Neural network

Convolution neural network

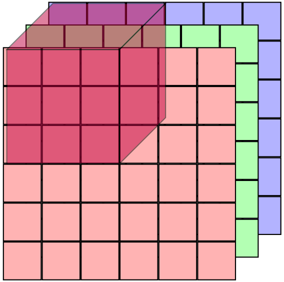
Khiem Nguyen

Email	khiem.nguyen@glasgow.ac.uk
MS Teams	khiem.nguyen@glasgow.ac.uk
Whatsapp	+44 7729 532071 (Emergency only)

May 18, 2025

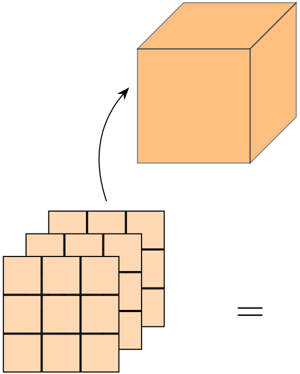
Table of Contents

Convolutions on RGB image



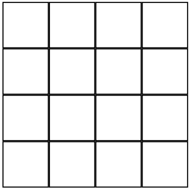
$6 \times 6 \times 3$

$*$



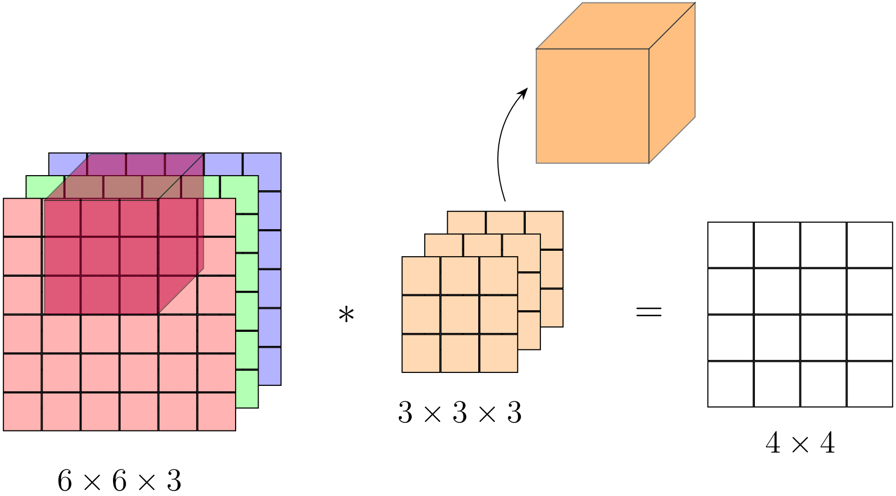
$3 \times 3 \times 3$

$=$

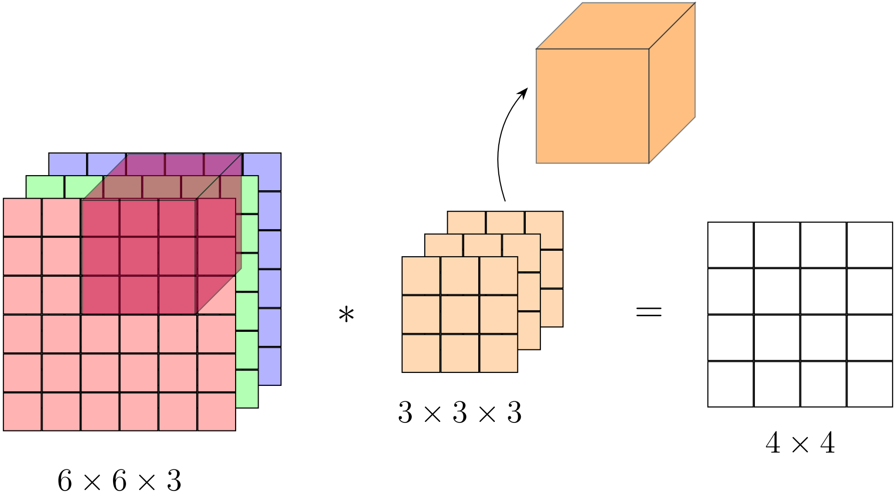


4×4

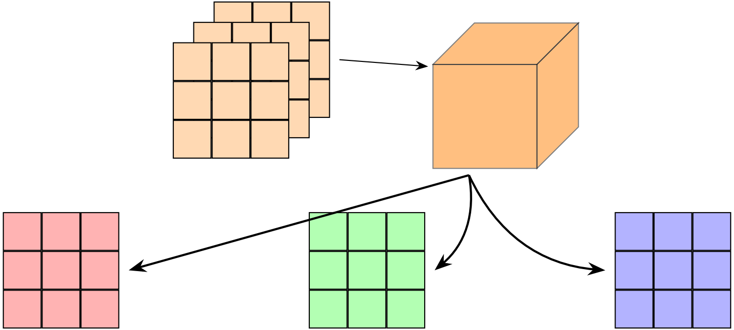
Convolutions on RGB image



Convolutions on RGB image



Convolutions on RGB image



Few words on drawing:

- We can think of the filter with three channels of the same color channels as the input.
- When we have more channels in deeper layers (more on this later), it probably does not make a lot of sense to think of color channels.

Movement of the filter/kernel

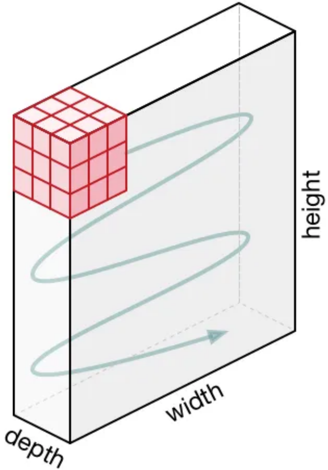
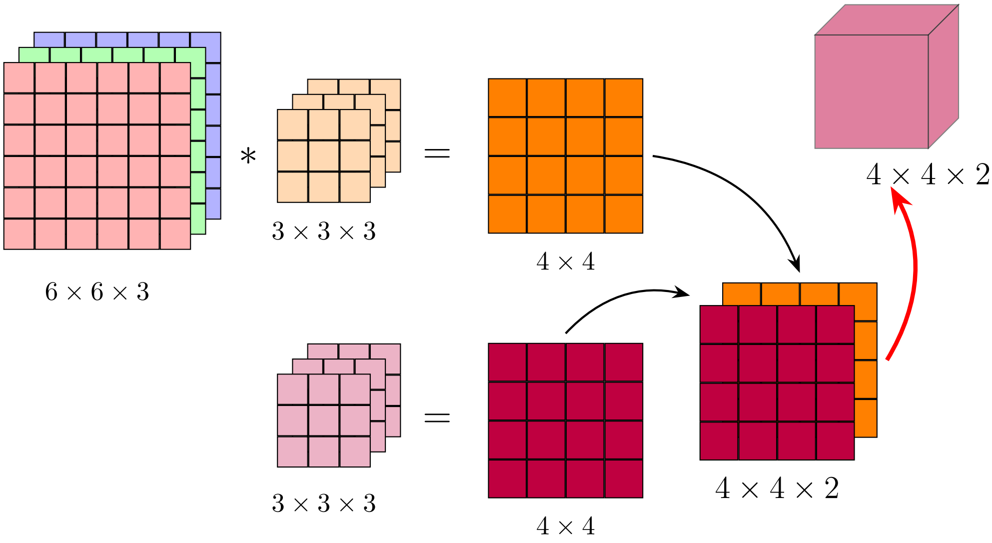
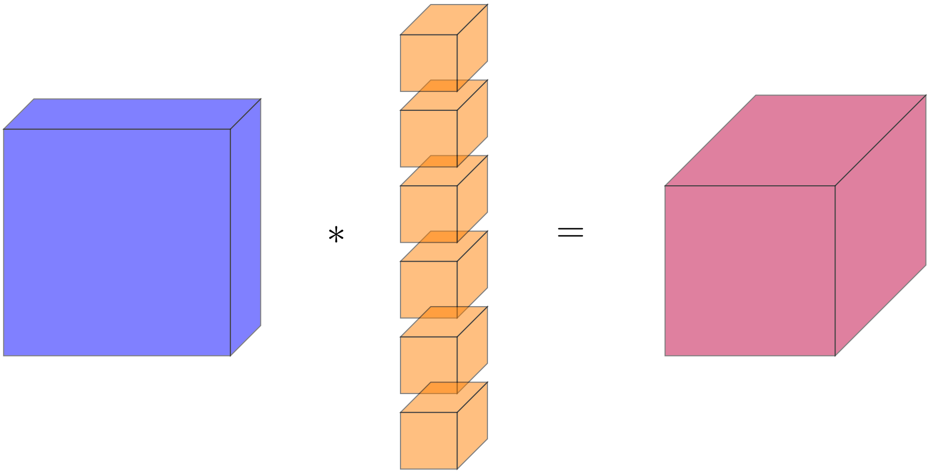


Figure: Movement of the kernel through one layer

Multiple filters

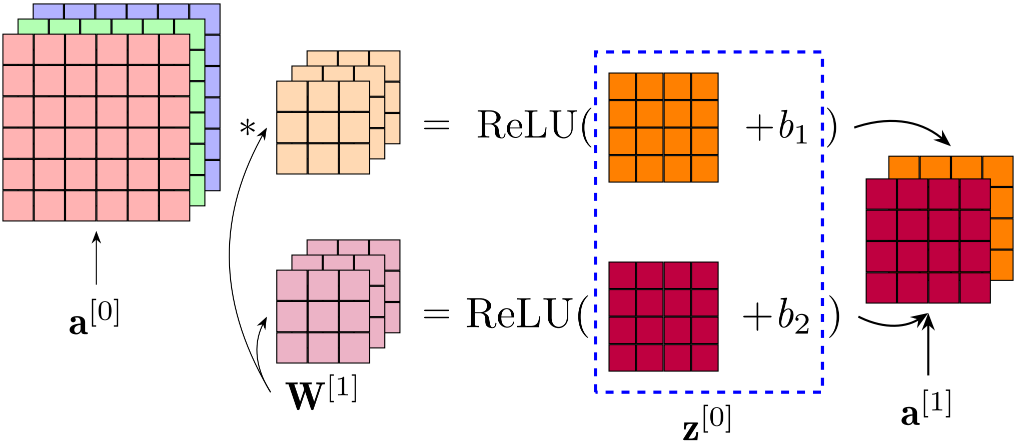


Multiple filters



$$(n \times n \times {\color{blue}n_C}) \quad * \quad \underbrace{(f \times f \times {\color{blue}n_C})}_{\text{using } n'_C \text{ filters}} \quad \Rightarrow \quad (n-f+1) \times (n-f+1) \times {\color{red}n'_C}$$

One layer of a Convolutional Network



$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{a}^{[0]} + \mathbf{b}^{[1]}$$

$$\mathbf{a}^{[1]} = g(\mathbf{z}^{[1]})$$

Example

Let us summarize the theory we have learned so far by an example:

Compute number of parameters in one layer!!!

If we have 10 filters of the size $(3 \times 3 \times 3)$ in one layer of a neural network, how many parameters does that layer have?

Summary of notation

If layer l is a convolution layer:

$$
\begin{aligned}
f^{[l]} &= \text{filter size} \\
p^{[l]} &= \text{padding} \\
s^{[l]} &= \text{stride} \\
n_C^{[l]} &= \text{number of filters} \\
&= \text{number of channels of the output "image"}
\end{aligned}
$$

- Each filter: $f^{[l]} \times f^{[l]} \times n_C^{[l-1]}$
- Activations: $a^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$
- Weights: $f^{[l]} \times f^{[l]} \times n_C^{[l-1]} \times n_C^{[l]}$

- Input: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_C^{[l-1]}$
- Output: $n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$

with

$$n_H^{[l]} = \left\lfloor \frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

$$n_W^{[l]} = \left\lfloor \frac{n_W^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

$$A^{[l]} \quad \rightarrow \quad m \times n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$$

Convolution Layer in PyTorch

Conv2d

CLASS torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [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.

This module supports TensorFloat32.

On certain ROCm devices, when using float16 inputs this module will use different precision for backward.

Documentation: Click on Me!

Convolution Layer in PyTorch

`nn.Conv2d`

In the simplest case, the input and output of the layer have the size

- Input size (N, C_{in}, H, W)
- Output size $(N, C_{\text{out}}, H, W)$
- To understand keyword argument `dilation`, cf. Visualization of convolution layer

➪ The computation in the `nn.Conv2d` is given by (from PyTorch description)

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

Convolution Layer in PyTorch

Input size (N, C_{in}, H, W)

Output size $(N, C_{\text{out}}, H, W)$

➪ Wee "cute" equation from PyTorch

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

➪ We need "decode" this wee cute equation

N_i	for the i^{th} input example
C_{out_j}	for the j^{th} channel in the output
C_{in_j}	number of channels of the input ('image')
k	running index for the k^{th} filter (kernel)

In deeper layer, the input does not carry the meaning of image we can see or visualize easily on the computer (too many channels).

Convolution Layer in PyTorch

Quick comparison `nn.Linear` versus `nn.Conv2d`

`nn.Linear`

- Input size (N, H_{in})
- Output size (N, H_{out})

`nn.Conv2d`

- Input size (N, C_{in}, H, W)
- Output size $(N, C_{\text{out}}, H, W)$

Trick If you kinda think of

H_{in} (`in_features`) as C_{in} (`in_channels`

H_{out} (`out_features`) as C_{out}

$\Rightarrow$ the number of neurons in fully connected layer is some what like number of channels in convolution layer.

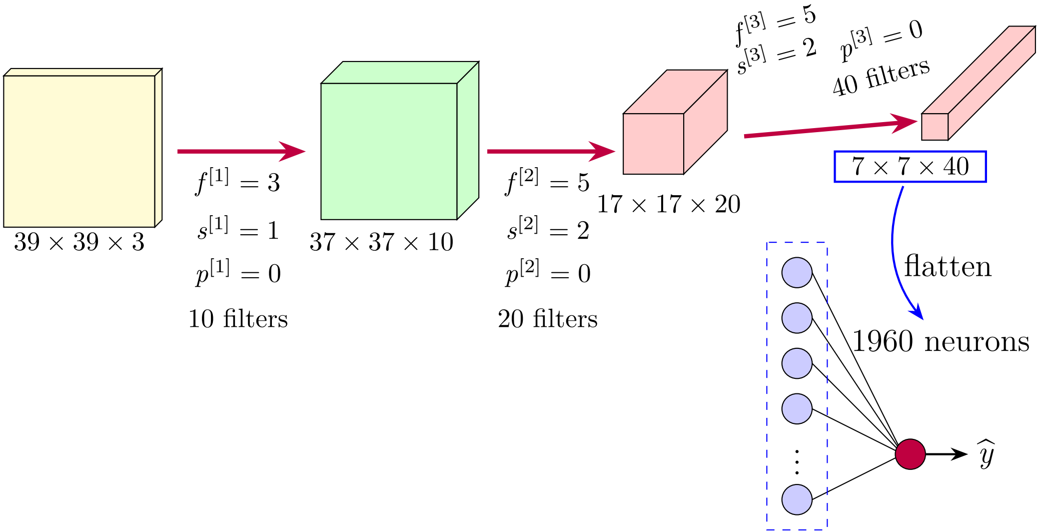
Note This is just a comparison for our memory. Of course, these concepts should not be mixed and confused.

Example of ConvNet

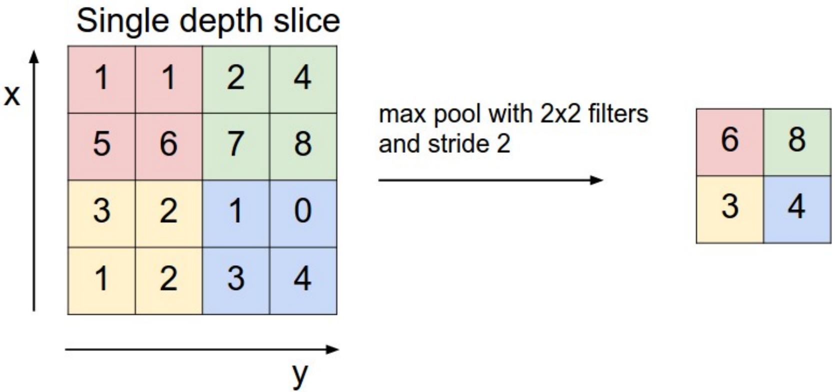
Congrat! Now you understand the key concept of Convolution Layer,

and how to construct it with PyTorch

Example of ConvNet



Pooling layer: Max Pooling



Hyperparameters

- Filter: $f = 2$
- Stride: $s = 2$

Pooling layer: Max pooling

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

→

9		

Hyperparameters

- Filter: $f = 2$
- Stride: $s = 2$

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

$\longrightarrow$

9	9	

Hyperparameters

- Filter: $f = 2$
- Stride: $s = 2$

Pooling layer: Max pooling

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

$\longrightarrow$

9	9	5

Hyperparameters

- Filter: $f = 2$
- Stride: $s = 2$

Pooling layer: Max pooling

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

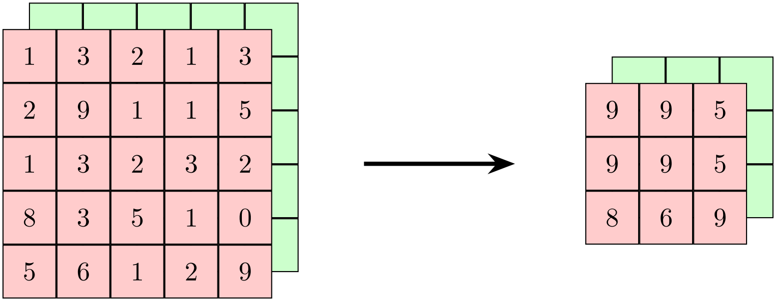
$\longrightarrow$

9	9	5
9	9	5
8	6	9

Hyperparameters

- Filter: $f = 2$
- Stride: $s = 2$

Pooling layer: Max pooling

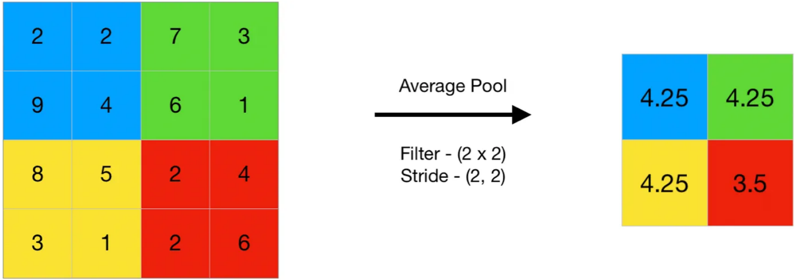


As before, we can apply the pooling over many channels in the image.

Hyperparameters

- Filter: $f = 2$
- Stride: $s = 2$

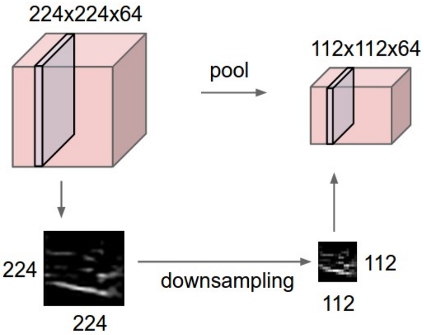
Pooling layer: Average pooling



Hyperparameters

- Filter: $f = 2$
- Stride: $s = 2$

Summary of pooling



- Reduce the size of “input” image
- Two main types: Max pooling or Average pooling
- Max pooling is used in most of the modern convolution neural networks
- Hyperparameters: filter size and stride
- **No parameters to learn!**

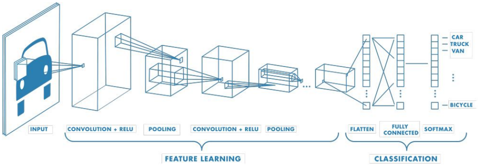
Types of layer in a convolutional network

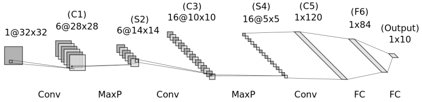
- Convolution (CONV/C)
- Pooling (POOL/P)
- Fully connected (FC)

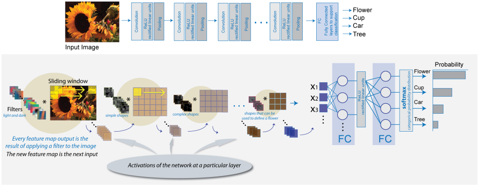
Table of Contents

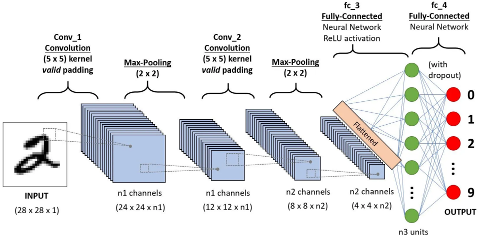
Various visualization of convolution neural networks

- There have been various ways of visualizing convolution neural networks.
- Although a CNN can be presented in different formats, it is important to interpret them with proper mechanism/mathematics behind the hood.









Various visualization of convolution neural networks

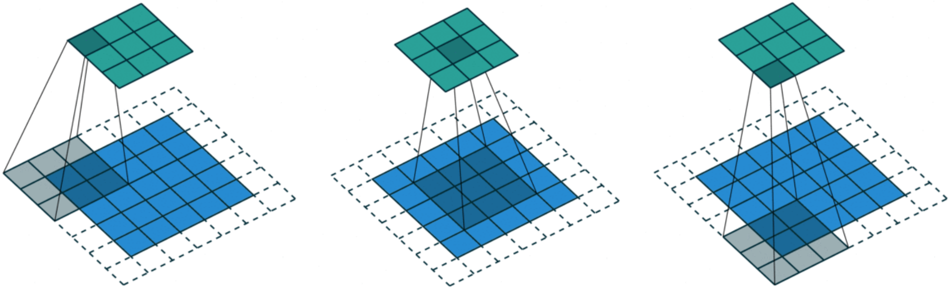
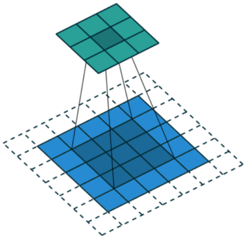
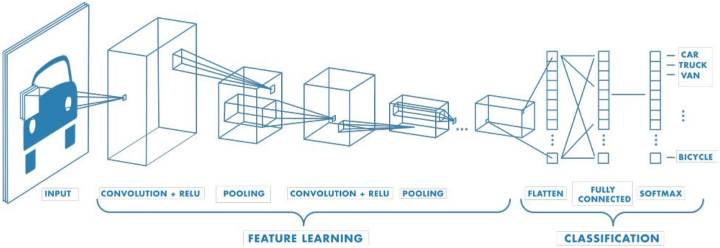


Figure: A common visualization of the convolution operation on one channel

Various visualization of convolution neural networks

Such visualization inspires the following representation





The visualization is absolutely useless and superficial as it is not an implementable architecture.

Various visualization of convolution neural networks

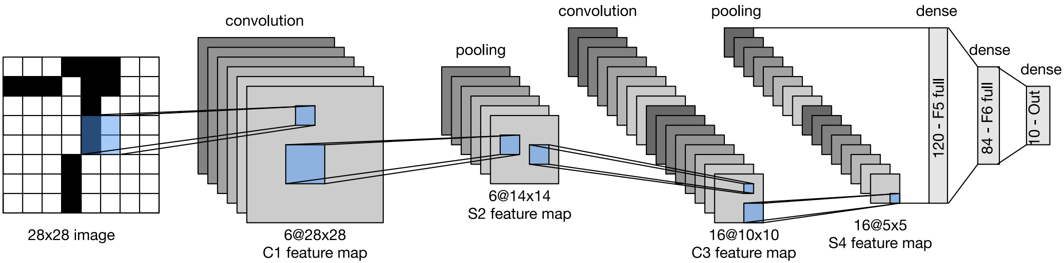
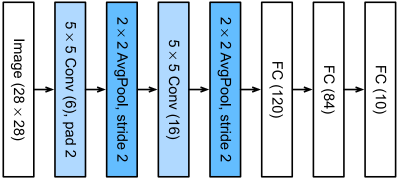


Figure: LeNet-5 Convolution Neural Network

- This convolution neural network was presented in 1998 by Yan LeCun (a god-level machine learning scientist).
- This visualization is useful as we know what we are dealing with.
- The "image" dimension $(n_H \times n_W \times n_C)$ is presented as $n_C\,@\,n_H \times n_W$.

Various visualization of convolution neural networks

- The modern networks are so big that it is difficult to visualize by drawing.
- Block diagrams in such cases are more useful.
- Top: LeNet, Bottom: AlexNet



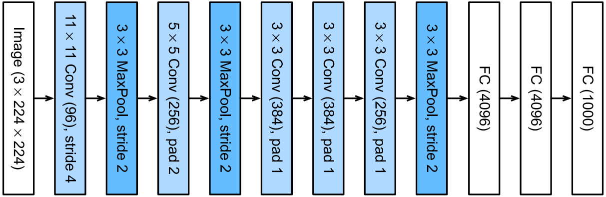


Table of Contents

Neural network example

	Activation shape	Activation size	# parameters
Input	$(32, 32, 3)$	?	?
CONV1 ($f = 5$, $s = 1$)	$(28, 28, 8)$	?	?
POOL1	$(14, 14, 8)$	?	?
CONV2 ($f = 5$, $s = 1$)	$(10, 10, 16)$	?	?
POOL2	$(5, 5, 16)$	?	?
FC3	$(120, 1)$	?	?
FC4	$(84, 1)$	?	?
Softmax	$(10, 1)$	?	?

Let us do an exercise: Compute the learnable parameters in the model!

Neural network example

	Activation shape	Activation size	# parameters
Input	$(32, 32, 3)$	3072	?
CONV1 ($f = 5, s = 1$)	$(28, 28, 8)$	6272	?
POOL1	$(14, 14, 8)$	1568	?
CONV2 ($f = 5, s = 1$)	$(10, 10, 16)$	1600	?
POOL2	$(5, 5, 16)$	400	?
FC3	$(120, 1)$	120	?
FC4	$(84, 1)$	84	?
Softmax	$(10, 1)$	10	?

$(A): \quad (5 \cdot 5 \cdot 3 + 1) \cdot 8 = 608$

$(B): \quad (5 \cdot 5 \cdot 8 + 1) \cdot 16 = 3216$

$(C): \quad 400 \cdot 120 + 120 = 48120$

$(D): \quad 128 \cdot 84 + 84 = 10164$

$(E): \quad 84 \cdot 10 + 10 = 850$

Neural network example

	Activation shape	Activation size	# parameters
Input	$(32, 32, 3)$	3072	0
CONV1 ($f = 5$, $s = 1$)	$(28, 28, 8)$	6272	(A) 608
POOL1	$(14, 14, 8)$	1568	0
CONV2 ($f = 5$, $s = 1$)	$(10, 10, 16)$	1600	(B) 3216
POOL2	$(5, 5, 16)$	400	0
FC3	$(120, 1)$	120	(C) 48120
FC4	$(84, 1)$	84	(D) 10164
Softmax	$(10, 1)$	10	(E) 850

$(A): \quad (5 \cdot 5 \cdot 3 + 1) \cdot 8 = 608$

$(B): \quad (5 \cdot 5 \cdot 8 + 1) \cdot 16 = 3216$

$(E): \quad 84 \cdot 10 + 10 = 850$

$(C): \quad 400 \cdot 120 + 120 = 48120$

$(D): \quad 120 \cdot 84 + 84 = 10164$

Table of Contents

Why look at classic convolution neural networks

Why look at classic convolution neural networks?

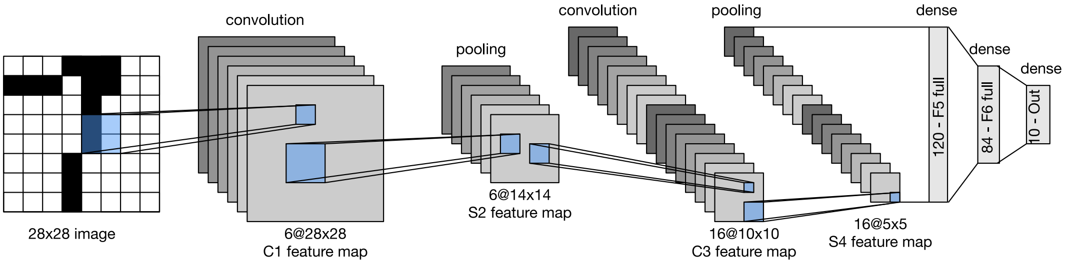
- Get intuition and update your own knowledge
- Understand the theory better (just like how we look at code and learn coding)
- One trained neural network working well on one computer vision task can be applied to other computer vision tasks through the concept of transfer learning

Classic convolution neural networks

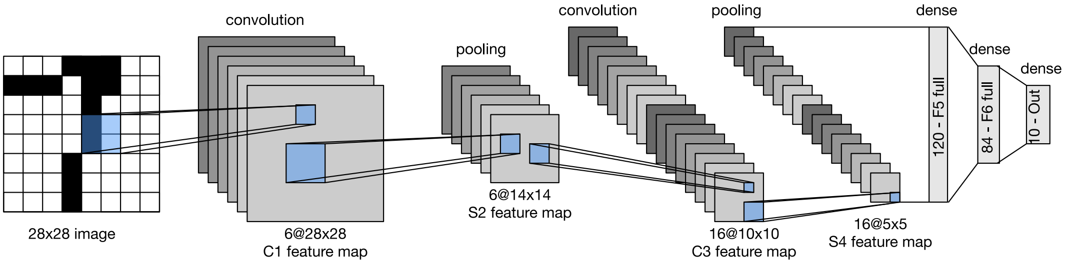
- **LeNet-5**: This CNN is considered very simple nowadays but was considered one of the pioneers in 1998. LeNet-1 was train in 1989.
  - Original paper
  - Tutorial
- **AlexNet**: A large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes.
  - Original paper.
  - Pretrained model from `torchvision`.
  - Pretrained model from PyTorch Hub
- **VGG**: Very Deep Convolutional Networks for Large-Scale Image Recognition.
  - Original paper.
  - Pretrained models from `torchvision`

Remark: LeNet-5 is so easy to train that I cannot find the pretrained model. The network, which was considered high-tech in the past, is now considered a simple network.

LeNet-5: Diagram

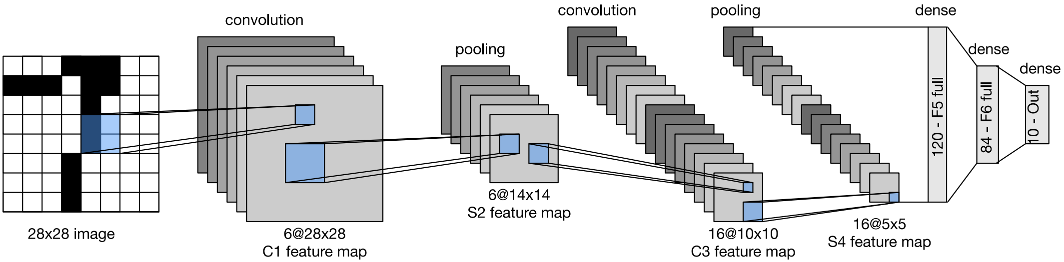


LeNet-5: Diagram



What is missing in this representation?

LeNet-5: Diagram



What is missing in this representation?

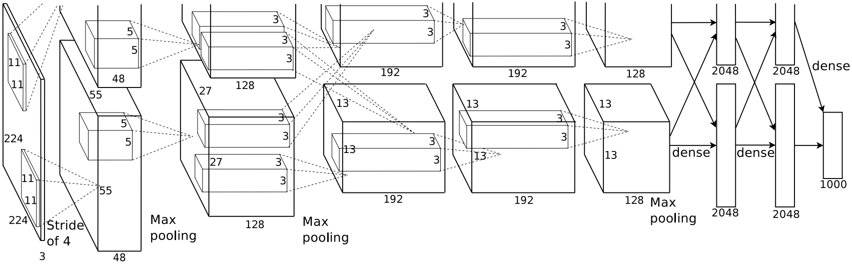
Yes! The activation functions.

LeNet-5: Architecture in table

# map	Layer Size	Feature Kernel size	Stride	Activation		
Input	Image	1	32×32			
1	Convolution	6	28×28	5×5	1	tanh
2	Average Pooling	6	14×14	2×2	2	tanh
3	Convolution	16	10×10	5×5	1	tanh
4	Average Pooling	16	10×10	5×5	1	tanh
5	Convolution	120	1×1	5×5	1	tanh
6	FC		84			tanh
Output	FC		10			softmax

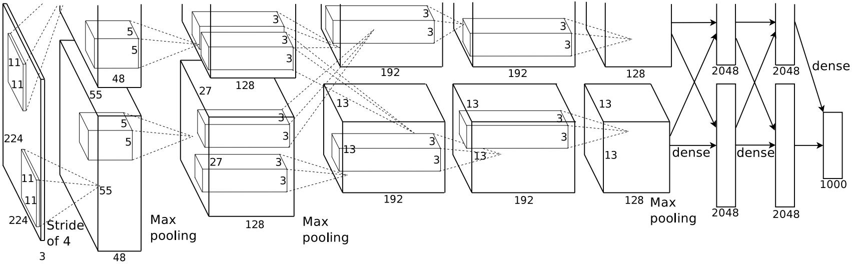
The original paper Click on me! experimented with both tanh and sigmoid as activation functions. The tutorial used ReLU. The idea is essentially the same.

AlexNet



This figure is from the original paper Click on Me!.

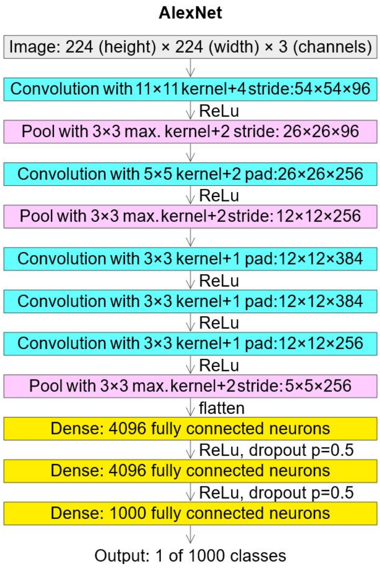
AlexNet



This figure is from the original paper Click on Me!.

- Yup! It looks like somebody cut the figure and it was missing something on the top.
- Everything is actually fine. One block is one filter; and we have many filters.
- Instead of drawing many filters stacked together in a big volume, it drew filters separately.

AlexNet



Little advice: *Always ask whether you learn the right thing.*

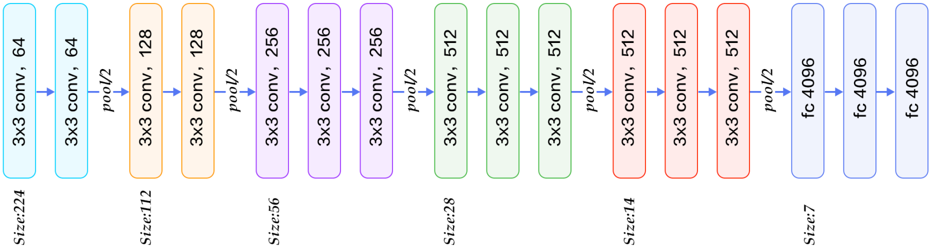
Explanation & background:

- If you work out the model, the input image should have the size

$$(227, 227, 3), \text{ not } (224, 224, 3).$$

- So the original paper has its own *minor* mistake (not *fundamental* mistake). So that's fine :D.
- This has been pointed out by "Andrej Karpathy" and *actually anybody who understand the theory.*

VGG



Try to understand what you end up seeing online: VGG
Original paper

Table of Contents

Putting it together

- Training set $(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(m)}, y^{(m)})$
- Loss function

$$\mathcal{L} = \frac{1}{m} \sum_{i=1}^{m} L(\widehat{y}^{(i)}, y^{(i)}),$$

  where
- Use gradient-based methods (such as gradient descent) to optimize *learnable parameters* to minimize the loss function $\mathcal{L}$.

Why convolutions?

- **Parameter sharing**: A feature detector (such as vertical edge detector) that's useful in one part of the image is probably useful in another part of the image.
- **Sparsity of connections**: In each layer, each output value depends only on small number of inputs.

Inpterpretation of a ConvNet architecture

Hopefully, after studying the basic building block of a convolution neural network, you can read and interpret convolution neural network architectures you bump into on the internet:

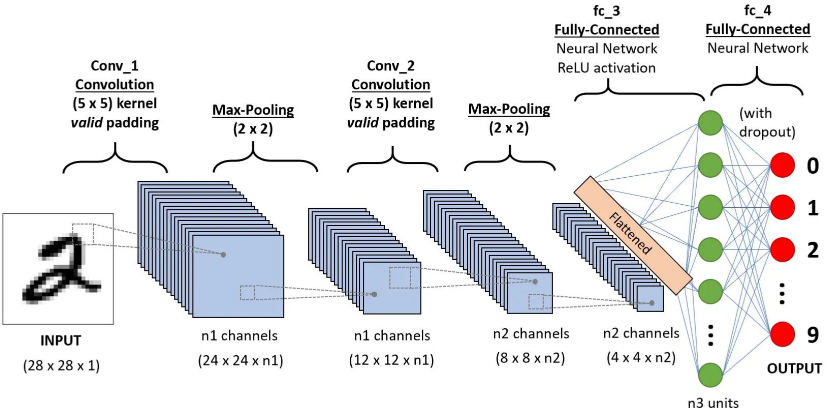


Figure: A ConvNet to classify handwritten digits