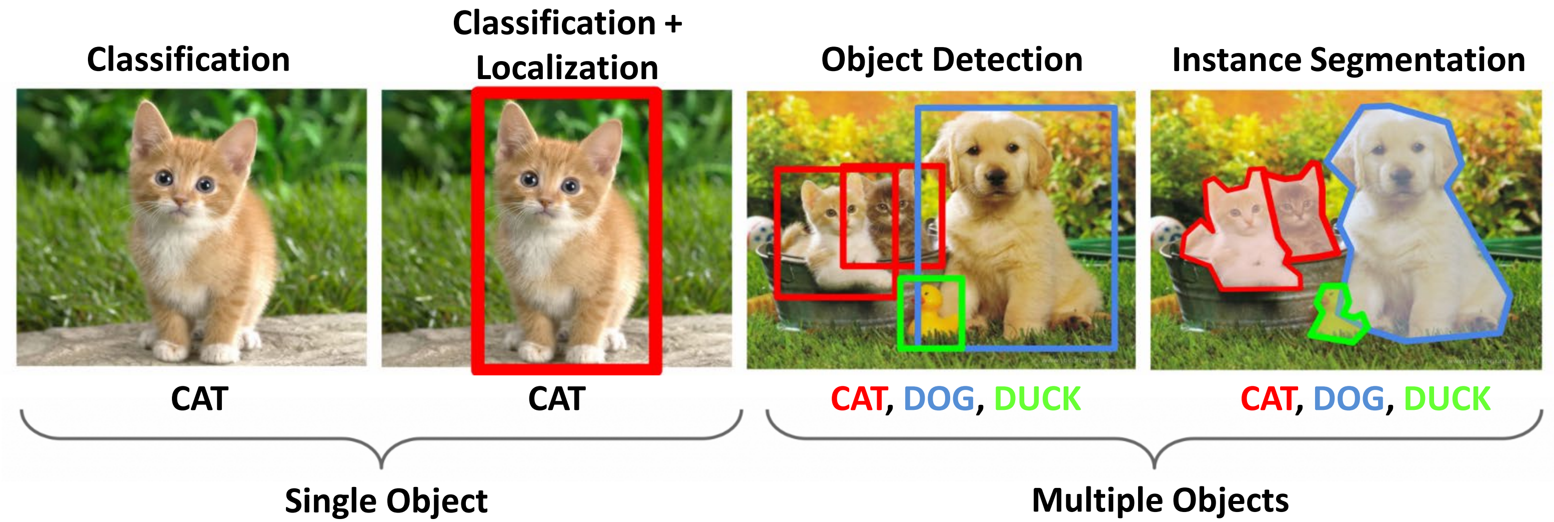


Convolution with kernel of size 3x3

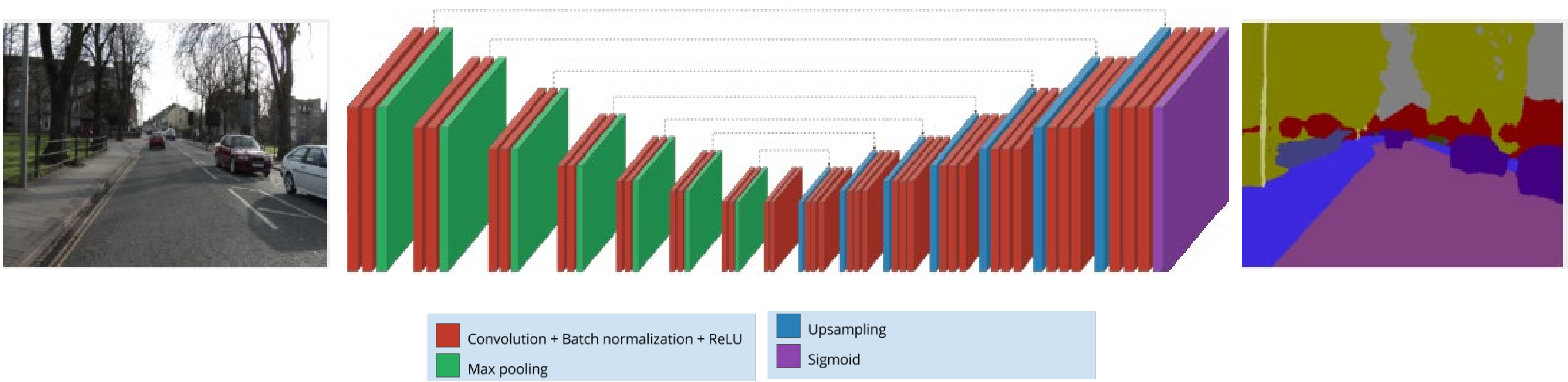


Convolution with kernel of size 1x1

Computer Vision Tasks for CNNs



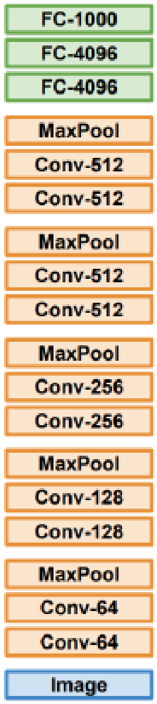
- **Semantic Image Segmentation** with CNNs
  - Convolutional Encoder-Decoder structure
  - Not instance-aware
  - Skip connections



Transfer Learning with CNNs

- **Common "Wisdom":** You need a lot of data to train a CNN
- **Solution**: **Transfer learning**
  - taking a model trained on a similar task that has lots of data and adopting this model to the task that may not have enough training data
  - This strategy is Pervasive!

Train on **ImageNet**



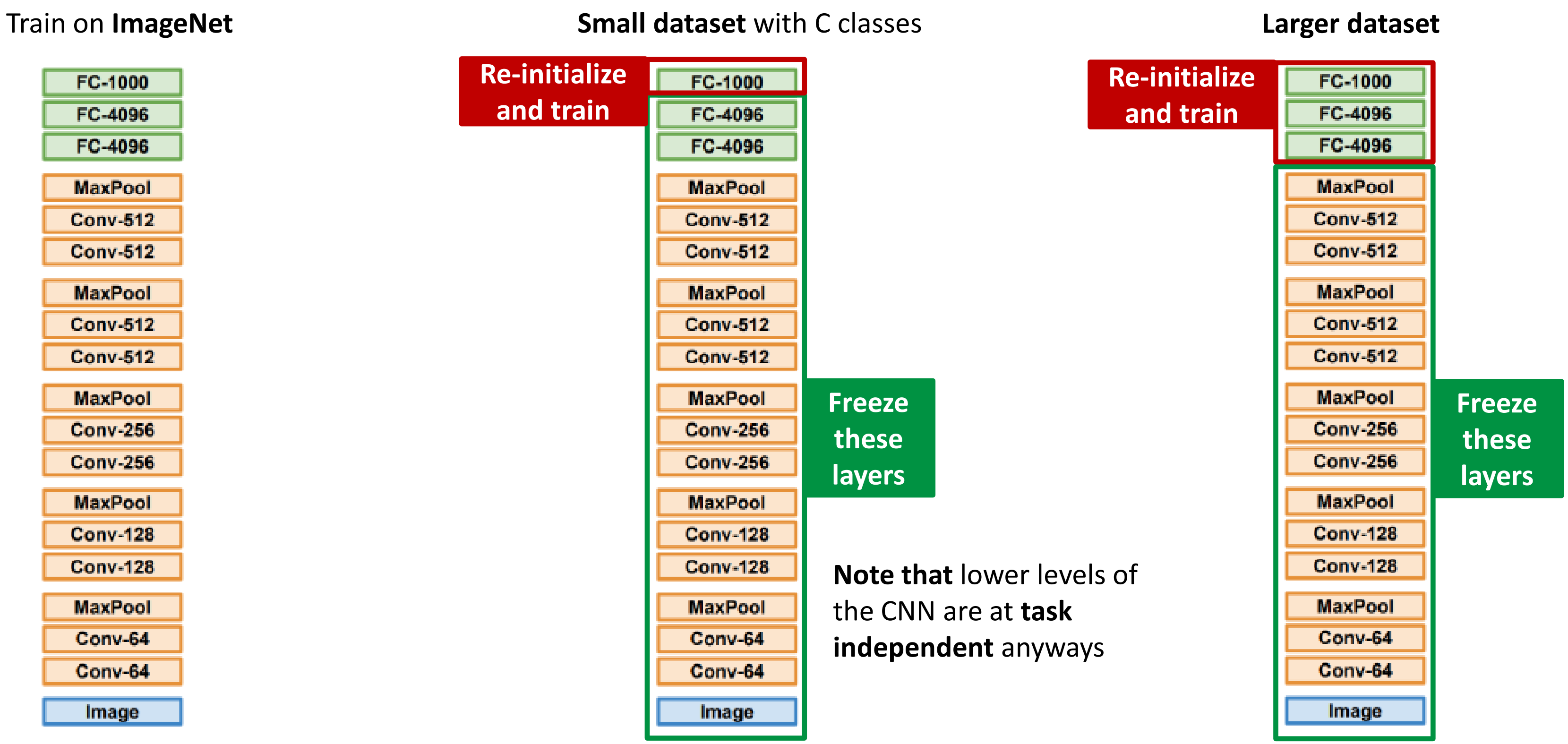
Why on **ImageNet**?

- Convenience, lots of **data**
- We know how to **train these well**

Note that, for some other tasks we would need to start with something else (e.g., videos for optical flow)

Transfer Learning with CNNs

- **Common "Wisdom":** You need a lot of data to train a CNN
- **Solution**: **Transfer learning**
  - taking a model trained on a similar task that has lots of data and adopting this model to the task that may not have enough training data
  - This strategy is Pervasive!



Conclusion

- A Convolutional Neural Network (CNN) is a special kind of **multi-layer feed-forward neural network** that can extract topological properties from an image.

- CNNs are **neurobiologically** motivated by the findings of locally sensitive and orientation-selective nerve cells in the visual cortex.

- CNNs can **recognize visual patterns** with extreme variability.

- If a **neuron** in the feature map **fires**, this corresponds to a **match with the template**.

- **Filter indices** in CNNs **are learned** directly by the network.

- Pooling (**subsampling**) layers reduce the spatial resolution of each feature map
  - By reducing the **spatial resolution** of the feature map, a **certain degree** of **shift** and **distortion invariance** is achieved.

- Like almost every other neural network, CNNs **are trained** with a version of the **back-propagation algorithm**.

Discussion

- Which of the following is the correct order for the typical CNN operation?
  - Convolution -> max pooling -> flattening -> full connection
  - Max pooling -> convolution -> flattening -> full connection
  - Flattening -> max pooling -> convolution -> full connection
  - None

- What are the benefits of using convolutional layers instead of having fully connected ones for image processing tasks?
  - Convolutional layers use
    - spatial context by only assigning weights to nearby pixels
    - translation invariance
    - lot fewer parameters due to weight sharing.

- What is the resulting image size if we apply a 3x3 convolution to an image with a size of 150x150? (no padding!)
  - 148x148

Discussion

- What is the resulting image size if we apply a pooling operation (with the size of 2x2) to an image with a size of 150x150?
  - 75x75

- Assume that we have an input tensor with the size of 128x128x16. How many parameters would a single 1x1 convolutional filter have, including the bias?
  - 16 (depth channel only) + 1 (bias) = 17

- Assume that we have a binary classification task of categorizing images as *dog* or *not dog.* We implement a CNN with a single output neuron. Let's assume that the output of this neuron is z. The final output of our network, $\hat{y}$ is then given by:

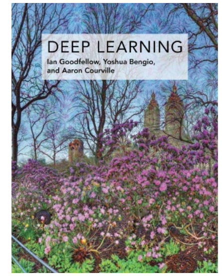
$$\boldsymbol{\hat{y} = sigmoid(ReLU(z))}$$

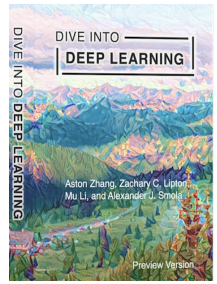
  We also have a threshold of 0.5 to classify all images as a dog, if their final value is $\hat{y} \geq 0.5$. What problem are we going to face in this case?

  - Since we first use ReLU and then sigmoid all the predictions will be positive. Note that ReLU response will be ***always*** $\geq \boldsymbol{0}$. When you pass this to Sigmoid ($\boldsymbol{S(x)} = \frac{\boldsymbol{1}}{\boldsymbol{1+e^{-x}}}$), the output will ***always be*** $\geq \boldsymbol{0.5}$. Therefore, $\boldsymbol{sigmoid(ReLU(z)) \geq 0.5\ \forall z}$.

Reading Material

- Deep Learning Book
  - Chapter [9.1, 9.2, 9.3, 9.4]

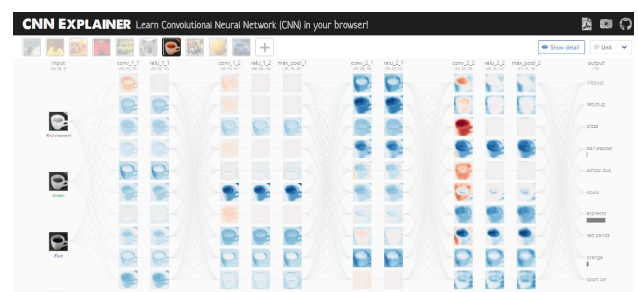




- Dive into Deep Learning Book
  - Chapter 7

- Video
  - Please check this video explaining how CNN features are learned in detail
    https://www.youtube.com/watch?v=N6wn8zMRlVE

- Blogs
  - https://medium.com/@subham.tiwari186/understanding-convolutional-neural-networks-cnns-72b62b17dd3a



- CNN Explainer:
  - https://poloclub.github.io/cnn-explainer/