DEEP LEARNING

Lecture 5: Basics of Convolutional Neural Networks

Dr. Yang Lu

Department of Computer Science and Technology

luyang@xmu.edu.cn

CNN Applications

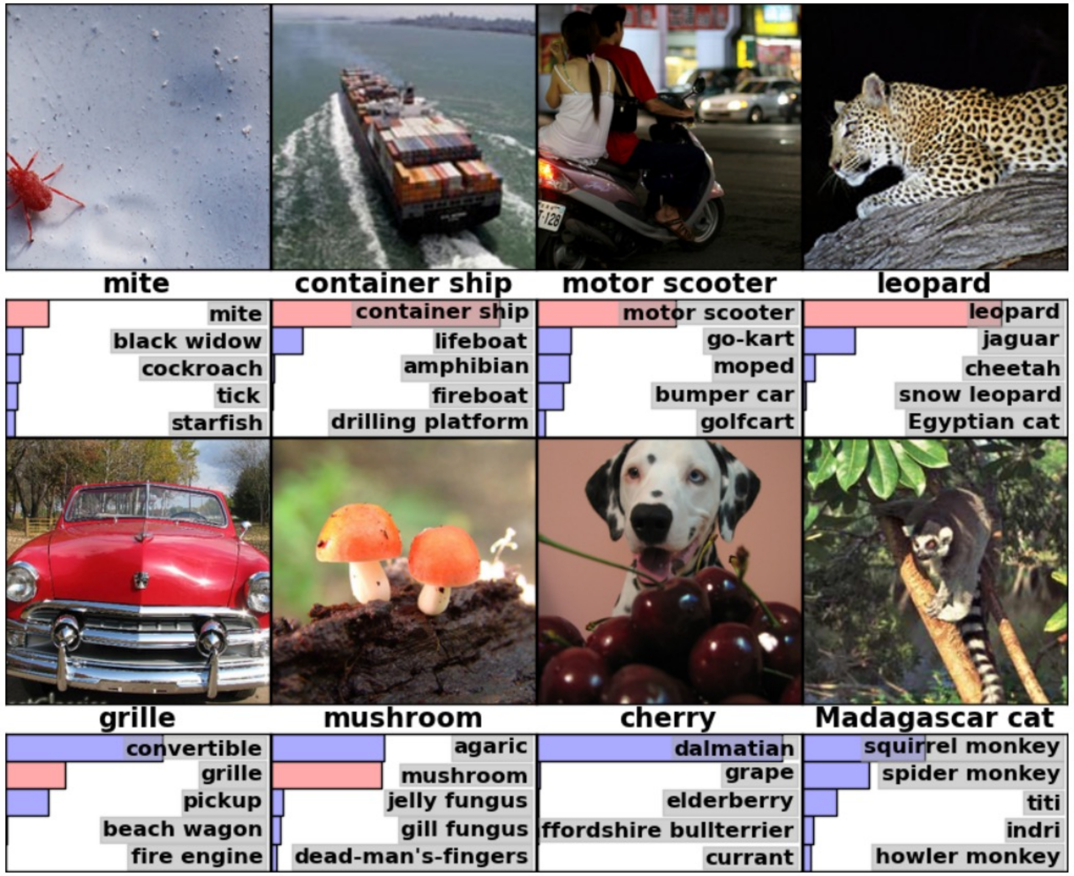


Image classification

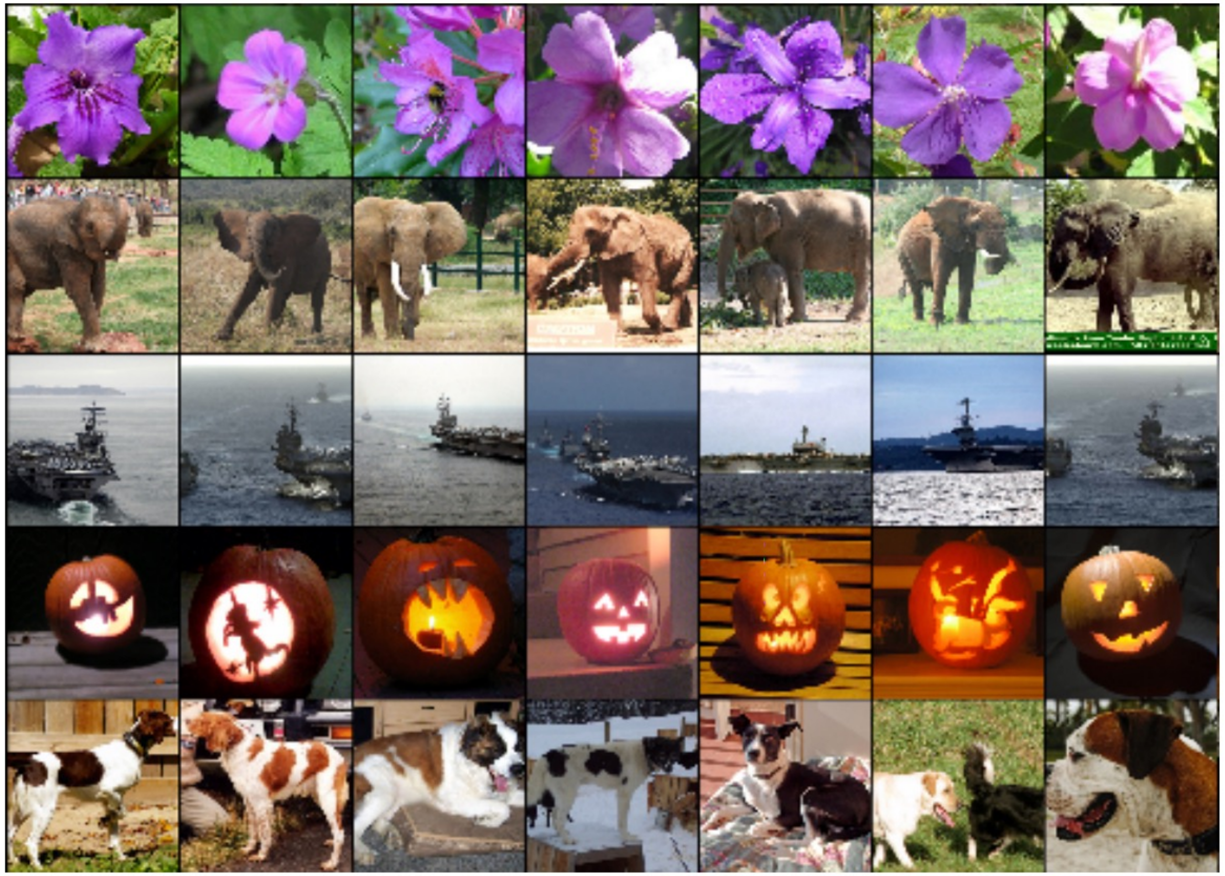
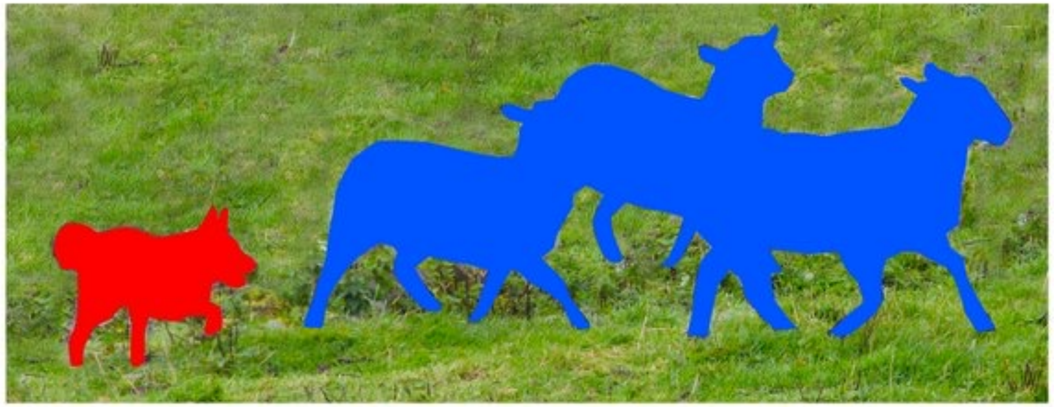


Image retrieval

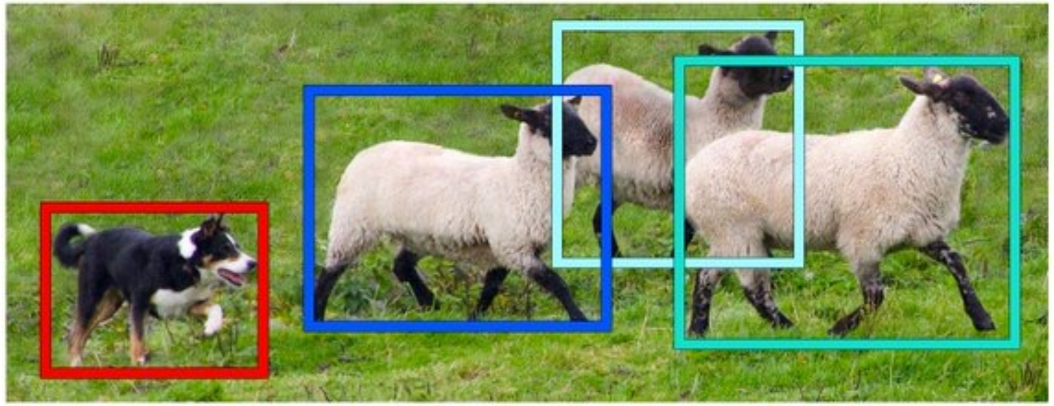
Image source: Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." In Advances in neural information processing systems, pp. 1097-1105. 2012.

CNN Applications

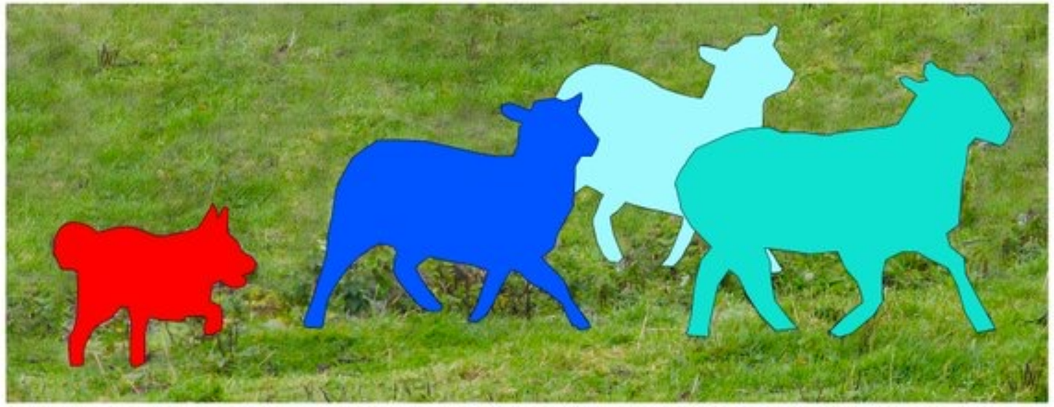
Image Recognition



Semantic Segmentation



Object Detection

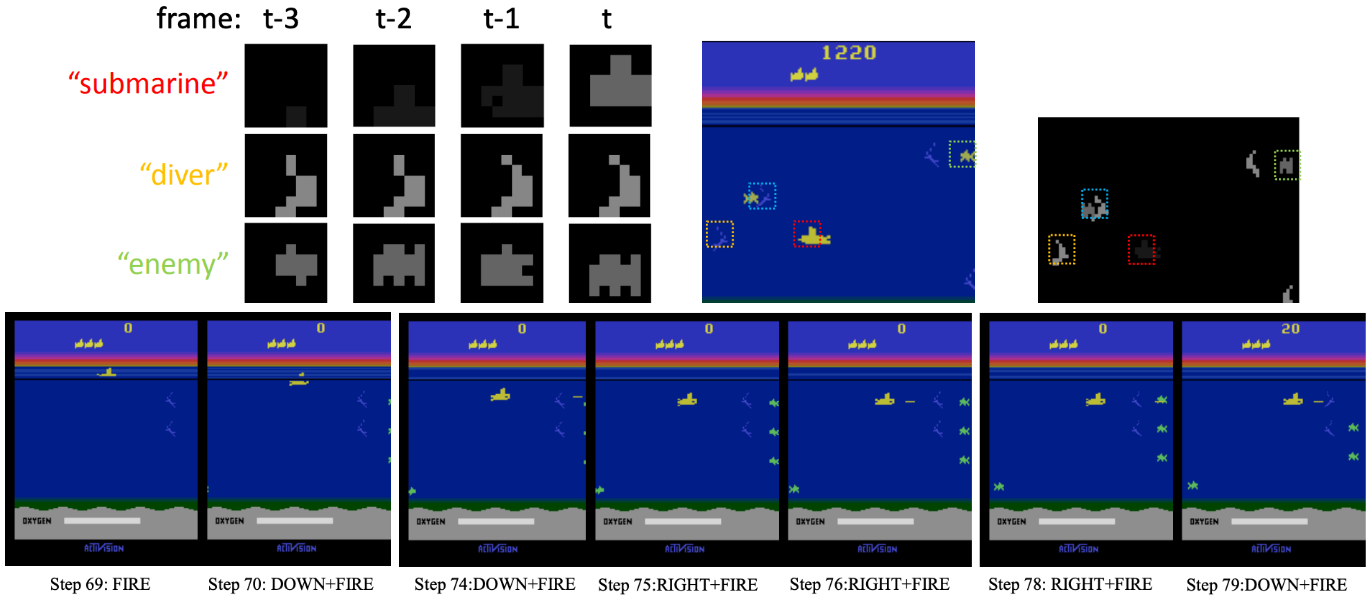


Instance Segmentation

Image source: https://towardsdatascience.com/how-to-handle-neural-network-output-a-practical-application-on-wildlife-drone-images-4670788d770d

CNN Applications

Pose estimation



Real-time Atari game play

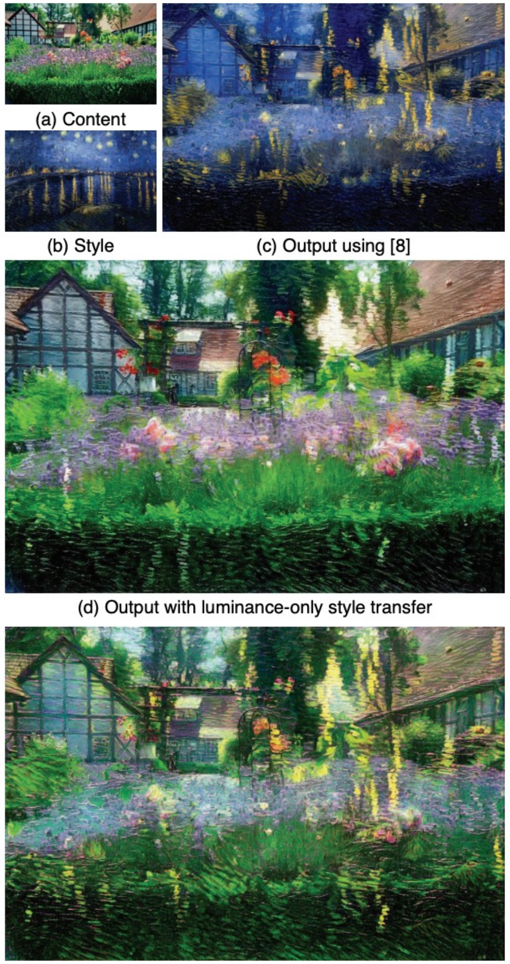
Image source: Toshev, Alexander, and Christian Szegedy. "Deeppose: Human pose estimation via deep neural networks." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1653-1660. 2014.
Guo, Xiaoxiao, Satinder Singh, Honglak Lee, Richard L. Lewis, and Xiaoshi Wang. "Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning." In Advances in neural information processing systems, pp. 3338-3346. 2014.

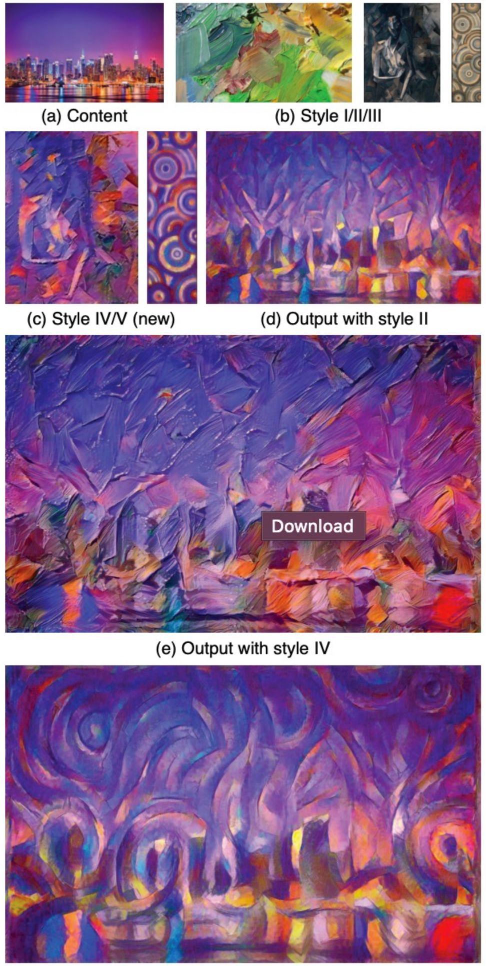
CNN Applications

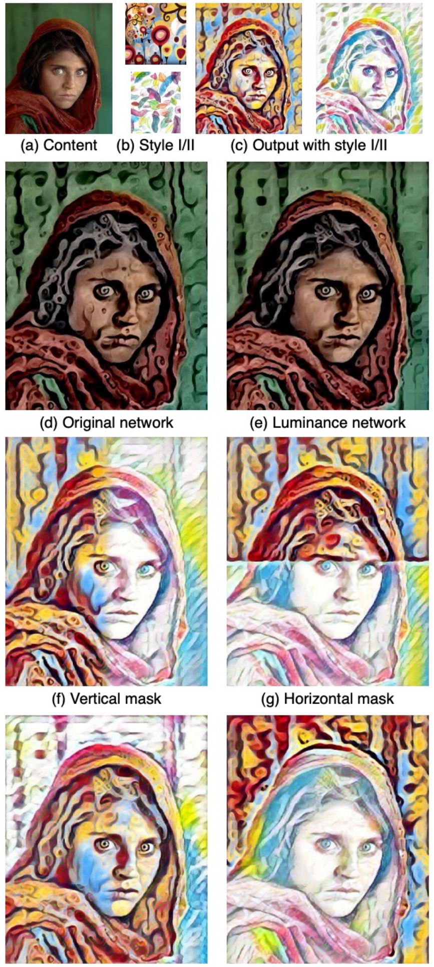
Image captioning

Image source: Vinyals, Oriol, Alexander Toshev, Samy Bengio, and Dumitru Erhan. "Show and tell: Lessons learned from the 2015 mscoco image captioning challenge." IEEE transactions on pattern analysis and machine intelligence 39, no. 4 (2016): 652-663.

CNN Applications







Style transfer

Image source: Gatys, Leon A., Alexander S. Ecker, Matthias Bethge, Aaron Hertzmann, and Eli Shechtman. "Controlling perceptual factors in neural style transfer." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3985-3993. 2017.

CNN Applications

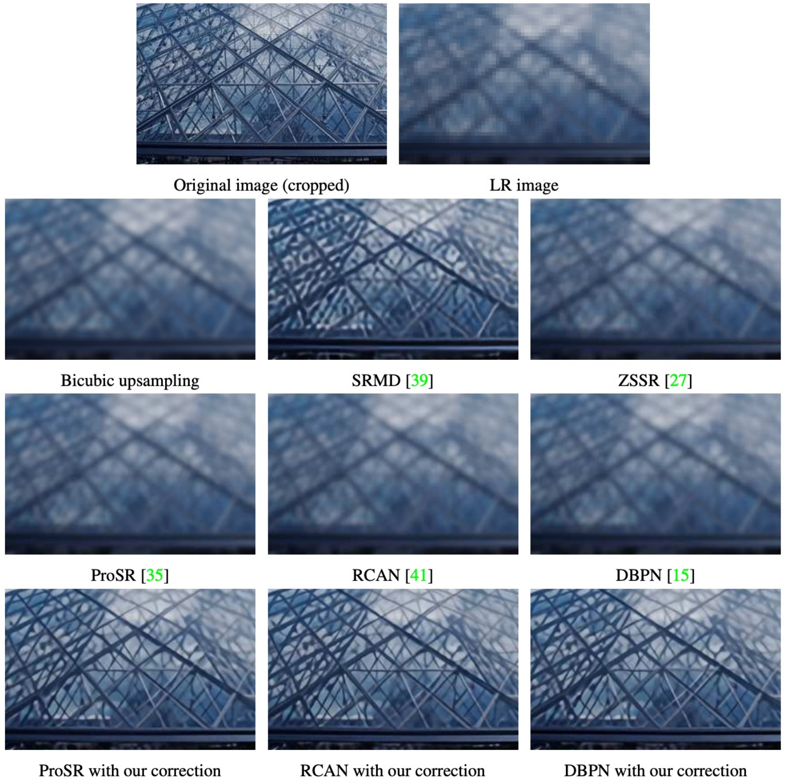
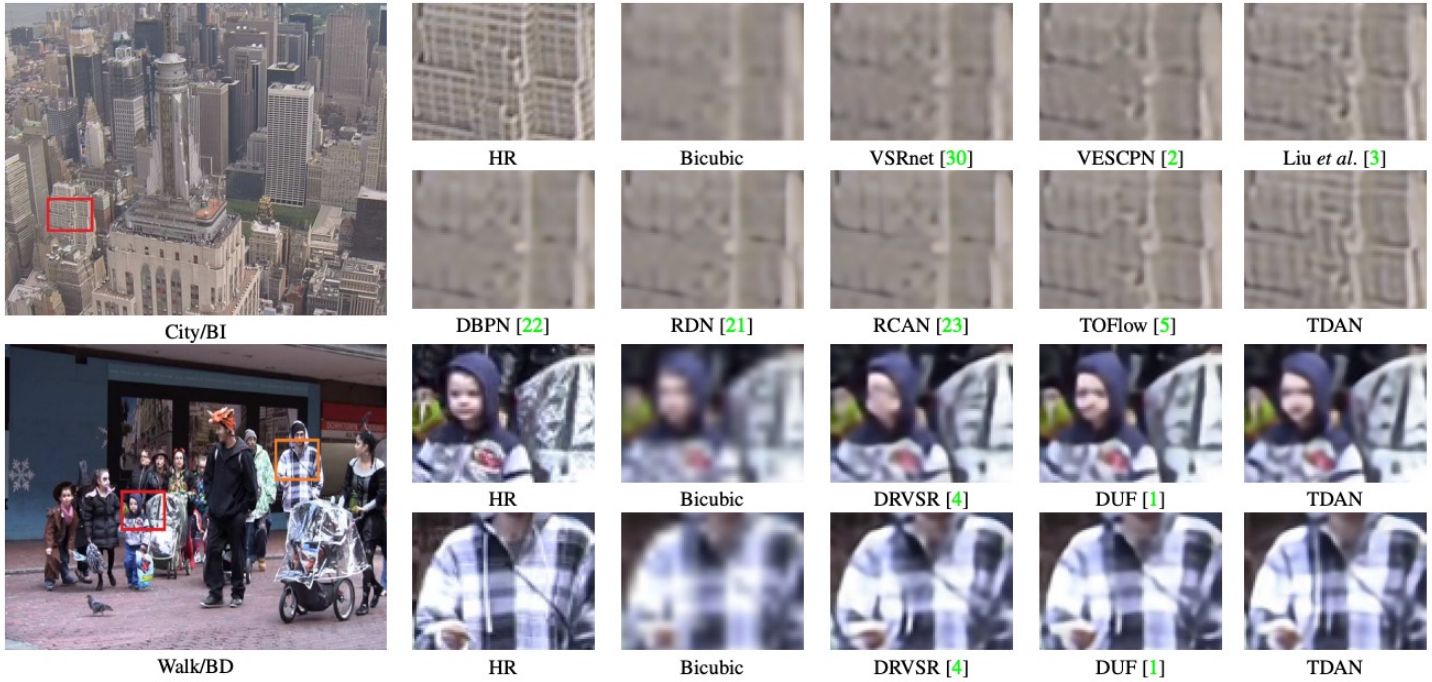


Image super-resolution

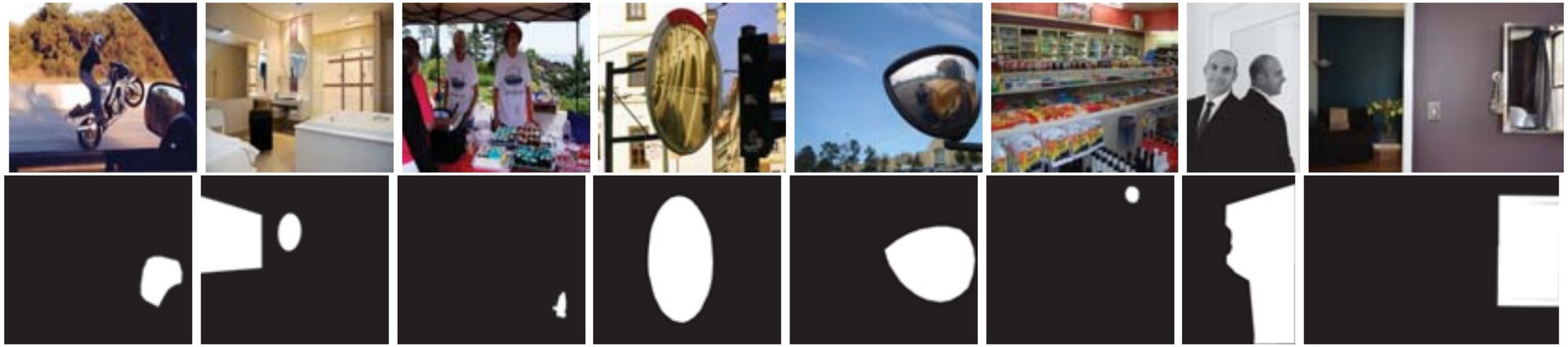


Video super-resolution

Image source: Hussein, Shady Abu, Tom Tirer, and Raja Giryes. "Correction Filter for Single Image Super-Resolution: Robustifying Off-the-Shelf Deep Super-Resolvers." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 1428-1437. 2020.
Tian, Yapeng, Yulun Zhang, Yun Fu, and Chenliang Xu. "TDAN: Temporally-Deformable Alignment Network for Video Super-Resolution." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 3360-3369. 2020.

CNN Applications

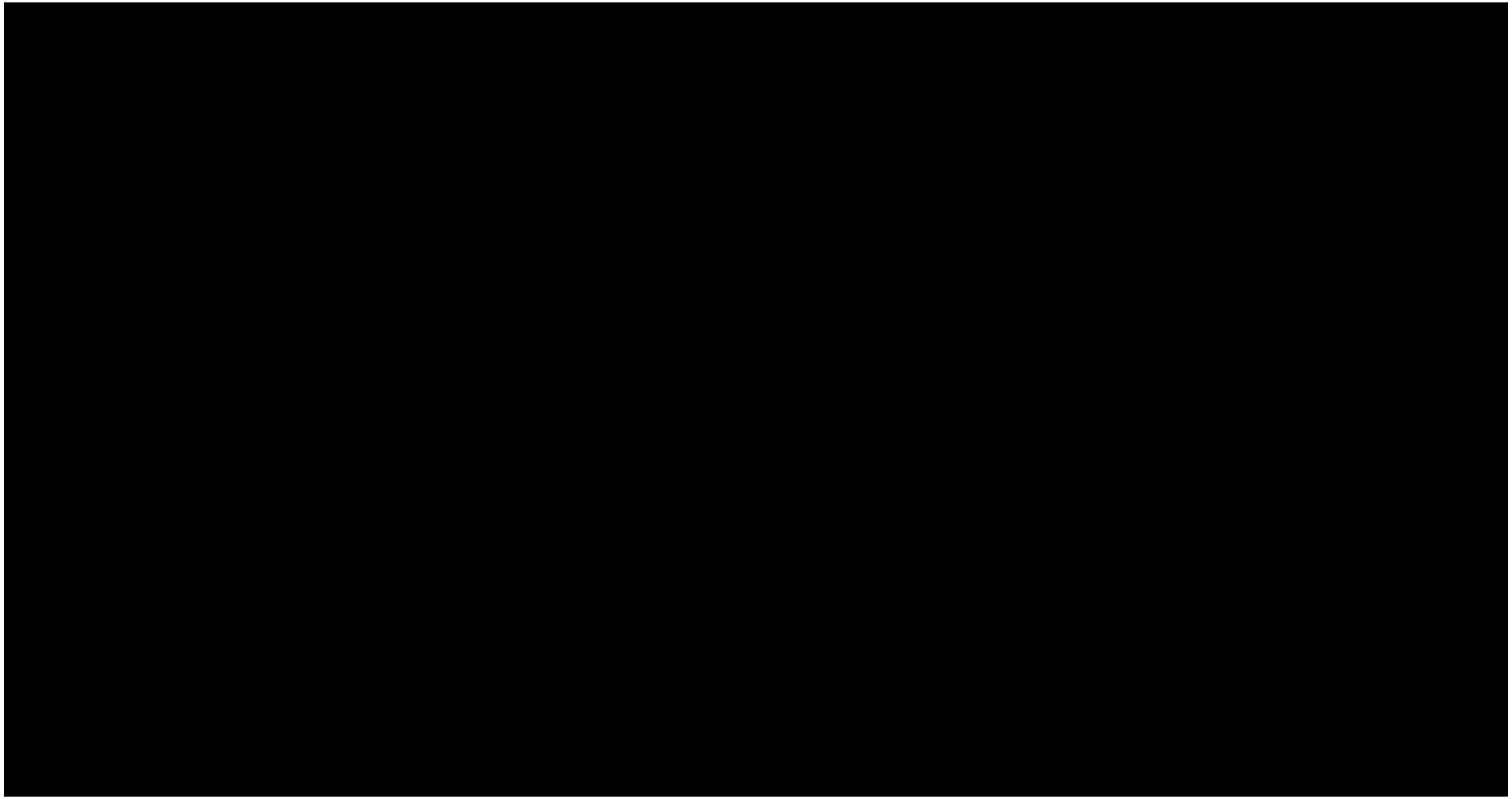
Rain and fog removal



Mirror detection

Image source: Li, Ruoteng, Robby T. Tan, and Loong-Fah Cheong. "All in One Bad Weather Removal Using Architectural Search." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 3175-3185. 2020.
Lin, Jiaying, Guodong Wang, and Rynson WH Lau. "Progressive Mirror Detection." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 3697-3705. 2020.

CNN Applications



Video frame interpolation

Video source: https://www.youtube.com/watch?v=5qAiffYFJh8

Paper source: https://arxiv.org/abs/2103.16206

Convolutional Neural Networks

- Recall in Lecture 2, we vectorize an image as the input of a neural network.
- What is the problem here?

Can maintain 2D structure through the whole network?

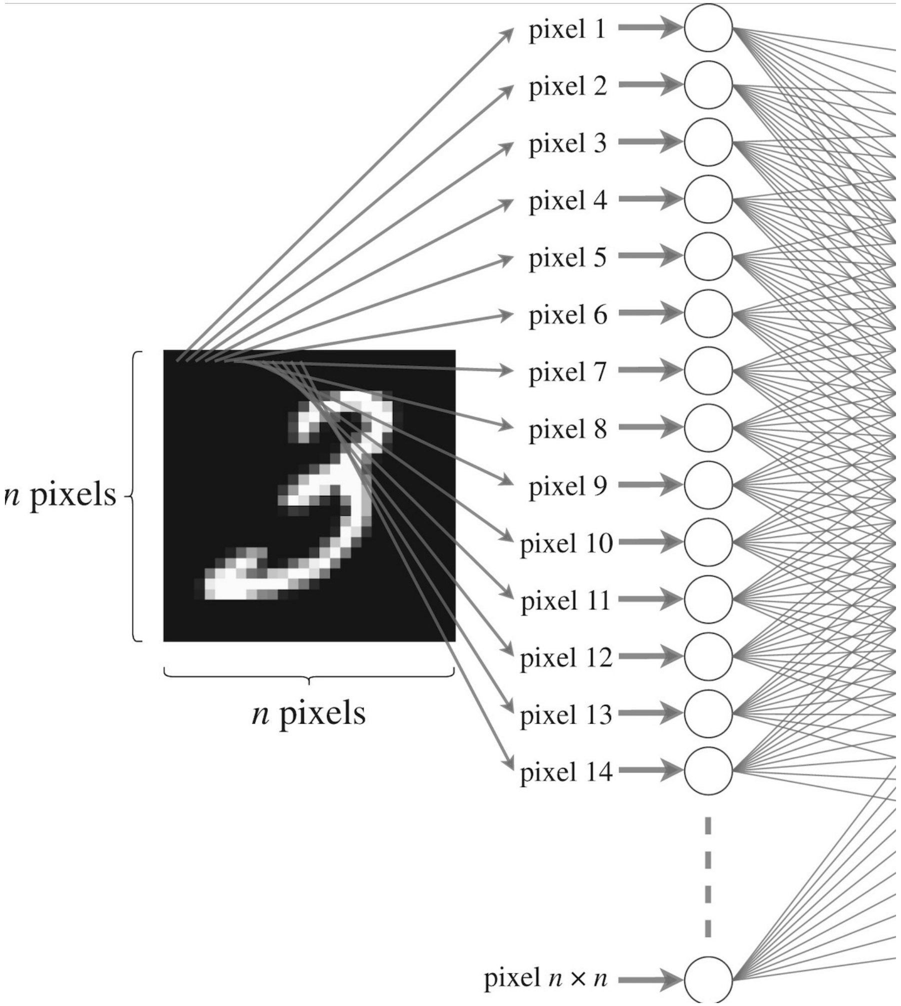


Image source: https://royalsocietypublishing.org/doi/10.1098/rsta.2019.0163

Convolutional Neural Networks

- Convolutional neural networks (CNNs) are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.
- Use the non-vectorized image as input with a 2D weight, which are called a filter or a kernel.
- We call the hidden outputs in CNNs feature map.

Convolutional Neural Networks

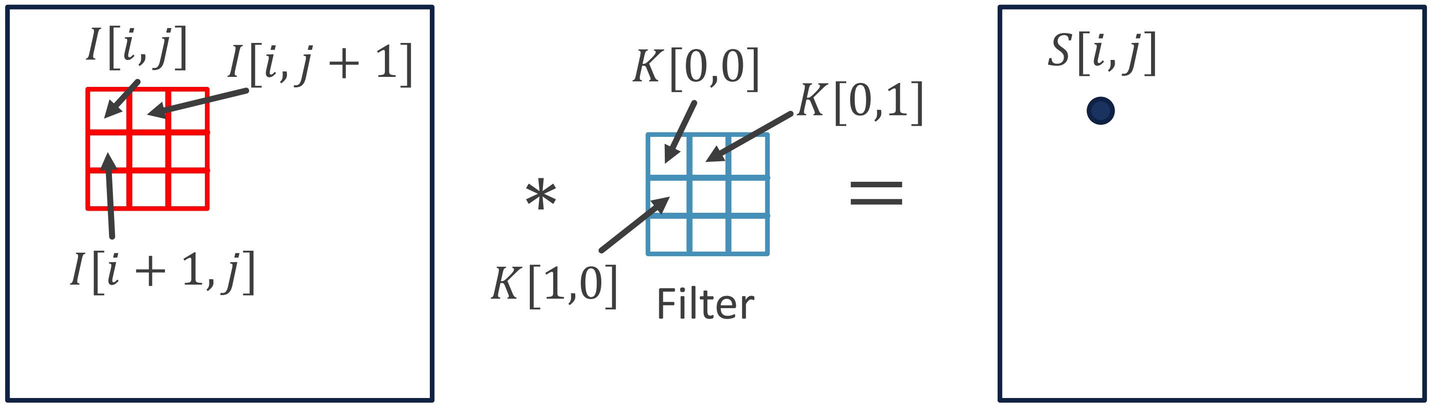
- The operation $*$ between the input image I and filter K to produce a new image S is called convolution, which is defined as:

pixel position

$$S[i,j] = (I * K)[i,j] = \sum_m \sum_n I[i+m, j+n] K[m,n].$$

offset

  - MLP: input $\boldsymbol{x}$, weight W, output $\boldsymbol{h}$.
  - CNN: input I, weight K, output S.



Input

Output (feature map)

Convolution

Input: 3×3

1	2	3
4	5	6
7	8	9

×

Filter: 2×2

10	20
30	40

=

Output: 2×2

370	

$$1 \times 10 + 2 \times 20 + 4 \times 30 + 5 \times 40 = 370$$

Convolution

Input: 3×3

1	2	3
4	5	6
7	8	9

×

Filter: 2×2

10	20
30	40

=

Output: 2×2

370	470

$$2\times10 + 3\times20 + 5\times30 + 6\times40 = 470$$

Convolution

Input: 3×3

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \times \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix} = \begin{bmatrix} 370 & 470 \\ 670 & \end{bmatrix}$$

Filter: 2×2

Output: 2×2

$$\boxed{4\times10 + 5\times20 + 7\times30 + 8\times40 = 670}$$

Convolution

Input: 3×3

1	2	3
4	5	6
7	8	9

×

Filter: 2×2

10	20
30	40

=

Output: 2×2

370	470
670	770

Now the problem: the size of the new image after convolution is shrunk.

$$5\times10 + 6\times20 + 8\times30 + 9\times40 = 770$$

Padding

- In order to keep the dimension of input and output matrix the same, we add padding.

Input: 3×3 + 1×1 padding

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

×

Filter: 3×3

10	20	30
40	50	60
30	20	10

=

Output: 3×3

300		

0×10 + 0×20 + 0×30 + 0×40 + 1×50 + 2×60 + 0×30 + 4×20 + 5×10 = 300

Padding

- In order to keep the dimension of input and output matrix the same, we add padding.

Input: 3×3 + 1×1 padding

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

×

Filter: 3×3

10	20	30
40	50	60
30	20	10

=

Output: 3×3

300	600	

0×10 + 0×20 + 0×30 + 1×40 + 2×50 + 3×60 + 4×30 + 5×20 + 6×10 = 600

Padding

- In order to keep the dimension of input and output matrix the same, we add padding.

Input: 3×3 + 1×1 padding

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

×

Filter: 3×3

10	20	30
40	50	60
30	20	10

=

Output: 3×3

300	600	500

$0\times10 + 0\times20 + 0\times30 + 2\times40 + 3\times50 + 0\times60 + 5\times30 + 6\times20 + 0\times10 = 500$

Stride

- Stride: skip a location of image.

  Input: 3×3 + 1×1 padding

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

×

Filter: 3×3

10	20	30
40	50	60
30	20	10

Stride: 2×2

=

Output: 2×2

300	

0×10 + 0×20 + 0×30 +
0×40 + 1×50 + 2×60 +
0×30 + 4×20 + 5×10 =
300

Stride

- Stride: skip a location of image.

  Input: 3×3 + 1×1 padding

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

×

Filter: 3×3

10	20	30
40	50	60
30	20	10

Stride: 2×2

=

Output: 2×2

300	500

0×10 + 0×20 + 0×30 + 2×40 + 3×50 + 0×60 + 5×30 + 6×20 + 0×10 = 500

Output Size

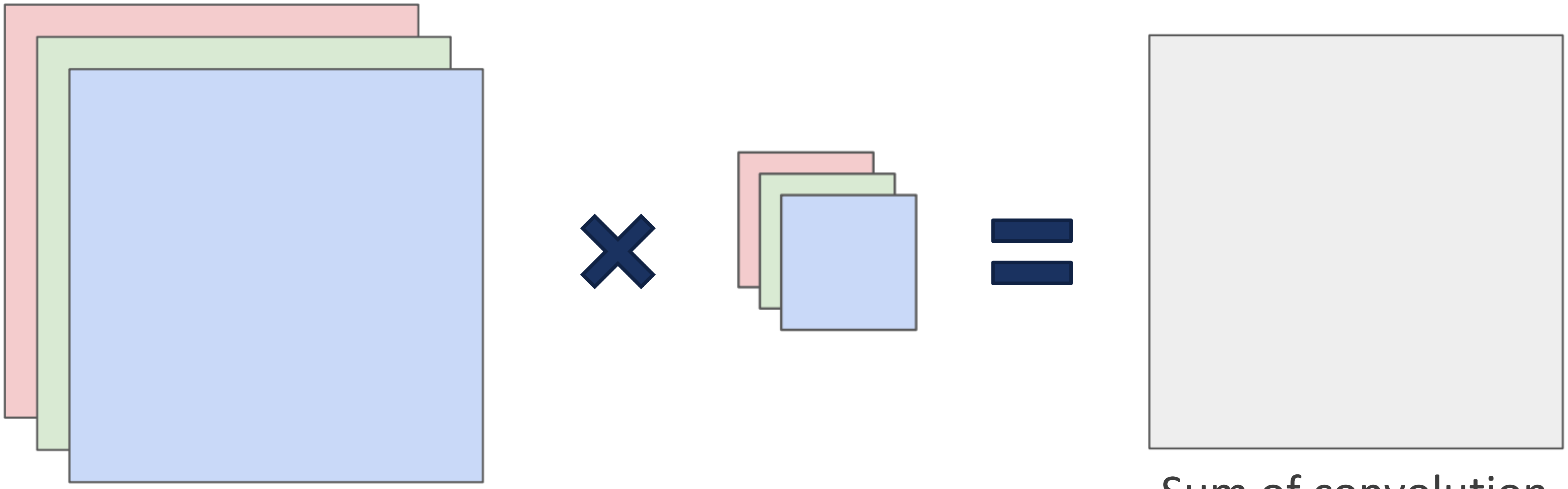
- Input size: $n_h \times n_w$; filter size: $k_h \times k_w$; padding size: $p_h \times p_w$, stride size: $s_h \times s_w$.
- Output size:

$$\left\lfloor \frac{n_h + 2p_h - k_h}{s_h} + 1 \right\rfloor \times \left\lfloor \frac{n_w + 2p_w - k_w}{s_w} + 1 \right\rfloor$$

- For example:
  - Input size 3×3, filter size 3×3, padding size 1×1, stride size 2×2.
  - Output size $\left\lfloor \frac{3+2-3}{2} + 1 \right\rfloor \times \left\lfloor \frac{3+2-3}{2} + 1 \right\rfloor = 2 \times 2$.

Channel and Depth

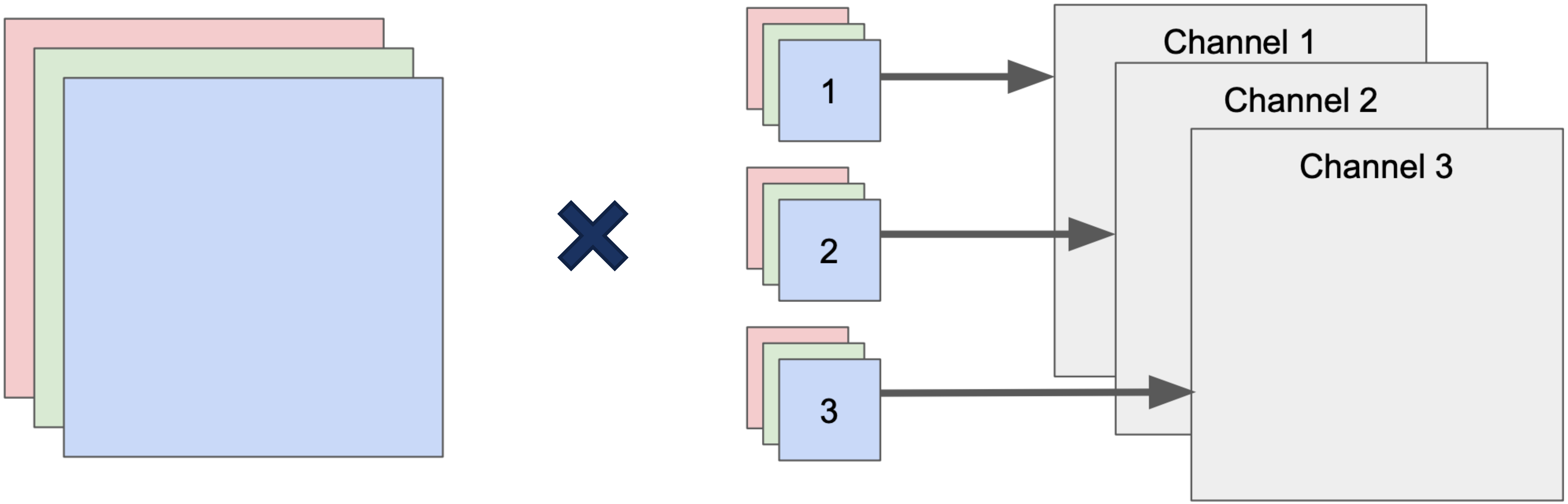
- The depth of the filter is same as the channel of the input image.
  - For an RGB image, we have three channels: red, green and blue.



Sum of convolution in each channel

Channel and Depth

- The depth of the feature map is a hyperparameter.
- It corresponds to the number of filters we would like to use, each learning to look for something different in the input.



Output Size with Depth

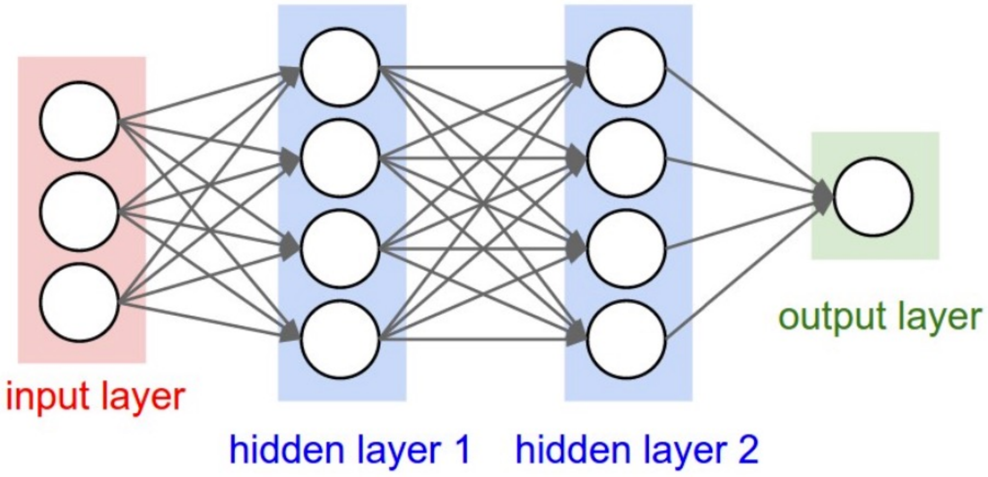
- Input size: $n_h \times n_w \times c_{in}$; filter size: $k_h \times k_w \times c_{in}$; filter number: c_{out}, padding size: $p_h \times p_w$, stride size: $s_h \times s_w$.
- Output size:

$$\left\lfloor \frac{n_h + 2p_h - k_h}{s_h} + 1 \right\rfloor \times \left\lfloor \frac{n_w + 2p_w - k_w}{s_w} + 1 \right\rfloor \times c_{out}$$

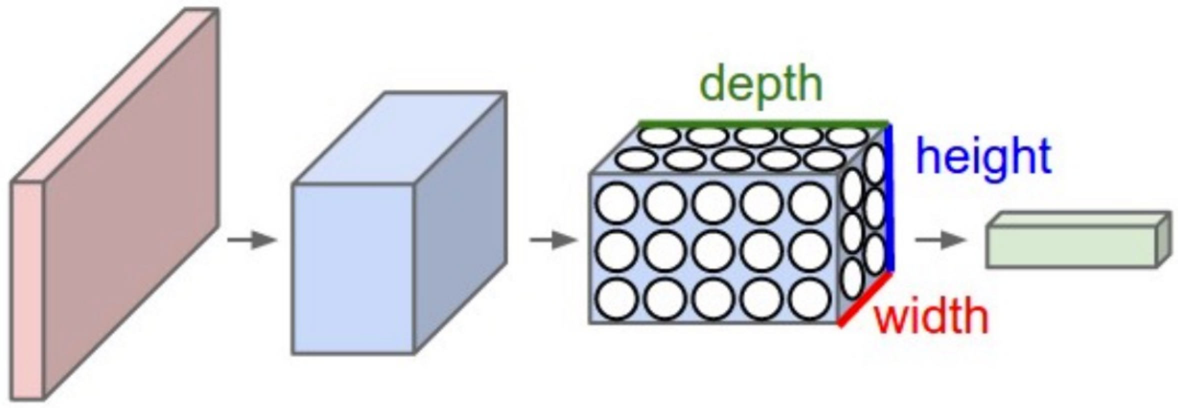
- For example:
  - Input size $5 \times 5 \times 3$, filters size $3 \times 3 \times 3$, filter number 5, padding size 1×1, stride size 2×2.
  - Output size $\left\lfloor \frac{5+2-3}{2} + 1 \right\rfloor \times \left\lfloor \frac{5+2-3}{2} + 1 \right\rfloor \times 5 = 3 \times 3 \times 5.$

Channel and Depth

- Every layer of a CNN transforms the 3D input volume to a 3D output volume of neuron activations.



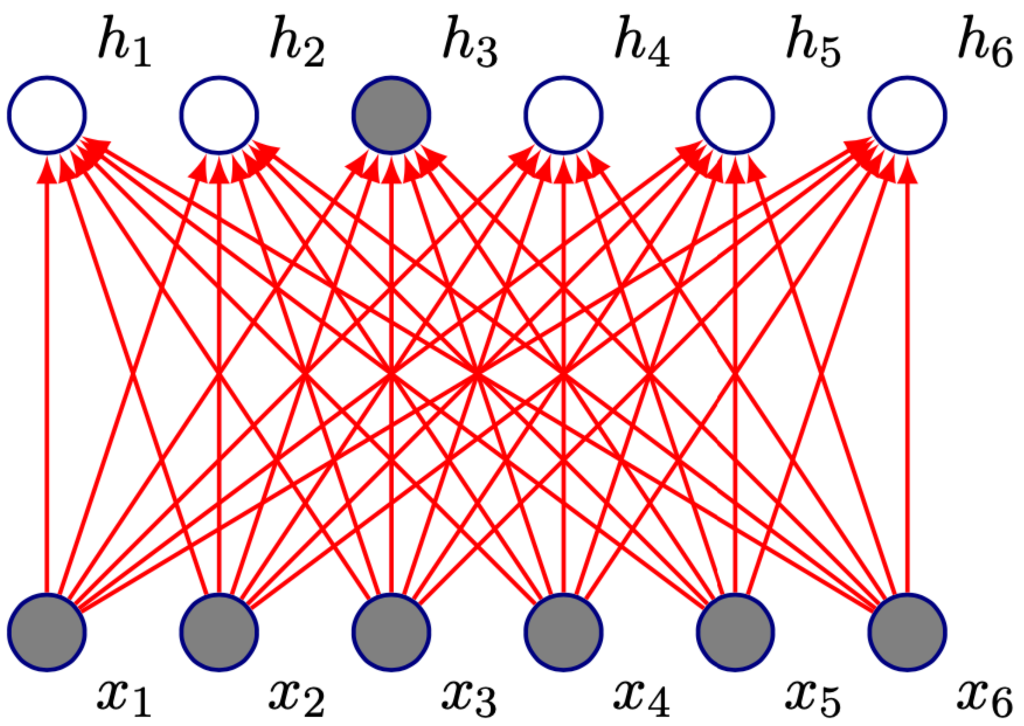
A regular 3-layer Neural Network.



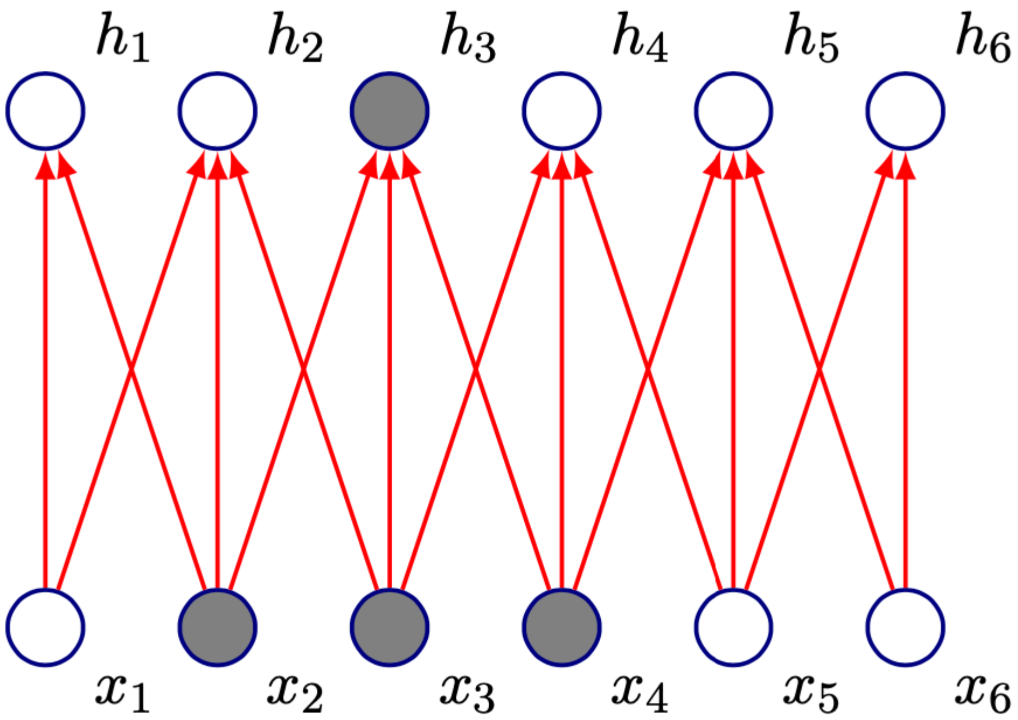
A CNN arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers

Image source: cs231n.github.io/convolutional-networks

Sparse Connectivity



Fully connected network: h_3 is computed by full matrix multiplication with no sparse connectivity.



Kernel of size 3, moved with stride of 1. h_3 only depends on x_2, x_3, x_4.

Image source: Lecture 7, CMSC 35246 Deep Learning Spring 2017, University of Chicago

Sparse Connectivity

- Input: 55×55×3, output: 55×55×96.
- If we adopt a fully connected layer, the number of parameters for one single layer is:

$$(55\times55\times3+1)\times55\times55\times96=2{,}635{,}670{,}400$$

- Now, if we use 96 11×11 filters with 5×5 padding and 1×1 stride.
- We can reduce the number of parameters to

$$(11\times11\times3+1)\times55\times55\times96=105{,}705{,}600$$

  - It is still unacceptable.

We use different filters for each pixel

Parameter Sharing

- We can dramatically reduce the number of parameters by making one reasonable assumption:

If one filter is useful to compute at some spatial position (x_1, y_1), it should also be useful to compute at a different position (x_2, y_2).

- We are going to constrain the neurons in each channel to use the same weights and bias.

Parameter Sharing

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

×

10	20	30
40	50	60
30	20	10

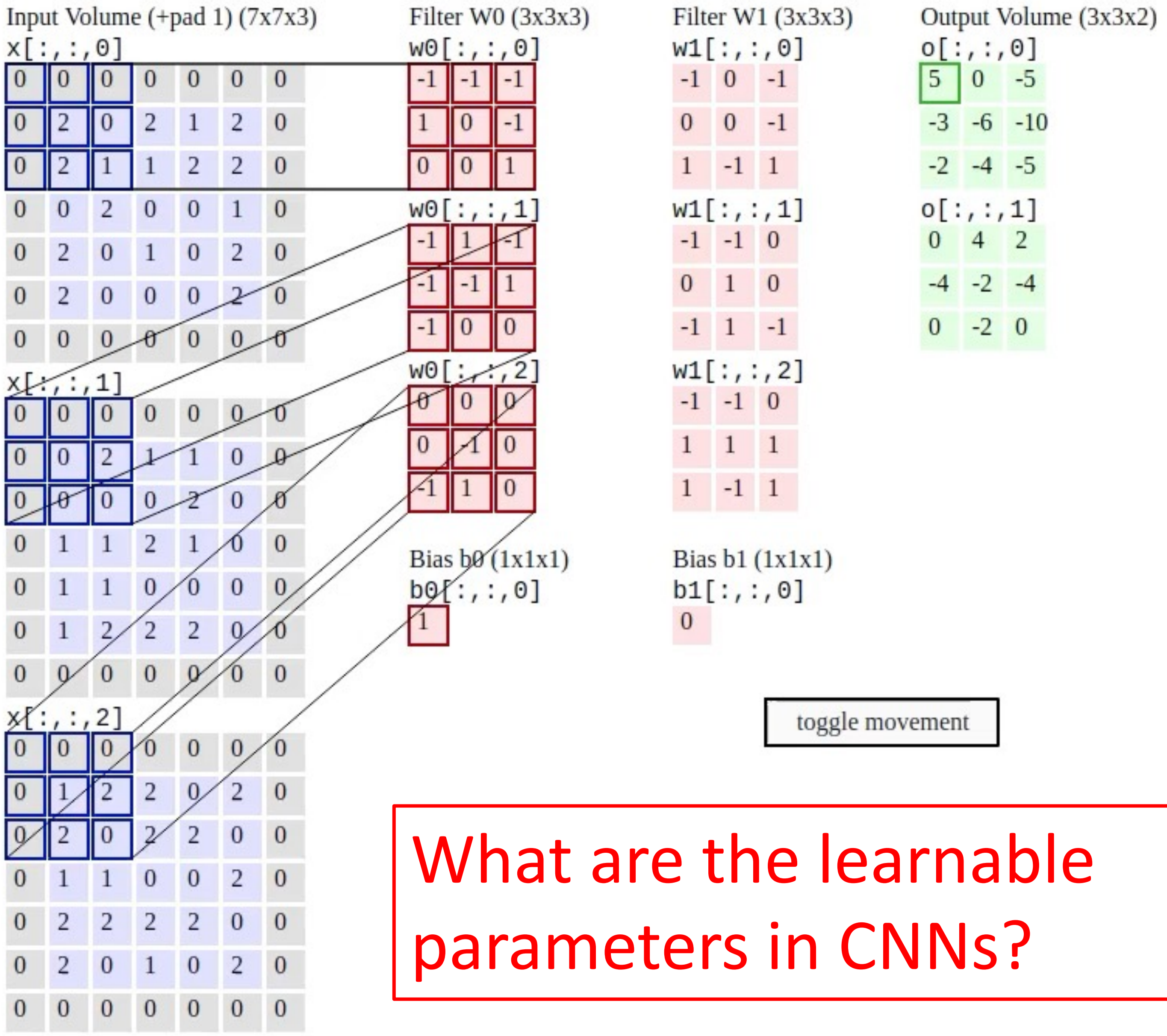
A filter is fixed for all pixel positions in a channel

Parameter Sharing

- Rather than learning a separate set of filter parameters for every location during convolution, we learn only one set.
- This does not affect the runtime of forward propagation, but it does further reduce the storage requirements.
- In the previous example, the number of parameters are reduced to

$$96\times(3\times11\times11+1)=34{,}944$$

  - 3000 times smaller than the non-sharing one.
  - 75,400 times smaller than the fully connected one.

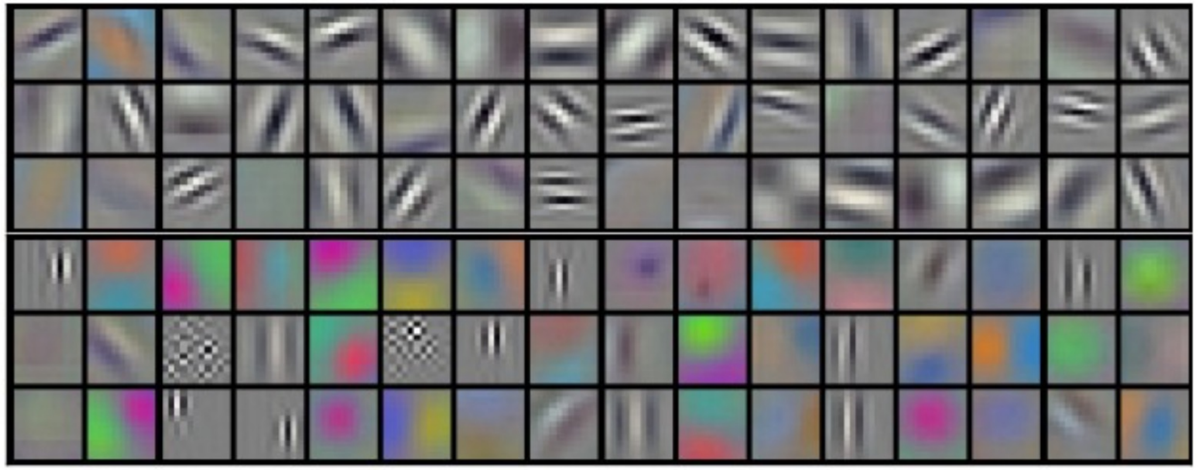


What are the learnable parameters in CNNs?

Image source: cs231n.github.io/convolutional-networks

Parameter Sharing

- Each of the 96 learned filters is of size 11×11×3.
- If detecting a horizontal edge is important at some location in the image, it should intuitively be useful at some other location.
- No need to relearn to detect a horizontal edge at every one of the 55×55 distinct locations in the Conv layer output volume.



Compare with the famous Sobel filter for edge detection:

X – Direction Kernel

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

Y – Direction Kernel

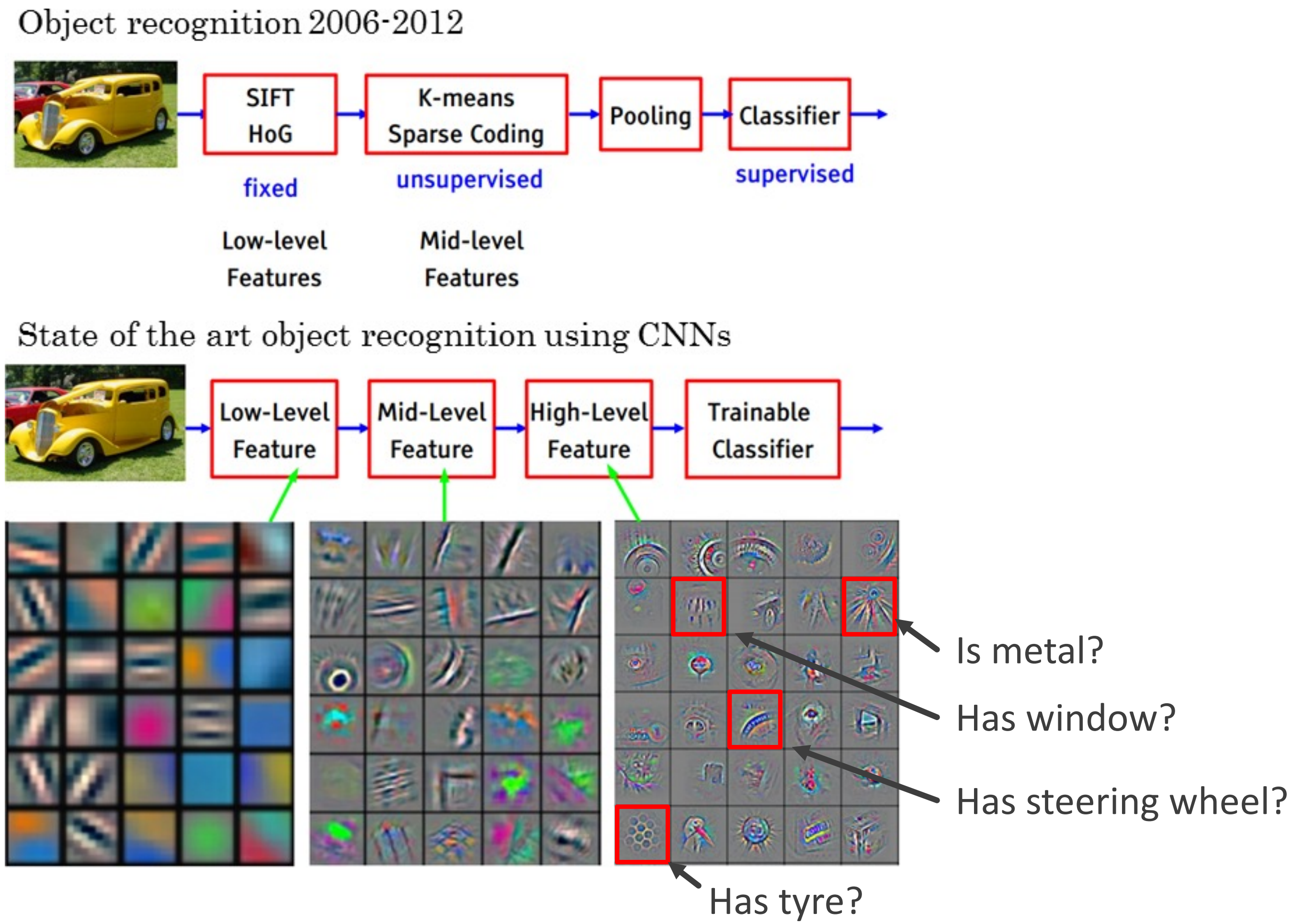
-1	-2	-1
0	0	0
1	2	1

Image source: cs231n.github.io/convolutional-networks

Parameter Sharing

- Parameter sharing is not only for reducing the number of parameters.
- It can also be treated as a regularized method for preventing overfitting.
- It forces the filters to learn some common patterns over the whole image, rather than some patterns specific at some positions.

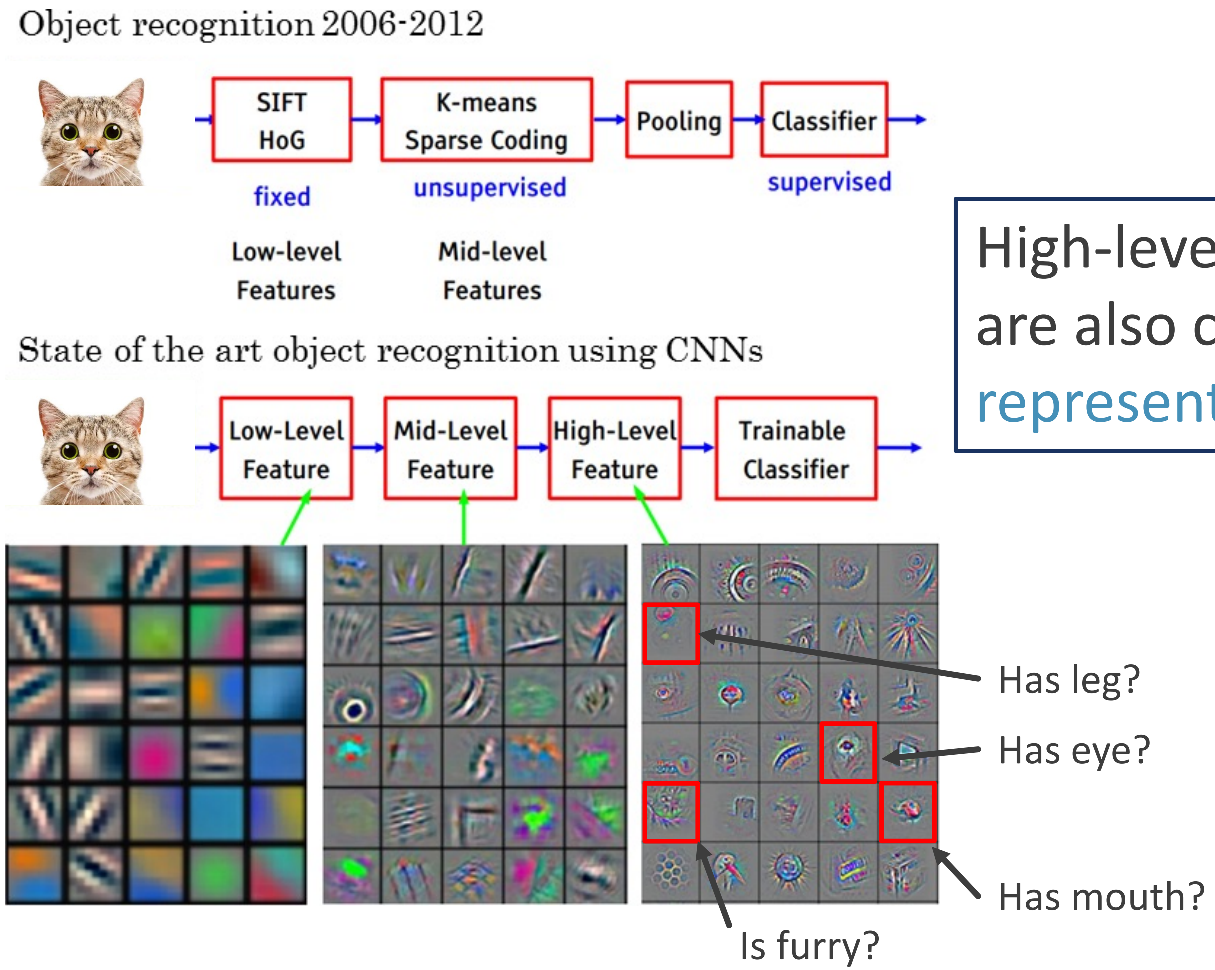
CNNs vs. Traditional Methods



High-level feature contains semantic information, indicating if this image has certain components or not.

Image source: https://rsipvision.com/exploring-deep-learning/

CNNs vs. Traditional Methods



High-level features are also called representations.

High-level feature contains semantic information, indicating if this image has certain components or not.

Image source: https://rsipvision.com/exploring-deep-learning/

Receptive Field

- The receptive field is defined as the size of the region in the input that produces the feature.
- It is a measure of association of an output feature to the input region.
- Increasing model depth is a straightforward way to increase receptive field.
- Is there any other way to increase the receptive field?

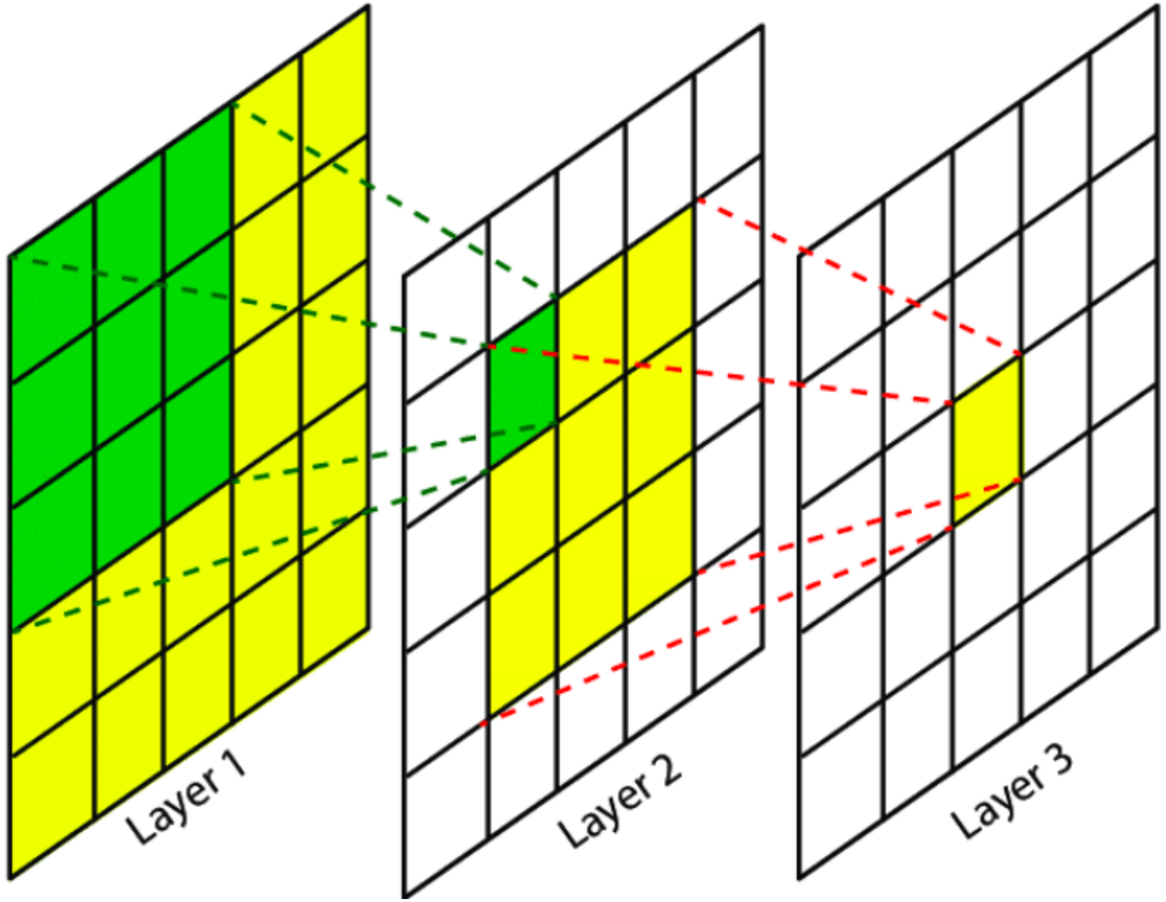
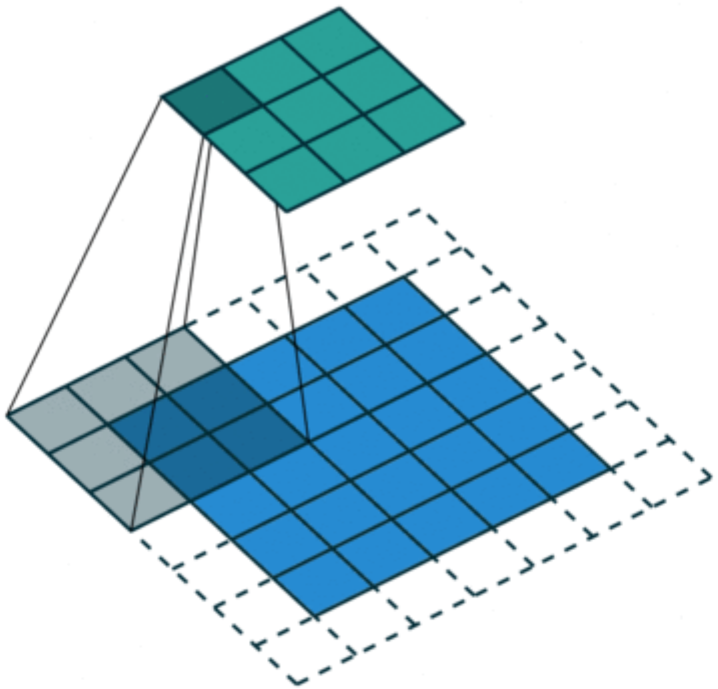


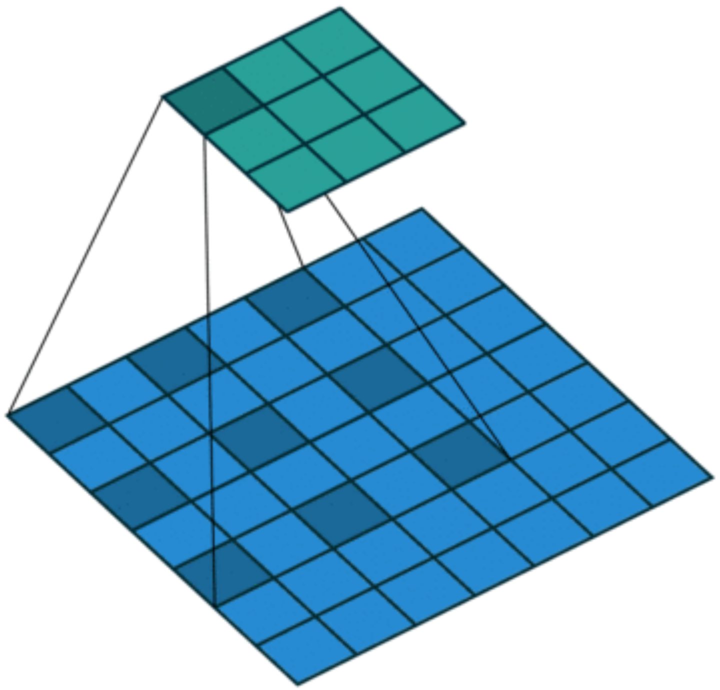
Image source: cs231n.github.io/convolutional-networks

Dilated Convolution

- As an alternative, dilated convolution allows to merge spatial information across the inputs much more aggressively with fewer layers.



Standard Convolution ($d = 1$)



Dilated Convolution ($d = 2$)

Image source: https://towardsdatascience.com/review-dilated-convolution-semantic-segmentation-9d5a5bd768f5

Output Size with Dilated Convolution

- Input size: $n_h \times n_w \times c_{in}$; filter size: $k_h \times k_w \times c_{in}$; filter number: c_{out}; padding size: $p_h \times p_w$, stride size: $s_h \times s_w$; dilation size: $d_h \times d_w$.
- Output size:

$$\left\lfloor \frac{n_h + 2p_h - (d_h \times k_h - 1) - 1}{s_h} + 1 \right\rfloor \times \left\lfloor \frac{n_w + 2p_w - (d_w \times k_w - 1) - 1}{s_w} + 1 \right\rfloor \times c_{out}$$

- For example:
  - Input size $11 \times 11 \times 3$, filter size $3 \times 3 \times 3$, padding size 1×1, stride size 2×2, dilation size 2×2.
  - Output size $\left\lfloor \frac{11+2-(2\times3-1)-1}{2} + 1 \right\rfloor \times \left\lfloor \frac{11+2-(2\times3-1)-1}{2} + 1 \right\rfloor \times 3 = 4 \times 4 \times 3$.

1D and 3D CNNs

- For image data, we are actually using 2D CNN.
  - It means the input data and the filter are both 2-dimentional.

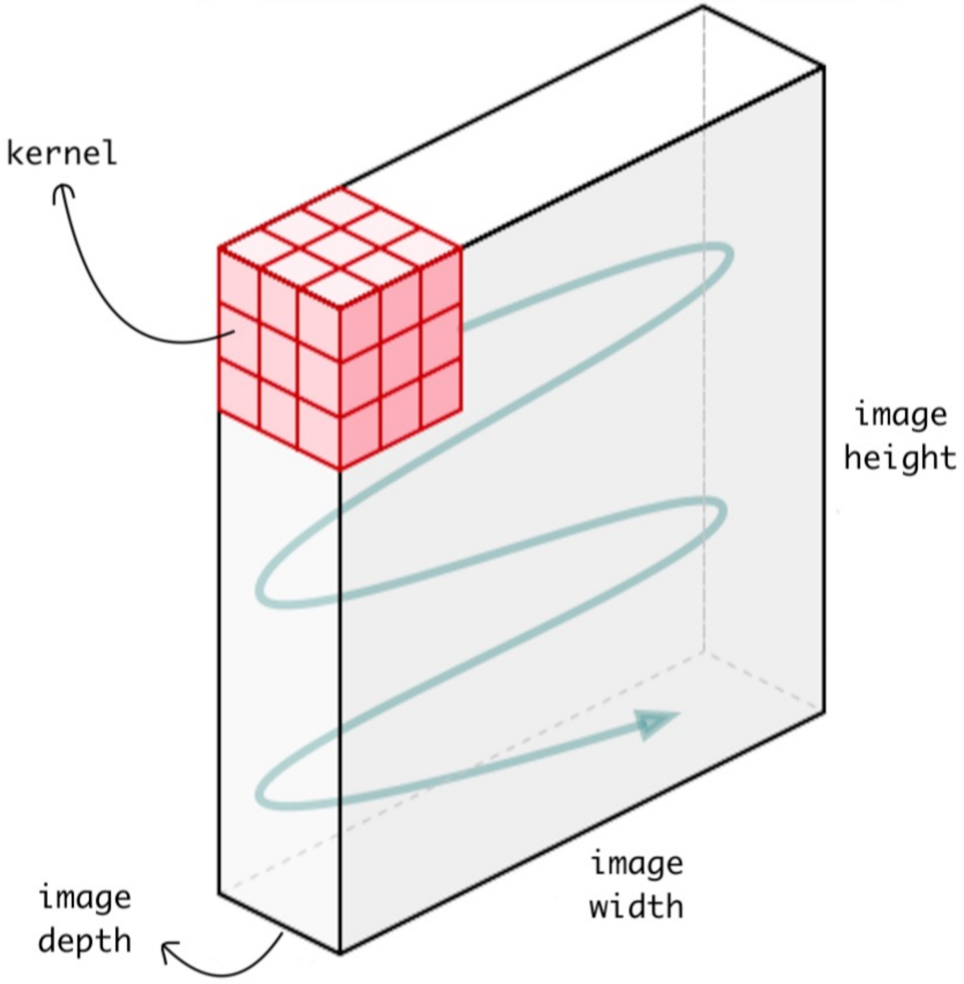


Image source: https://towardsdatascience.com/understanding-1d-and-3d-convolution-neural-network-keras-9d8f76e29610

1D and 3D CNNs

- For 1D data like signal or time-series data, we can adopt 1D CNN with 1D filter.
  - It is just a sparse and parameter-shared version of MLP.

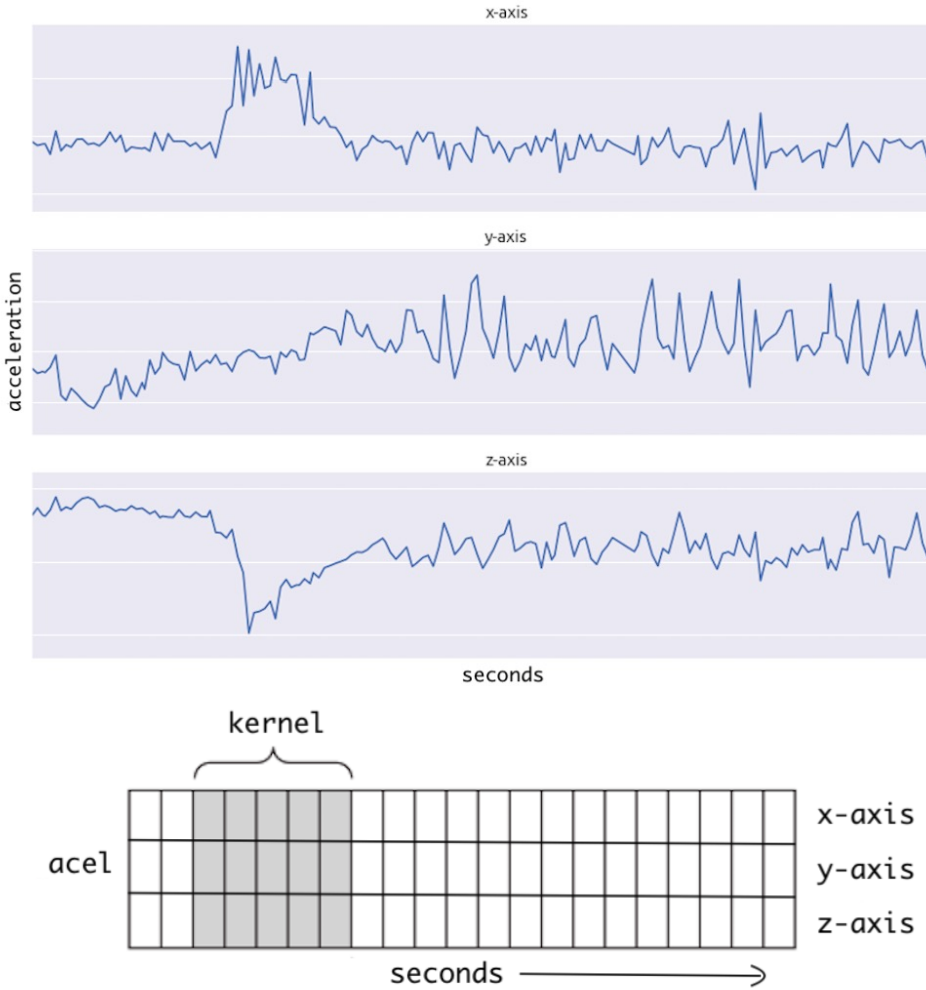
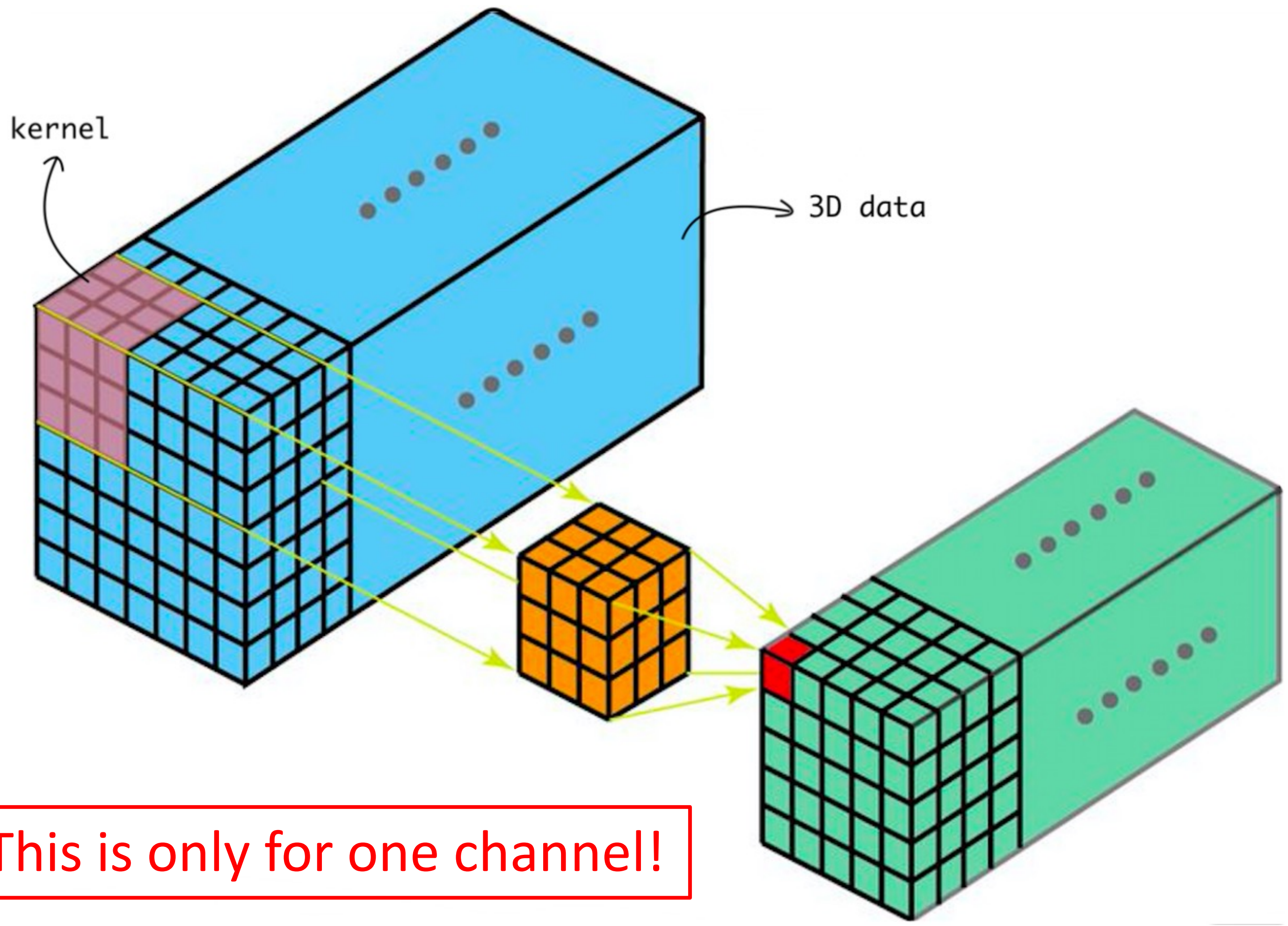


Image source: https://towardsdatascience.com/understanding-1d-and-3d-convolution-neural-network-keras-9d8f76e29610

1D and 3D CNNs

- For 3D data like hyperspectral images, medical images, or videos, we can generalize 2D CNNs to 3D.

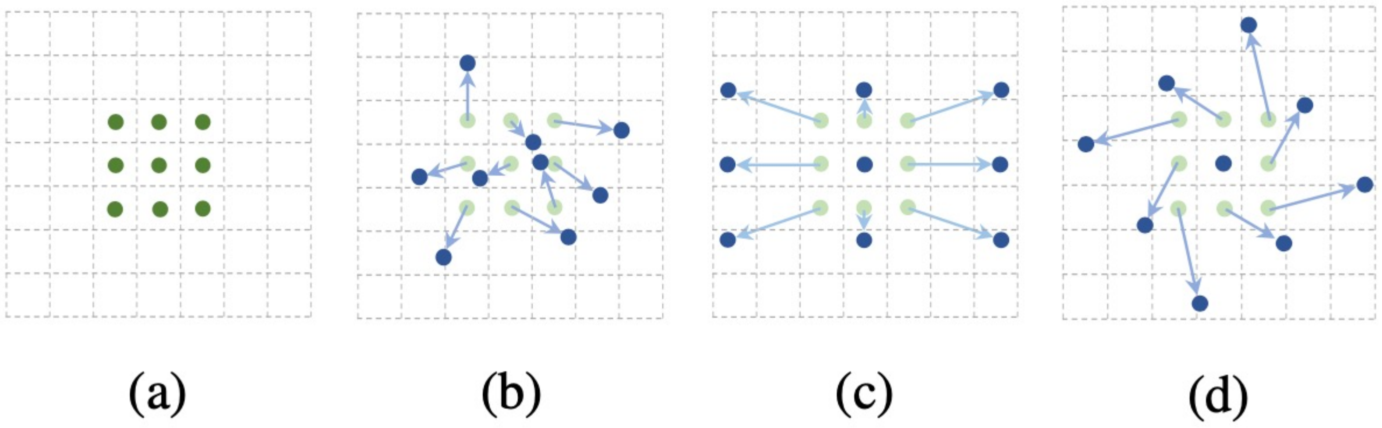


This is only for one channel!

Image source: https://towardsdatascience.com/understanding-1d-and-3d-convolution-neural-network-keras-9d8f76e29610

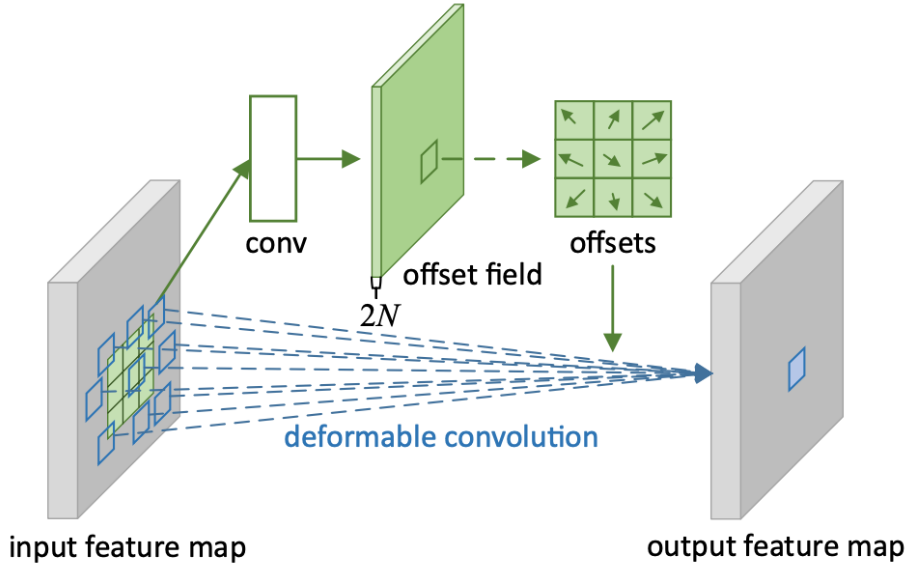
Deformable Convolutional Networks

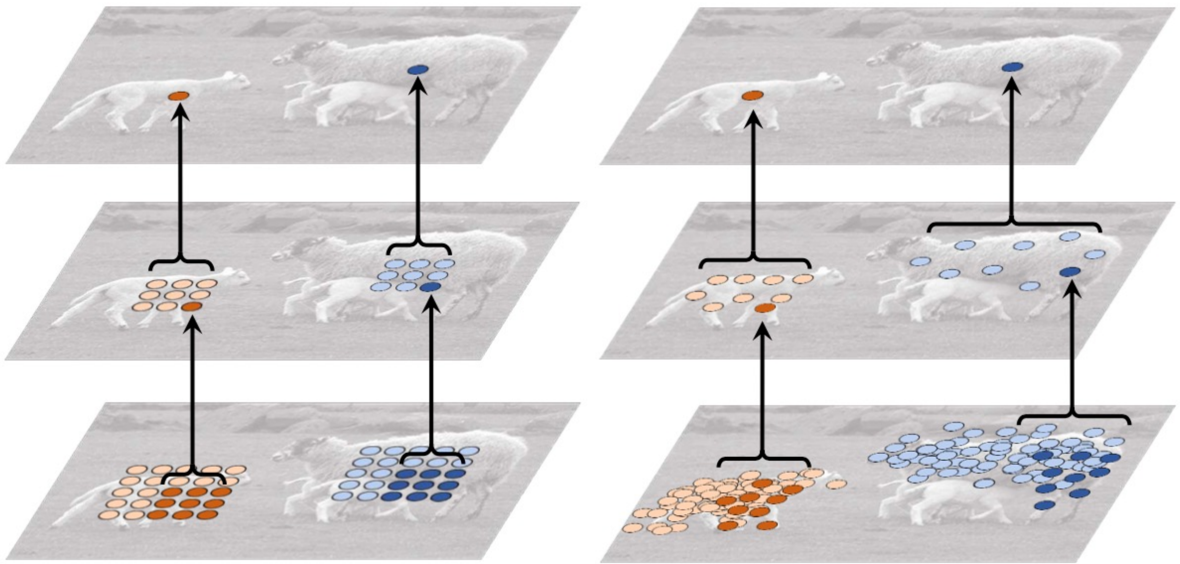
- One problem of CNNs: the shape of filters is fixed (e.g. square-like), but the shape of objects is variant.



- (a) regular sampling grid (green points) of standard convolution.
- (b) deformed sampling locations (dark blue points) with augmented offsets (light blue arrows) in deformable convolution.
- (c)(d) are special cases of (b).

Deformable Convolutional Networks





$$\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n + \Delta \mathbf{p}_n).$$

Filter weights and position offsets are learnable parameters

Source: W. H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, “Deformable Convolutional Networks,” in ICCV, 2017, p. 6003.

Convolutional Layer in PyTorch

Docs > torch.nn > Conv2d

CONV2D

CLASS `torch.nn.Conv2d(in_channels: int, out_channels: int, kernel_size: Union[T, Tuple[T, T]], stride: Union[T, Tuple[T, T]] = 1, padding: Union[T, Tuple[T, T]] = 0, dilation: Union[T, Tuple[T, T]] = 1, groups: int = 1, bias: bool = True, padding_mode: str = 'zeros')` [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

Parameters

- **in_channels** (*int*) – Number of channels in the input image
- **out_channels** (*int*) – Number of channels produced by the convolution
- **kernel_size** (*int or tuple*) – Size of the convolving kernel
- **stride** (*int or tuple, optional*) – Stride of the convolution. Default: 1
- **padding** (*int or tuple, optional*) – Zero-padding added to both sides of the input. Default: 0
- **padding_mode** (*string, optional*) – `'zeros'`, `'reflect'`, `'replicate'` or `'circular'`. Default: `'zeros'`
- **dilation** (*int or tuple, optional*) – Spacing between kernel elements. Default: 1
- **groups** (*int, optional*) – Number of blocked connections from input channels to output channels. Default: 1
- **bias** (*bool, optional*) – If `True`, adds a learnable bias to the output. Default: `True`

Source: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

```python
import torch
rand_input = torch.randn(20, 16, 50, 100)
```

Input shape in PyTorch is:
$[N, C_{in}, H, W]$

```python
# With square kernels and equal stride
m = torch.nn.Conv2d(16, 33, 3, stride=2)
output = m(rand_input)
print(output.shape)
```

```
torch.Size([20, 33, 24, 49])
```

```python
# non-square kernels and unequal stride and with padding
m = torch.nn.Conv2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2))
output = m(rand_input)
print(output.shape)
```

```
torch.Size([20, 33, 28, 100])
```

```python
# non-square kernels and unequal stride and with padding and dilation
m = torch.nn.Conv2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2), dilation=(3, 1))
output = m(rand_input)
print(output.shape)
```

```
torch.Size([20, 33, 26, 100])
```

Convolutional Layer in TensorFlow

```
tf.keras.layers.Conv2D(
    filters, kernel_size, strides=(1, 1), padding='valid', data_format=None,
    dilation_rate=(1, 1), groups=1, activation=None, use_bias=True,
    kernel_initializer='glorot_uniform', bias_initializer='zeros',
    kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None,
    kernel_constraint=None, bias_constraint=None, **kwargs
)
```

Arguments	
`filters`	Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
`kernel_size`	An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
`strides`	An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
`padding`	one of "`valid`" or "`same`" (case-insensitive).

```python
import tensorflow as tf
rand_input = tf.random.normal([20, 50, 100, 16])
```

Input shape in TF is: $[N, H, W, C_{in}]$

```python
# With square kernels and equal stride
m = tf.keras.layers.Conv2D(33, 3, strides=2)
output = m(rand_input)
print(output.shape)
```

```
(20, 24, 49, 33)
```

```python
# non-square kernels and unequal stride and with padding
p = tf.keras.layers.ZeroPadding2D((4, 2))
m = tf.keras.layers.Conv2D(33, (3, 5), strides=(2, 1))
output = m(p(rand_input))
print(output.shape)
```

Special padding size can't be assigned in conv2d

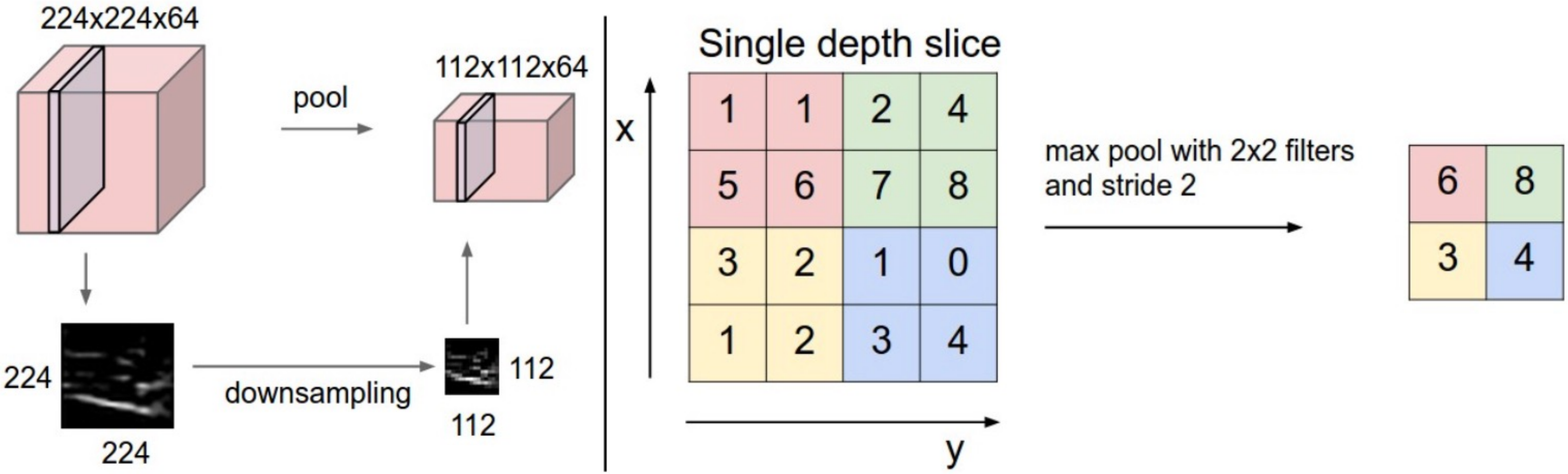
```
(20, 28, 100, 33)
```

stride>1 is incompatible with dilation_rate >1

```python
# non-square kernels and equal stride and with padding and dilation
p = tf.keras.layers.ZeroPadding2D((4, 2))
m = tf.keras.layers.Conv2D(33, (3, 5), strides=(1, 1), dilation_rate=(3, 1))
output = m(p(rand_input))
print(output.shape)
```

```
(20, 52, 100, 33)
```

Pooling

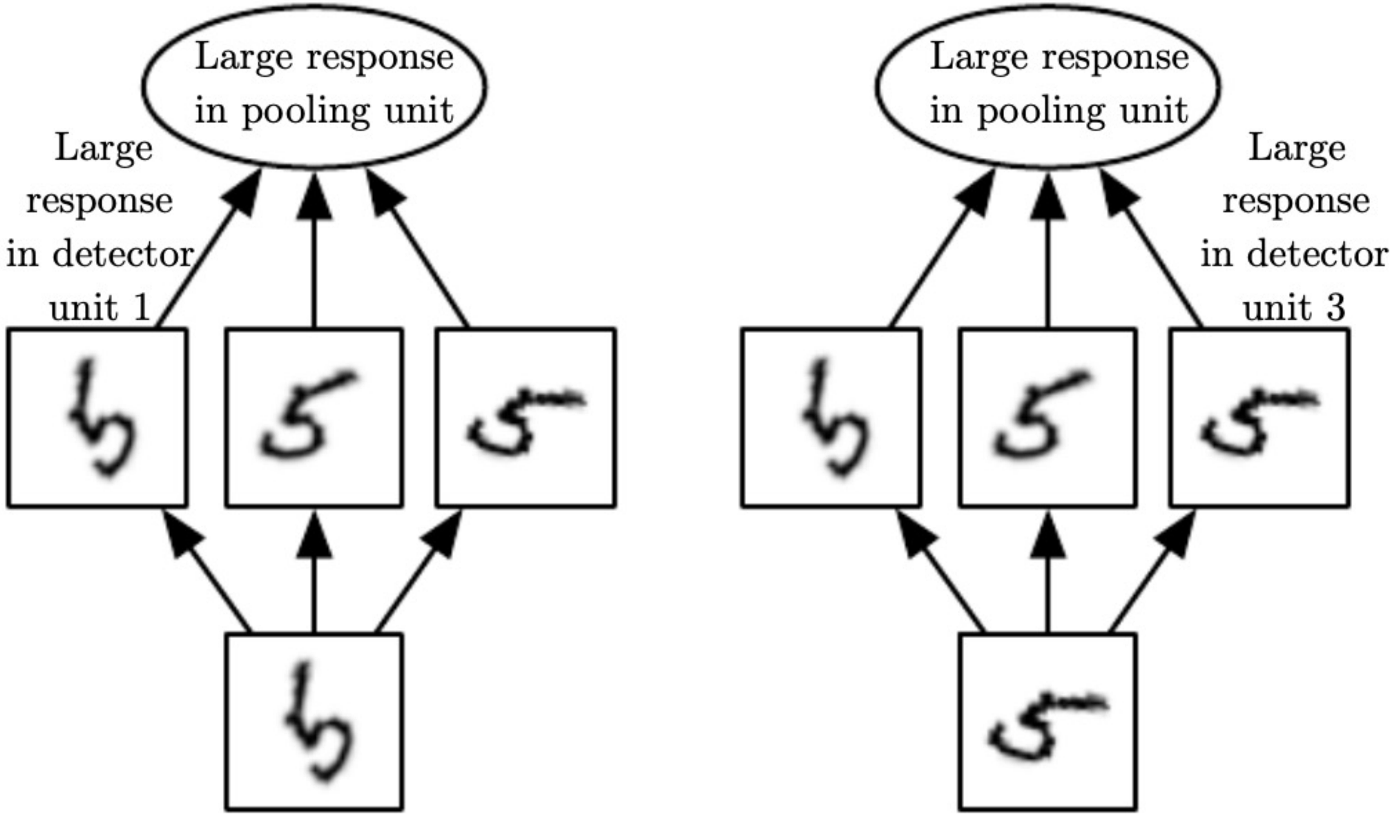


- Pooling layer downsamples the volume spatially, independently in each channel of the input volume.

Image source: cs231n.github.io/convolutional-networks

Pooling

- On one hand, pooling increases the receptive field.
- On the one hand, pooling introduces invariance.



The max pooling unit then has a large activation regardless of which detector unit was activated.

Image source: Figure 9.9, Goodfellow, Bengio, and Courville, Deep Learning, Cambridge: MIT press, 2016.

Pooling Layer in PyTorch

Docs > torch.nn > MaxPool2d

MAXPOOL2D

CLASS `torch.nn.MaxPool2d(kernel_size: Union[T, Tuple[T, ...]], stride: Optional[Union[T, Tuple[T, ...]]] = None, padding: Union[T, Tuple[T, ...]] = 0, dilation: Union[T, Tuple[T, ...]] = 1, return_indices: bool = False, ceil_mode: bool = False)` [SOURCE]

Applies a 2D max pooling over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C, H, W), output (N, C, H_{out}, W_{out}) and `kernel_size` (kH, kW) can be precisely described as:

$$\mathrm{out}(N_i, C_j, h, w) = \max_{m=0,\ldots,kH-1} \max_{n=0,\ldots,kW-1} \mathrm{input}(N_i, C_j, \mathrm{stride}[0] \times h + m, \mathrm{stride}[1] \times w + n)$$

Parameters

- **kernel_size** – the size of the window to take a max over
- **stride** – the stride of the window. Default value is `kernel_size`
- **padding** – implicit zero padding to be added on both sides
- **dilation** – a parameter that controls the stride of elements in the window
- **return_indices** – if `True`, will return the max indices along with the outputs. Useful for `torch.nn.MaxUnpool2d` later
- **ceil_mode** – when True, will use *ceil* instead of *floor* to compute the output shape

Source: https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html

Pooling Layer in PyTorch

```
# pool of square window of size=3, stride=2
m = torch.nn.MaxPool2d(3, stride=2)
output = m(rand_input)
print(output.shape)
```

```
torch.Size([20, 16, 24, 49])
```

```
# pool of non-square window
m = torch.nn.MaxPool2d((3, 2), stride=(2, 1))
output = m(rand_input)
print(output.shape)
```

```
torch.Size([20, 16, 24, 99])
```

Pooling Layer in TensorFlow

```
tf.keras.layers.MaxPool2D(
    pool_size=(2, 2), strides=None, padding='valid', data_format=None, **kwargs
)
```

Arguments	
`pool_size`	integer or tuple of 2 integers, window size over which to take the maximum. `(2, 2)` will take the max value over a 2x2 pooling window. If only one integer is specified, the same window length will be used for both dimensions.
`strides`	Integer, tuple of 2 integers, or None. Strides values. Specifies how far the pooling window moves for each pooling step. If None, it will default to `pool_size`.
`padding`	One of `"valid"` or `"same"` (case-insensitive). "valid" adds no zero padding. "same" adds padding such that if the stride is 1, the output shape is the same as input shape.
`data_format`	A string, one of `channels_last` (default) or `channels_first`. The ordering of the dimensions in the inputs. `channels_last` corresponds to inputs with shape `(batch, height, width, channels)` while `channels_first` corresponds to inputs with shape `(batch, channels, height, width)`. It defaults to the `image_data_format` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "channels_last".

Source: https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool2D

Pooling Layer in TensorFlow

```
# pool of square window of size=3, strides=2
m = tf.keras.layers.MaxPooling2D(3, strides=2)
output = m(rand_input)
print(output.shape)
```

```
(20, 24, 49, 16)
```

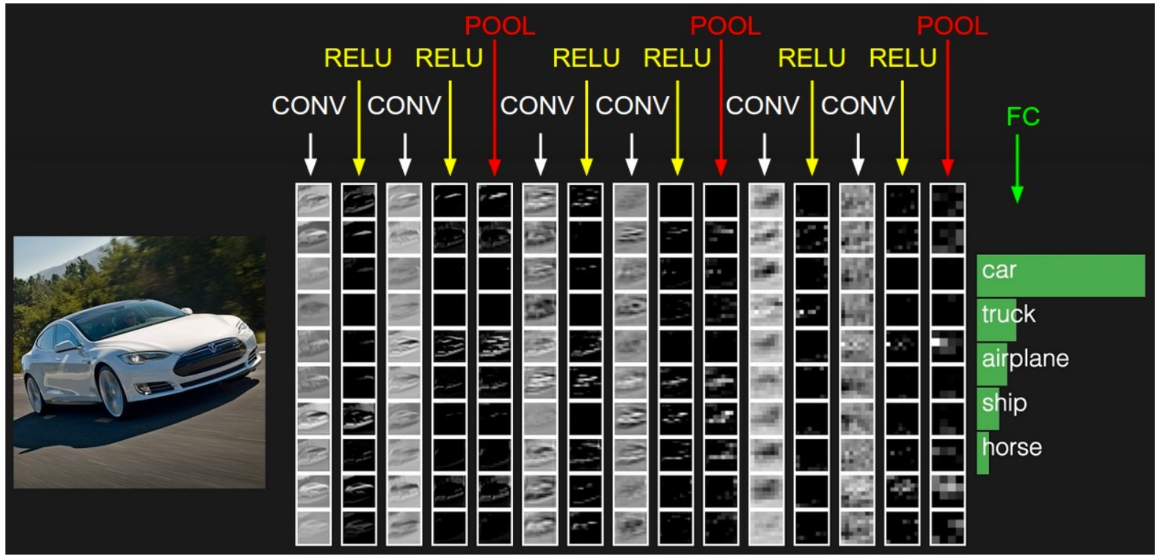
```
# pool of non-square window
m = tf.keras.layers.MaxPooling2D((3, 2), strides=(2, 1))
output = m(rand_input)
print(output.shape)
```

```
(20, 24, 99, 16)
```

CNN Architecture

A typical CNN consists of four basic modules:

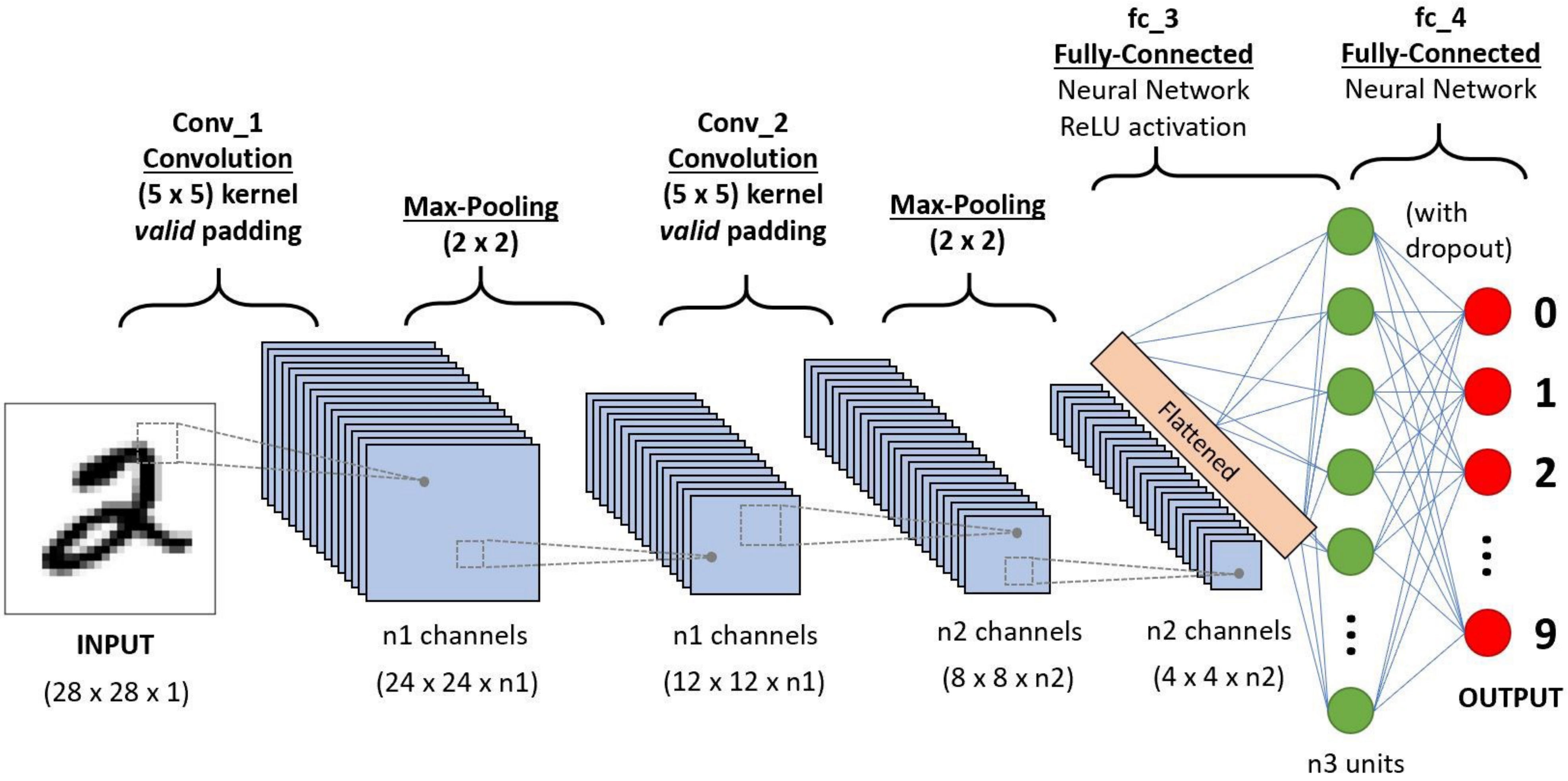
- CONV layer will compute the output of neurons that are connected to local regions in the input.
- RELU layer will apply an elementwise activation function.
- FC (i.e. fully-connected) layer will compute the class scores.



- POOL layer will perform a downsampling operation along the spatial dimensions (width, height).

Image source: cs231n.github.io/convolutional-networks

CNN Architecture



CNN Architecture

- We can also use global average pooling (GAP) to replace flattening.

Image source: B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, “Learning Deep Features for Discriminative Localization,” in CVPR, 2016, vol. 2016-Decem, pp. 2921–2929.

CNN Architecture

- The most common form of a CNN architecture:
  - stacks a few CONV-RELU layers;
  - follows them with POOL layers;
  - repeats this pattern until the image has been merged spatially to a small size;
  - transits to fully-connected layers to produce output (e.g. class scores).
- The most common CNN architecture follows the pattern:

  INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC

  - The * indicates repetition, and the indicates an optional pooling layer.
  - N >= 0 (and usually N <= 3), , K >= 0 (and usually K < 3).

CNN Architecture

- INPUT -> FC.
  - A simple linear classifier. Here N = M = K = 0 .
- INPUT -> CONV -> RELU -> FC.
  - Only CONV layer and RELU layer are used.
- INPUT -> [CONV -> RELU -> POOL]*2 -> FC -> RELU -> FC.
  - There is a single CONV layer between every POOL layer.
- INPUT -> [CONV -> RELU -> CONV -> RELU -> POOL]*3 -> [FC -> RELU]*2 -> FC.
  - Two CONV layers stacked before every POOL layer.
  - This is generally a good idea for larger and deeper networks, because multiple stacked CONV layers can develop more complex features of the input volume before the destructive pooling operation.

Filter Size

Is a stack of three 3×3 CONV layers equivalent to a single 7×7 CONV layer?

- No. There are several disadvantages for using filters with large size:
  - Less powerful: the neurons would be computing a linear function over the input, while the three stacks of CONV layers contain non-linearities that make their features more expressive.
  - More parameters: if both the input and output of a layer have depth C, 7×7 CONV layer would contain $C\times(7\times7\times C) = 49C^2$, while the three 3×3 CONV layers only contains $3\times C\times(3\times3\times C) = 27C^2$.
- Intuitively, stacking CONV layers with tiny filters as opposed to having one CONV layer with big filters allows us to express more powerful features of the input, and with fewer parameters.

Layer Sizing Patterns

The common rules of thumb for sizing the architectures:

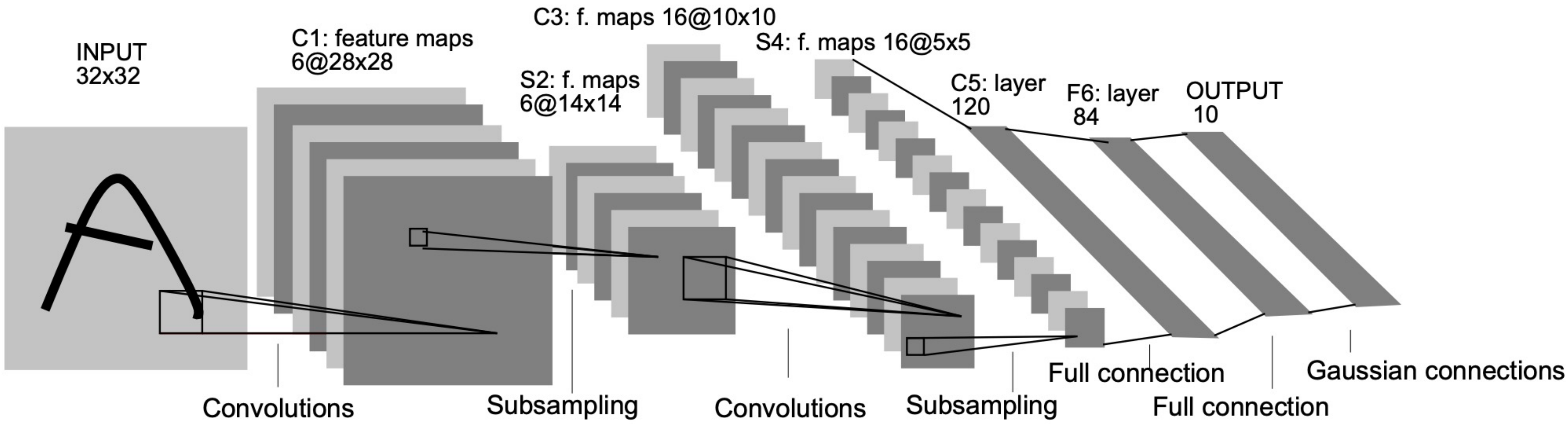
- The INPUT layer (that contains the image) should be divisible by 2 many times.
  - E.g. 32 (e.g. CIFAR-10), 64, 96 (e.g. STL-10), or 224 (e.g. common ImageNet ConvNets), 384, and 512.
- The CONV layers should be
  - using small filters (e.g. 3x3 or at most 5x5),
  - using a stride of 1x1,
  - padding the input volume with zeros in such way that the conv layer does not alter the spatial dimensions of the input.

Layer Sizing Patterns

- The pool layers are in charge of downsampling the spatial dimensions of the input.
  - The most common setting is to use max-pooling with 2×2 receptive fields, and with a stride of 2×2.
  - Note that this discards exactly 75% of the activations in an input volume (due to downsampling by 2 in both width and height).
  - Another slightly less common setting is to use 3×3 receptive fields with a stride of 2.
  - It is very uncommon to see receptive field sizes for max pooling that are larger than 3 because the pooling is then too lossy and aggressive. This usually leads to worse performance.

Classical CNN Architectures

Gradient-based learning applied to document recognition
Y LeCun, L Bottou, Y Bengio… - Proceedings of the …, 1998 - ieeexplore.ieee.org
Multilayer neural networks trained with the back-propagation algorithm constitute the best example of a successful gradient based learning technique. Given an appropriate network architecture, gradient-based learning algorithms can be used to synthesize a complex decision surface that can classify high-dimensional patterns, such as handwritten characters, with minimal preprocessing. This paper reviews various methods applied to handwritten character recognition and compares them on a standard handwritten digit …

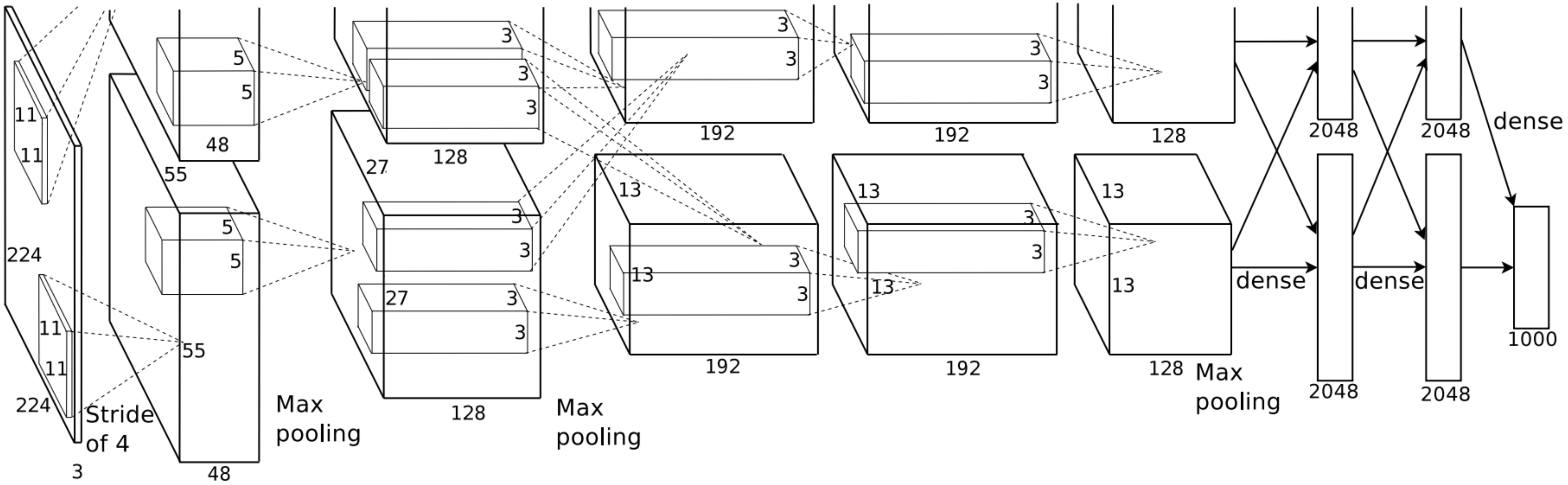


Architecture of LeNet-5

Image source: LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86, no. 11 (1998): 2278-2324.

Classical CNN Architectures

Imagenet classification with deep convolutional neural networks
A Krizhevsky, I Sutskever, GE Hinton - Advances in **neural** …, 2012 - papers.nips.cc
We trained a large, deep convolutional neural network to classify the 1.3 million high-resolution images in the LSVRC-2010 ImageNet training set into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 39.7\% and 18.9\% which is …
Cited by 70691 Related articles All 133 versions



Architecture of AlexNet

Image source: Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." In Advances in neural information processing systems, pp. 1097-1105. 2012.

Conclusion

After this lecture, you should know:

- What is convolution and filter.
- What are the commonly used layers in CNN.
- How to calculate the output size of after a convolutional layer.
- Why do we need pooling.
- What are the typical CNN architectures.

Suggested Reading

- Deep learning textbook chapter 9.
- cs231n CNN tutorial
- Conv Nets: A Modular Perspective
- Understanding Convolutions

Thank you!

- Any question?
- Don’t hesitate to send email to me for asking questions and discussion. ☺