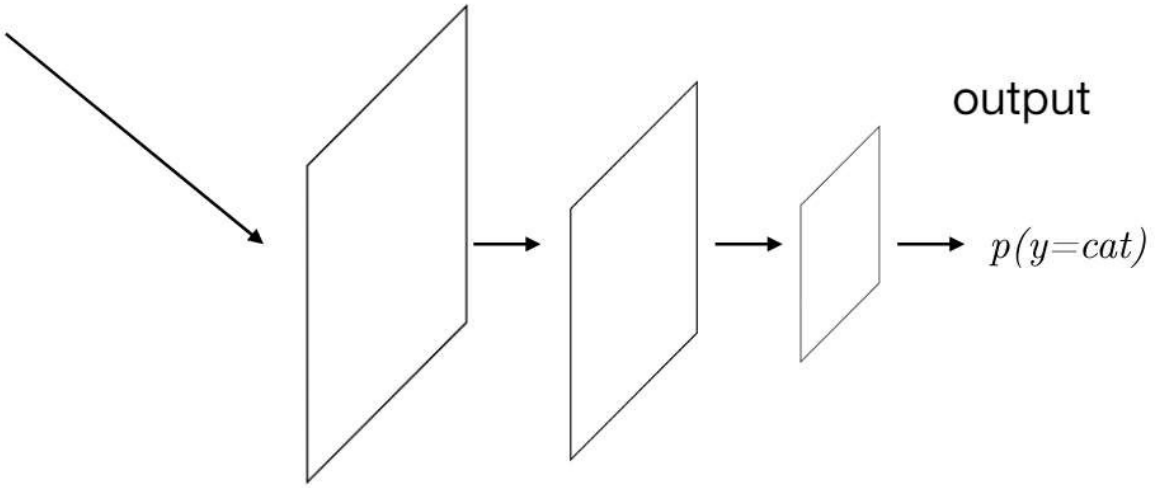
CNN

CNN for image classification

Image Source:
twitter.com%2Fcats&psig=AOvVaw30_o-PCM-K21DiMAJQimQ4&ust=1553887775741551

Image Source: https://www.pinterest.com/pin/244742560974520446



Object detection

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 779-788).

Image segmentation

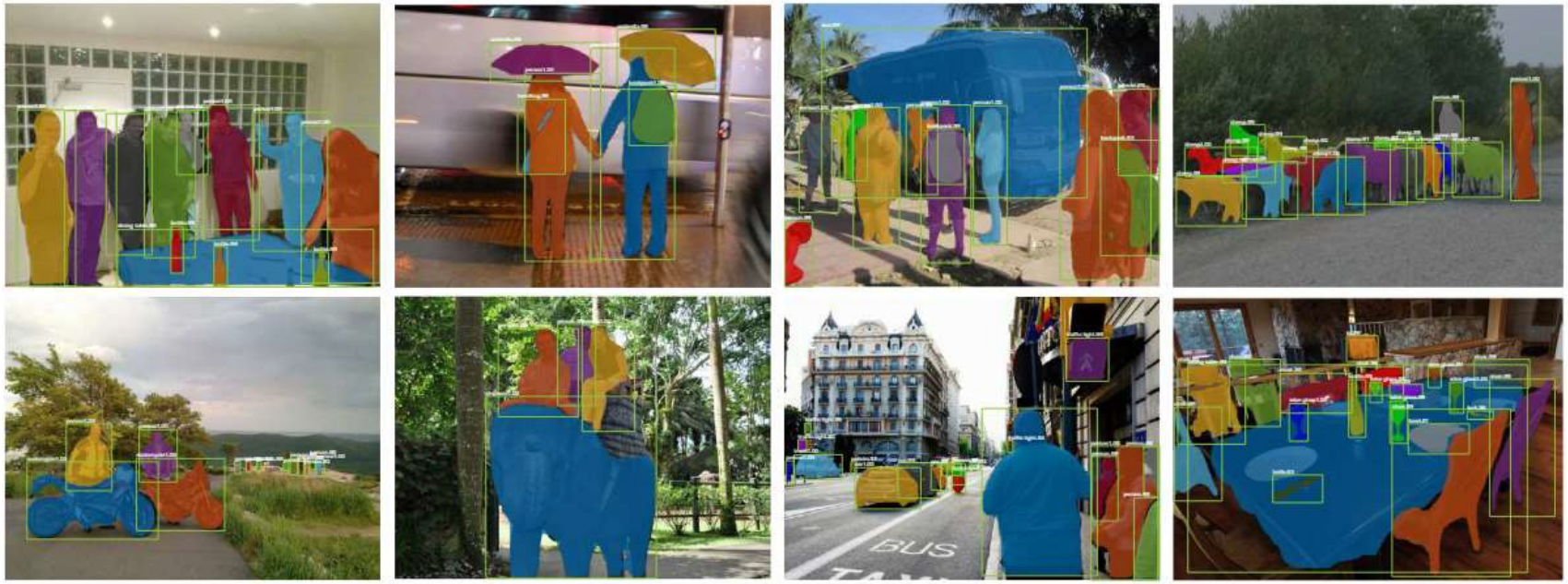
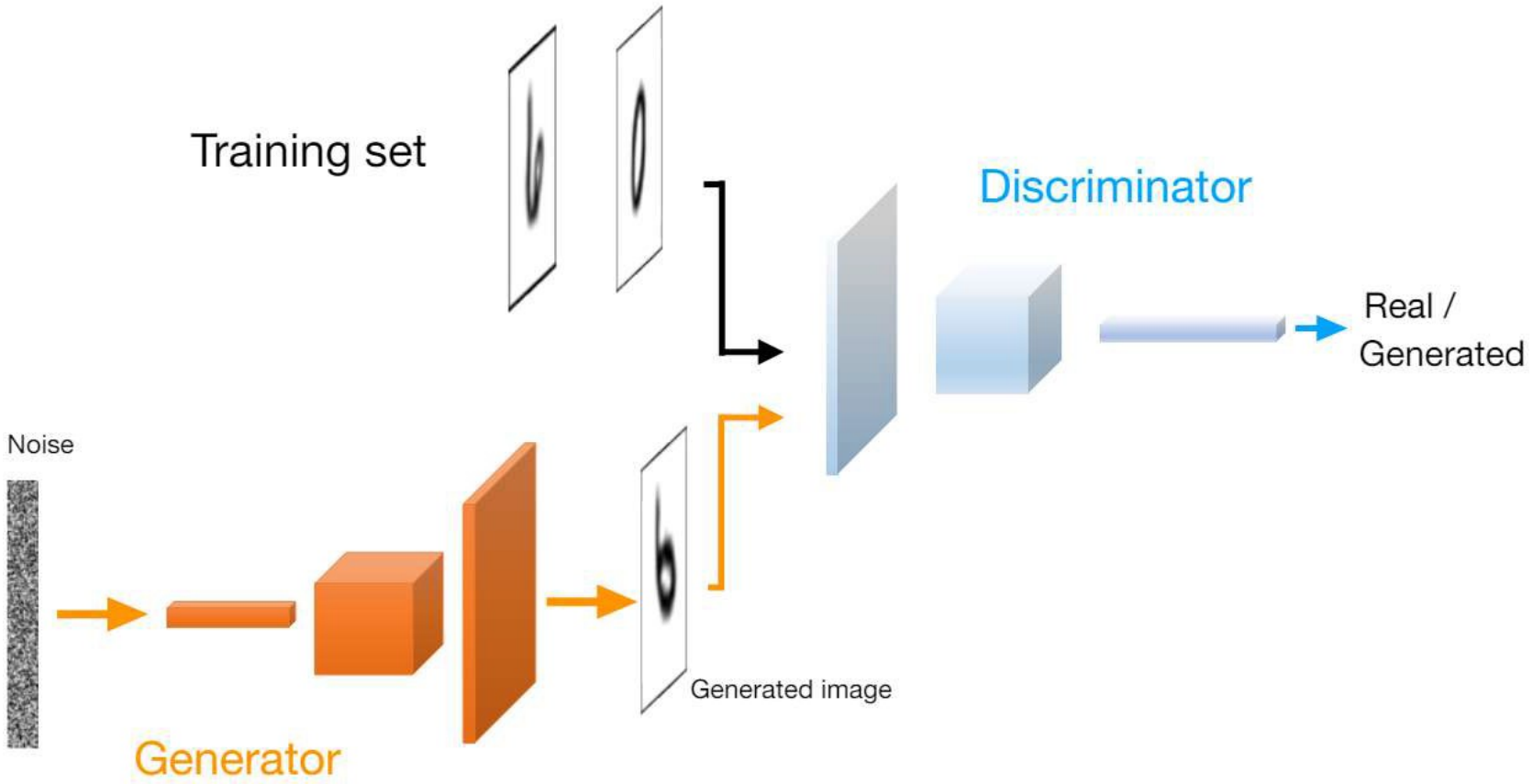


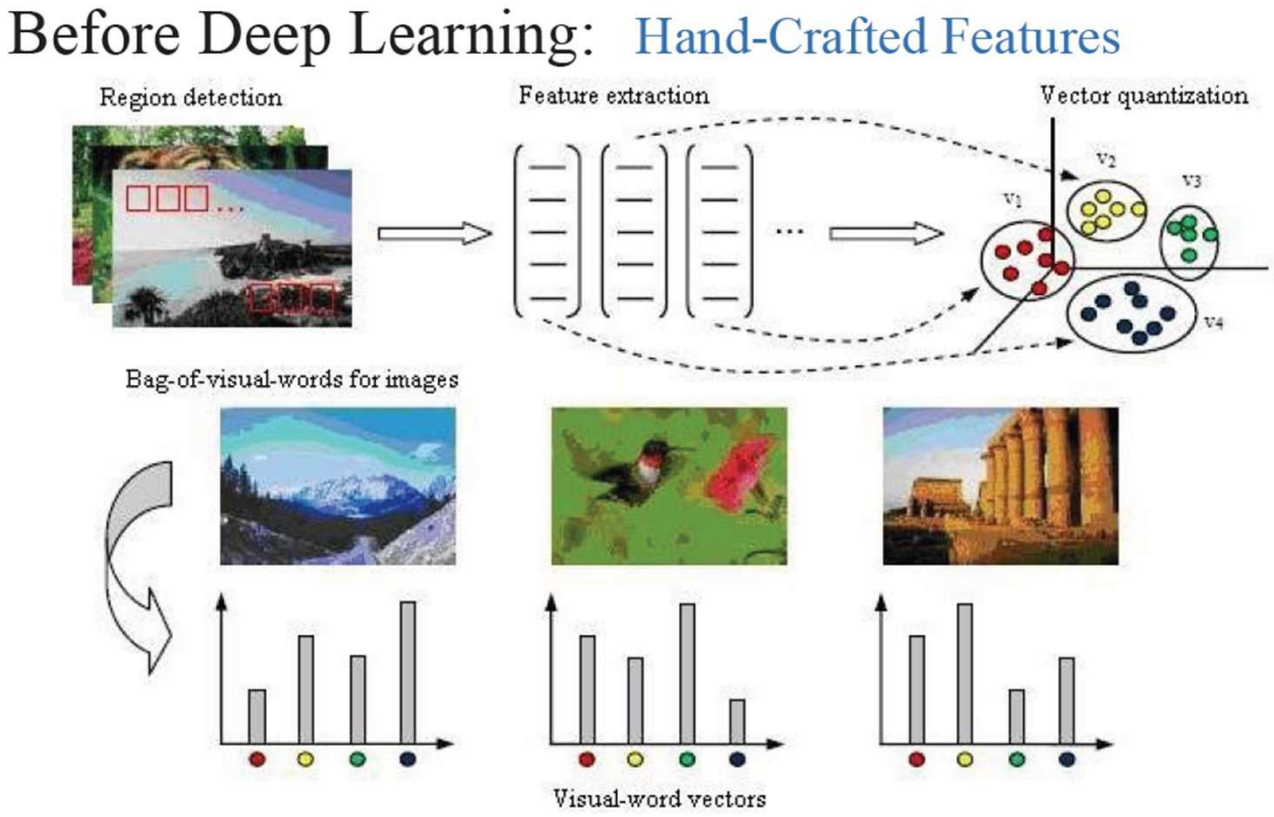
Figure 2. **Mask R-CNN** results on the COCO test set. These results are based on ResNet-101 [15], achieving a *mask* AP of 35.7 and running at 5 fps. Masks are shown in color, and bounding box, category, and confidences are also shown.

He, Kaiming, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. "Mask R-CNN." In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2961-2969. 2017.

Image generation



History of Computer Vision



History of Computer Vision

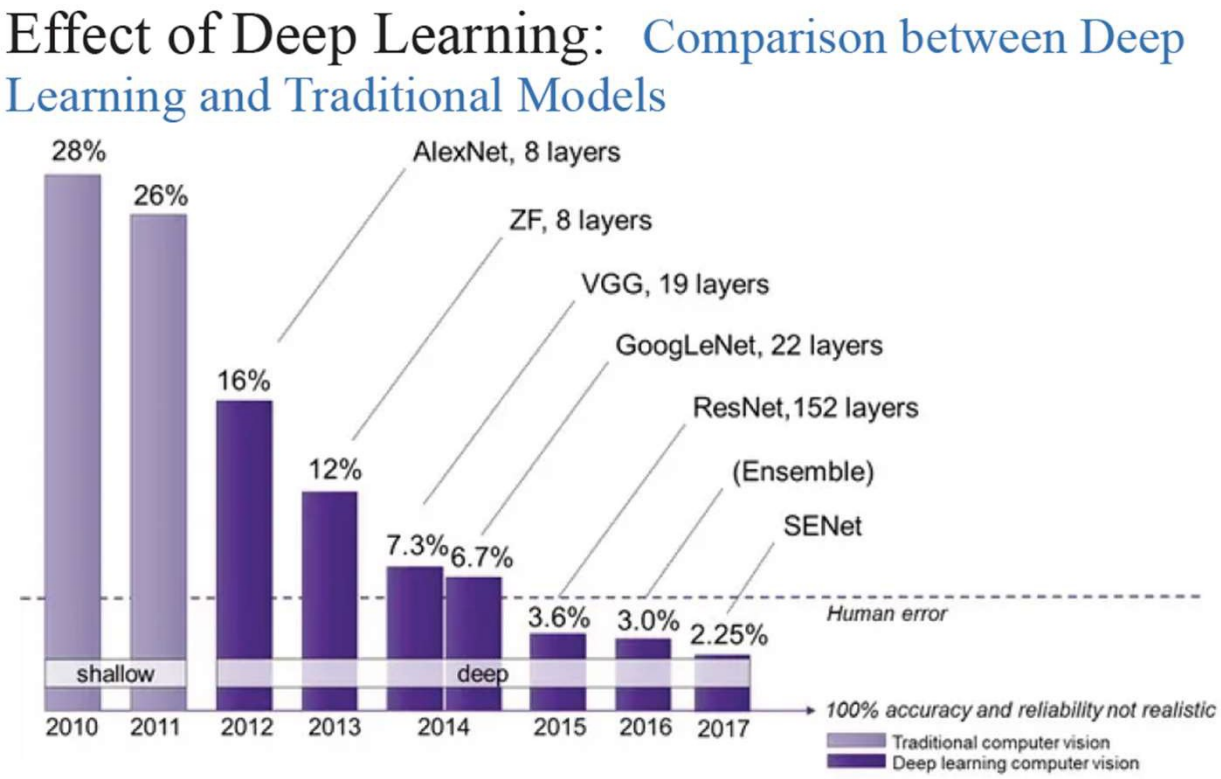
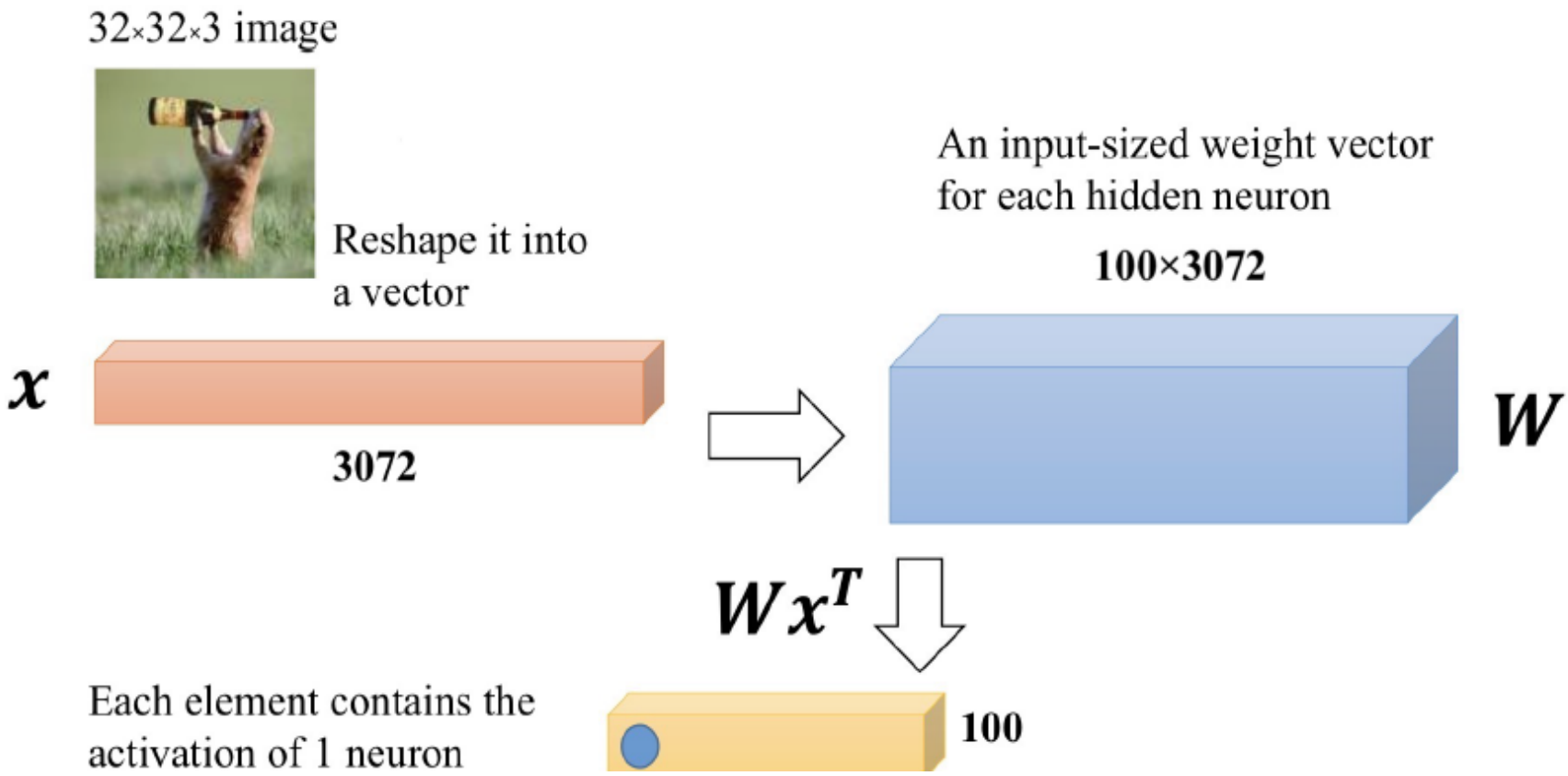


Figure: ImageNet Competetion Results Over Time, Source

Using Fully Connected Artichecture

- What we've been using?
  - **Dense** Vector Multiplication



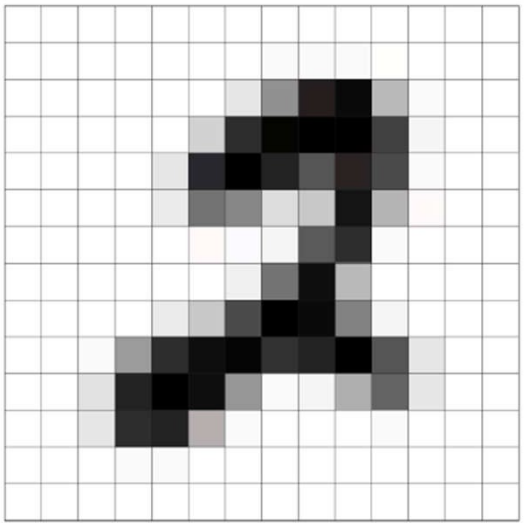
- What is the Problem with FC ?

- Neurons in a single layer operate independently and **do not share connections**, even for inputs.

- To be shift-invariant, **many samples** (in various time or locations) must be shown to them.

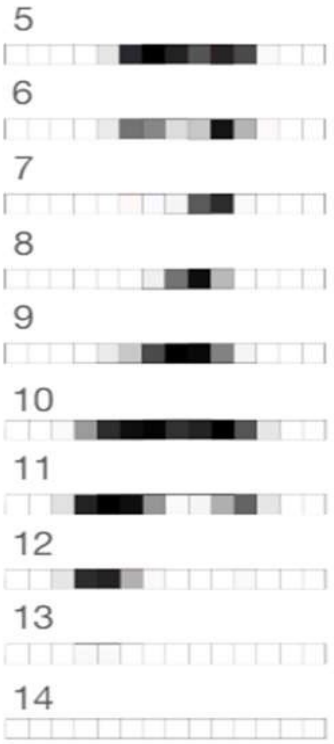
- **Regular Neural Nets don't scale well to full images**.
  - Parameters would **add up** quickly!
  - Full connectivity is wasteful and the huge number of parameters would quickly **lead to overfitting**.

Multi layer perceptron on image data

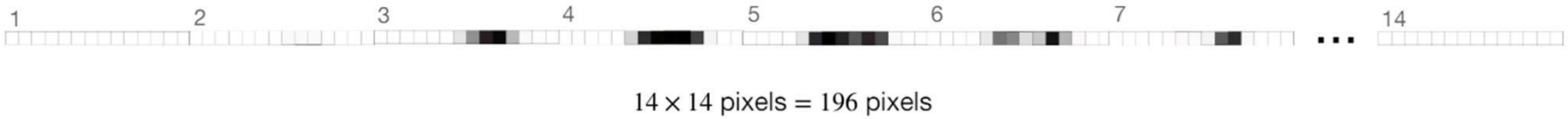


14x14-dimensional image

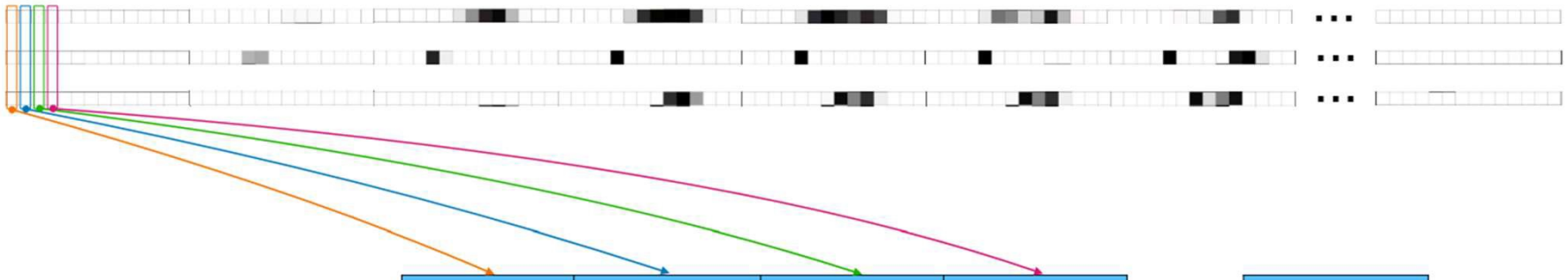
Concatenate the rows



get a 196-dimensional row vector

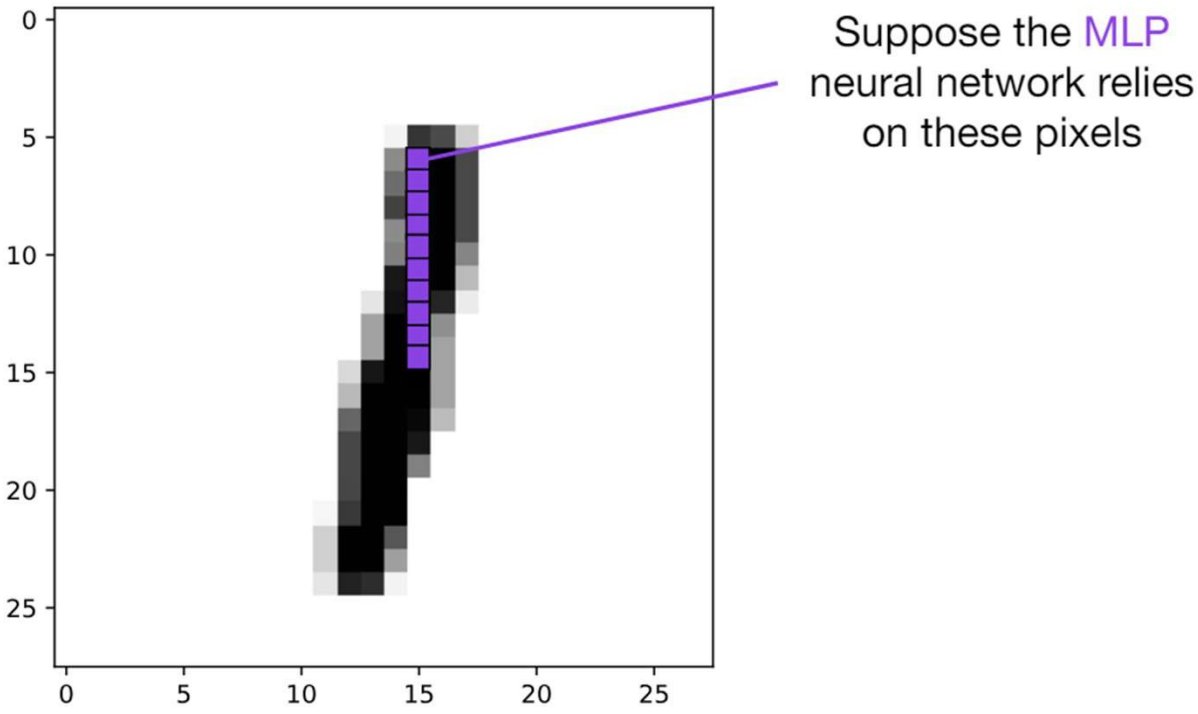


Training examples

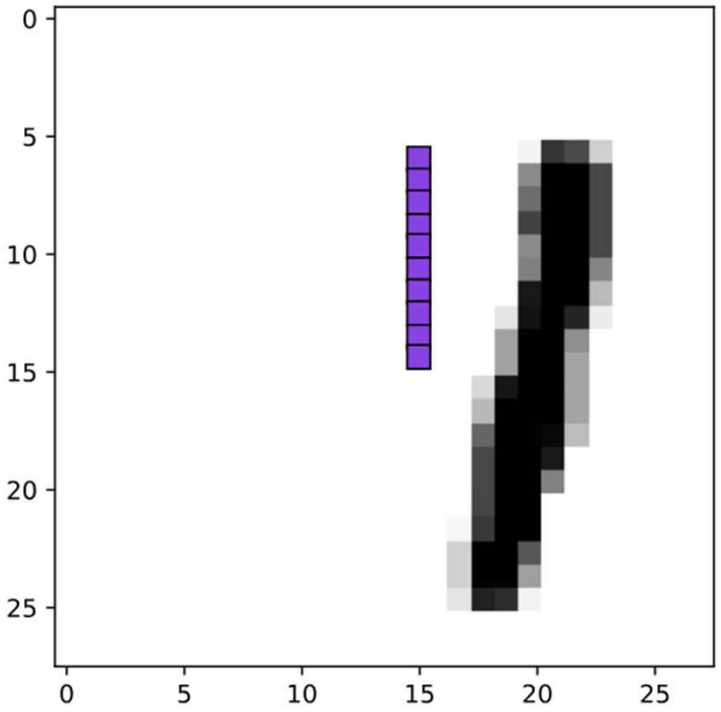


Position 1	Position 2	Position 3	Position 4	...	Position 196
0	0	0	0		0
0	0	0	0	...	0
0	0	0	0		0
...	...	...	...		...

Problem with MLP for images



Problem with MLP for images



An MLP will not recognize the digit if it is in slightly different position

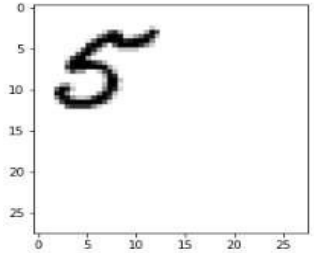
Why Image Classification is Hard

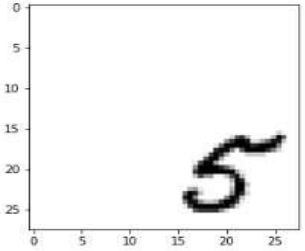
Different lighting, contrast, viewpoints, etc.

Image Source:
twitter.com%2Fcats&psig=AOvVaw30_o-PCM-K21DiMAJQimQ4&ust=1553887775741551

Image Source: https://www.123rf.com/photo_76714328_side-view-of-tabby-cat-face-over-white.html

Or even simple translation





This is hard for traditional methods like multi-layer perceptrons, because the prediction is basically based on a sum of pixel intensities

Main concept behind convolutional neural networks

- Sparse-connectivity: A single element in the feature map is connected to only a small patch of pixels. (This is very different from connecting to the whole input image, in the case of multilayer perceptrons.)
- Parameter-sharing: The same weights are used for different patches of the input image.
- Many layers: Combining extracted local patterns to global patterns

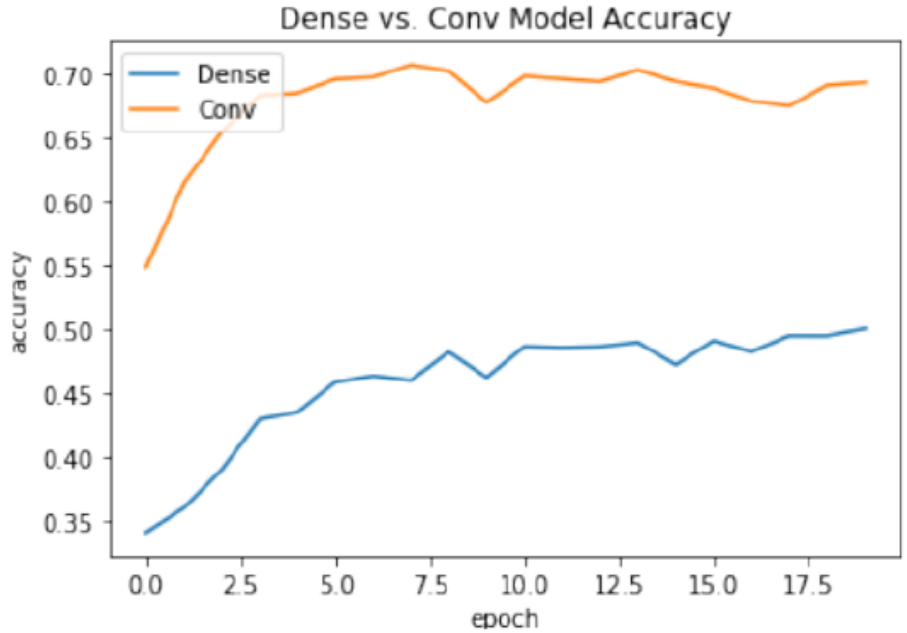
Fully connected Vs Convolution

Why Use Convolution:

- **Exploit spatial structure:** Convolution sees local patterns by using filters over small patches of the image.
- **Reduce parameters:** By sharing weights across the image, convolution reduce the number of parameters.
- **Build hierarchical features:** Convolution starts with small patterns, then combines them to recognize more complex patters.

FC vs CNN

Model Architecture	# Params	Test Accuracy
CNN	**2M**	**67.8%**
FCN	9M	50.6%



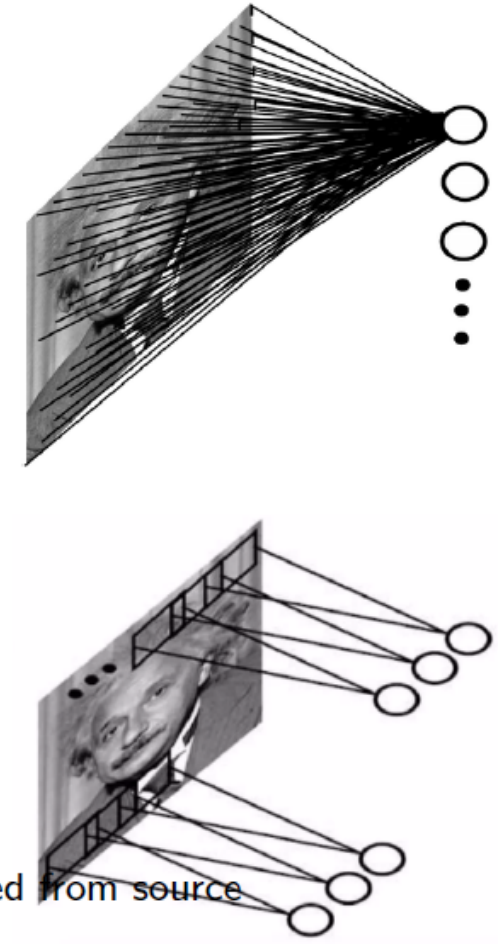
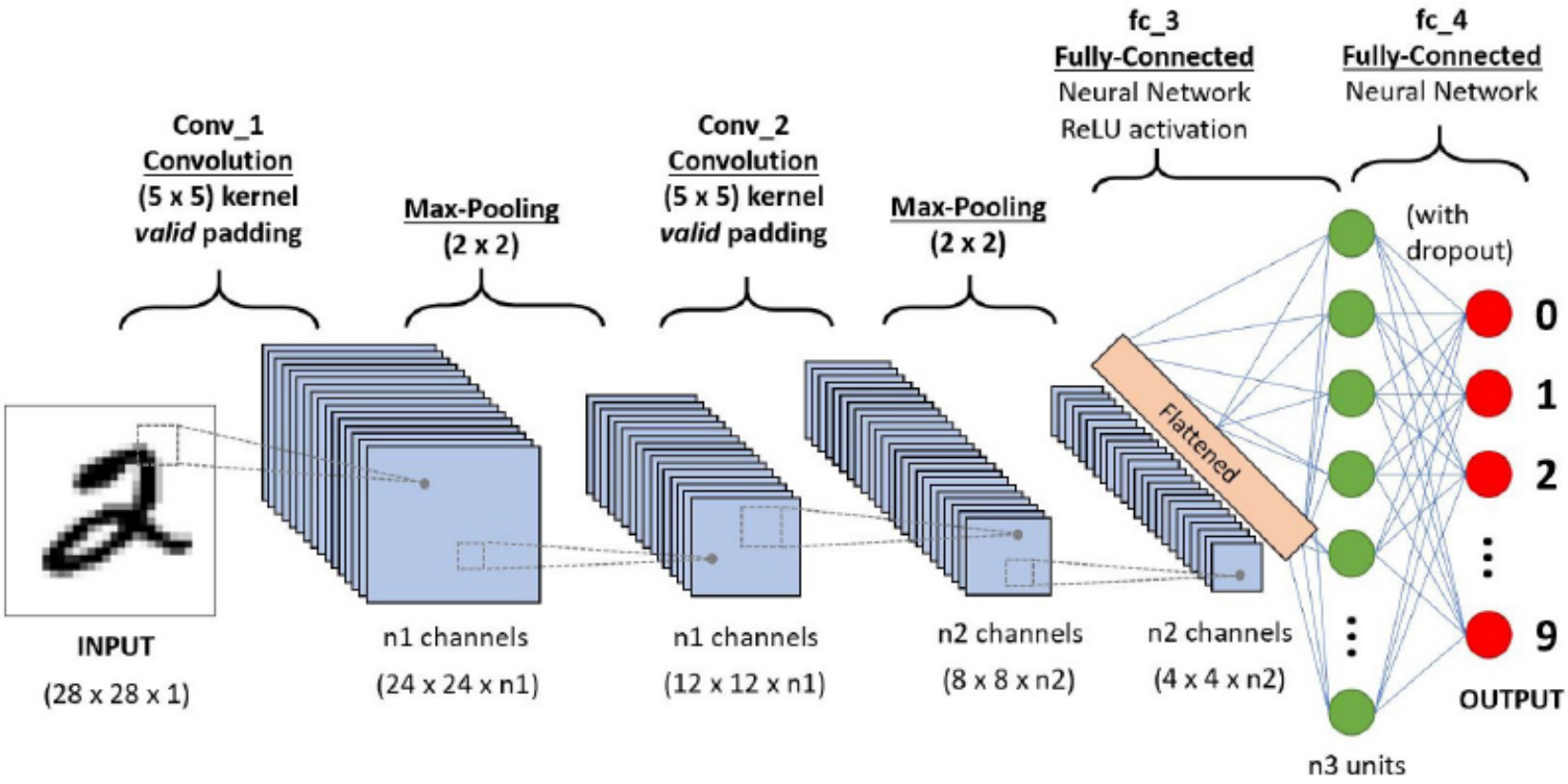


Figure adapted from source

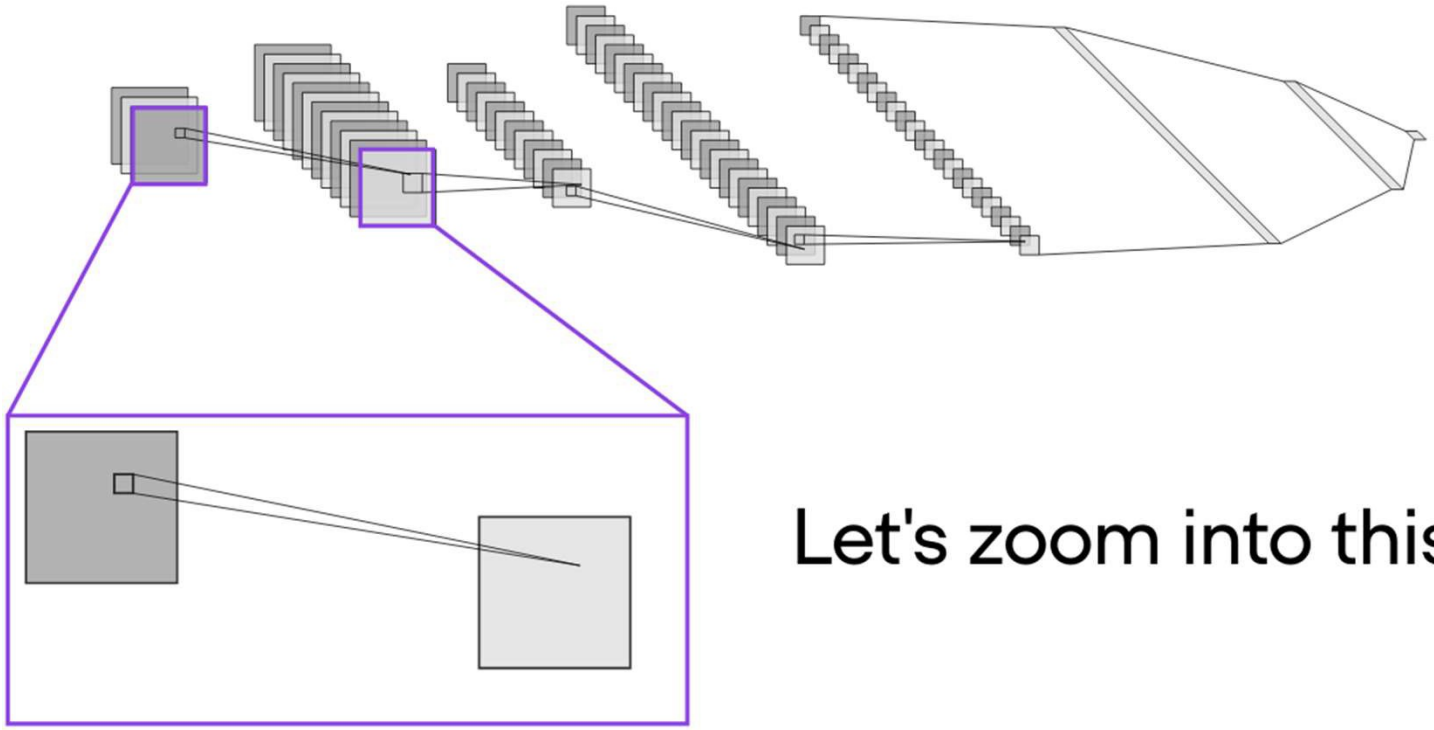
Looking at convolutional layers in more detail



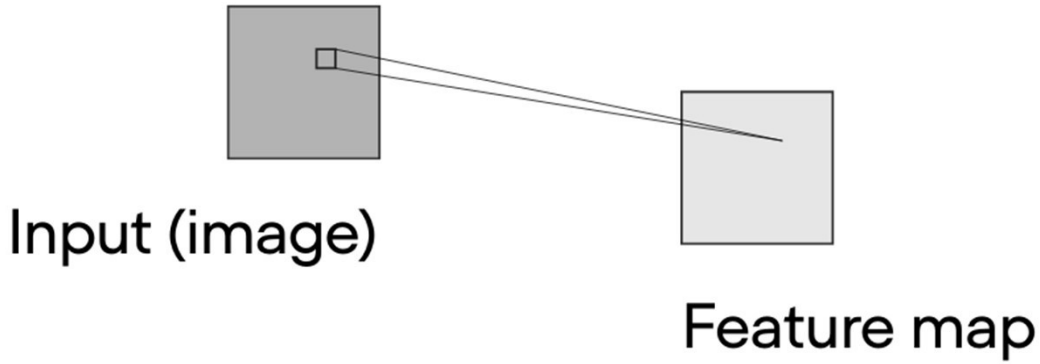
CNN basic structure

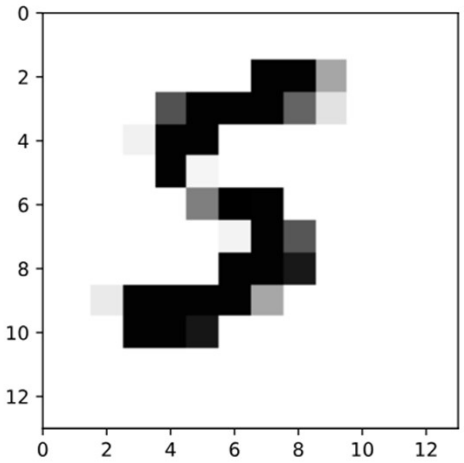
Three Main Types of Layers

- **Convolutional Layer**
  - Output of neurons are connected to **local regions** in the input.
  - Applying the **same filter** across the entire image.
  - CONV layer's parameters consist of a set of **learnable filters**.
- **Pooling Layer**
  - Performs a **downsampling** operation along the spatial dimensions.
- **Fully-Connected Layer**
  - Typically used in the final stages of the network to combine high-level features and **make predictions**.

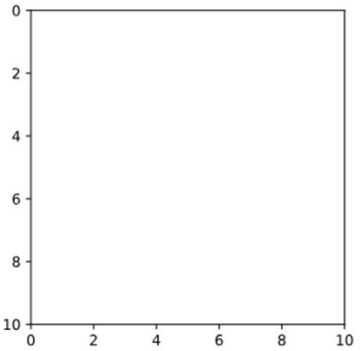


Let's zoom into this part

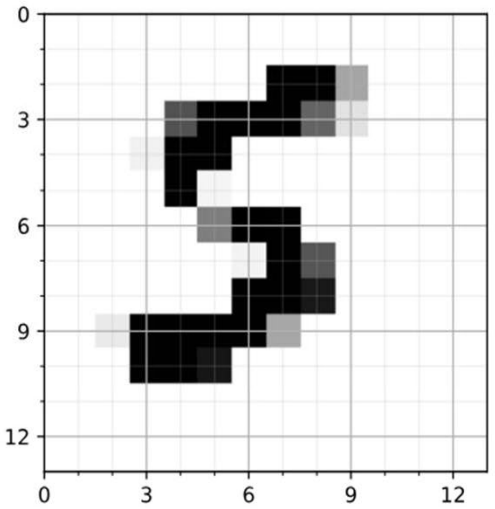

Input (image)
Feature map



Input (image)

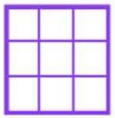


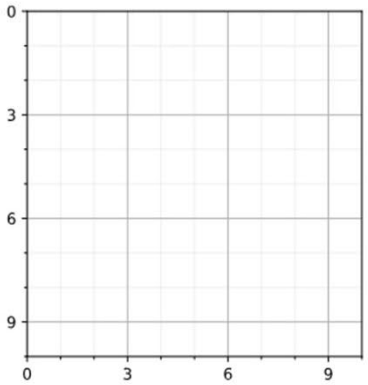
Feature map



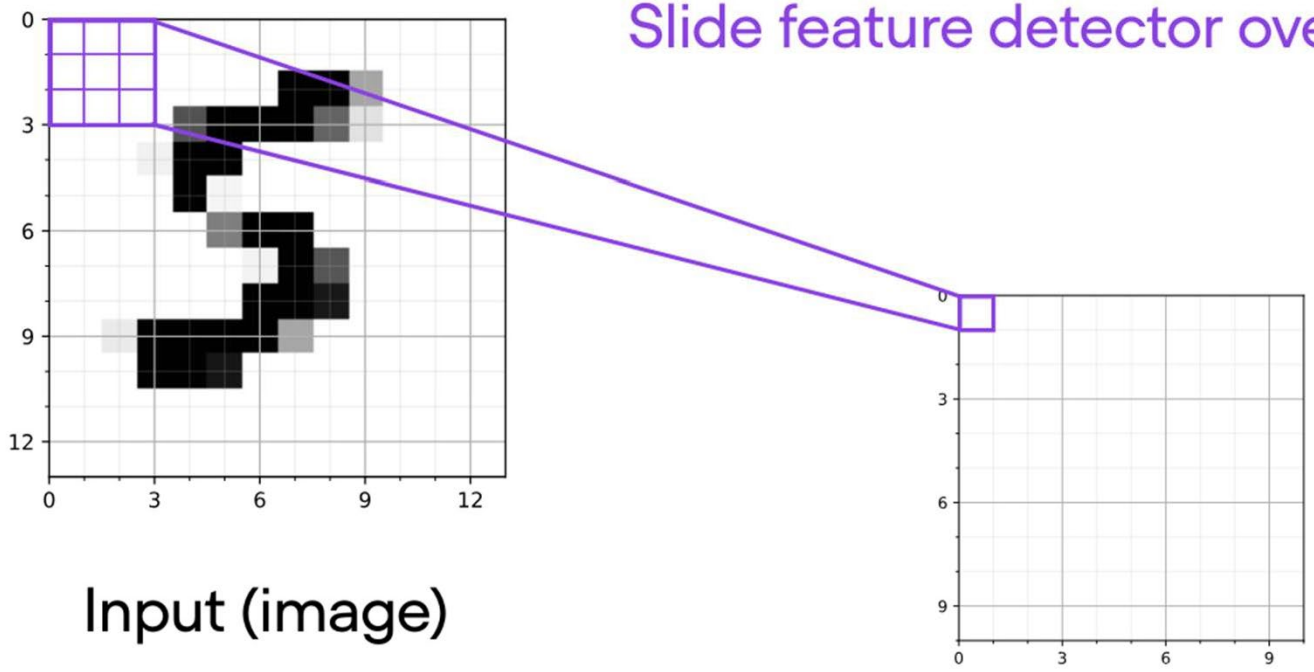
Input (image)

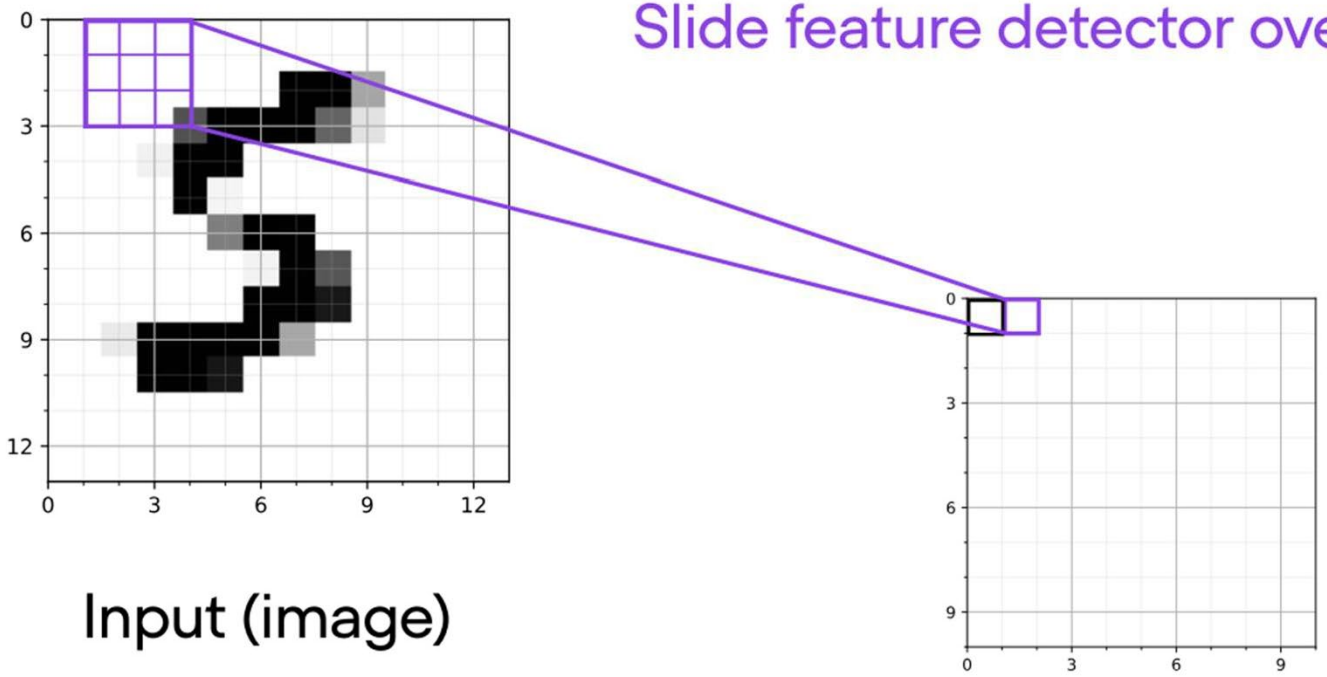
3x3 feature detector (kernel, filter)

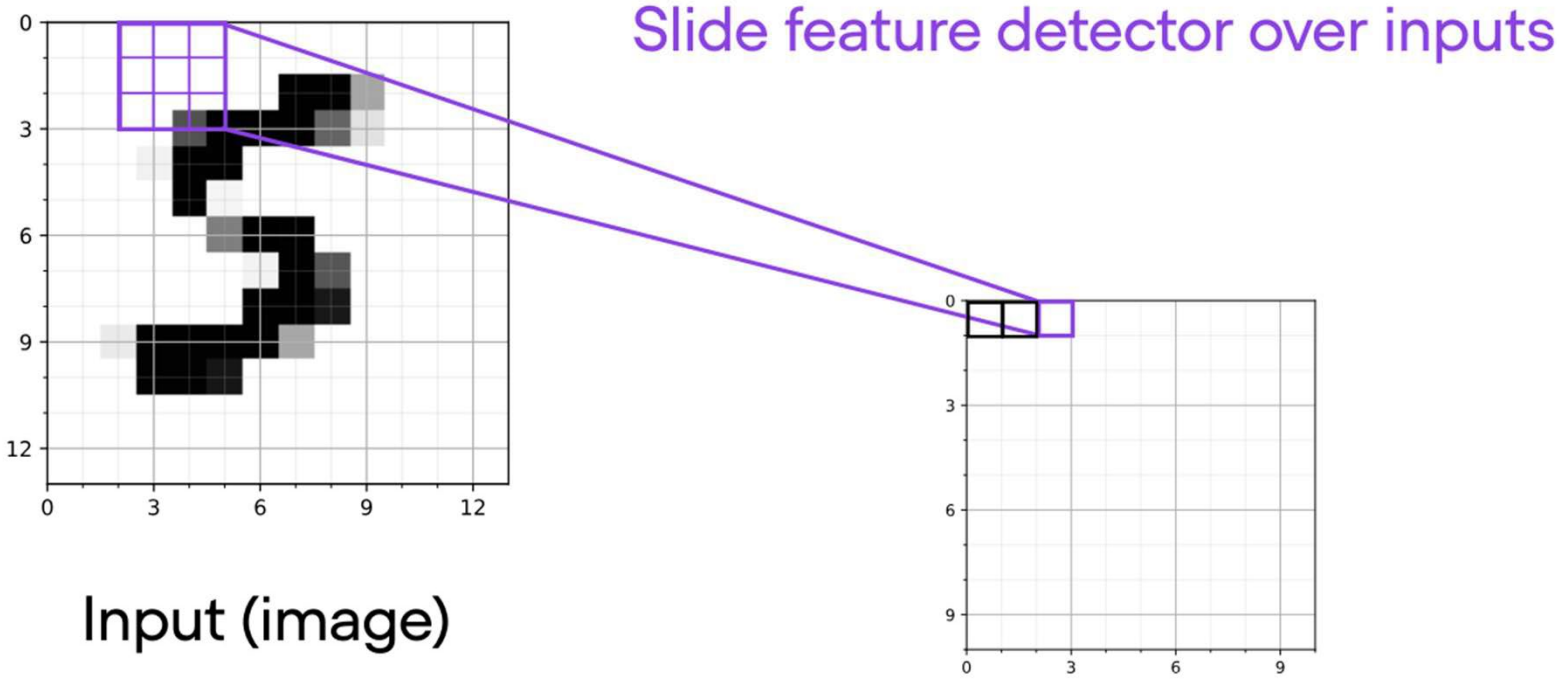


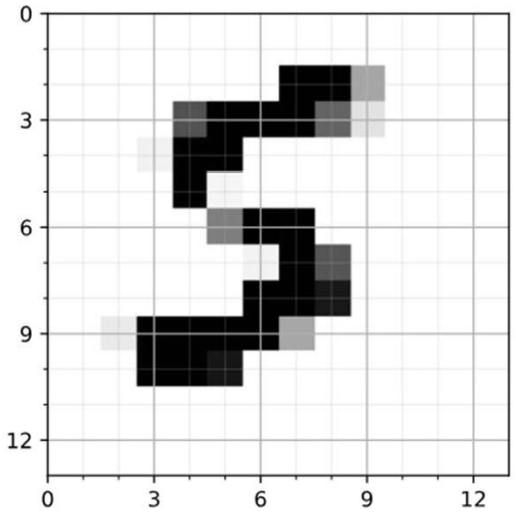


Feature map


Slide feature detector over inputs
0
3
6
9
12
0
3
6
9
12
Input (image)
0
3
6
9
0
3
6
9
Feature map

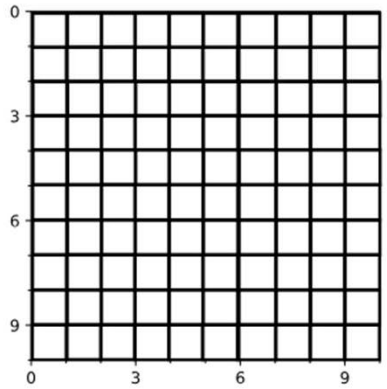

Slide feature detector over inputs
0
3
6
9
12
0
3
6
9
12
Input (image)
0
3
6
9
0
3
6
9
Feature map


Slide feature detector over inputs
0
3
6
9
12
0
3
6
9
12
0
3
6
9
0
3
6
9
Input (image)

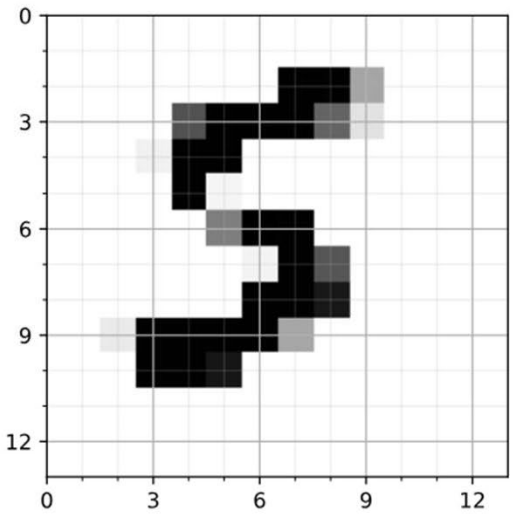


Input (image)

Slide feature detector over inputs

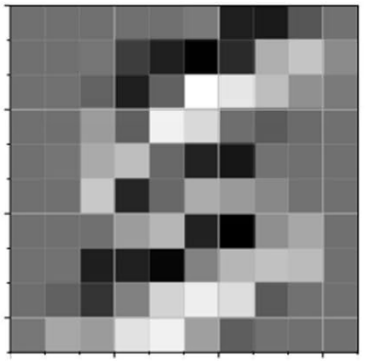


Feature map



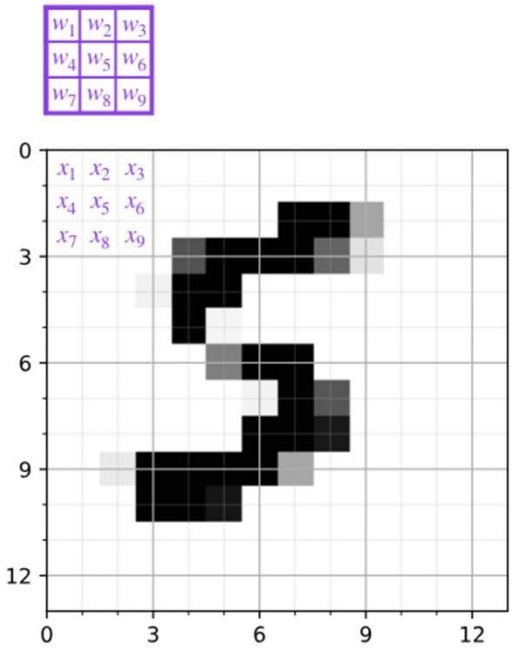
Input (image)

Slide feature detector over inputs



Feature map

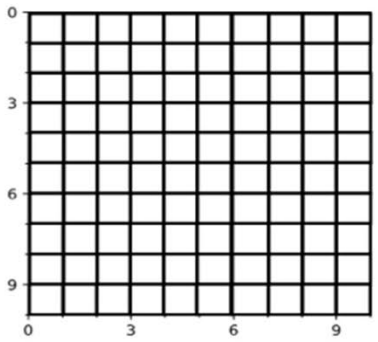
What is happening during this operation?

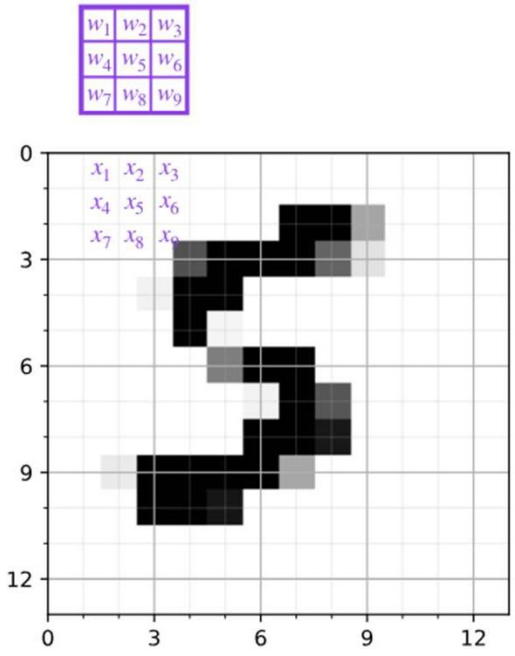


The inputs (x's) differ as we slide over the image.

The weights (w's) do not differ. → weight sharing

$$z = b + \sum_j w_j x_j$$





weight sharing

- Rationale: A feature detector that works well in one region may also work well in another region
- A reduction in parameters to fit (compared to MLP)

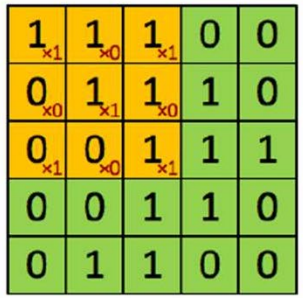
Convolution

- This is how we calculate the convolutional layer's output:

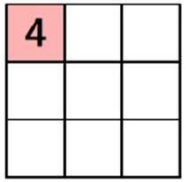
$$ConvolvedFeature(i,j) = (I * K)(i,j) = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} I(i+a, j+b)K(a,b)$$

I: Input Image
K: Our Kernel
k_h and k_w: The height and width of the Kernel

Figure: Convolving Kernel



Image



Convolved Feature

- This is how we calculate the convolutional layer's output:

$$ConvolvedFeature(i,j) = (I * K)(i,j) = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} I(i+a, j+b)K(a,b)$$

I: Input Image
K: Our Kernel
k_h and k_w: The height and width of the Kernel

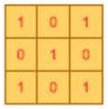
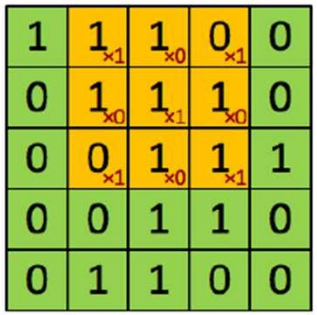
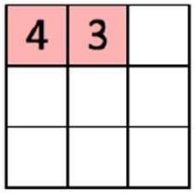


Figure: Convolving Kernel



Image



Convolved Feature

- This is how we calculate the convolutional layer's output:

$$ConvolvedFeature(i,j) = (I * K)(i,j) = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} I(i+a, j+b)K(a,b)$$

I: Input Image
K: Our Kernel
k_h and k_w: The height and width of the Kernel

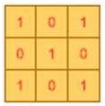
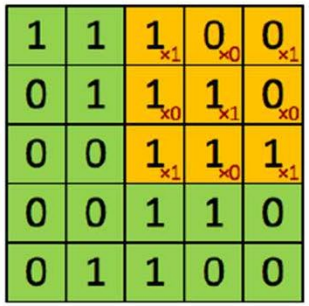
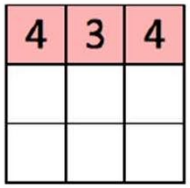


Figure: Convolving Kernel



Image



Convolved Feature

- This is how we calculate the convolutional layer's output:

$$ConvolvedFeature(i,j) = (I * K)(i,j) = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} I(i+a, j+b)K(a,b)$$

I: Input Image
K: Our Kernel
k_h and k_w: The height and width of the Kernel

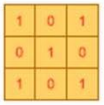
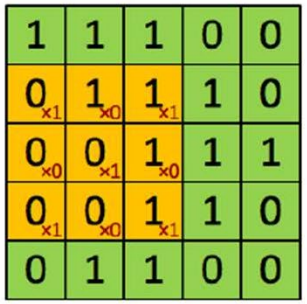
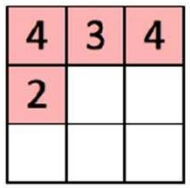


Figure: Convolving Kernel



Image



Convolved Feature

- This is how we calculate the convolutional layer's output:

$$ConvolvedFeature(i,j) = (I * K)(i,j) = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} I(i+a, j+b)K(a,b)$$

I: Input Image
K: Our Kernel
k_h and k_w: The height and width of the Kernel

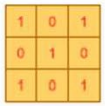
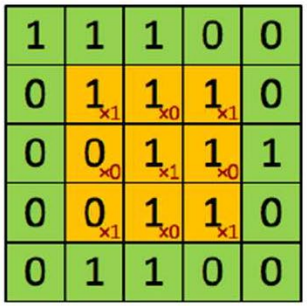
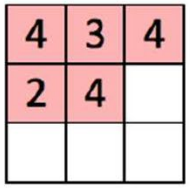


Figure: Convolving Kernel



Image



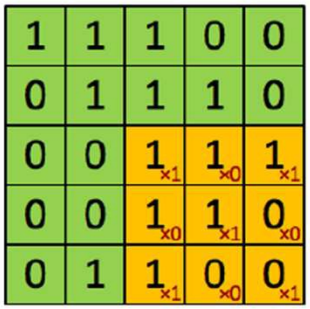
Convolved Feature

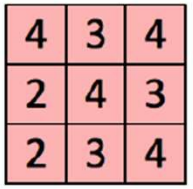
- This is how we calculate the convolutional layer's output:

$$ConvolvedFeature(i,j) = (I * K)(i,j) = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} I(i+a, j+b)K(a,b)$$

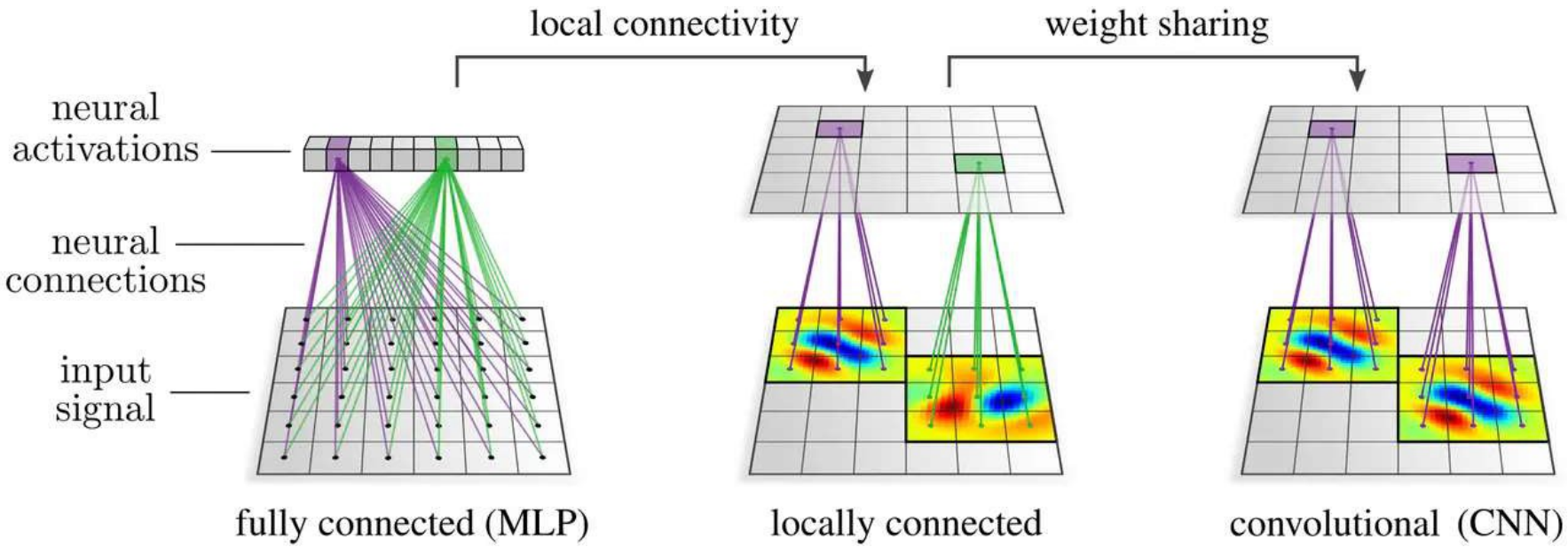
I: Input Image
K: Our Kernel
k_h and k_w: The height and width of the Kernel

Figure: Convolving Kernel





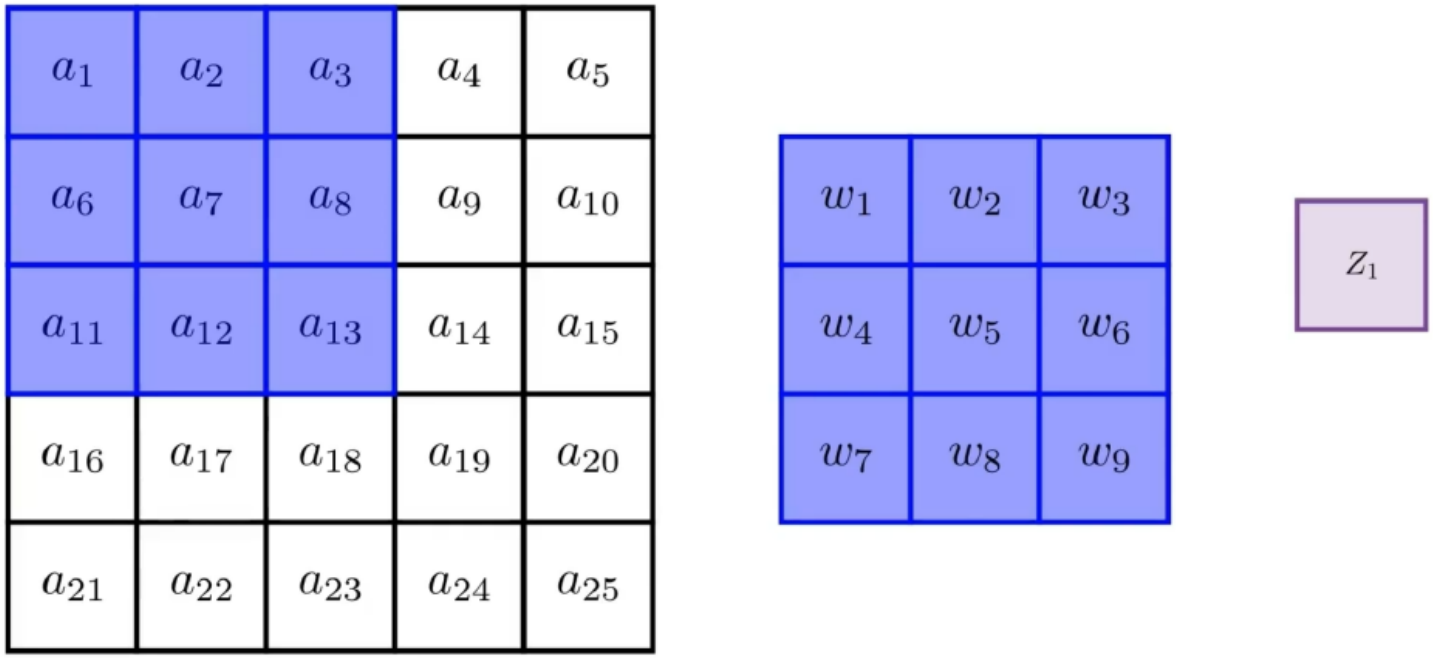
Locality and Weight sharing



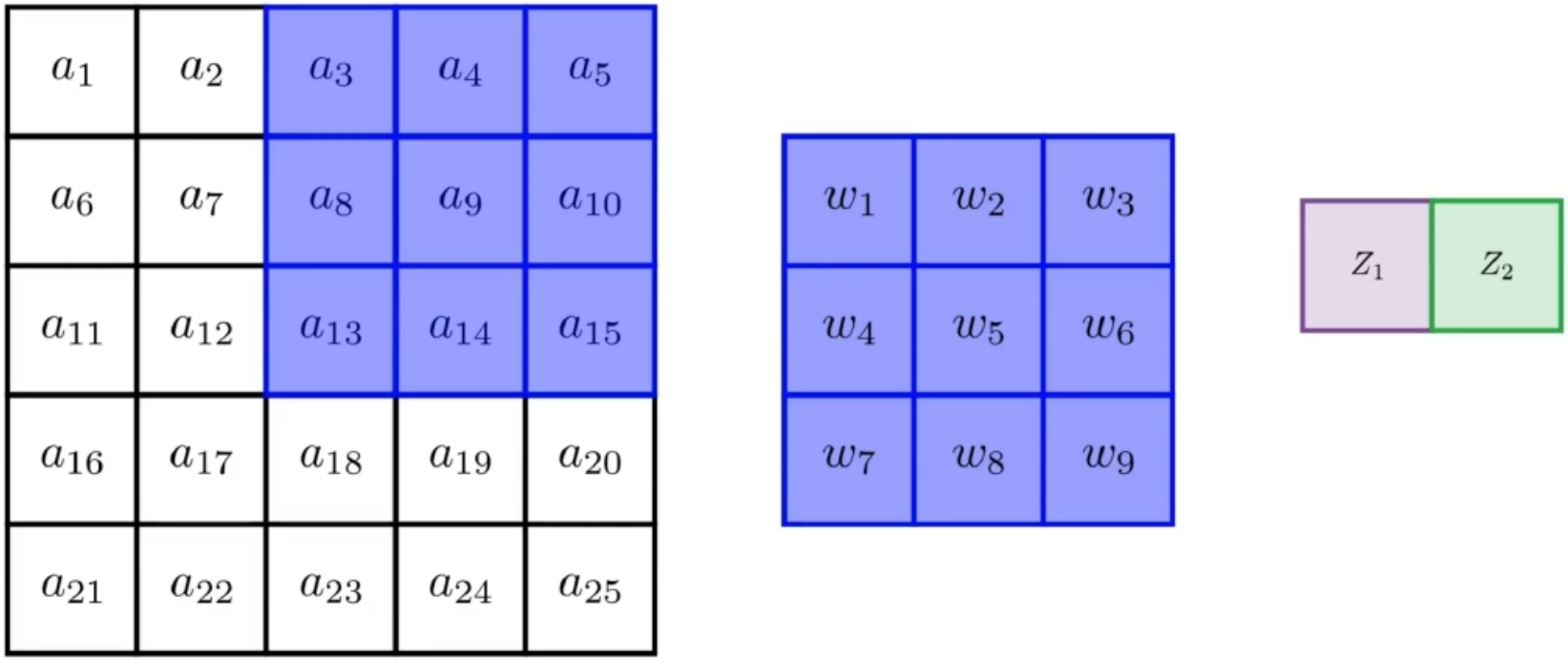
Backpropagation in CNNs

- Same overall concept as before: Multivariable chain rule, but now with an additional weight sharing constraint

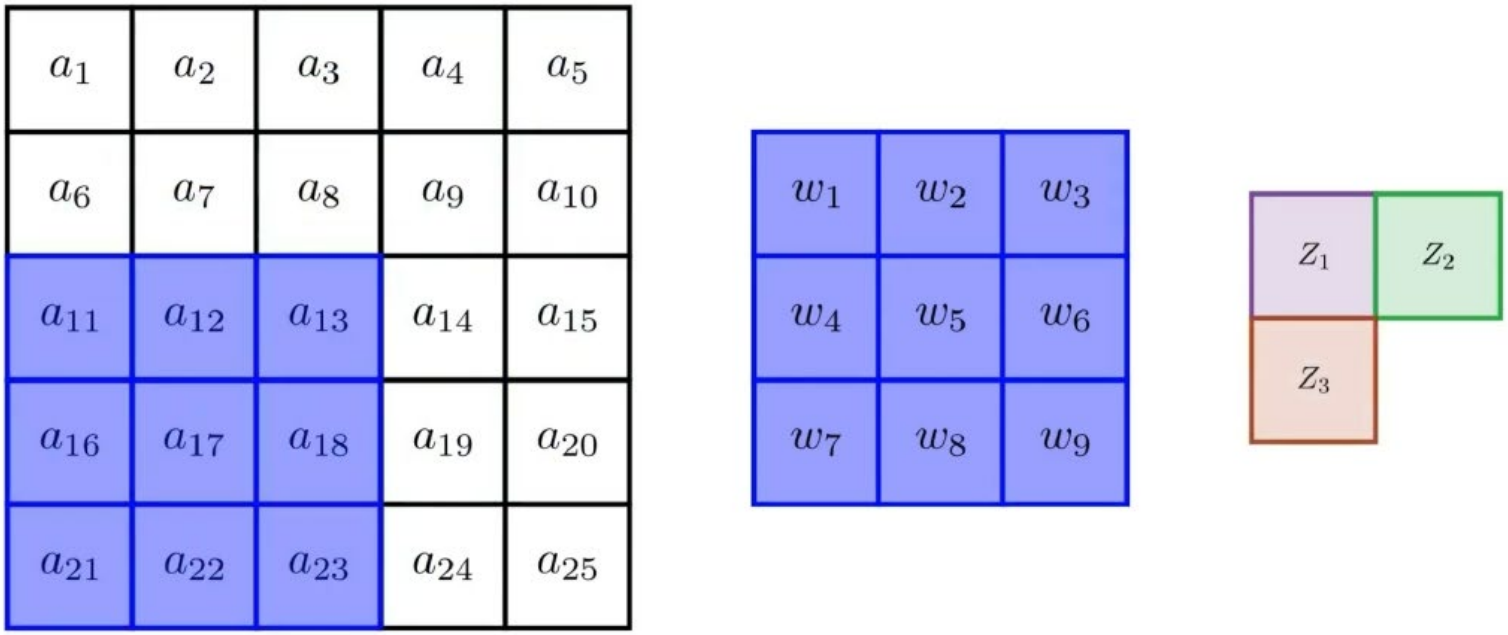
Convolution



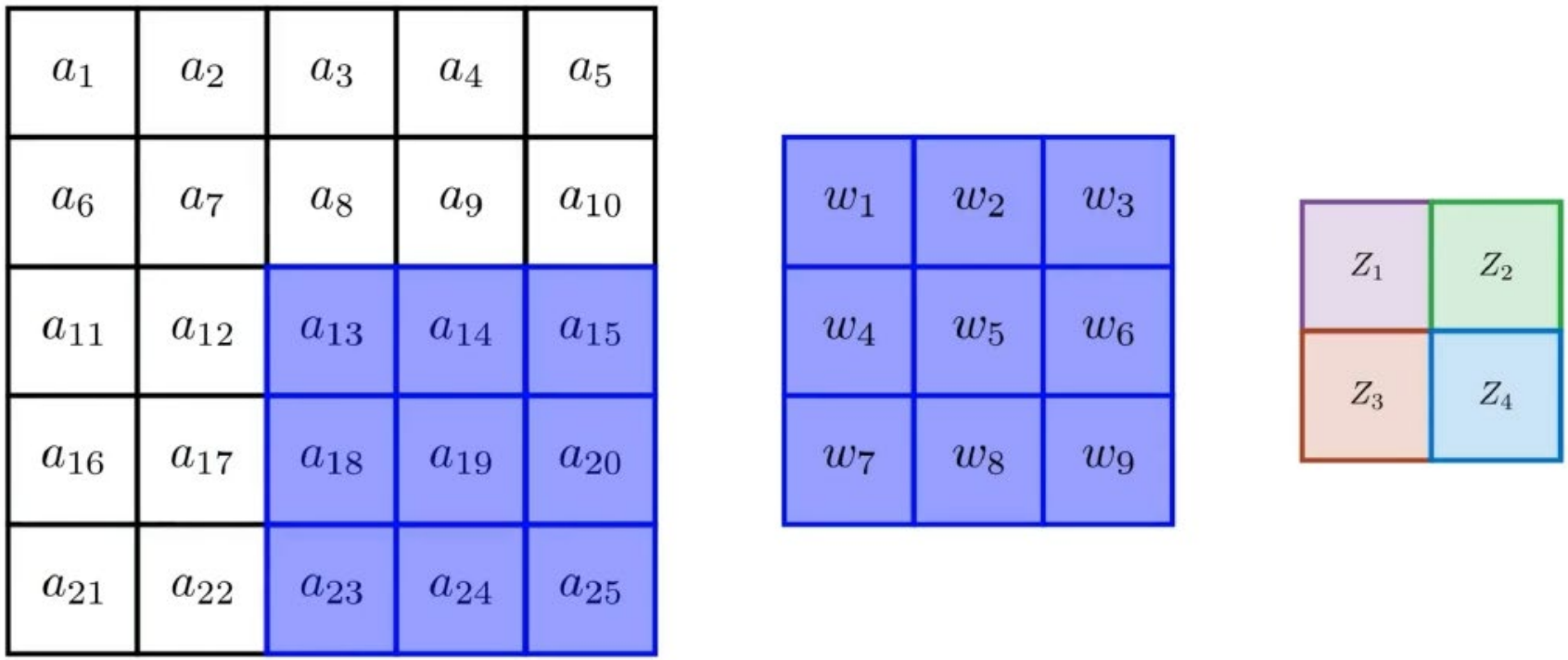
Convolution



Convolution



Convolution



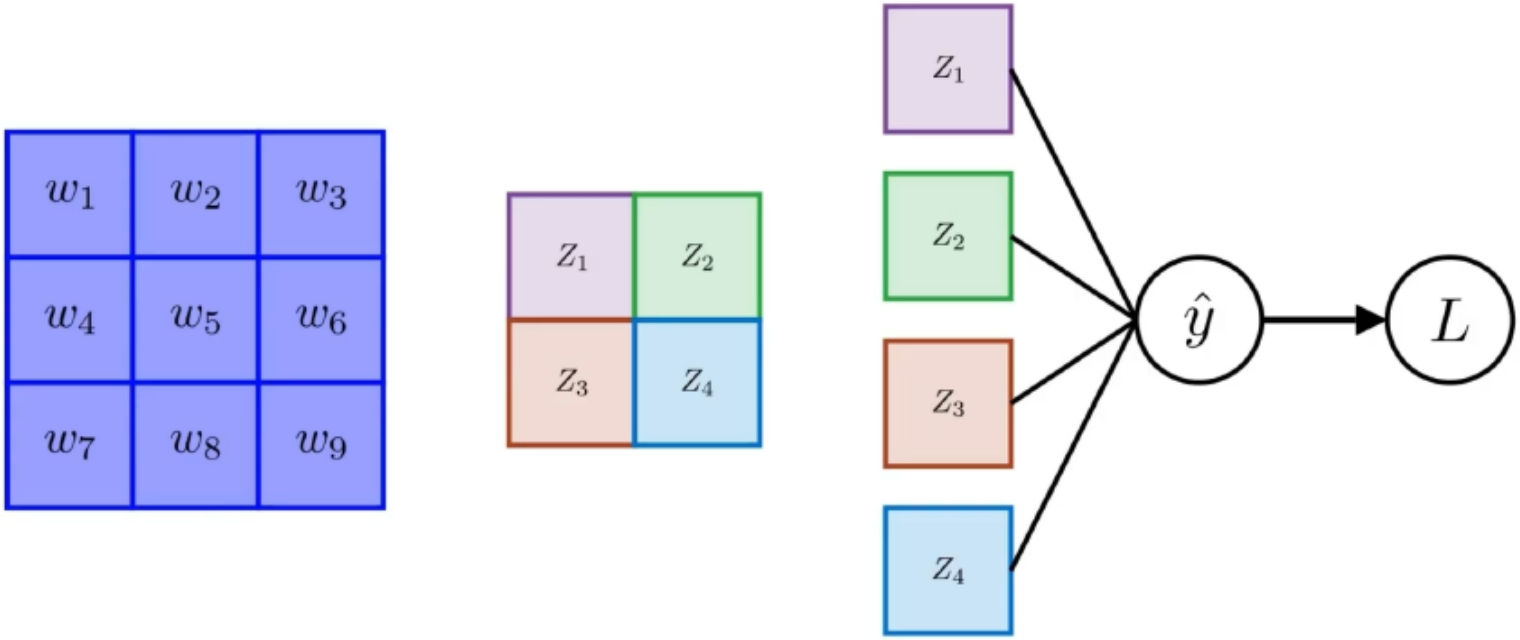
$$z_1 = w_1 \times a_1 + w_2 \times a_2 + w_3 \times a_3 + w_4 \times a_4 + + \cdots + w_9 \times a_{13}$$

$$z_2 = w_1 \times a_3 + w_2 \times a_4 + w_3 \times a_5 + w_4 \times a_8 + \cdots + w_9 \times a_{15}$$

$$z_3 = w_1 \times a_{11} + w_2 \times a_{12} + w_3 \times a_{13} + w_4 \times a_{16} + \cdots + w_9 \times a_{23}$$

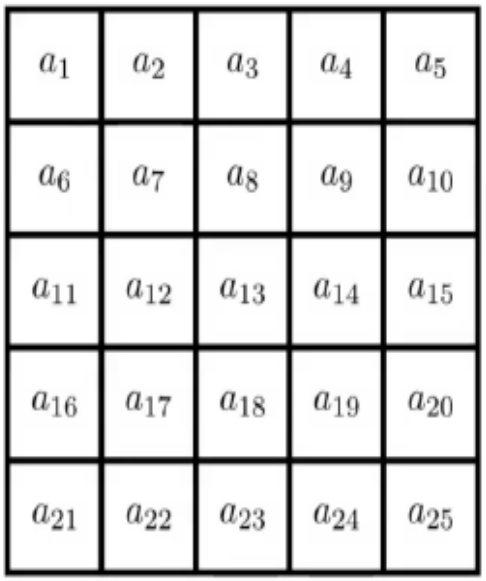
$$z_4 = w_1 \times a_{13} + w_2 \times a_{14} + w_3 \times a_{15} + w_4 \times a_{18} + \cdots + w_9 \times a_{25}$$

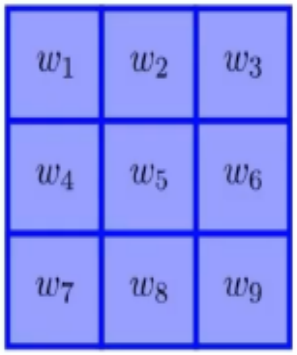
- For easier computation, we will assume one layer of convolution and a perceptron as our whole network.
- **Recall**: $w_i^* = w_i - \alpha \times \frac{\partial L}{\partial w_i}$



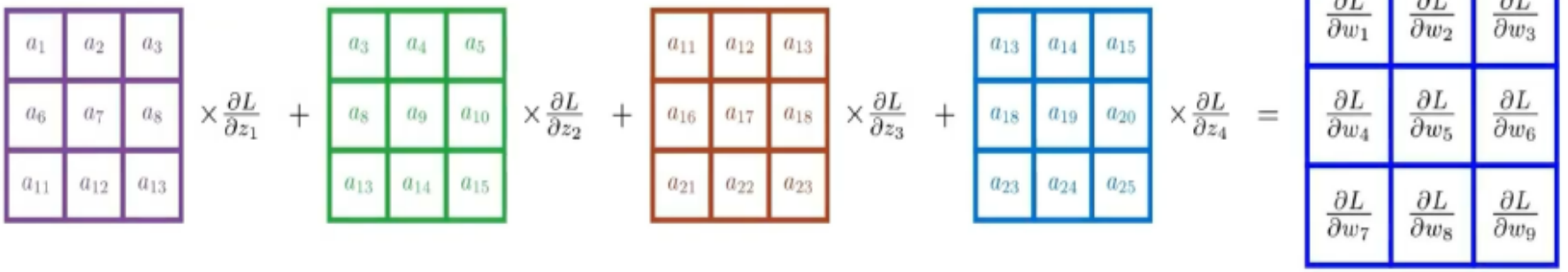
- We can easily calculate the gradients:

$$\frac{\partial L}{\partial w_1} = \frac{\partial z_1}{\partial w_1}\frac{\partial L}{\partial z_1} + \frac{\partial z_2}{\partial w_1}\frac{\partial L}{\partial z_2} + \frac{\partial z_3}{\partial w_1}\frac{\partial L}{\partial z_3} + \frac{\partial z_4}{\partial w_1}\frac{\partial L}{\partial z_4}$$
$$= a_1\frac{\partial L}{\partial z_1} + a_3\frac{\partial L}{\partial z_2} + a_{11}\frac{\partial L}{\partial z_3} + a_{13}\frac{\partial L}{\partial z_4}$$

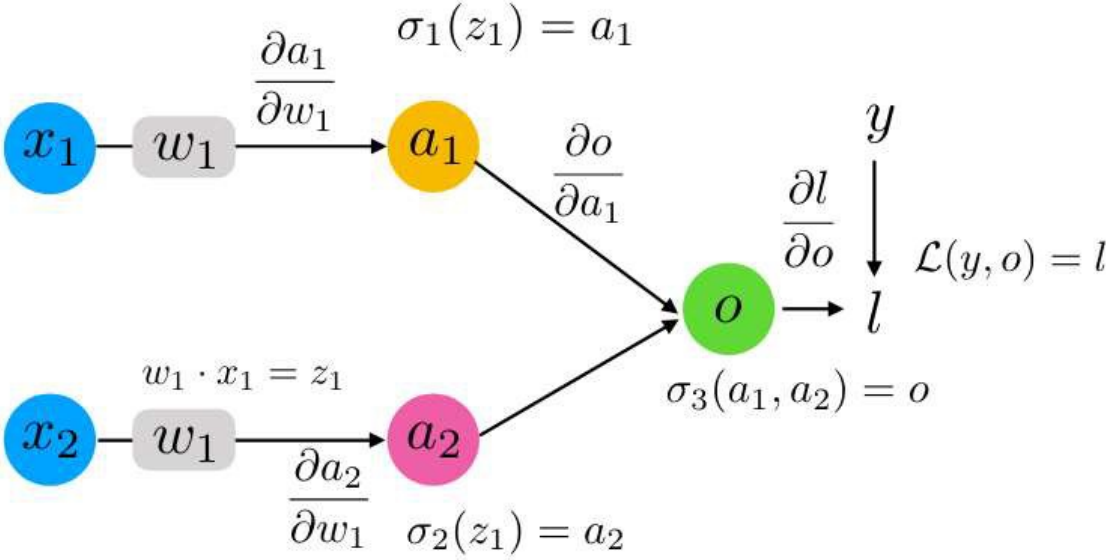




$$\frac{\partial L}{\partial w_1} = a_1\frac{\partial L}{\partial z_1} + a_3\frac{\partial L}{\partial z_2} + a_{11}\frac{\partial L}{\partial z_3} + a_{13}\frac{\partial L}{\partial z_4}$$
$$\frac{\partial L}{\partial w_2} = a_2\frac{\partial L}{\partial z_1} + a_4\frac{\partial L}{\partial z_2} + a_{12}\frac{\partial L}{\partial z_3} + a_{14}\frac{\partial L}{\partial z_4}$$
$$\frac{\partial L}{\partial w_3} = a_3\frac{\partial L}{\partial z_1} + a_5\frac{\partial L}{\partial z_2} + a_{13}\frac{\partial L}{\partial z_3} + a_{15}\frac{\partial L}{\partial z_4}$$
$$\frac{\partial L}{\partial w_4} = a_6\frac{\partial L}{\partial z_1} + a_8\frac{\partial L}{\partial z_2} + a_{16}\frac{\partial L}{\partial z_3} + a_{18}\frac{\partial L}{\partial z_4}$$
$$\frac{\partial L}{\partial w_5} = a_7\frac{\partial L}{\partial z_1} + a_9\frac{\partial L}{\partial z_2} + a_{17}\frac{\partial L}{\partial z_3} + a_{19}\frac{\partial L}{\partial z_4}$$
$$\frac{\partial L}{\partial w_6} = a_8\frac{\partial L}{\partial z_1} + a_{10}\frac{\partial L}{\partial z_2} + a_{18}\frac{\partial L}{\partial z_3} + a_{20}\frac{\partial L}{\partial z_4}$$
$$\frac{\partial L}{\partial w_7} = a_{11}\frac{\partial L}{\partial z_1} + a_{13}\frac{\partial L}{\partial z_2} + a_{21}\frac{\partial L}{\partial z_3} + a_{23}\frac{\partial L}{\partial z_4}$$
$$\frac{\partial L}{\partial w_8} = a_{12}\frac{\partial L}{\partial z_1} + a_{14}\frac{\partial L}{\partial z_2} + a_{22}\frac{\partial L}{\partial z_3} + a_{24}\frac{\partial L}{\partial z_4}$$
$$\frac{\partial L}{\partial w_9} = a_{13}\frac{\partial L}{\partial z_1} + a_{15}\frac{\partial L}{\partial z_2} + a_{23}\frac{\partial L}{\partial z_3} + a_{25}\frac{\partial L}{\partial z_4}$$



Graph with Weight Sharing



Upper path

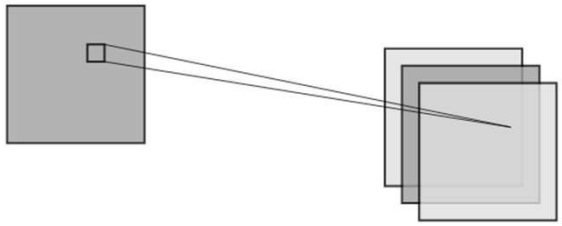
$$\frac{\partial l}{\partial w_1} = \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1} + \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_2} \cdot \frac{\partial a_2}{\partial w_1}$$ (multivariable chain rule)

Lower path

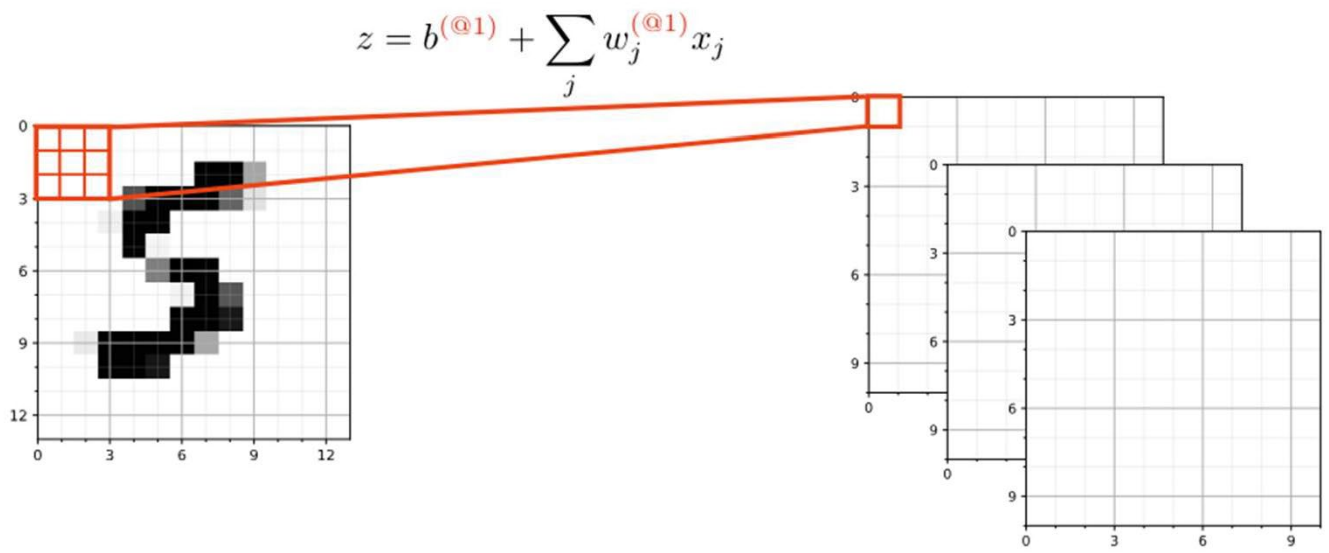
- Which problem is solved with weigh sharing and locality?

Multiple Output Channels

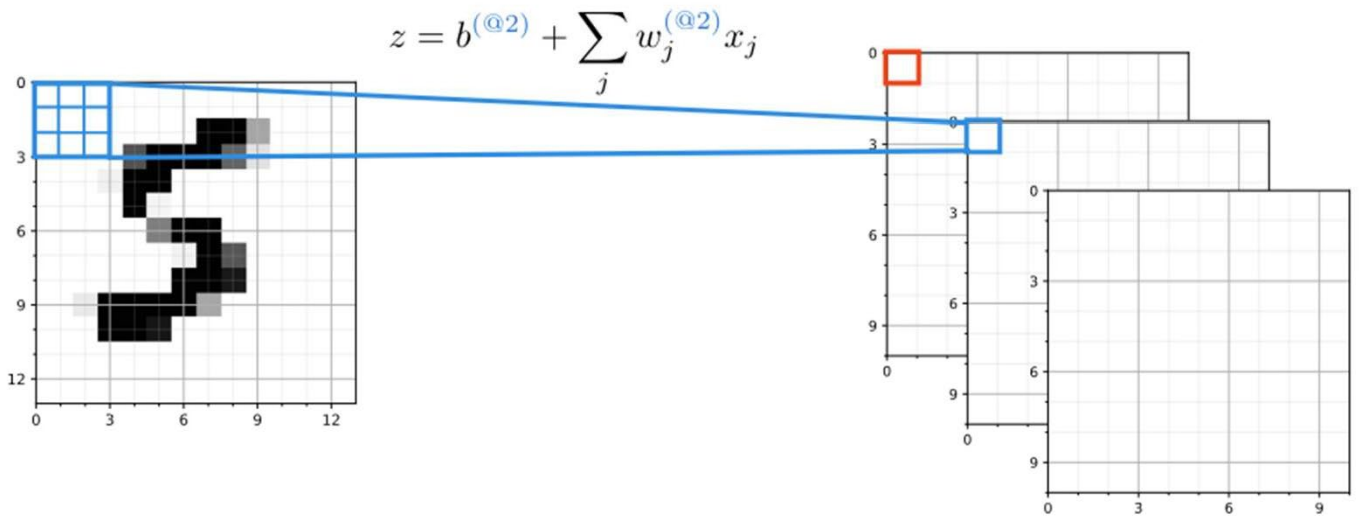
1 input channel, 3 output channels



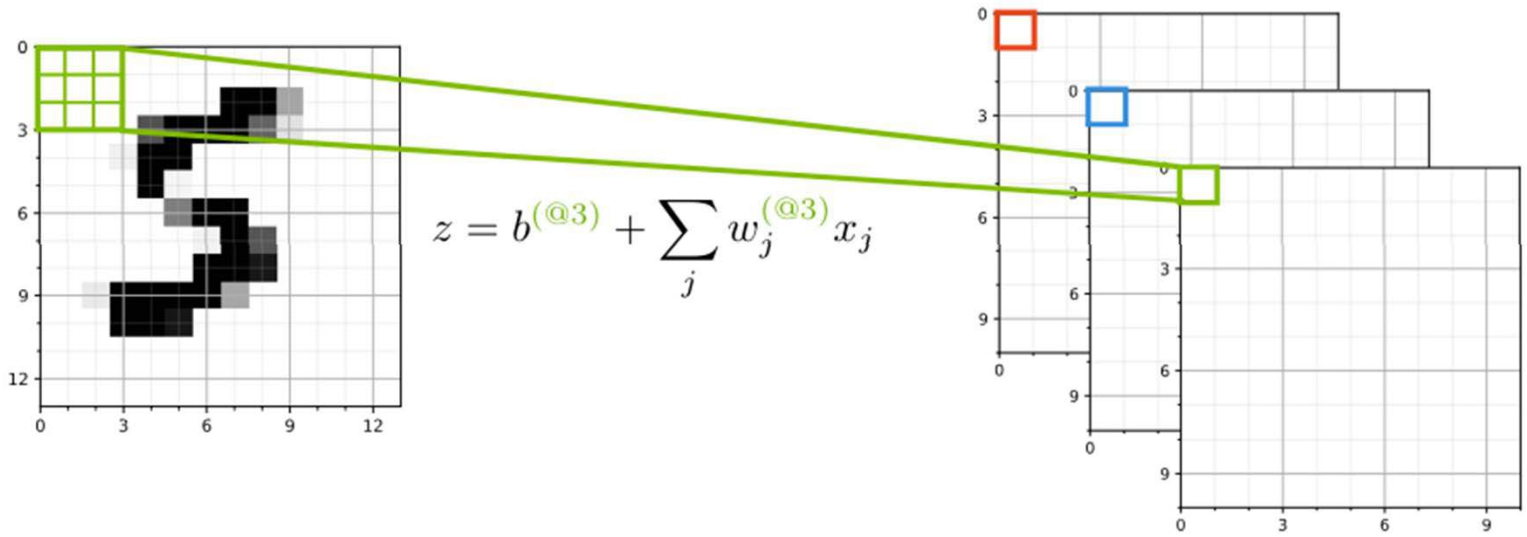
1 input channel 3 output channel



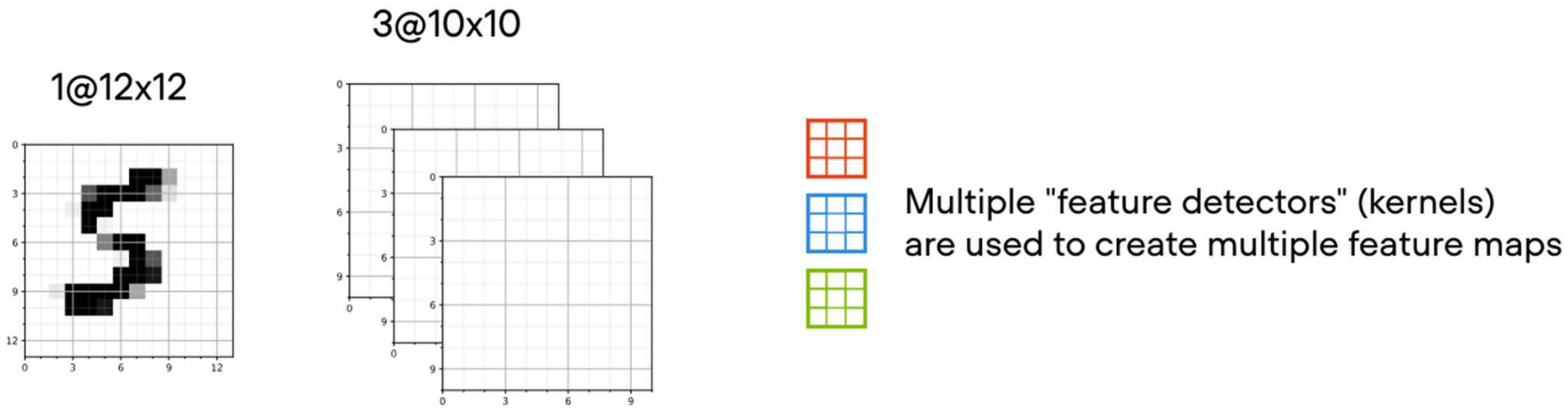
1 input channel 3 output channel



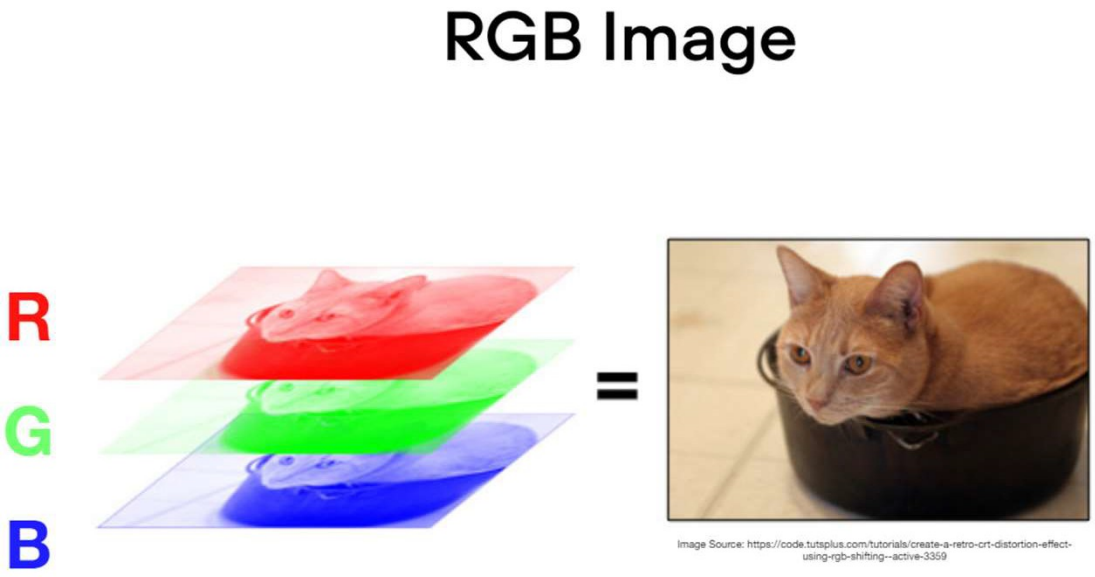
1 input channel 3 output channel



1 input channel 3 output channel



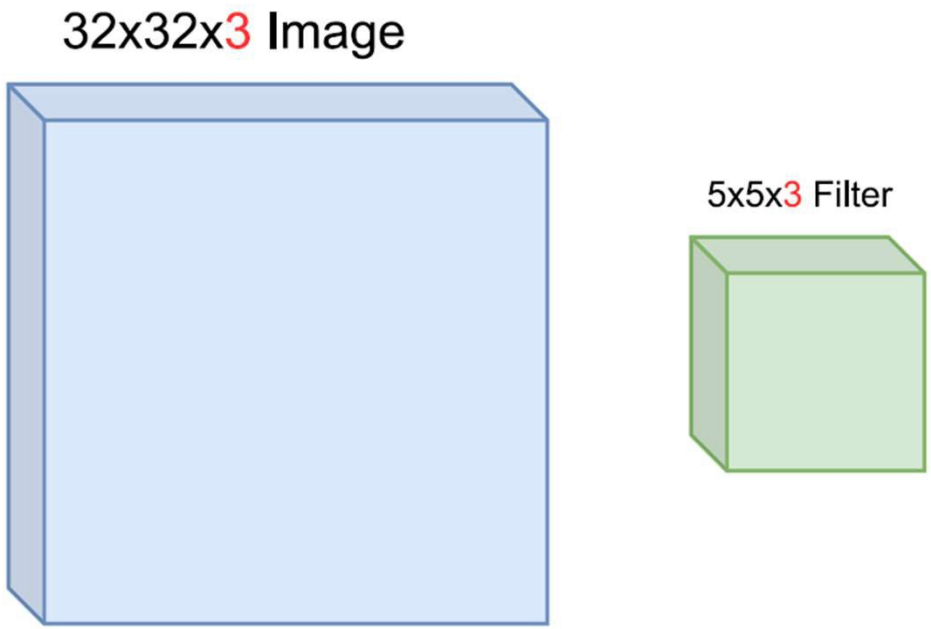
Multiple input channel



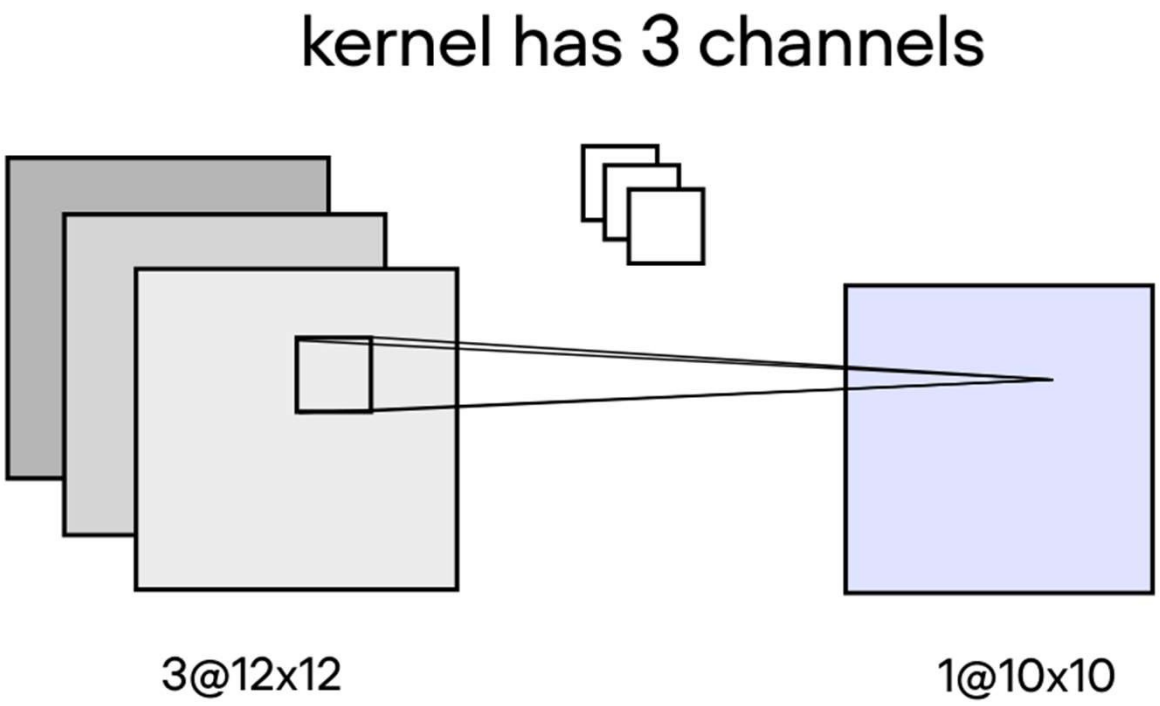
Color image as a stack of matrices

Input channels

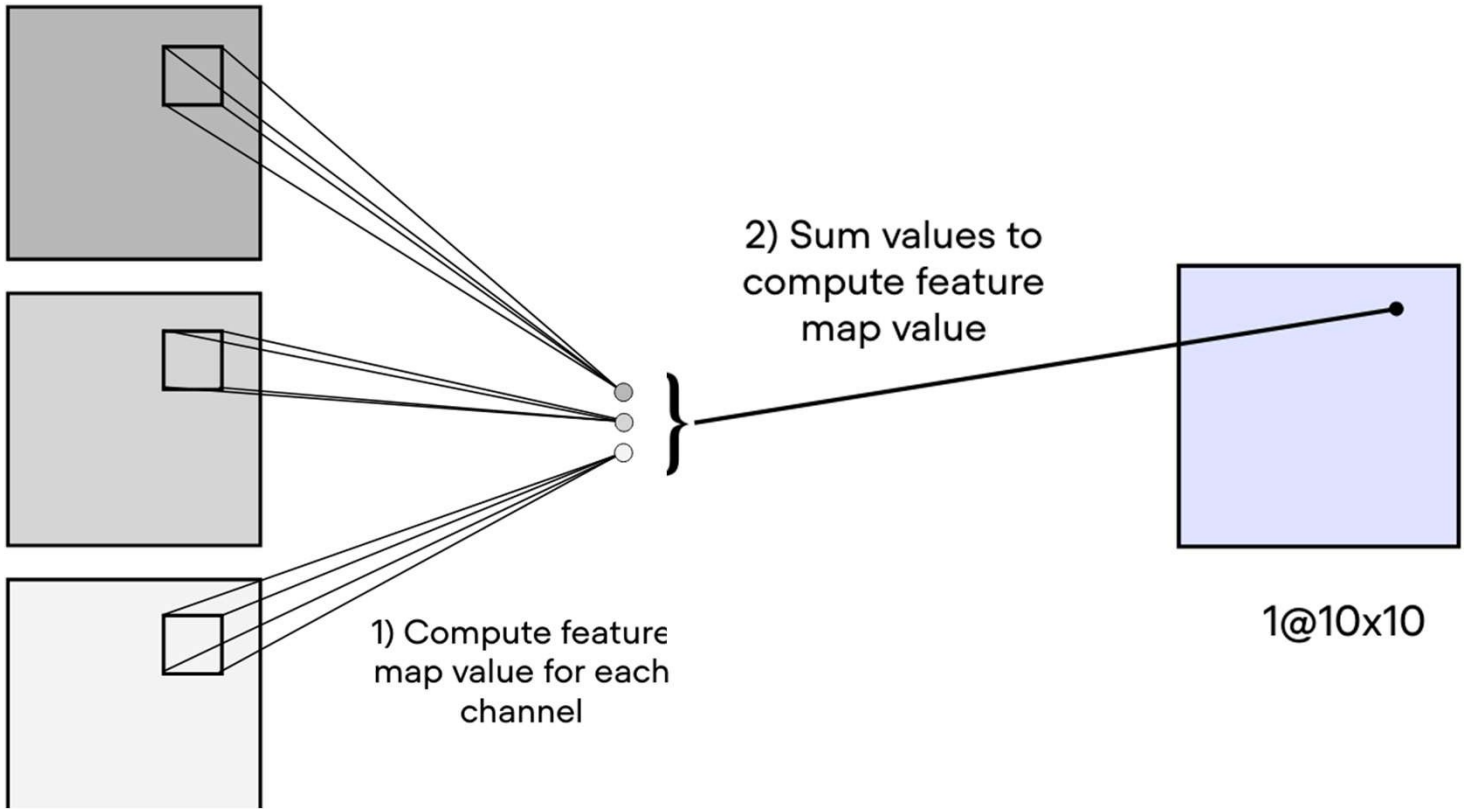
Filters always extend the full depth of the input.
(#Input channels == #Filter Channels)



3 input channel, 1 output channel

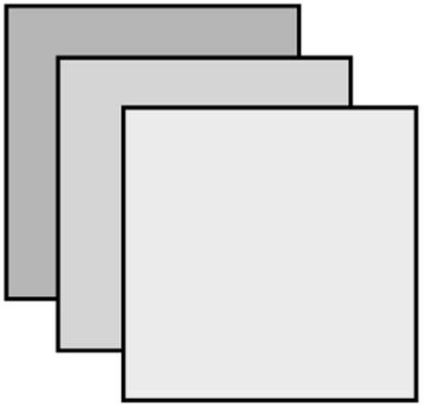


3 input channel, 1 output channel



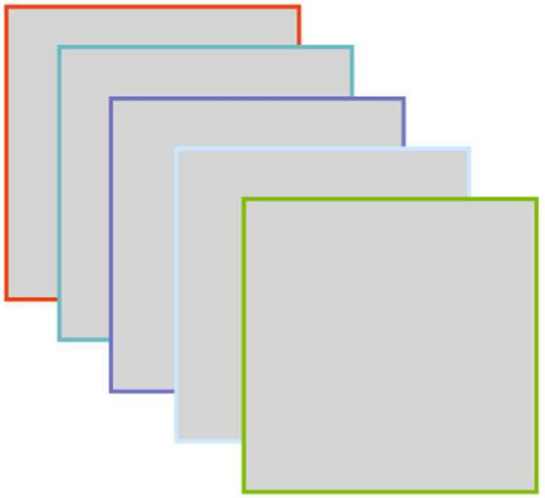
Multiple input and output channels

3 input channels



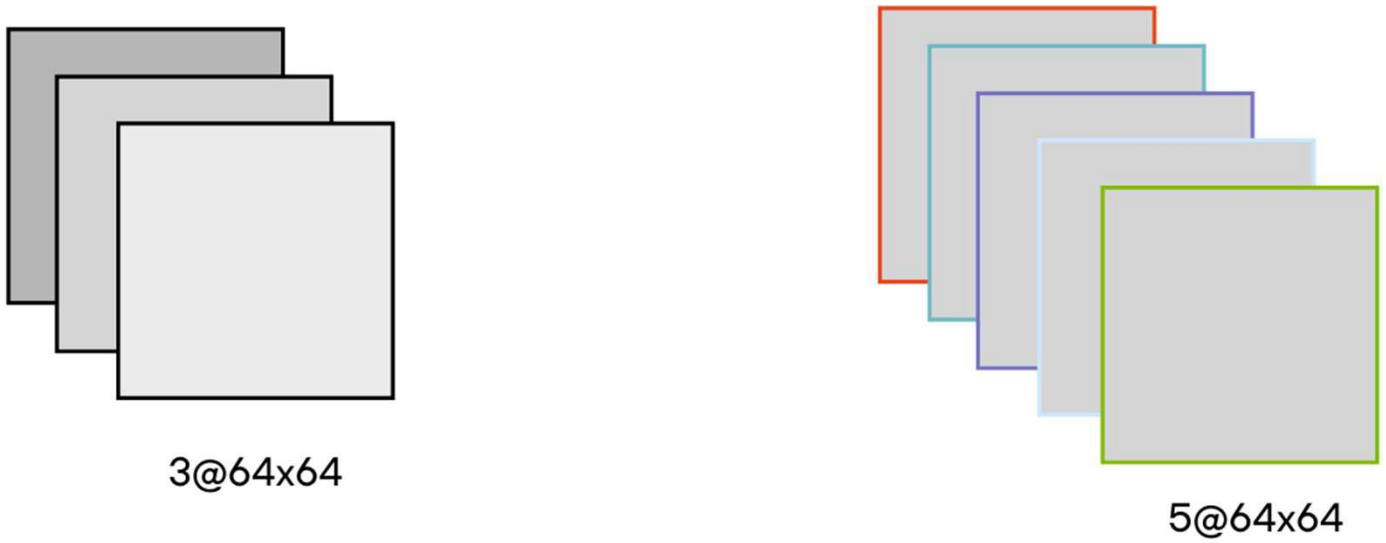
3@64x64

5 output channels

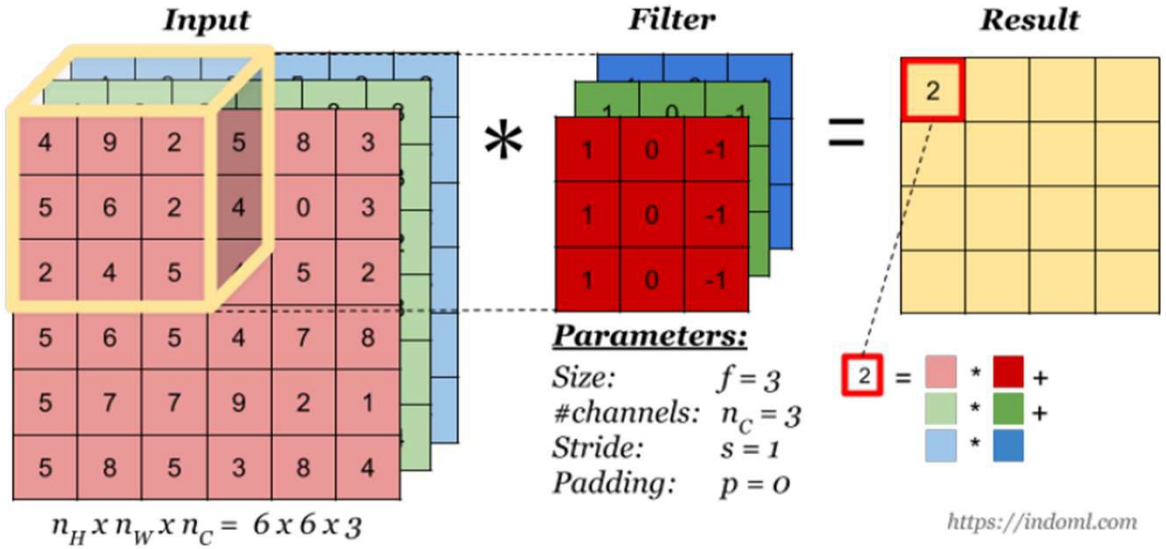


5@64x64

Multiple input and output channels

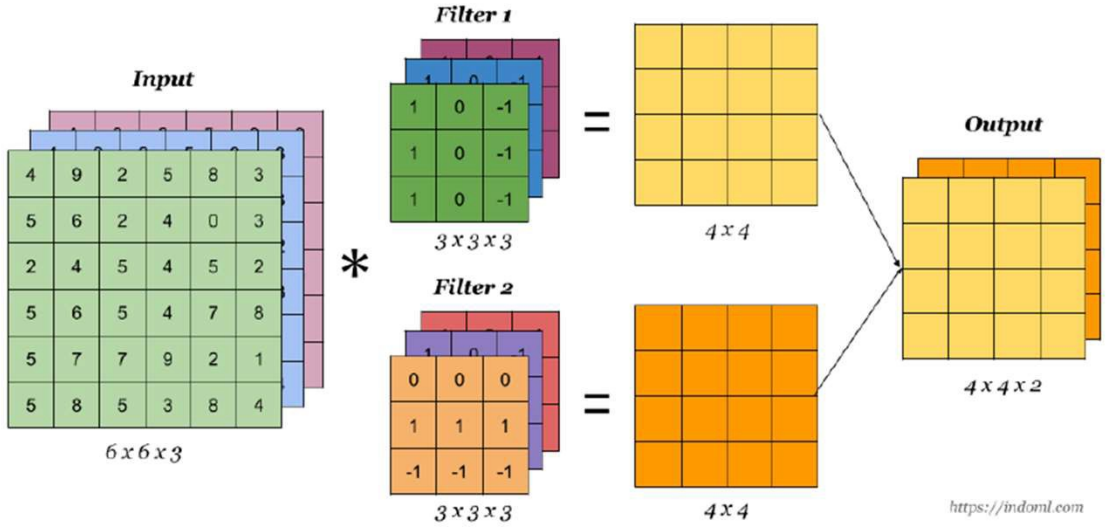


- As said before: ”Filters always extend the full depth of the input. (#Input channels == #Filter Channels)”

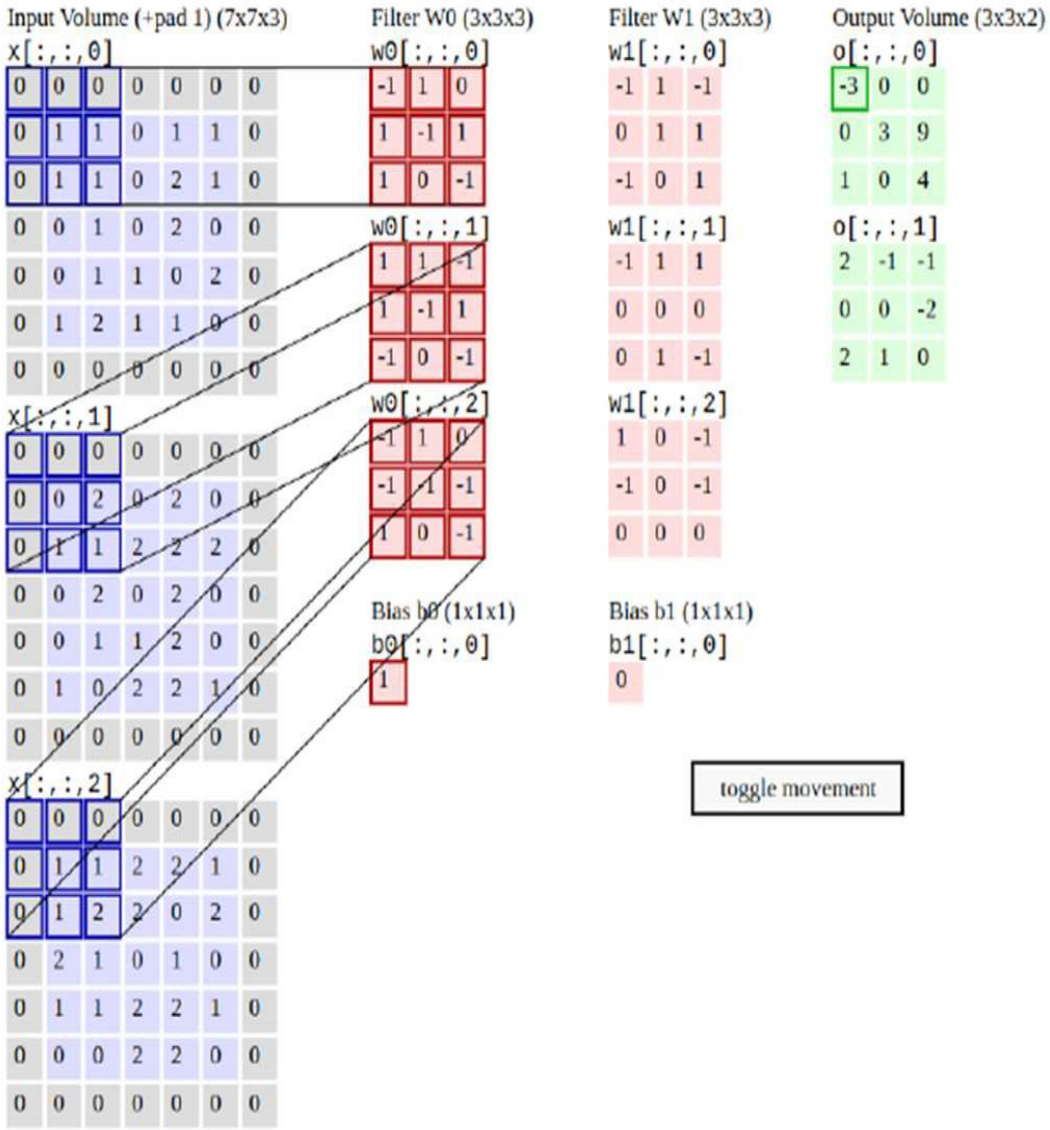


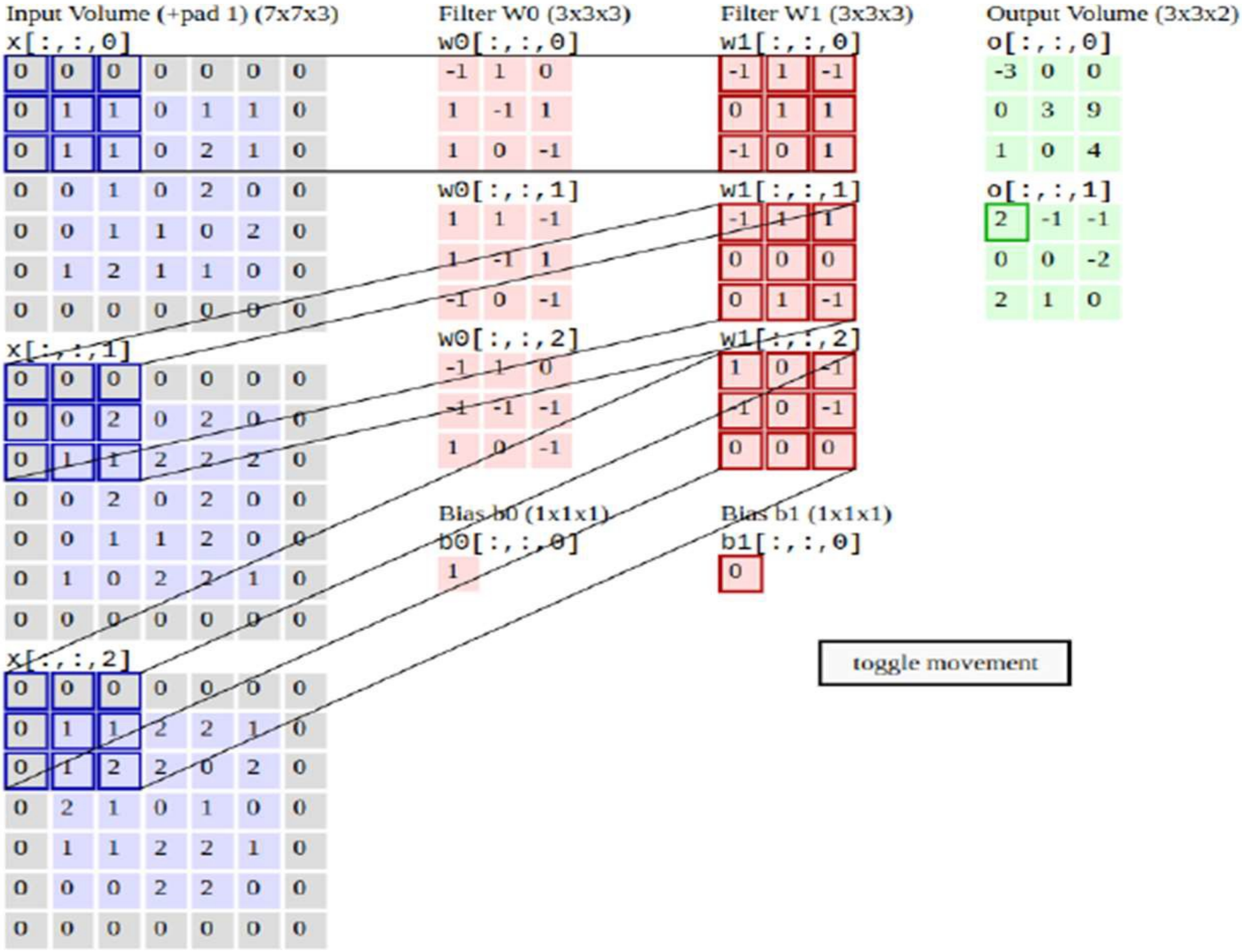
- As said before: ”Filters always extend the full depth of the input volume. (#Input channels == #Filter Channels)”

- Each filter results in one output channel. We can apply multiple different filters to obtain multiple output channels. Each channel can learn something distinct.

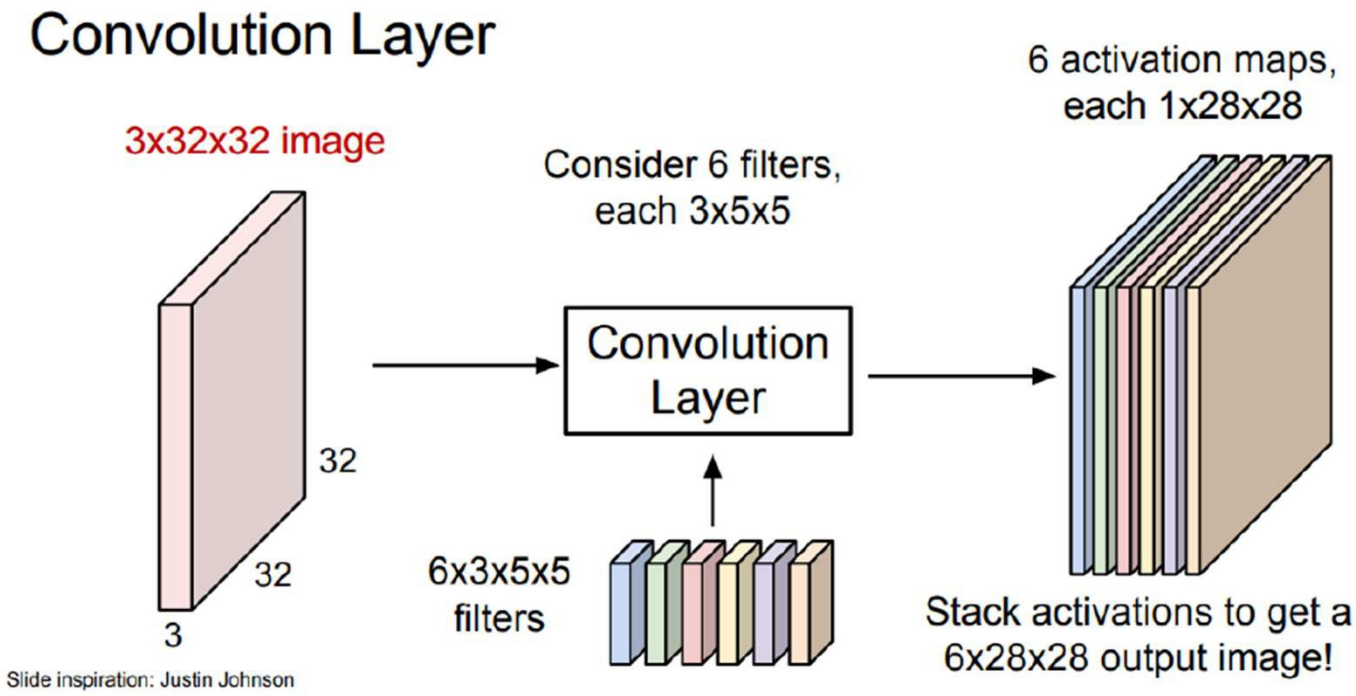


Let's have a closer look to how the calculations with more than one filter work:




Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 1 0 1 1 0
0 1 1 0 2 1 0
0 0 1 0 2 0 0
0 0 1 1 0 2 0
0 1 2 1 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 0 2 0 2 0 0
0 1 1 2 2 2 0
0 0 2 0 2 0 0
0 0 1 1 2 0 0
0 1 0 2 2 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 1 1 2 2 1 0
0 1 2 2 0 2 0
0 2 1 0 1 0 0
0 1 1 2 2 1 0
0 0 0 2 2 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
-1 1 0
1 -1 1
1 0 -1
w0[:,:,1]
1 1 -1
1 -1 1
-1 0 -1
w0[:,:,2]
-1 1 0
-1 -1 -1
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 -1
0 1 1
-1 0 1
w1[:,:,1]
-1 1 1
0 0 0
0 1 -1
w1[:,:,2]
1 0 -1
-1 0 -1
0 0 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-3 0 0
0 3 9
1 0 4
o[:,:,1]
2 -1 -1
0 0 -2
2 1 0
toggle movement

- We apply several filters and stack the output together to get a multilayer output, with each layer representing something it learned.

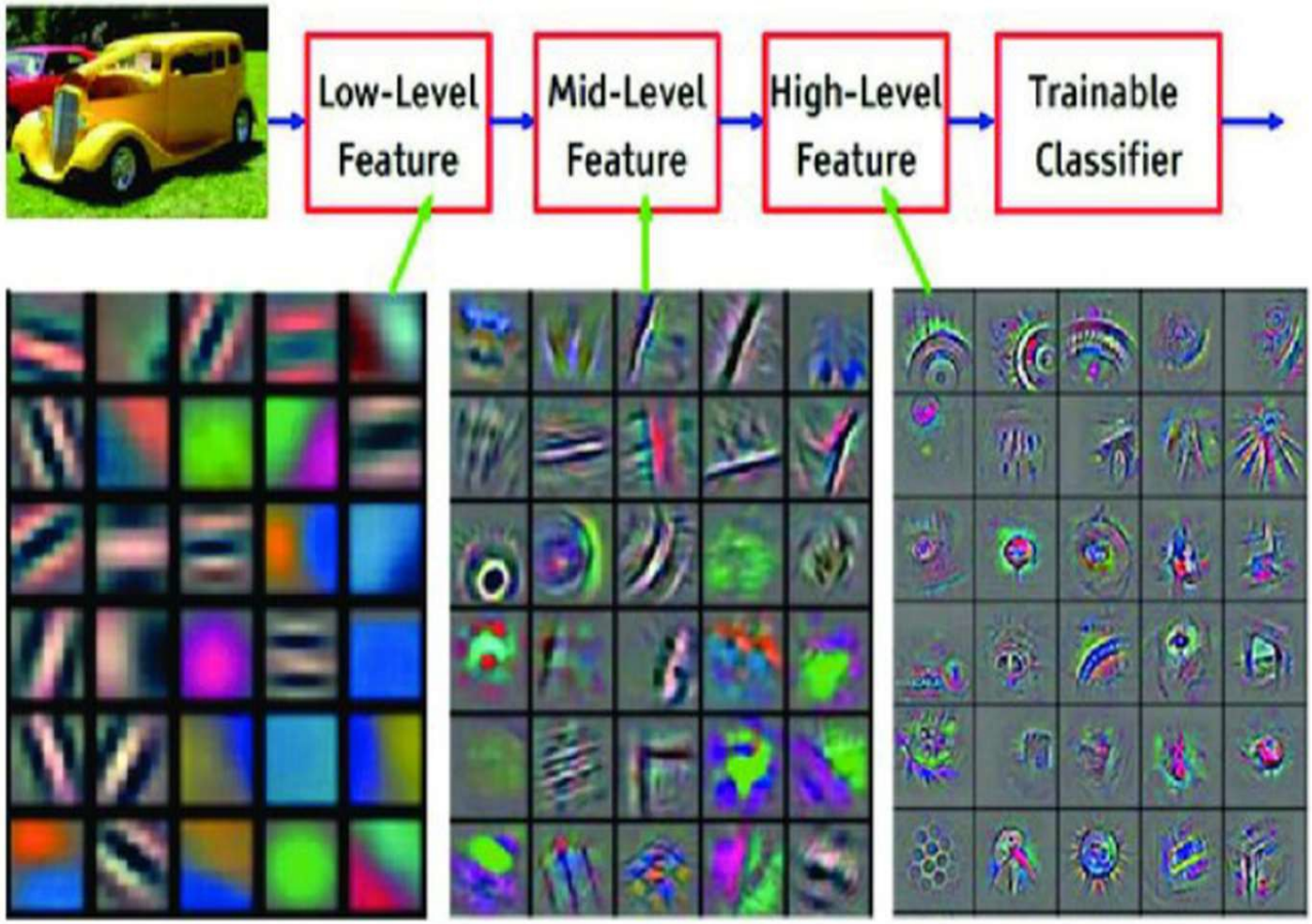


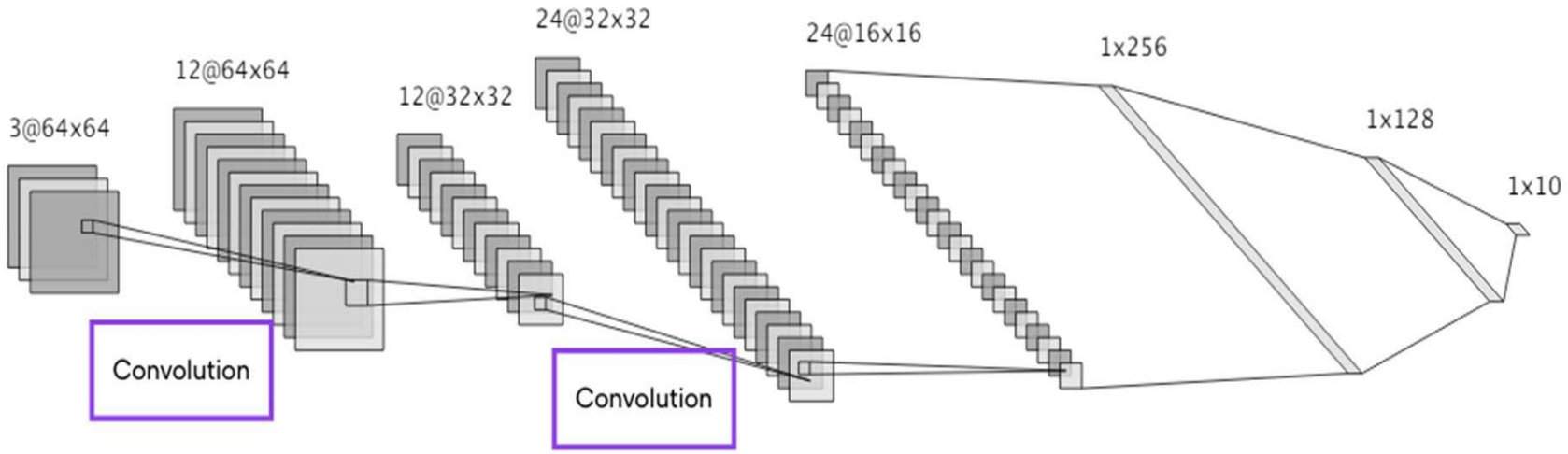
Question: What do convolutional filters at different levels of a ConvNet learn?

Answer:

- Filters in the **early layers** typically detect **simple features** such as edges, textures, and basic shapes.
- As we move **deeper** into the network, the filters learn **more complex and abstract features**, such as specific parts of objects (e.g., a nose, eyes, or other high-level patterns).

- We apply several filters and stack the output together to get a multilayer output, with each layer representing something it learned.
- Some kernels learn to recognize vertical lines, some circles, and some, specific objects:



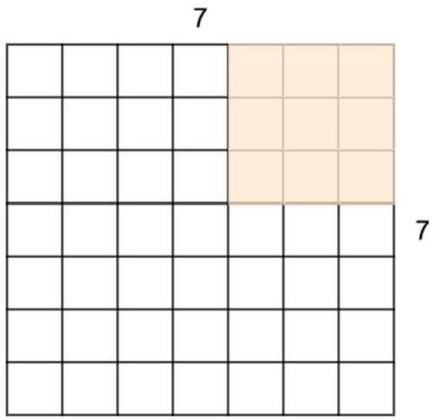

3@64x64
12@64x64
Convolution
12@32x32
24@32x32
Convolution
24@16x16
1x256
1x128
1x10

What is a Stride

The amount of movement between applications of the filter to the input image is referred to as the stride, and it is almost always symmetrical in height and width dimensions.

Closer look

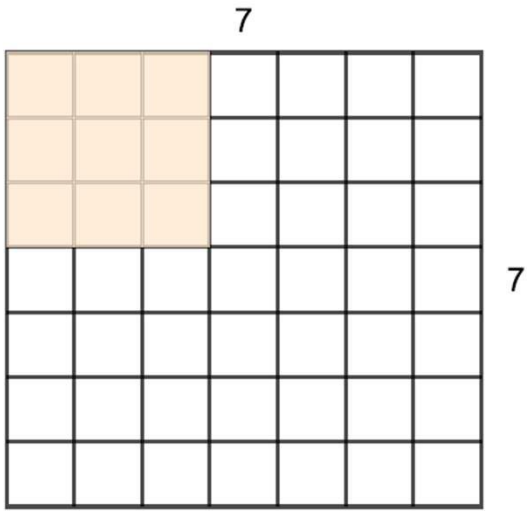
- 7x7 input with 3x3 filter
- This was a Stride 1 filter
- => Outputs 5x5



Strides

Now let's use Stride 2

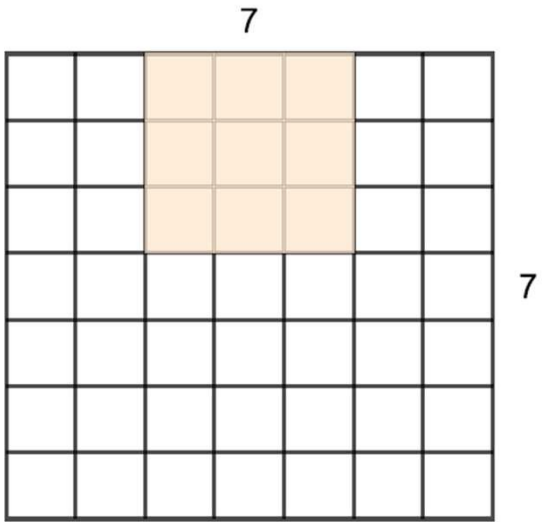
- 7x7 input with 3x3 filter



Strides

Now let's use Stride 2

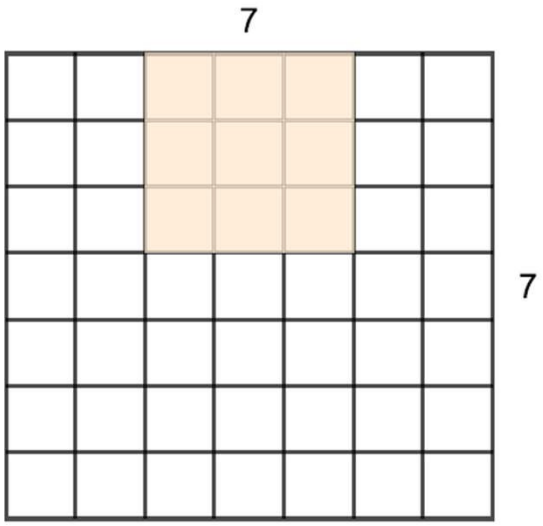
- 7x7 input with 3x3 filter



Strides

Now let's use Stride 2

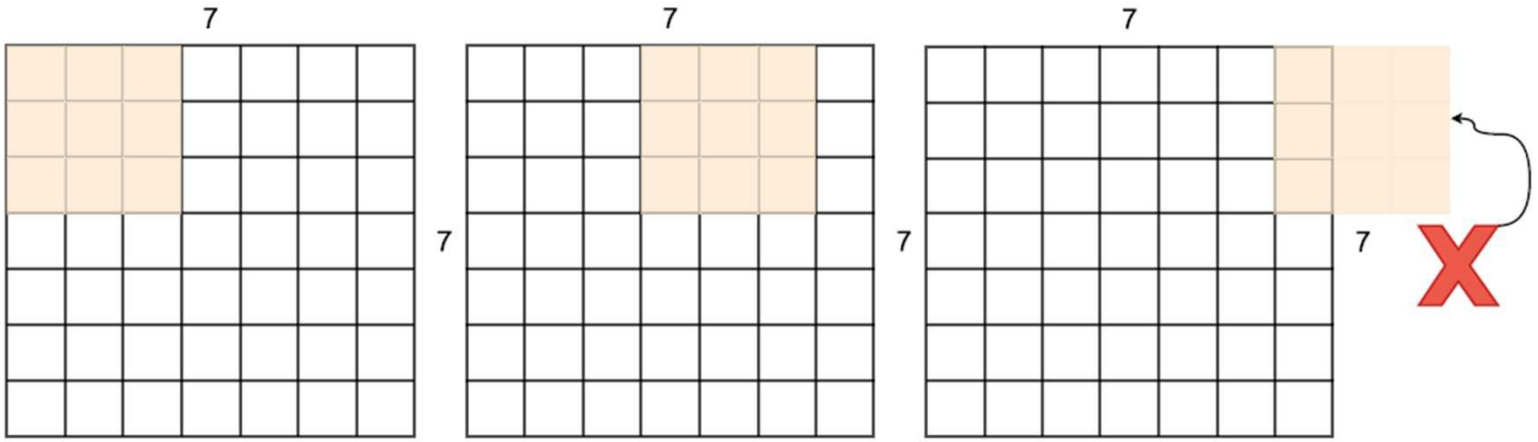
- 7x7 input with 3x3 filter



Strides

Stride 3?

- 7x7 input with 3x3 filter

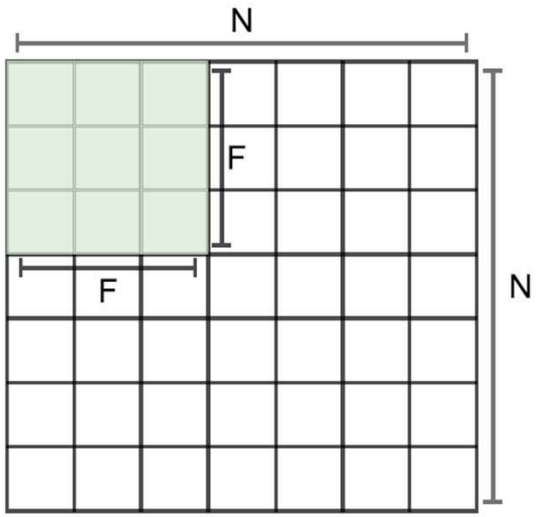


Strides

So 7x7 input with 3x3 filter and stride 3 doens't work!

Let't do the calculations:

$$OutputSize = (N - F)/Stride + 1$$

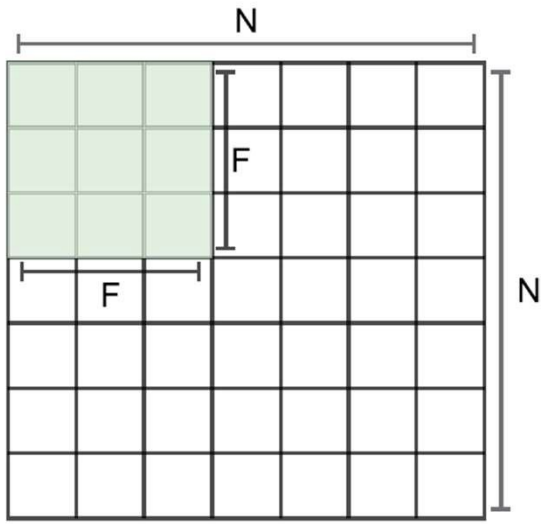


Strides

$$OutputSize = (N - F)/Stride + 1$$

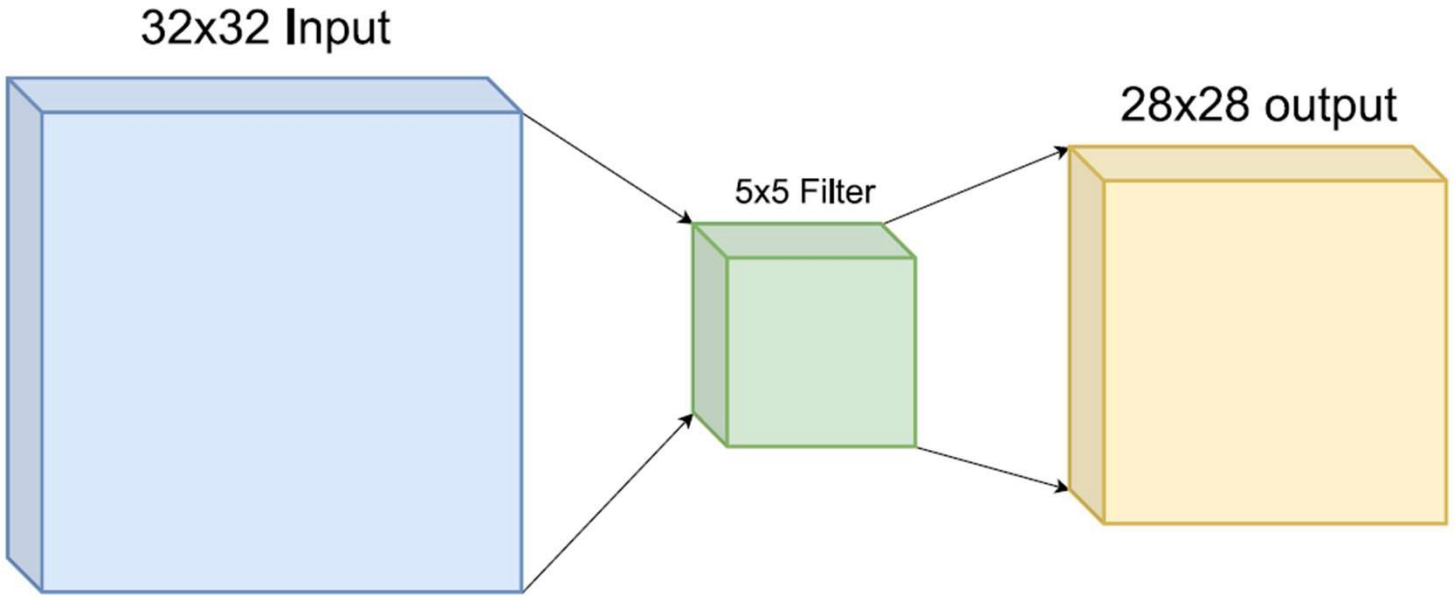
N = 7, F = 3 =>

- Stride 1 ==> $(7 - 3)/1 + 1 = 5$
- Stride 2 ==> $(7 - 3)/2 + 1 = 3$
- Stride 3 ==> $(7 - 3)/3 + 1 = 2.33$:))



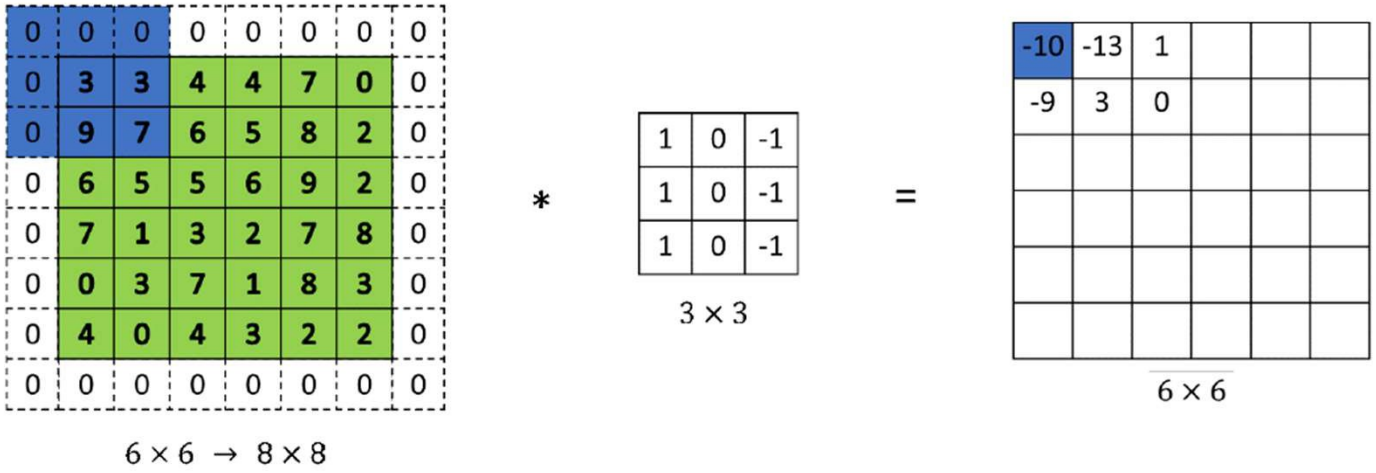
Padding

1. Borders don't get enough attention.
2. Outputs shrink!



Padding

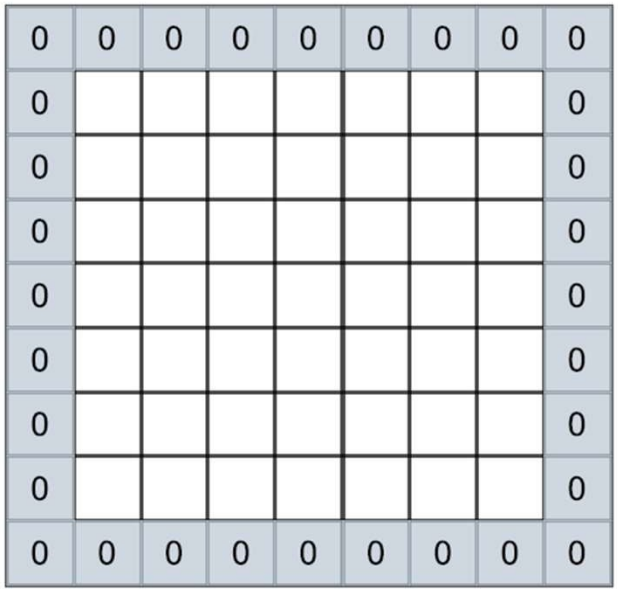
- We can use Padding to preserve the dimensionality
- It ensures that all pixels are used equally frequently



Padding

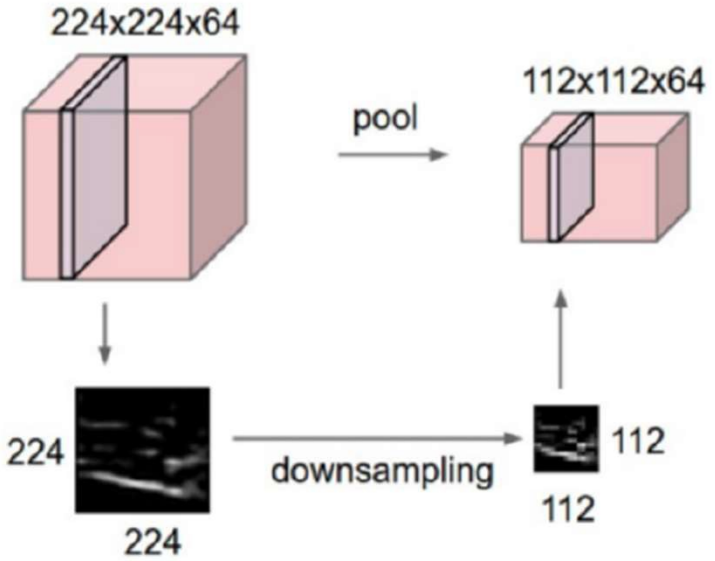
$$OutputSize = (N + 2P - F)/Stride + 1$$

It is common to use $P = (F - 1)/2$ with stride 1 to preserve the input size.

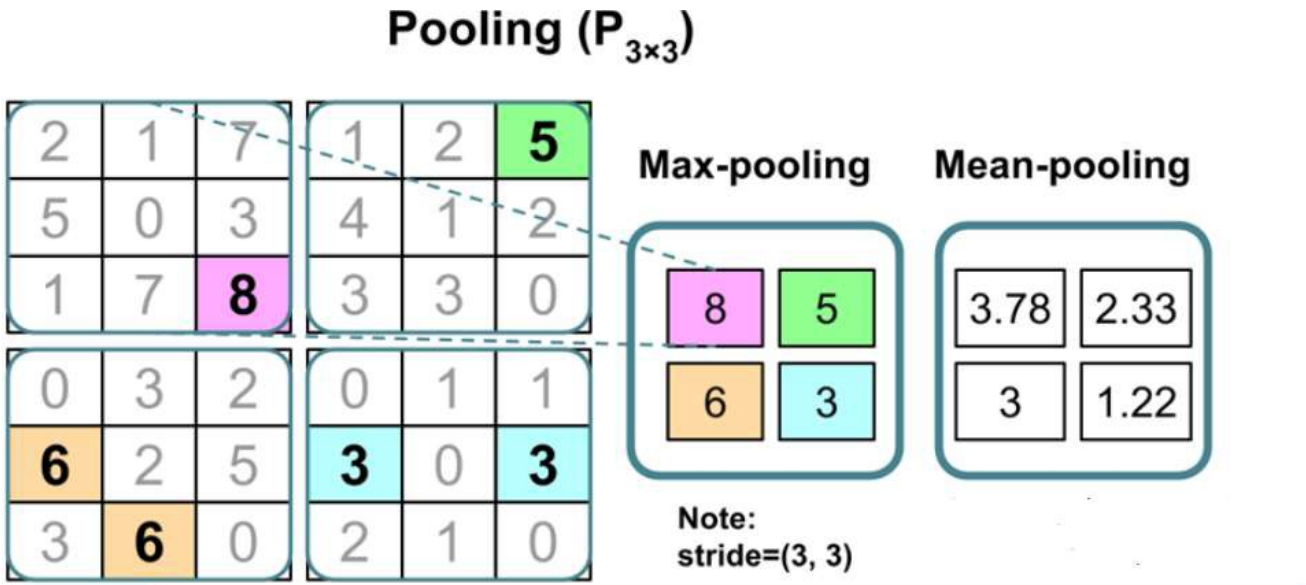


Pooling

Pooling is performed in neural networks to reduce variance and computation complexity



Pooling Layers Can Help With Local Invariance



Note that typical pooling layers do not have any learnable parameters

Downside: Information is lost.
May not matter for classification, but applications where relative position is important (like face recognition)

In practice for CNNs: some image preprocessing still recommended

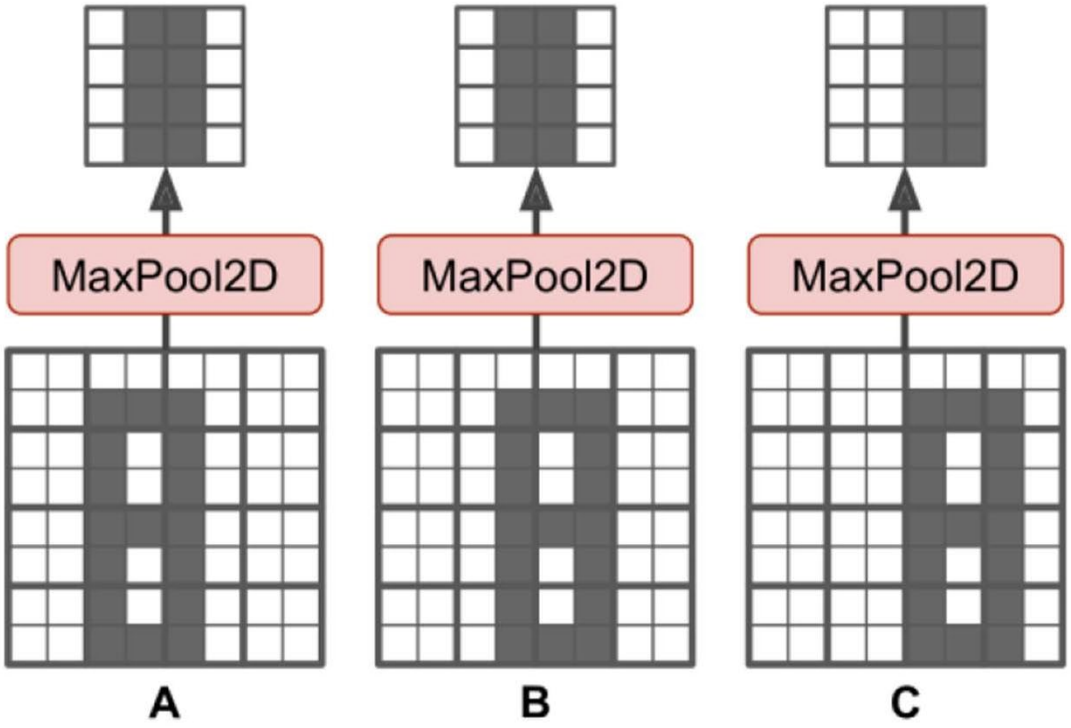
Pooling benefits

A Pooling layer's goal is to sub-sample the input in order to reduce:

- The computational load
- The memory usage
- The number of parameters
- Limiting the risk of overfitting

Max Pooling

Most common type of pooling layer
Invariance to small translations



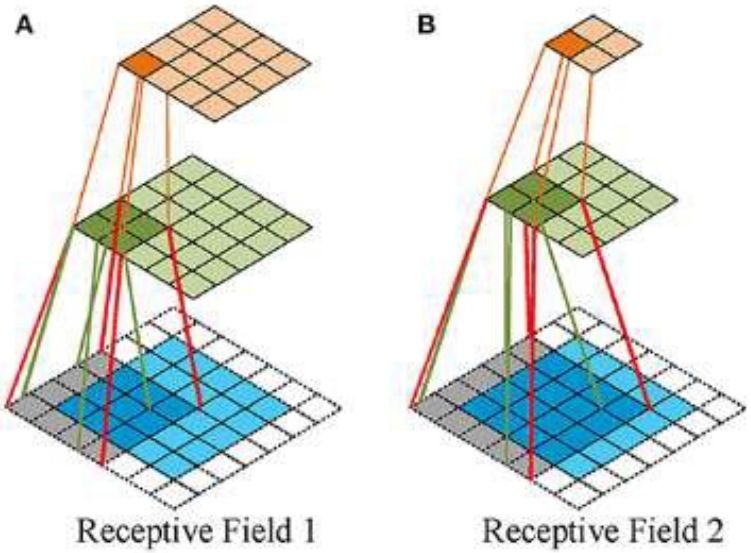
Smaller filter size or larger ?

- Using smaller filter size is better or larger?

Distributing the scan

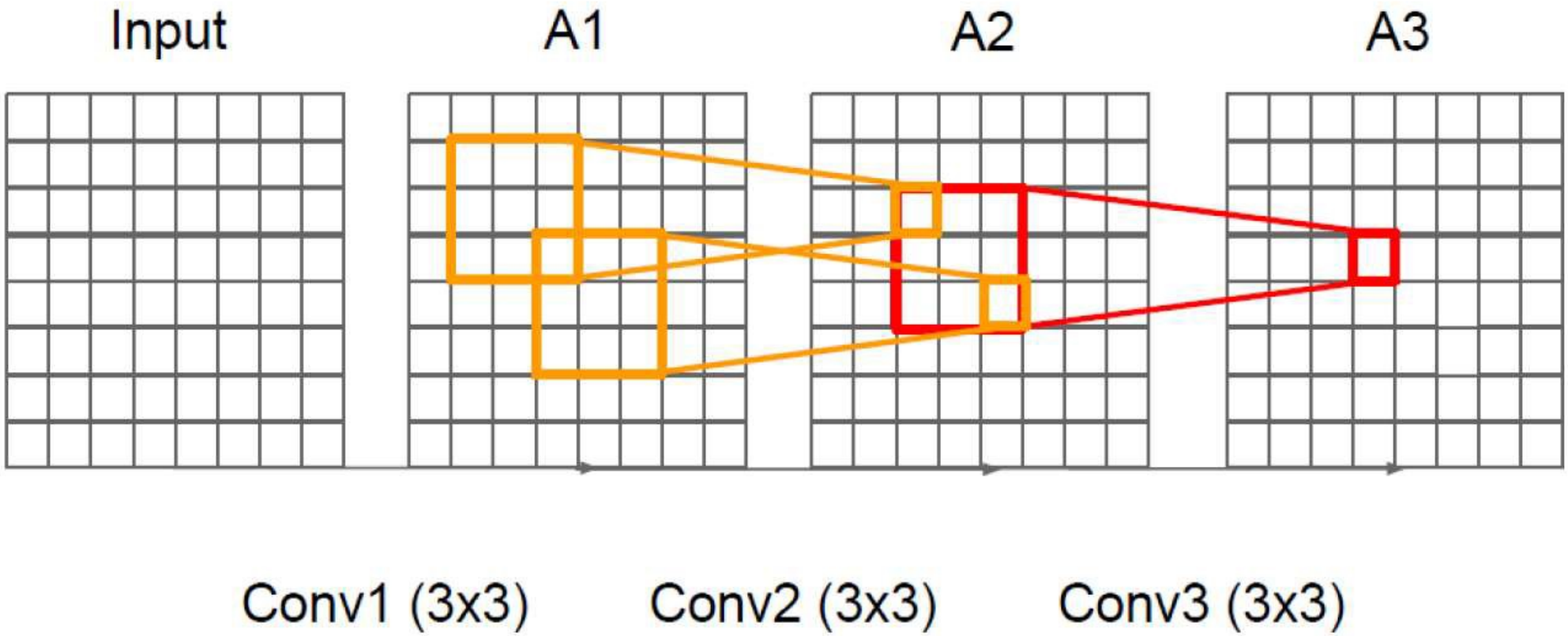
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates blocks of outputs from the first layer

Receptive field

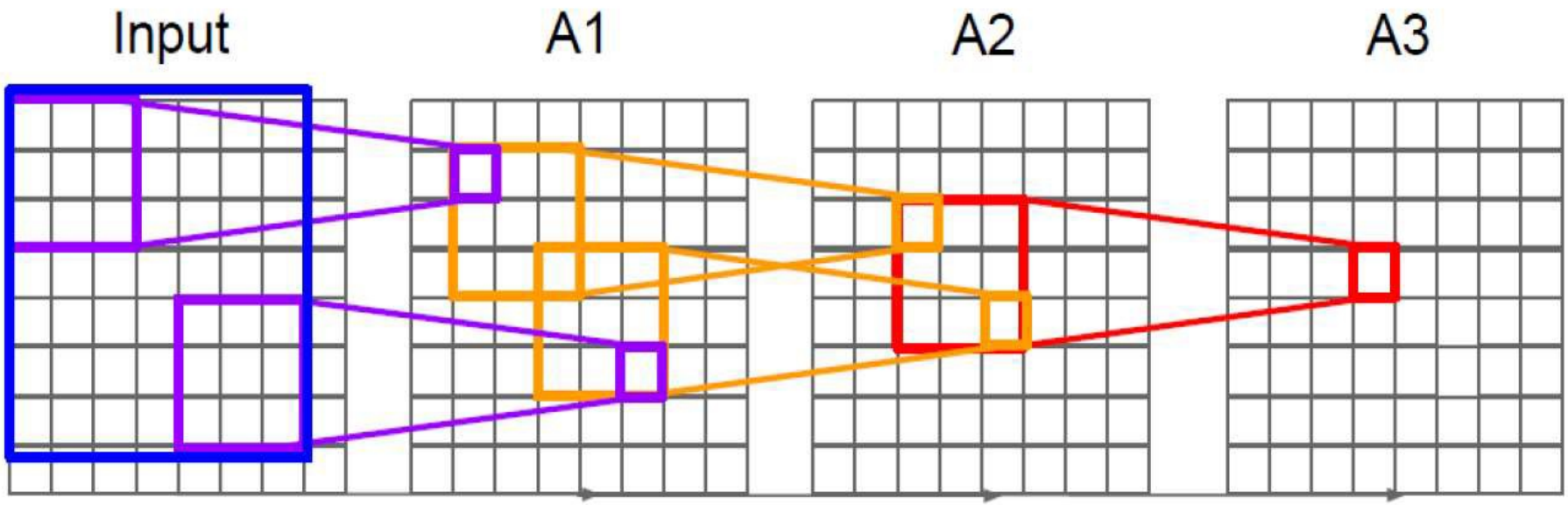


Receptive field a deep CNN. (A) The first convolving: a 3 × 3 kernel over a 5 × 5 input padded with a 1 × 1 border of zeros using unit strides. The second convolving: a 2 × 2 kernel using unit strides. (B) The first convolving: a 4 × 4 kernel over a 5 × 5 input padded with a 1 × 1 border of zeros using unit strides. The second convolving: a 2 × 2 kernel using 2 × 2 strides.

What is the effective receptive field of three 3×3 conv (stride 1) layers?



What is the effective receptive field of three 3×3 conv (stride 1) layers?



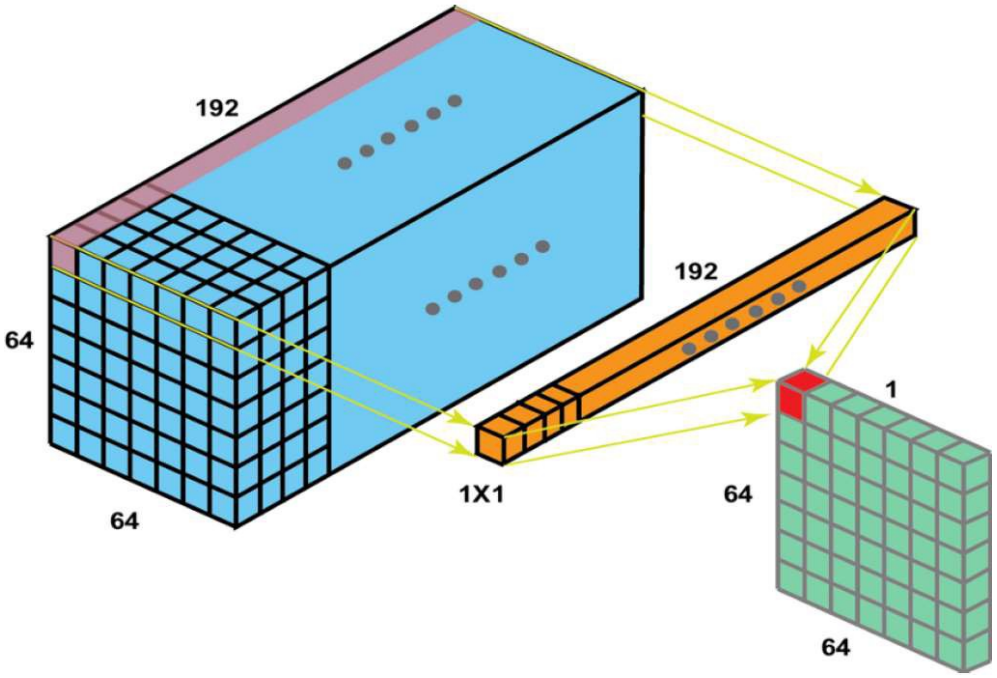
Power of small filter

- Suppose input is H*W*C and we use convolutions of C filters

Stride 1 and Padding to preserve H , W

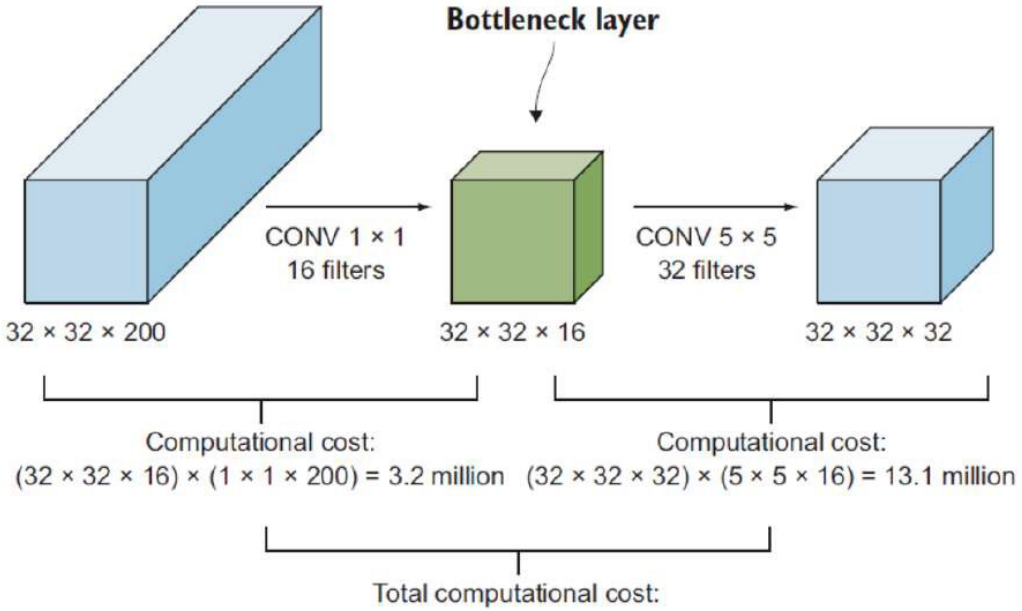
- One convolution with 7*7 filters
  - Number of weights : (7*7*C) *C(for the number of filter) = $49\,C^2$
- 3 convolution with 3*3 filters
  - Number of weights: (3*3*C) *C + (3*3)C*C + (3*3*C)*C = $27\,C^2$

1*1 Convolution



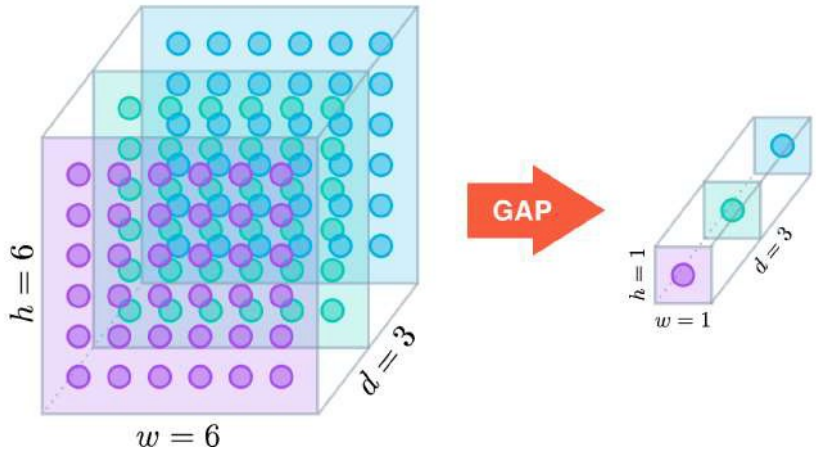
1*1 Convolution use case

- Assume we want to transform a $32 \times 32 \times 200$ tensor to a $32 \times 32 \times 32$ one using 32 5×5 filters. Thus, we need $(32 \times 32 \times 200) \times (5 \times 5 \times 32) \approx 163M$ operations!
- Instead we can use 1×1 convolution:



Global Average Pooling

Similar to max pooling layers, GAP layers are used to reduce the spatial dimensions of a three-dimensional tensor. However, GAP layers perform a more extreme type of dimensionality reduction, where a tensor with dimensions $h \times w \times d$ is reduced in size to have dimensions $1 \times 1 \times d$. GAP layers reduce each $h \times w$ feature map to a single number by simply taking the average of all hw values.



Why Gap

- GAP is used to replace the traditional fully connected layers in CNN.
- There is no parameter to optimize in the GAP thus overfitting is avoided at this layer
- GAP sums out the spatial information, thus it is more robust to spatial translations of the input.

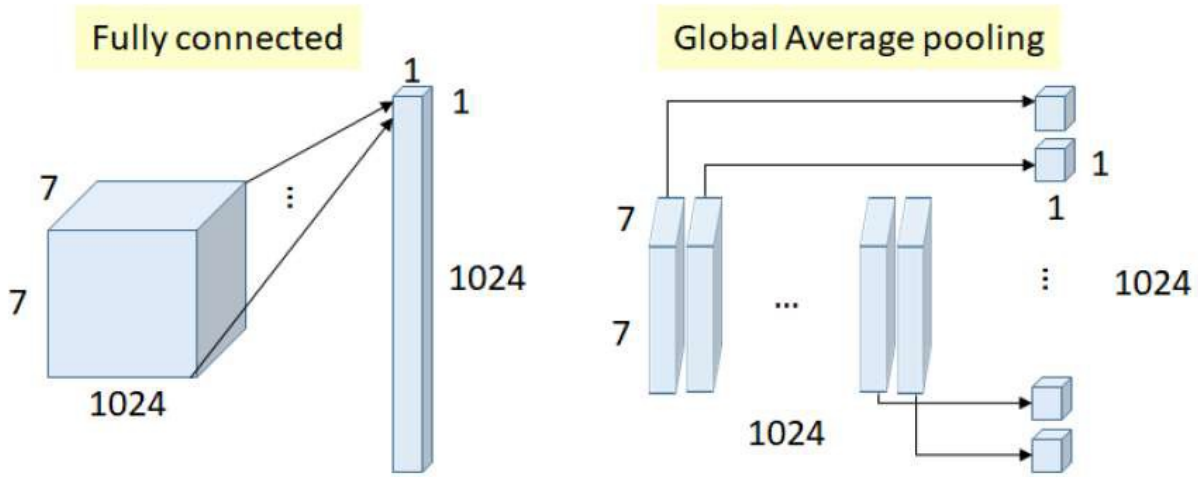


Figure: FC Layer vs GAP Layer