CNN

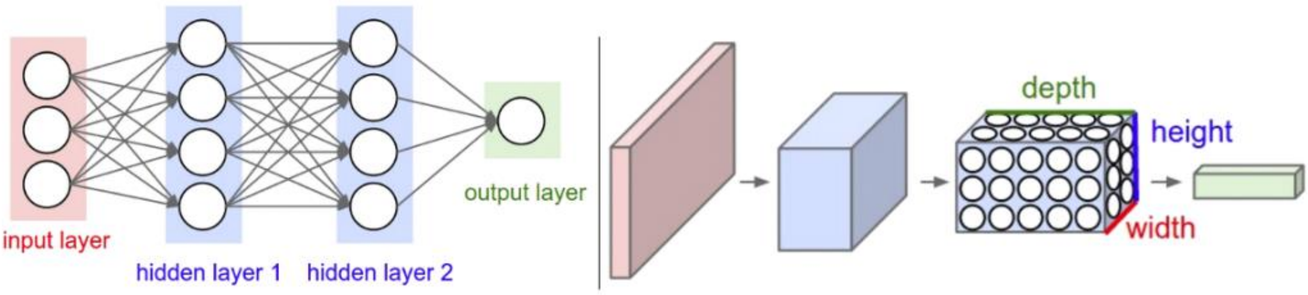
2024年9月23日 22:33

Architecture

Overview

- Regular Neural Nets - don't work well with images, especially with large imgs (eg: 200x200x3), full connectivity is a waste of neurons

- Convolution Neural Nets
  - each layer have neurons arranged in 3D (weight, height, depth)
  - depth refers to 3rd dimension of a layer, not depth of full network



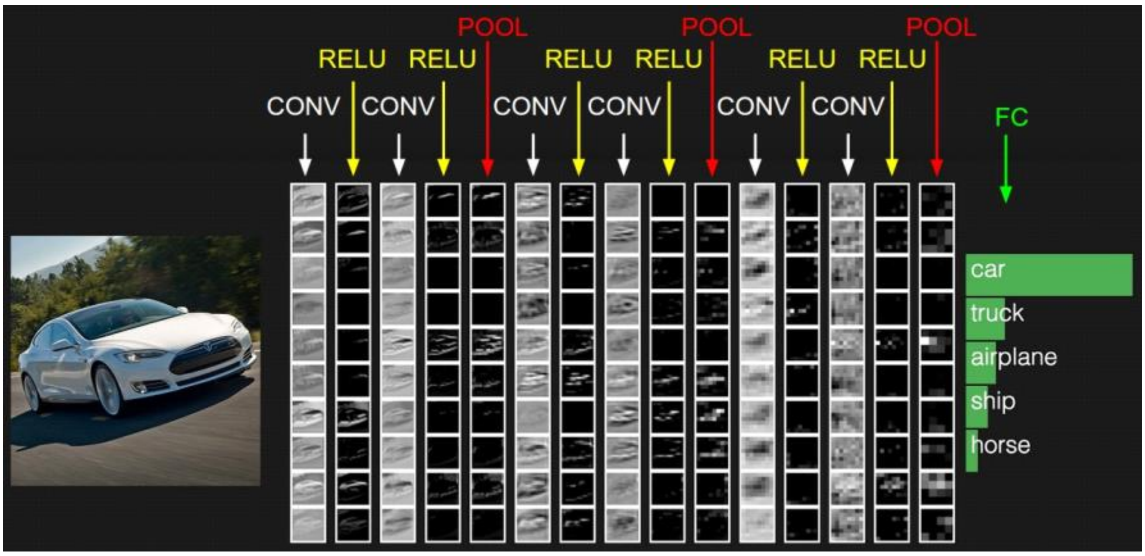
Layers used to build CNN:

3 Main Types:

- Convolutional Layer (CONV)
- Pooling Layer (POOL)
- Fully-Connected Layer (FC)

An example: [INPUT - CONV - RELU - POOL - FC]

- INPUT [32x32x3]: image with 32x32, 3 channels RGB
  - CONV: eg 12 filters is used, result in volume [32x32x12]
- RELU: activation function
- POOL: downsampling operation along the spatial dimensions, result in volume [16x16x12]
- FC: volume of [1x1x10], 10 corresponds to 10 categories of CIFAR-10



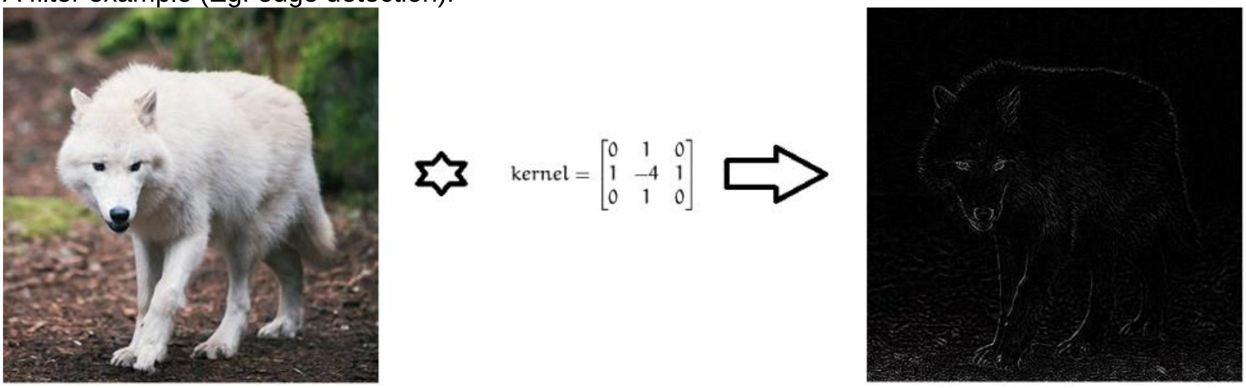
Convolutional Layer

Overview:

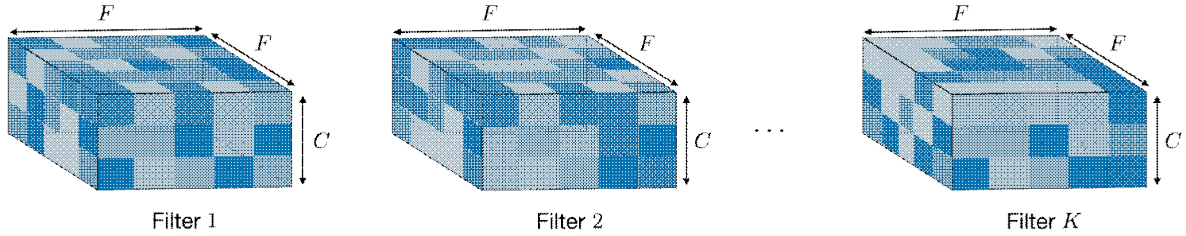
- The CONV layer's parameters consist of a set of learnable filters
- Every filter is small spatially (along width and height), but extends through the full depth of the input volume
- During the forward pass, we slide (more precisely, convolve) each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position.
- Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer, or eventually entire honeycomb or

wheel-like patterns on higher layers of the network

A filter example (Eg: edge detection):

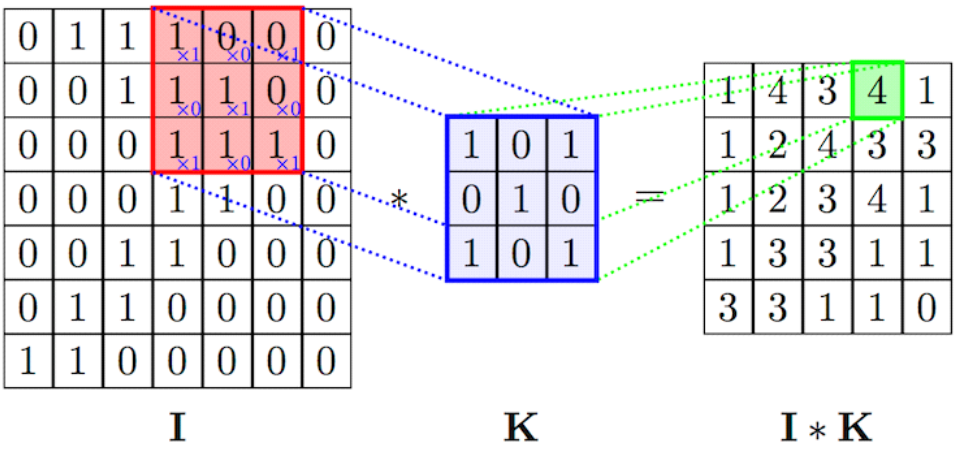


How a filter looks like in 3D:



Convolution operation:

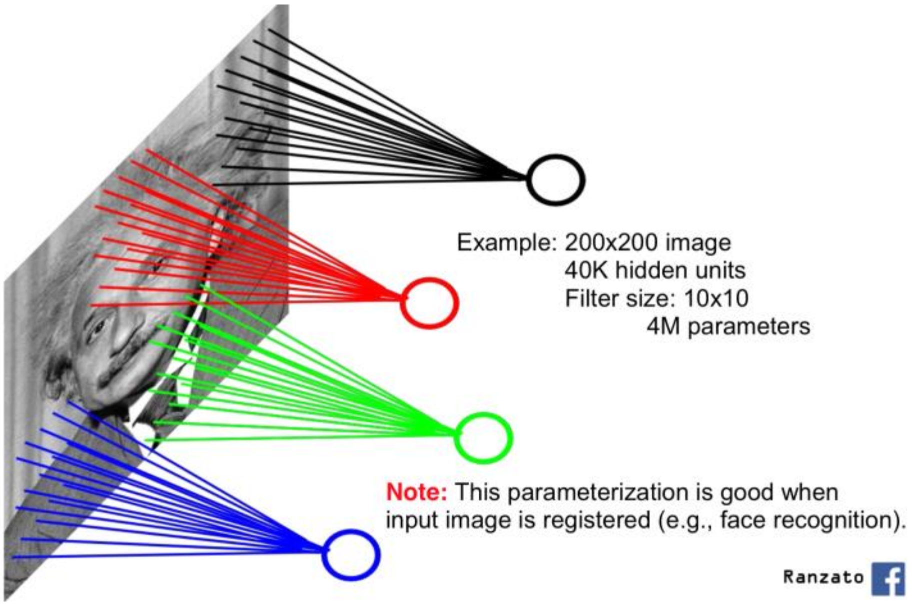
- Each convolution filter represents specific feature set e.g eyes, ears etc.
- The output signal strength is not dependent on where the features are located, but simply whether the features are present.
- A cat could be sitting in different positions, and the CNN algorithm would still be able to recognize it.



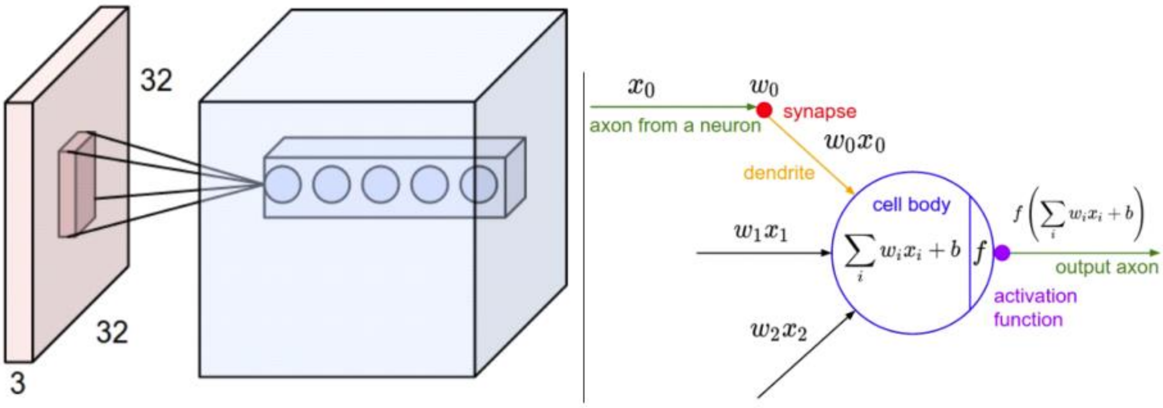
Local Connectivity:

Main Points:

- Instead of connect neurons to all neurons in the prev layer, will connect each neuron to only a local region of the input volume.
- This local region is defined by the **receptive field** (or filter size), which is width and height of the region.
- Each neuron is always full along the depth dimension, refer as depth connectivity
(eg: in RGB img, each neuron will be connected to all 3 channels)



Example:
- Input: an RGB CIFAR img, size of [32x32x3]
- If the receptive field (filter size) is 5x5, each neuron in Conv Layer will have weights to a [5x5x3] region in the input volume
- A total of 5*5*3 = 75 weights (+1 bias parameter)
- The extent of the connectivity along the depth axis must be 3, since this is the depth of the input volume.

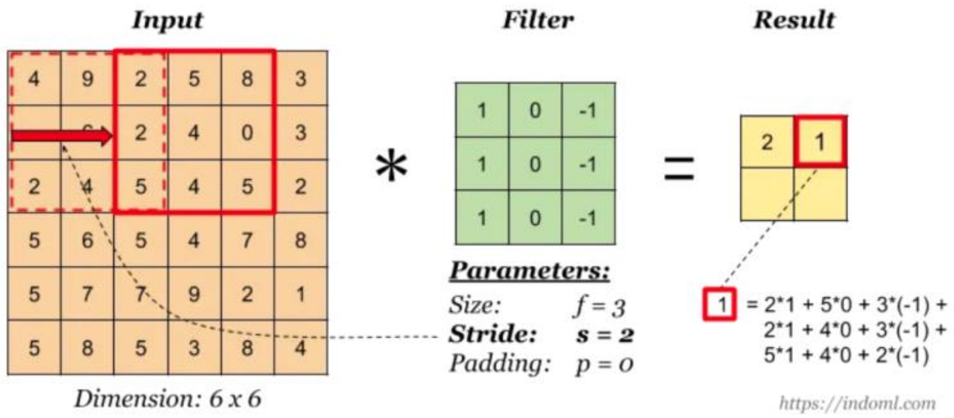


Spatial Arrangements:
Determined by 3 hyperparameters: **depth, stride, zero-padding**

Depth:
- The depth of the output volume corresponds to the number of filters used in the Conv layer
- Eg: 1st Conv layer takes as input the raw img, diff neurons along depth dim activate in the presence of oriented edges, or blobs of color.

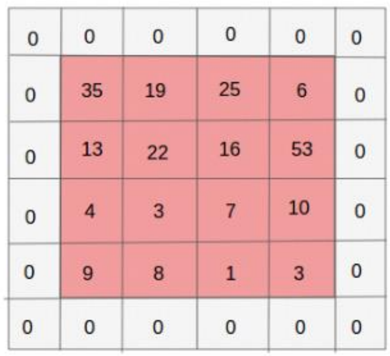
Stride:
- The stride determines how the filter moves across the input volume.
- A stride of 1 means the filter moves one pixel at a time, while a stride of 2 means it moves two pixels at a time.

Example stride of 2:



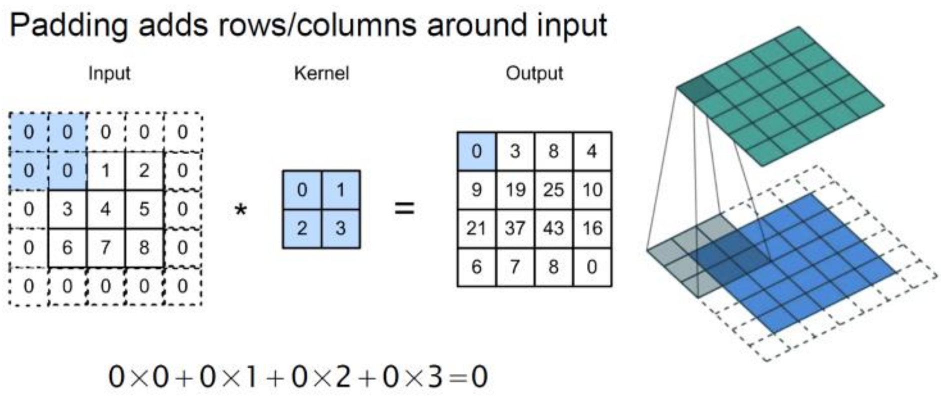
Zero-padding:

What it does?
Adding zeros around the border of the input volume



Why it's useful:

- To control the spatial size of the output volume.
- Padding can help preserve the spatial dimensions of the input volume.
- Eg: if you want the output volume to have the same width and height as the input, you can use appropriate padding.



Spatial Size Calculation:

Spatial size of the output volume is:

$$\text{Output Size} = \frac{W - F + 2P}{S} + 1$$

Where:

- W: input volume size (width or height)
- F: filter size (receptive field size of conv layer neurons)
- P: amount of zero-padding
- S: stride

Example:

- a 7x7 input and a 3x3 filter with stride 1 and pad 0, we would get a 5x5 output.
- a 7x7 input and a 3x3 filter with stride 2 and pad 0, we would get a 3x3 output.

The use of zero-padding:

- In general, setting zero padding to be P=(F−1)/2 when the stride is S=1 ensures that the input volume and output volume will have the same size spatially.

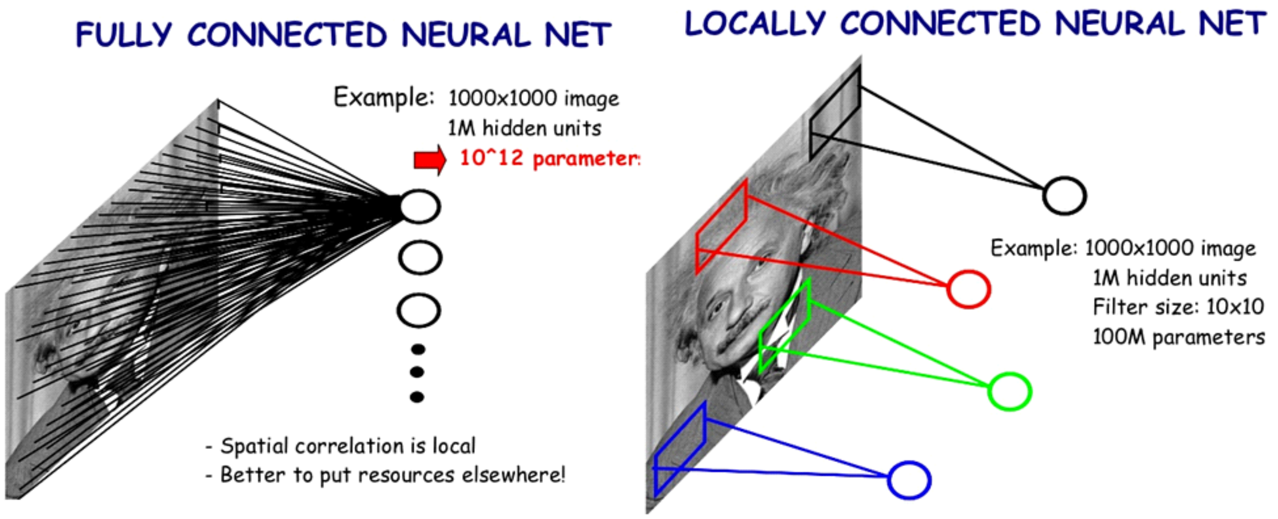
Constraints on strides

- Note that certain combinations of inputs (W, F, P, S) can be invalid, as they do not produce integer output size
- Eg: Input size W=10, no-padding P=0, filter size F=3, then it's impossible to use stride S=2, since (W−F+2P)/S+1=(10−3+0)/2+1=4.5 (not an integer, indicating that the neurons don't "fit" neatly and symmetrically across the input)
- CNN library might handle this by throwing an exception, zero-padding the rest, or cropping the input to make it fit.

Parameter Sharing:

Without parameter sharing:

- Without parameter sharing, each neuron in the convolutional layer would have its own set of weights and biases.
- For example, in the first convolutional layer of AlexNet, there are 55x55x96 = 290,400 neurons, each with 11x11x3 = 363 weights and 1 bias. **This results in a total of 290,400 x 364 = 105,705,600 parameters, which is extremely high.**



Main idea:

- Parameter sharing is a technique used to significantly reduce the number of parameters in the network.
- If one feature is useful to compute at some spatial position (x,y), then it should also be useful to compute at a different position (x2,y2).

Depth slices:

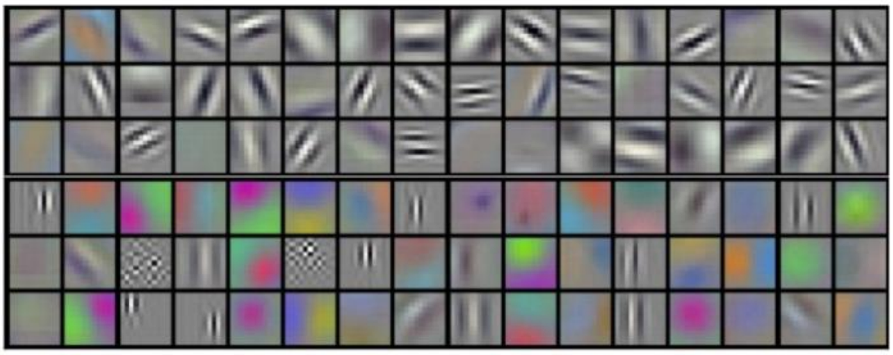
- A depth slice refers to a 2-dimensional slice of the output volume along the depth dimension. For example, a volume of size 55x55x96 has 96 depth slices, each of size 55x55.
- In parameter sharing, all neurons in a single depth slice use the same set of weights and bias. This means that instead of having unique weights for each neuron, we have one set of weights shared across all neurons in the depth slice.

Reduction in Parameters:

- With parameter sharing, the first convolutional layer in the example would have only 96 unique sets of weights (one for each depth slice). **Each set of weights has 11x11x3 = 34,848 unique weights, plus 96 biases, resulting in a total of 34,944 parameters.**
- This is a dramatic reduction compared to the 105,705,600 parameters without parameter sharing.

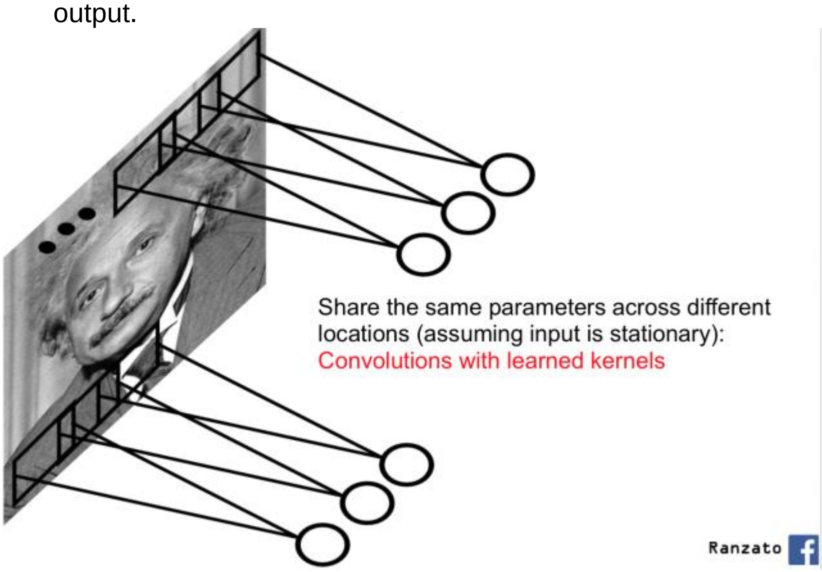
Example of filters learned in a CNN:

- Each of the 96 filters shown here is of size [11x11x3], and each one is shared by the 55*55 neurons in one depth slice.
- If detecting a horizontal edge is important at some location in the image, it should intuitively be useful at some other location as well due to the translationally-invariant structure of images.
- Therefore no need to relearn to detect a horizontal edge at every one of the 55*55 distinct locations in the Conv layer output volume.

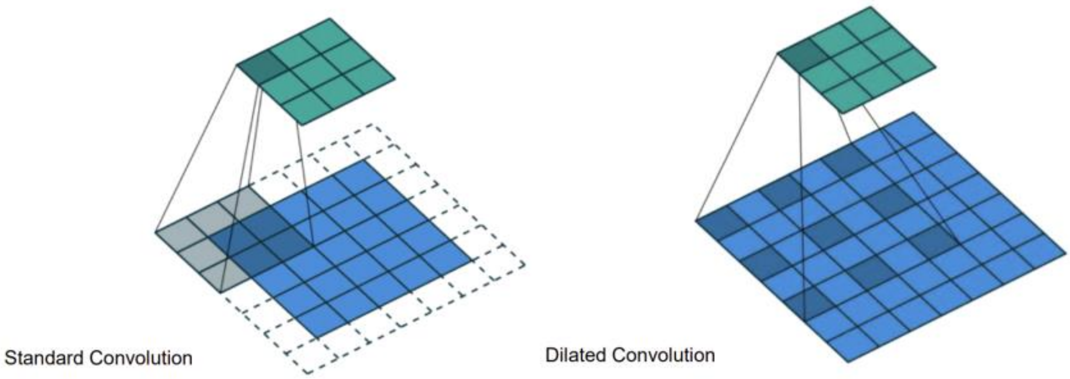


Convolution operation:

- Since all neurons in a depth slice use the same weights, the forward pass of the convolutional layer can be computed as a convolution of the filter (set of weights) with the input volume. This is why the layer is called a Conv Layer.
- The filter (or kernel) slides over the input volume, applying the same weights at each position to produce the output.



Dilated Convolution:

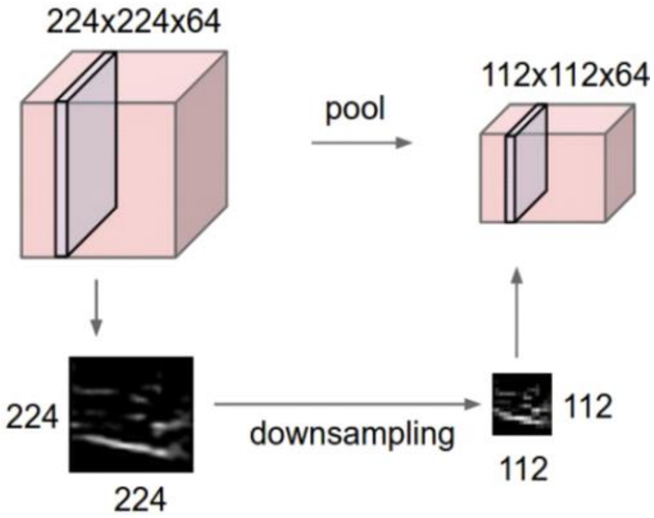


Summary of Conv layer:

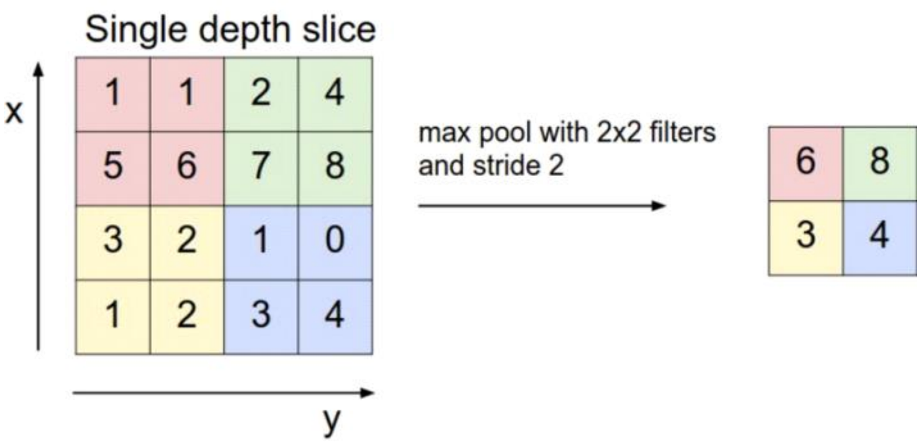
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Pooling Layer

Pooling Layer is used to reduce the spatial dimensions of the input volume, which helps in reducing the number of parameters and computation, and also helps control overfitting.



Types:
- Max Pooling (Takes the max value)
- Average Pooling (Takes the avg value)

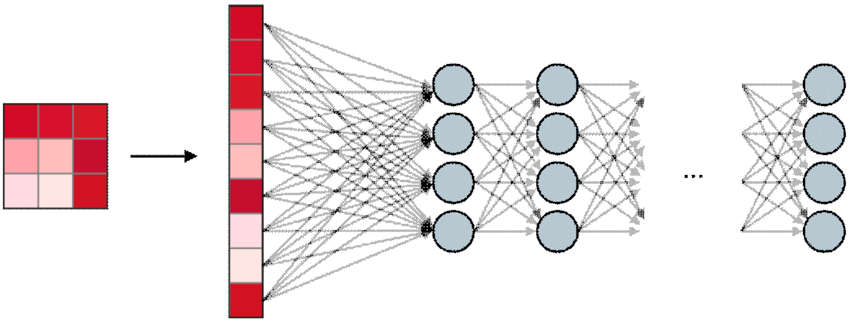


A detailed explaination:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- For Pooling layers, it is not common to pad the input using zero-padding.

Fully Connected Layer

- Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks.
- Their activations can hence be computed with a matrix multiplication followed by a bias offset.



Normalization Layer

Many types of normalization layers have been proposed for use in CNN architectures, sometimes with the intentions of implementing inhibition schemes observed in the biological brain. However, these layers have since fallen out of favor because in practice their contribution has been shown to be minimal, if any.

Layer Patterns

- The most common form of a ConvNet architecture stacks a few CONV-RELU layers, follows them with POOL layers, and repeats this pattern until the image has been merged spatially to a small size.
- At some point, it is common to transition to fully-connected layers. The last fully-connected layer holds the output, such as the class scores.

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

where the `*` indicates repetition, and the `POOL?` indicates an optional pooling layer. Moreover, `N >= 0` (and usually `N <= 3`), `M >= 0`, `K >= 0` (and usually `K < 3`). For example, here are some common ConvNet architectures you may see that follow this pattern:

- `INPUT -> FC`, implements a linear classifier. Here `N = M = K = 0`.
- `INPUT -> CONV -> RELU -> FC`
- `INPUT -> [CONV -> RELU -> POOL]*2 -> FC -> RELU -> FC`. Here we see that there is a single CONV layer between every POOL layer.
- `INPUT -> [CONV -> RELU -> CONV -> RELU -> POOL]*3 -> [FC -> RELU]*2 -> FC` Here we see 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.

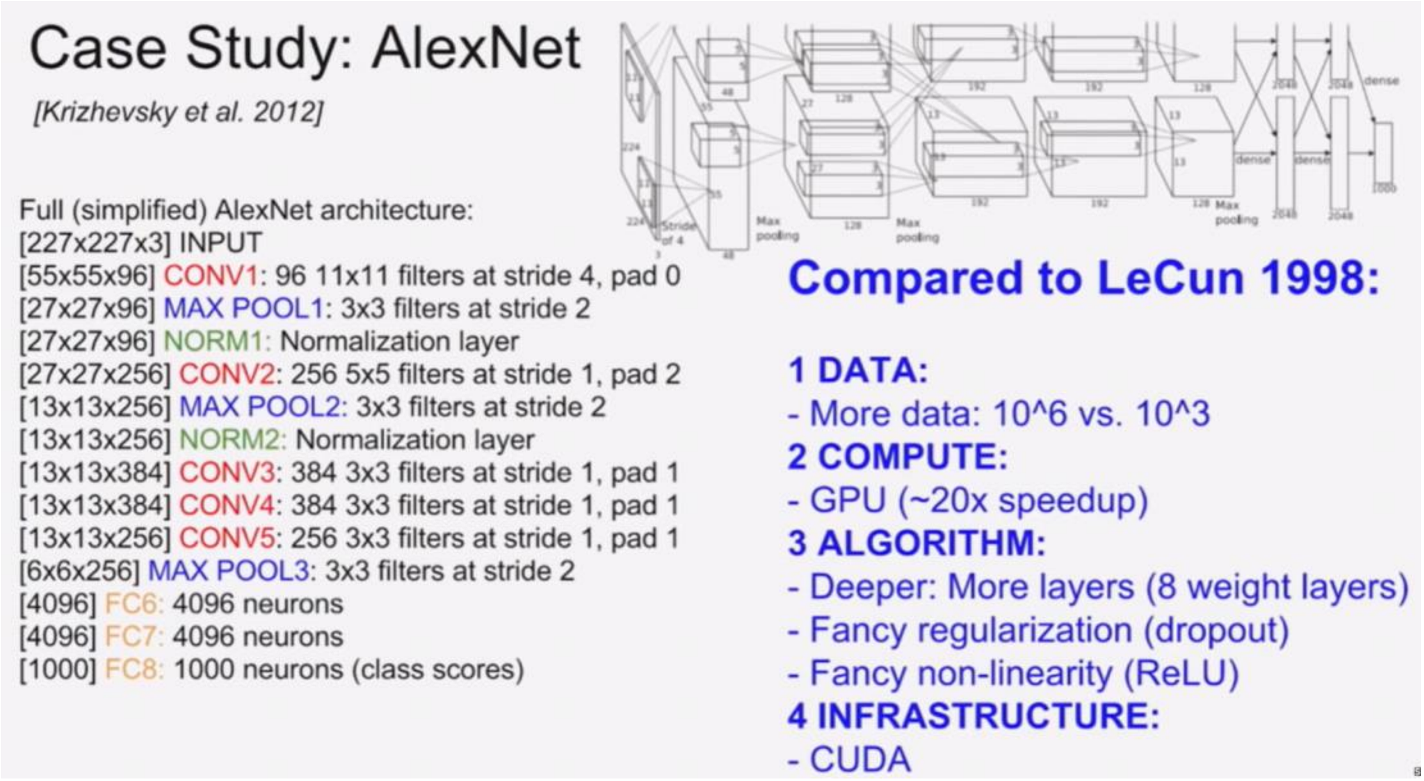
Case studies:

Classic CNN:

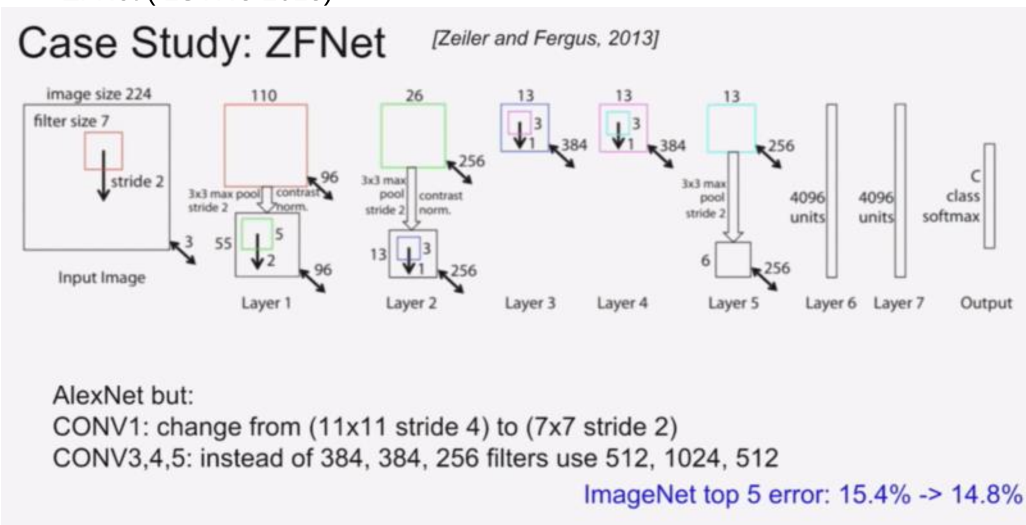
- LeNet-5

Modern CNN:

- AlexNet (ILSVRC 2012)



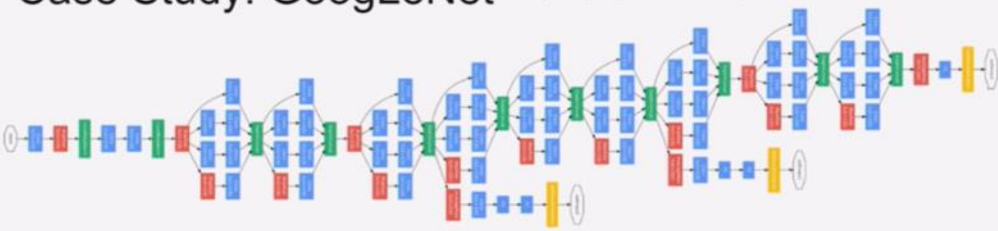
- ZFNet (ILSVRC 2013)

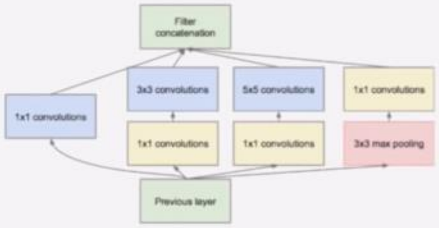


- GoogLeNet (ILSVRC 2014)

Case Study: GoogLeNet

[Szegedy et al., 2014]





Inception module

ILSVRC 2014 winner (6.7% top 5 error)

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Fun features:

- Only 5 million params!
(Removes FC layers
completely)

Compared to AlexNet:
- 12X less params
- 2x more compute
- 6.67% (vs. 16.4%)

- VGGNet (ILSVRC 2014)

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

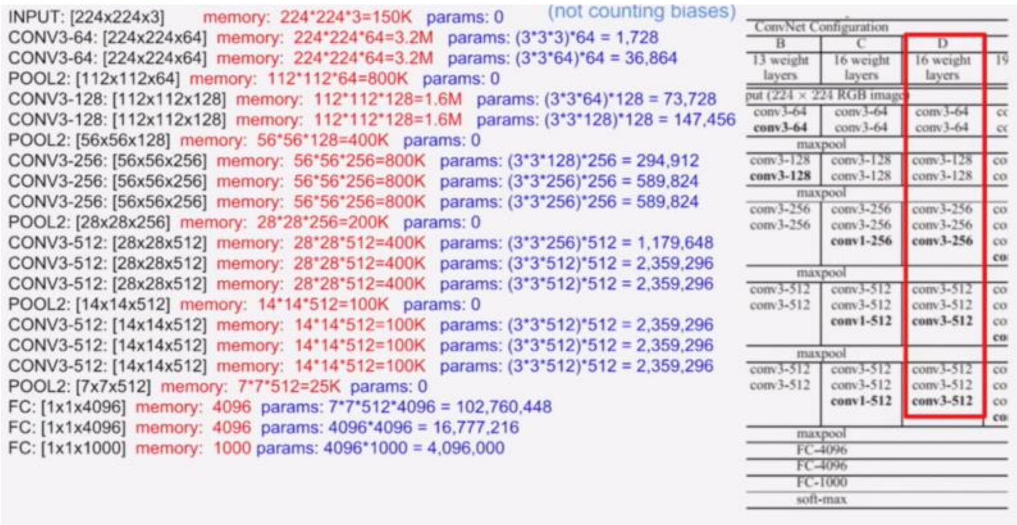
best model

11.2% top 5 error in ILSVRC 2013
->
7.3% top 5 error

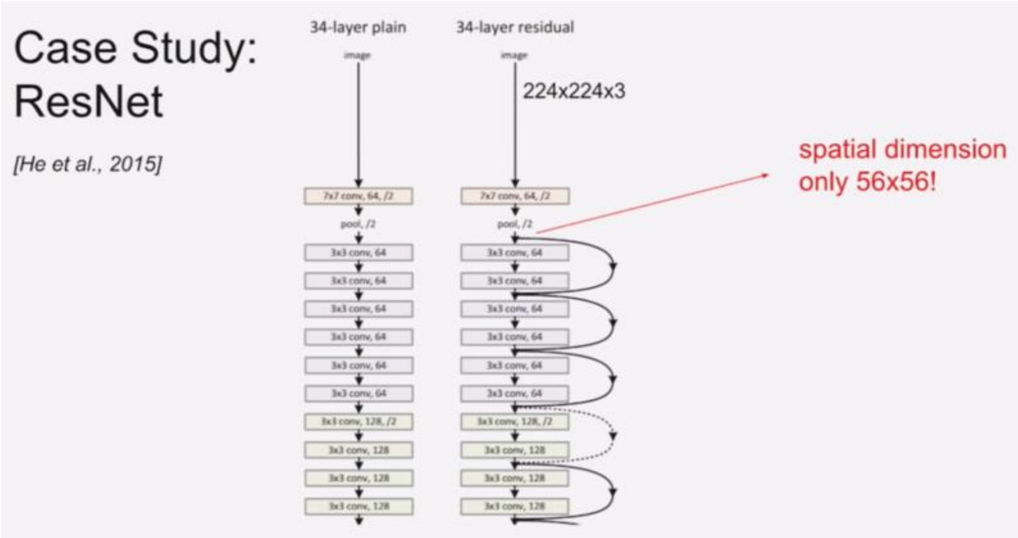
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

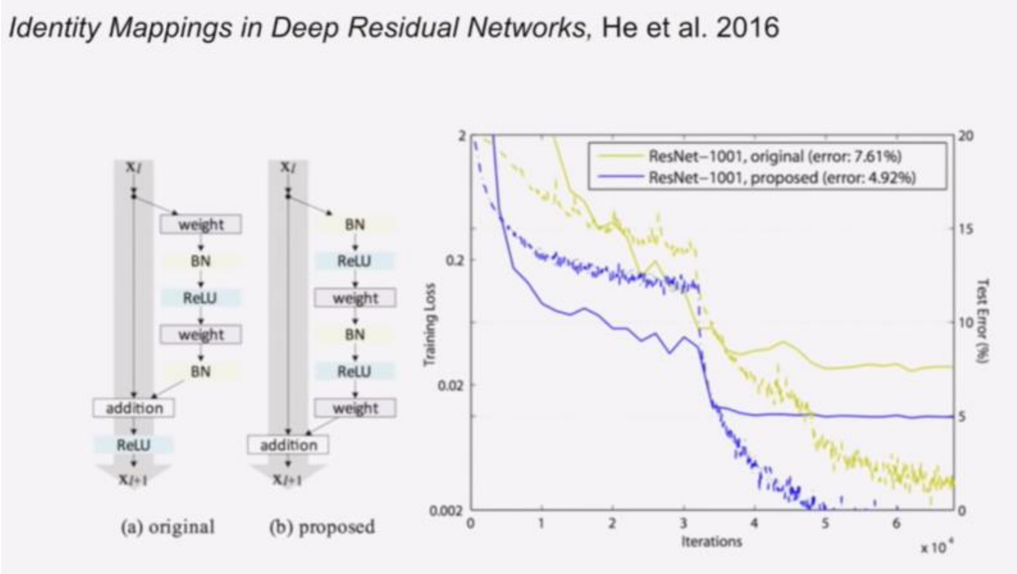
Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144



- ResNet (ILSVRC 2015)





PyTorch:

Format [Link]:

- torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')

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, tuple or str, optional*) – Padding added to all four sides of the input. Default: 0
- **padding_mode** (*str, 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`

Shape:

- Input: $(N, C_{in}, H_{in}, W_{in})$ or (C_{in}, H_{in}, W_{in})
- Output: $(N, C_{out}, H_{out}, W_{out})$ or $(C_{out}, H_{out}, W_{out})$, where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

Examples:

```
>>> # With square kernels and equal stride
>>> m = nn.Conv2d(16, 33, 3, stride=2)
>>> # non-square kernels and unequal stride and with padding
>>> m = nn.Conv2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2))
>>> # non-square kernels and unequal stride and with padding and dilation
>>> m = nn.Conv2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2), dilation=(3, 1))
>>> input = torch.randn(20, 16, 50, 100)
>>> output = m(input)
```

References

CNN Links:

CNN Stanford CS231n Lecture Note	CS231n CNN
Github CNN Animations	CNN Animations
CNN Toronto Lecture Notes	CNN Toronto
ConvNetJS CIFAR-10 demo	CNN Demo
YouTube CNN for Computer Vision	YouTube