Machine Learning for Robotics: **Convolutional Neural Networks**

Prof. Navid Dadkhah Tehrani

To process images with MLP, we had to flatten the images.

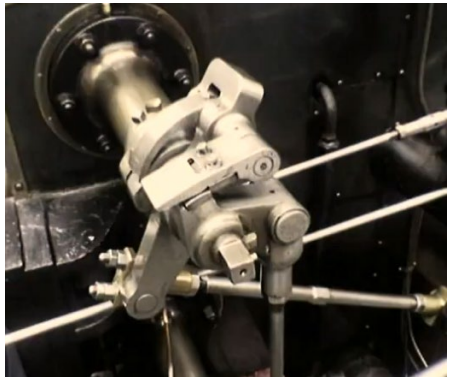
MLPs process images pixel by pixel and that's not very efficient as the image height and width increases.
Also humans don't perceive images pixel by pixel.

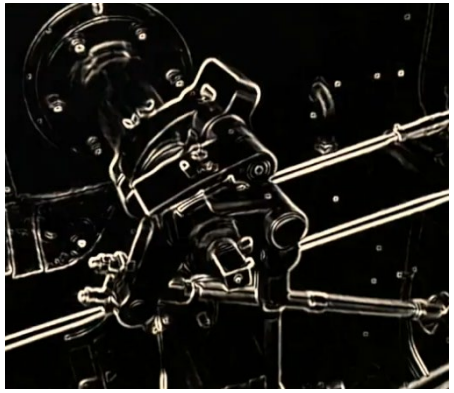
In CNNs, the underlying assumption is that pixels next to each other have local dependencies and are not independent of each other.

CNNs can take the input image as is w/o a need to flatten it (flattening would mess up the local dependencies)

Before looking at CNNs let's see what the convolutional filters are image processing.

In the old days in computer vision would extract features from images with convolutional filters.





In old days, Many research papers were publishing different numerical value for convolutional filters that would extract certain features.

Example of horizontal line detector convolutional filter

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

-1	-1	-1
2	2	2
-1	-1	-1

Convolutional kernel

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

activation

2	1	0
0	0	0
1	2	3

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

-1	-1	-1
2	2	2
-1	-1	-1

Convolutional kernel

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

with padding

activation

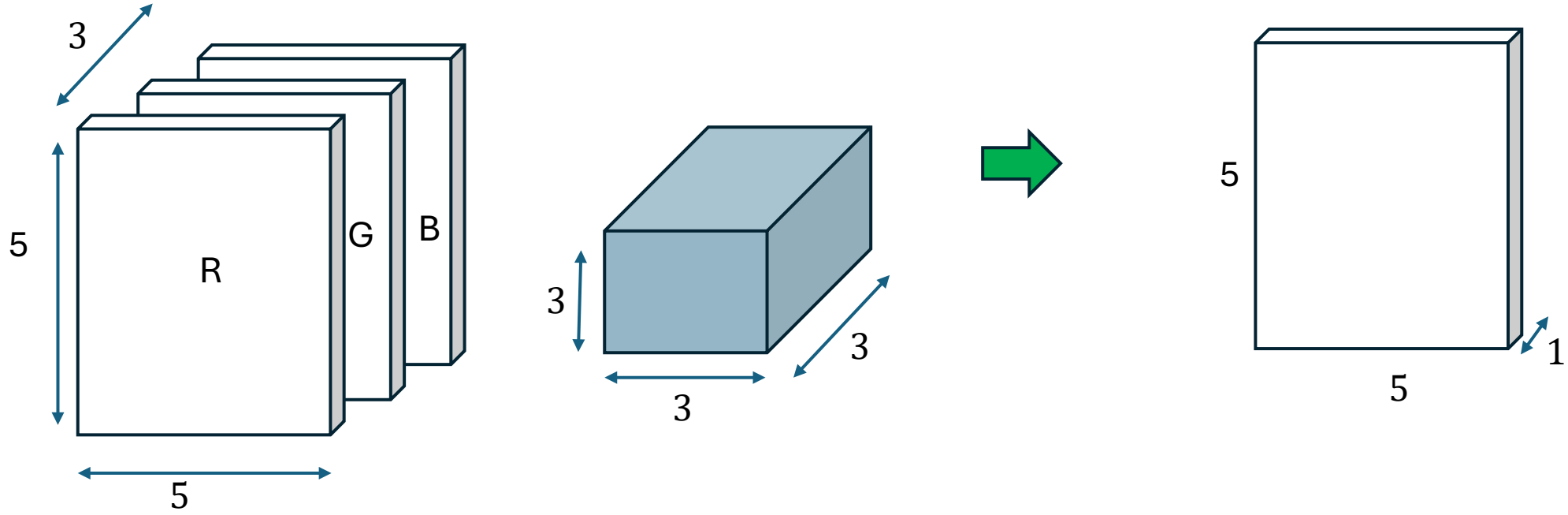
0	0	-1	0	0
2	2	1	0	0
0	0	0	0	0
0	1	2	3	2
2	3	3	3	2

3D convolution

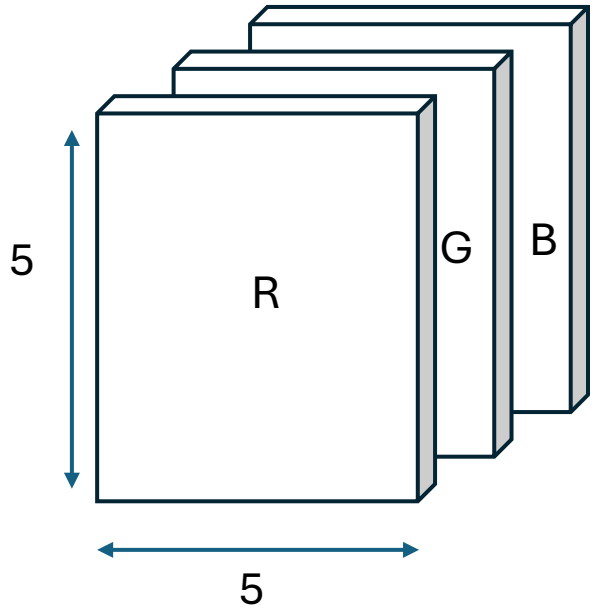
How do we apply convolution filter to RGB image?

In gray scale image, the image was 5*5*1 and the kernel was 3*3*1.

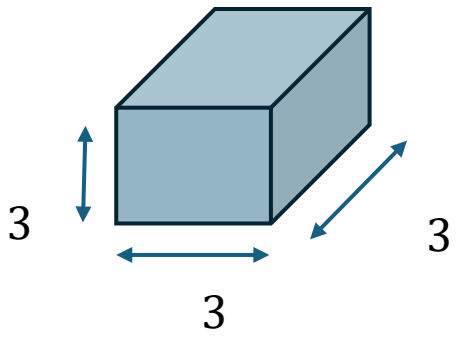
Now if the image if 5*5*3, the kernel must be 3*3*3 as well because this way the kernel can mix up the different RGB channels.



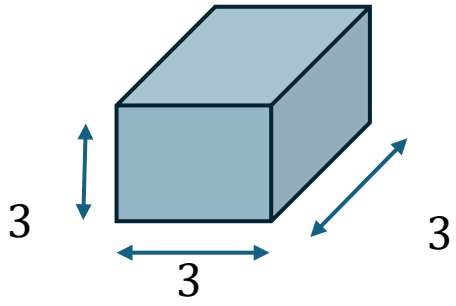
In this example, applying each kernel requires 27 pointwise multiplications ($\rightarrow \sum_{i=1}^{27} multipication$)
Notice that the result is 5*5*1 not 5*5*3 . In other words, a single kernel brings the depth back to 1.
If we want the result to have depth of more that one, we can have more than one kernel.



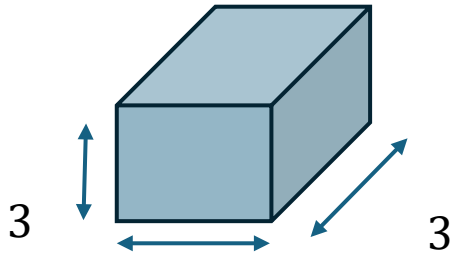
Kernel 1



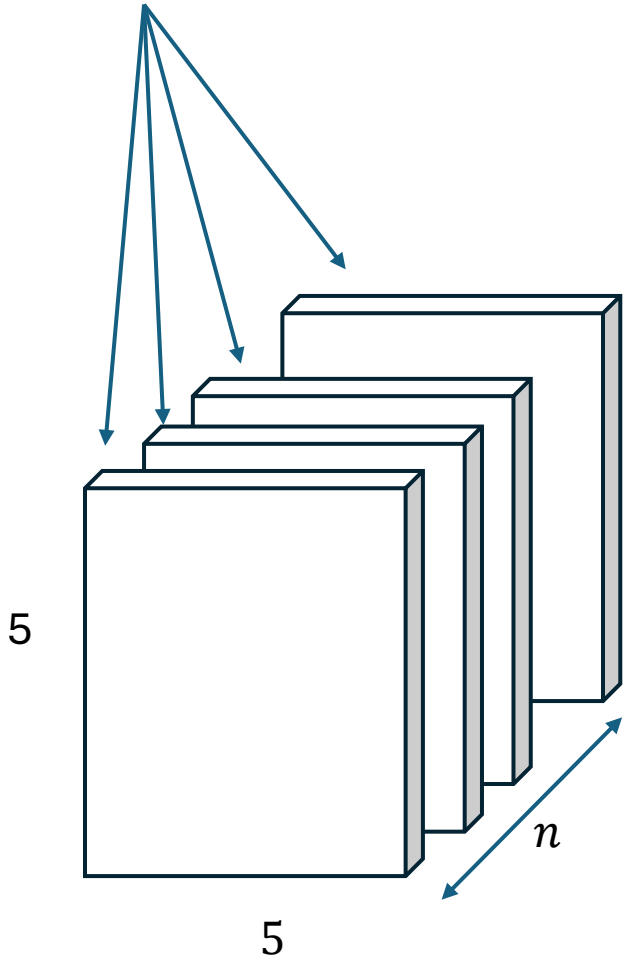
Kernel 2



Kernel n



These are called features or channels
(each feature has a depth of one)



Each of these n kernel can be specific for a task, for example kernel #1 is for horizontal line detection, kernel #2 is for vertical line detection, kernel #3 for 45 degree line detection, etc.

In convolutional neural networks,
We first decide how many kernel we want.
We also decide the size of the kernels (3*3, 5*5, ...)
Then we only initialize these kernels with some random weights and then the kernel weights get learned via gradient descent.

We can also have many layers such as MLP. For each the feature of the first layer become the input to the next layer.

$$\text{Image } (3 * H * W) \xrightarrow{\text{n covolution kernels}} n * H * W \xrightarrow{\text{activation}} n * H * W \xrightarrow{\text{m covolution kernels}} m * H * W \ldots.$$

We can also visualize what kernels are being learns.

Receptive Field

How many pixel from the original image affect a pixel in the feature map.

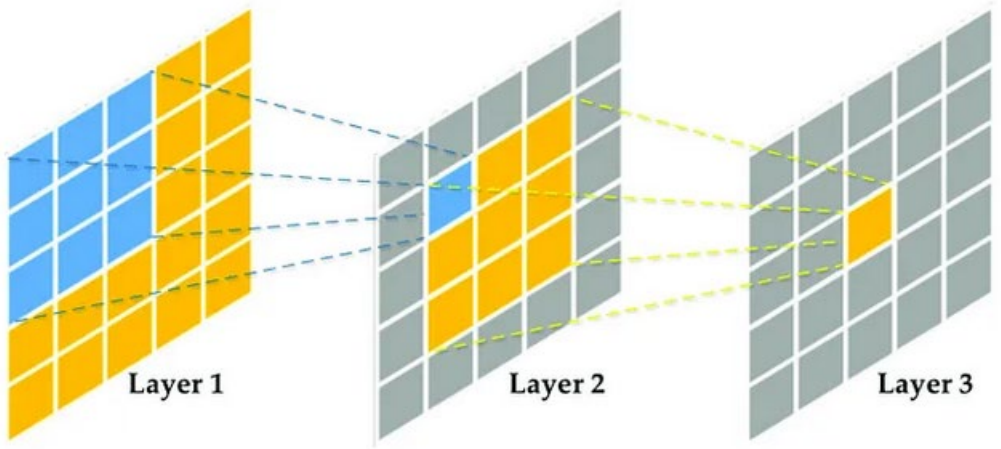
To have a larger receptive field:

- Use a larger kernel (for example 7-by-7)
- Use a Sequence smaller convolutions (for example multiple layers of 3-by-3)

Many architectures only use 3-by-3 convolutions but use multiple layers of them.

In many tasks, especially dense prediction tasks like semantic image segmentation or depth estimation where we make a prediction for every single pixel in the input image,
 it is critical for each output pixel to have a big receptive field,
such that no important information is left out when making the prediction.

Because of small receptive field, CNNs are not fully invariant to scale, rotation and translation.
They are only invariant to small changes.



1-by-1 convolution filter

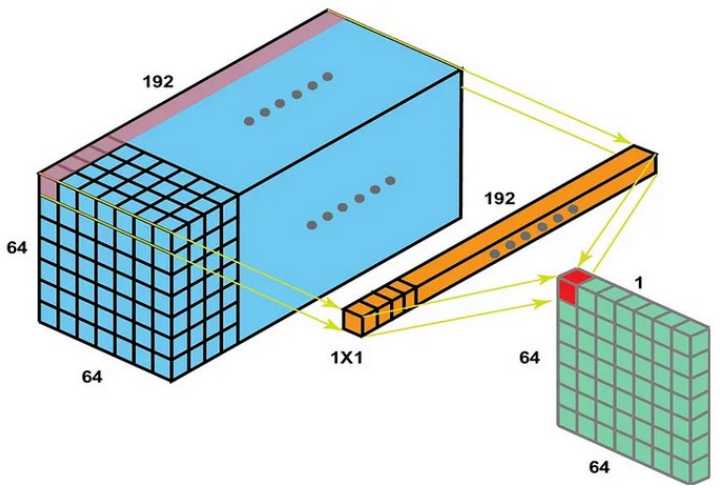
1-by-1 convolution is used in many modern CNN architectures.

Pixel in a channel: refers to 3D dimension pixel. For example, if a layer has 192 channel, each pixel has a depth of 80.

it combines channels together and it acts as a dense layer across channels.

In order words, a 1-by-1 convolution layer performs fully connected functionality on the input/output pixel.

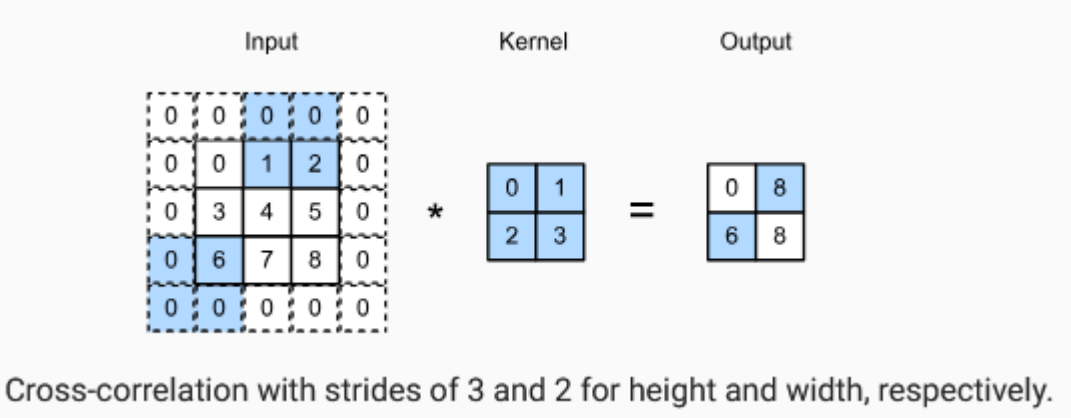
We will see in the future that Resnet uses both 3-by-3 and 1-by-1 convolutions.



Stride

The stride is how many pixel we move the kernel each time.

Stride is used to reduce the size the output after the applying the convolution Kernel.



Cross-correlation with strides of 3 and 2 for height and width, respectively.

$$output\ width = \frac{w - k + 2p}{s} + 1$$

w: $input\ width$
k: $kernel\ size$
p: $padding\ size$
s: $stride\ size$

Example: input image is 32 by 32 and kernel size is 5*5 with no padding and stride of one , what the dimension of output feature?

Answer: $= \frac{32-5+0}{1} + 1 = 28$

If the stride is 2, then $= \frac{32-5+0}{2} + 1 = 14$

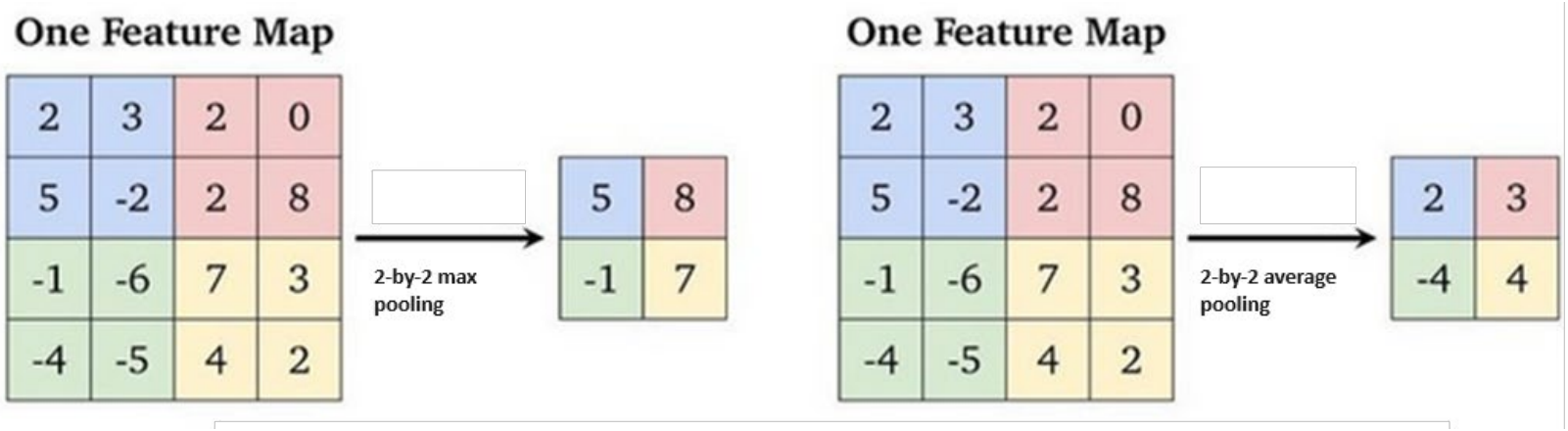
Comparison of the number of weights in MLP versus CNN for image processing

Q: Say we have 28*28 image (Mnist dataset). And we apply ten 5*5 convolution filter. How many weight neural network has?
A: $10 * 5 * 5 + 10 \ = 260 \ (including\ biases)$

Q: if we apply fully connected layer with output of 10 neurons, how many weights we have?
A: $28 * 28 * 10 + 10 = 7850$

Max Pooling/ Average pooling Layer

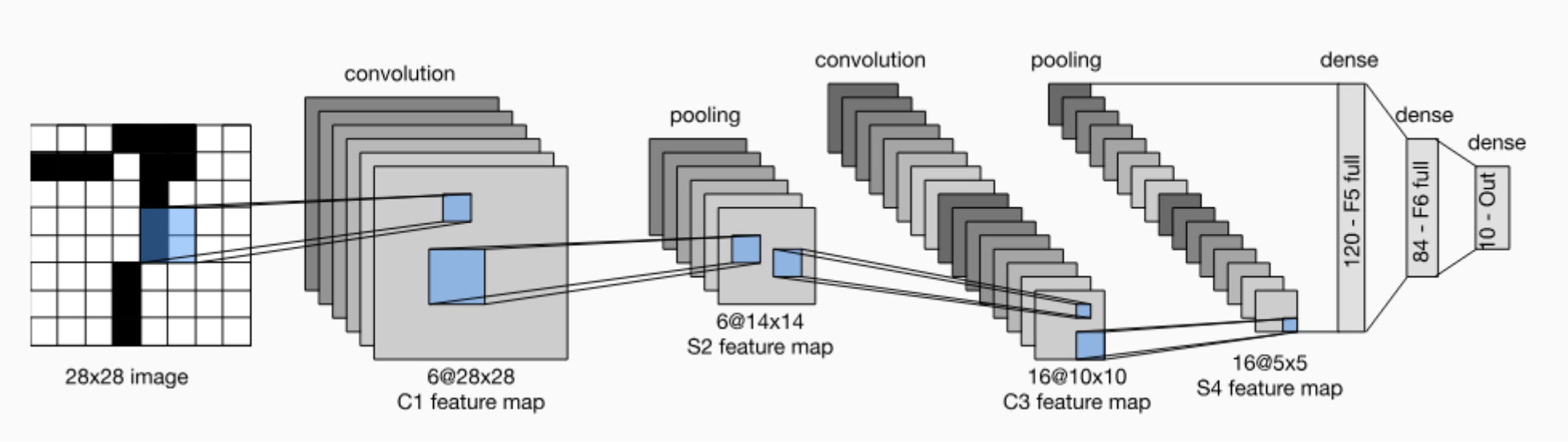
This is another way to reduce the size of the output after applying convolution kernel.



As you recall before, stride is another way to reduce the dimension of output.
The downside of the pooling layer is that it does not have learnable parameters.

LeNet

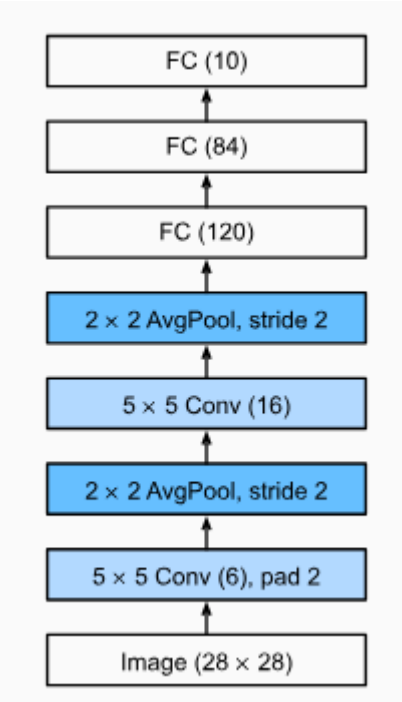
It is the first CNN architecture that was published in 1998 for classification. At the time the Relu was not invented and the architecture used sigmoid activation.



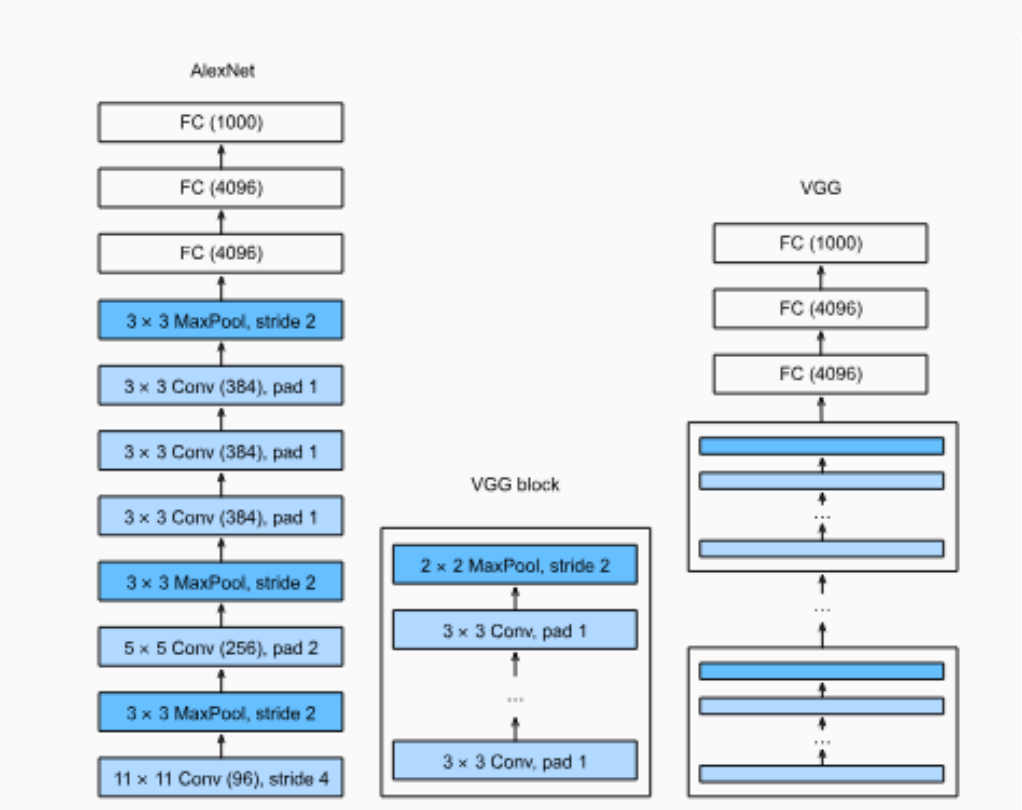

```
self.features = torch.nn.Sequential(

    torch.nn.Conv2d(in_channels, 6, kernel_size=5),
    torch.nn.Tanh(),
    torch.nn.MaxPool2d(kernel_size=2),
    torch.nn.Conv2d(6, 16, kernel_size=5),
    torch.nn.Tanh(),
    torch.nn.MaxPool2d(kernel_size=2)
)

self.classifier = torch.nn.Sequential(
    torch.nn.Linear(16*5*5, 120),
    torch.nn.Tanh(),
    torch.nn.Linear(120, 84),
    torch.nn.Tanh(),
    torch.nn.Linear(84, num_classes),
)
```

Note: as mentioned earlier, the pooling layer does not have learnable parameters. We can always replace the pooling layer with conv layer with desired stride.

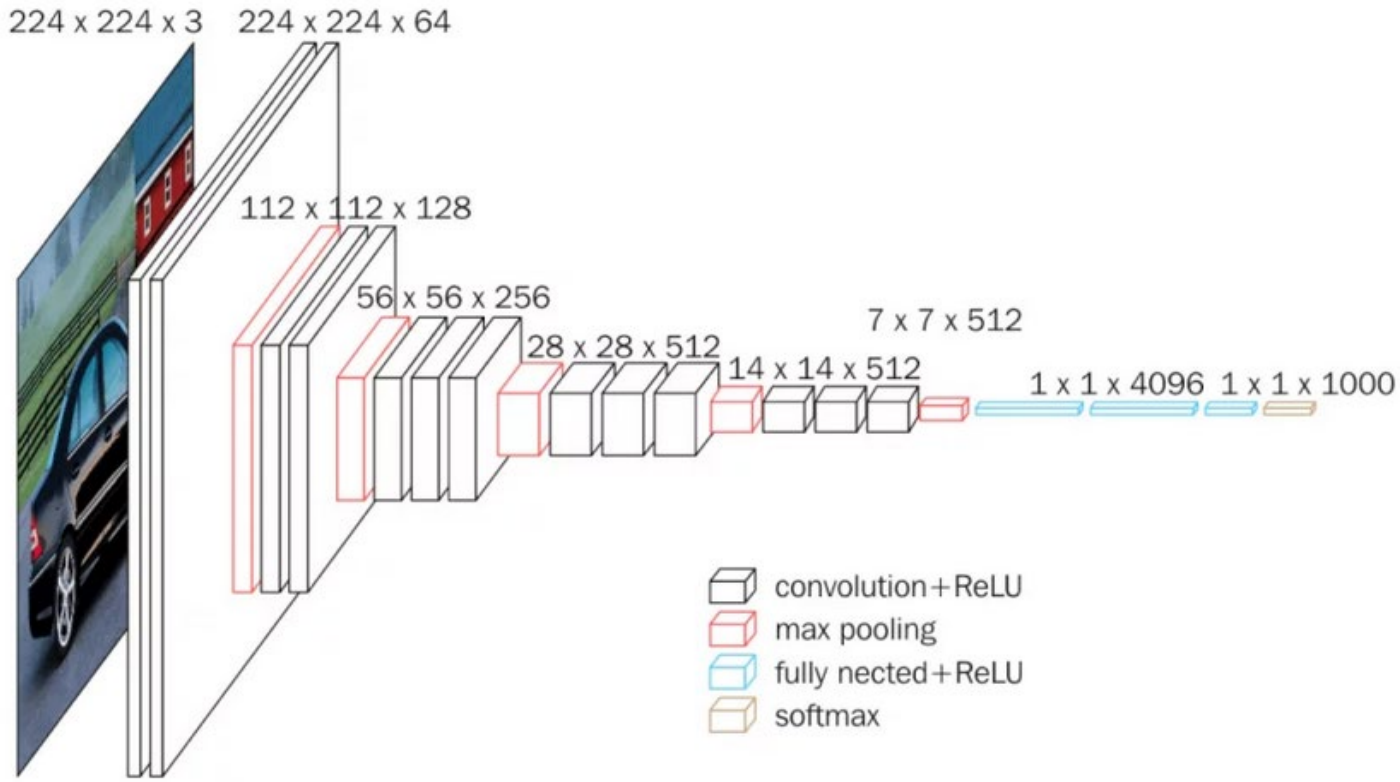
Can you calculate how many weights LeNet has?



AlexNet and VGG



VGG-16



[1] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, 2012
[2] K. Simonyan, A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, 2014

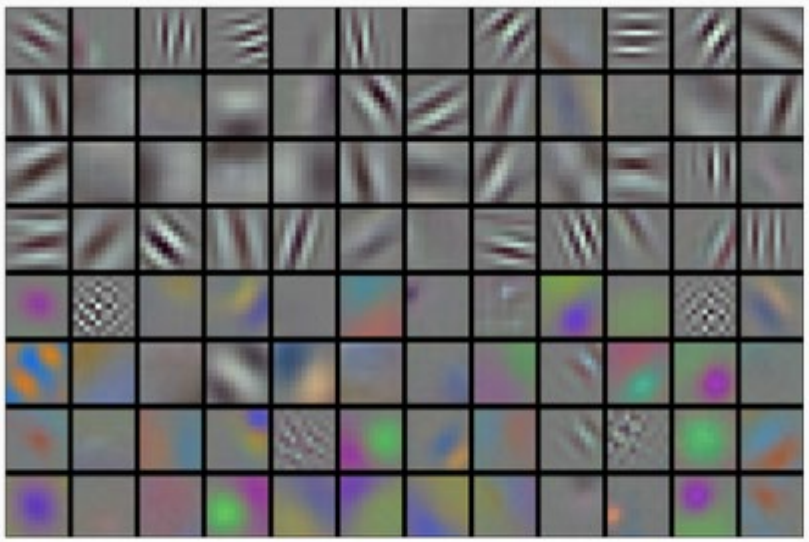


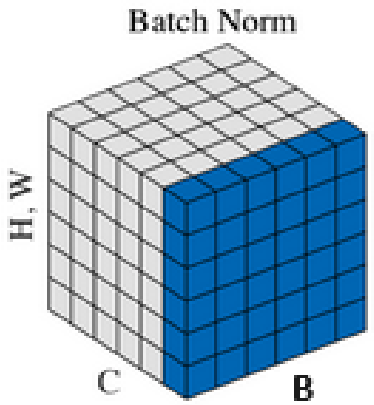
Fig. 8.1.1 Image filters learned by the first layer of AlexNet. Reproduction courtesy of Krizhevsky *et al.* (2012).

In the lowest layers of the network, the model learns feature extractors that resembled some traditional filters.
Higher layers in the network might build upon these representations to represent larger structures, like eyes, noses, blades of grass, and so on.

Dropout and Batchnorm in CNNs

In 1D version of batchnorm in MLP, the dimension of each layer was $B * m$ (B: batch size, m: number of neurons in the layer). And we calculated β, γ for each m (we go over the batch for each m to calculate mean and std.)

In 2D batchnorm in CNN, each layer has dimension of B$* C * H * W$. So we have C independent β, γ. In other words we average the image over the batch for each channel independently.



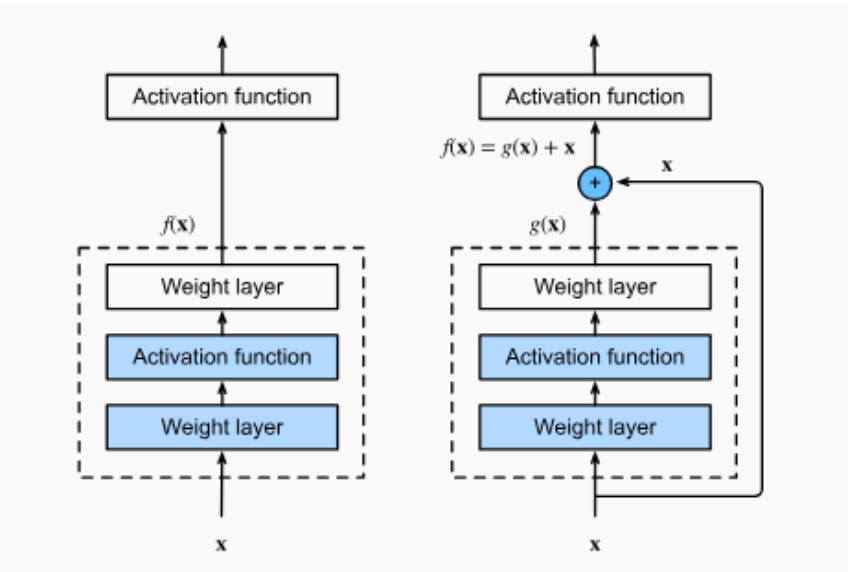
For dropout in CNNs, we drop a channel randomly with some probability.

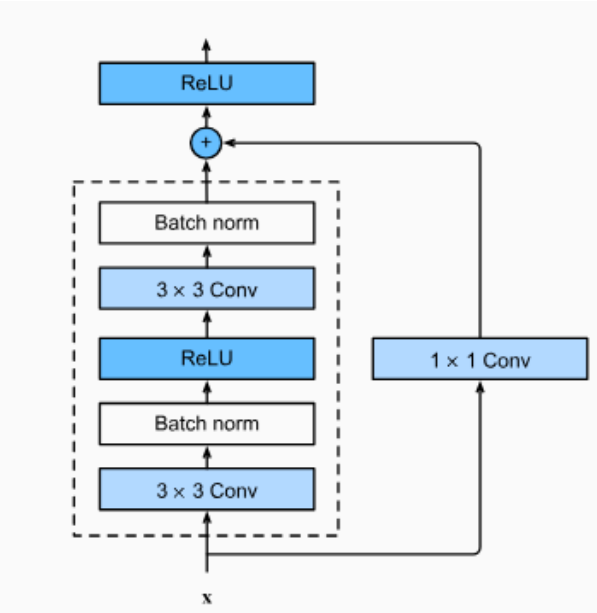
Resnet

The key feature in resent is "skip connections" that allow gradients to flow through the network more effectively. This make it possible to train much deeper networks than before (ResNet-18, ResNet-50, ResNet-152)

Residual blocks are the building blocks of ResNet.

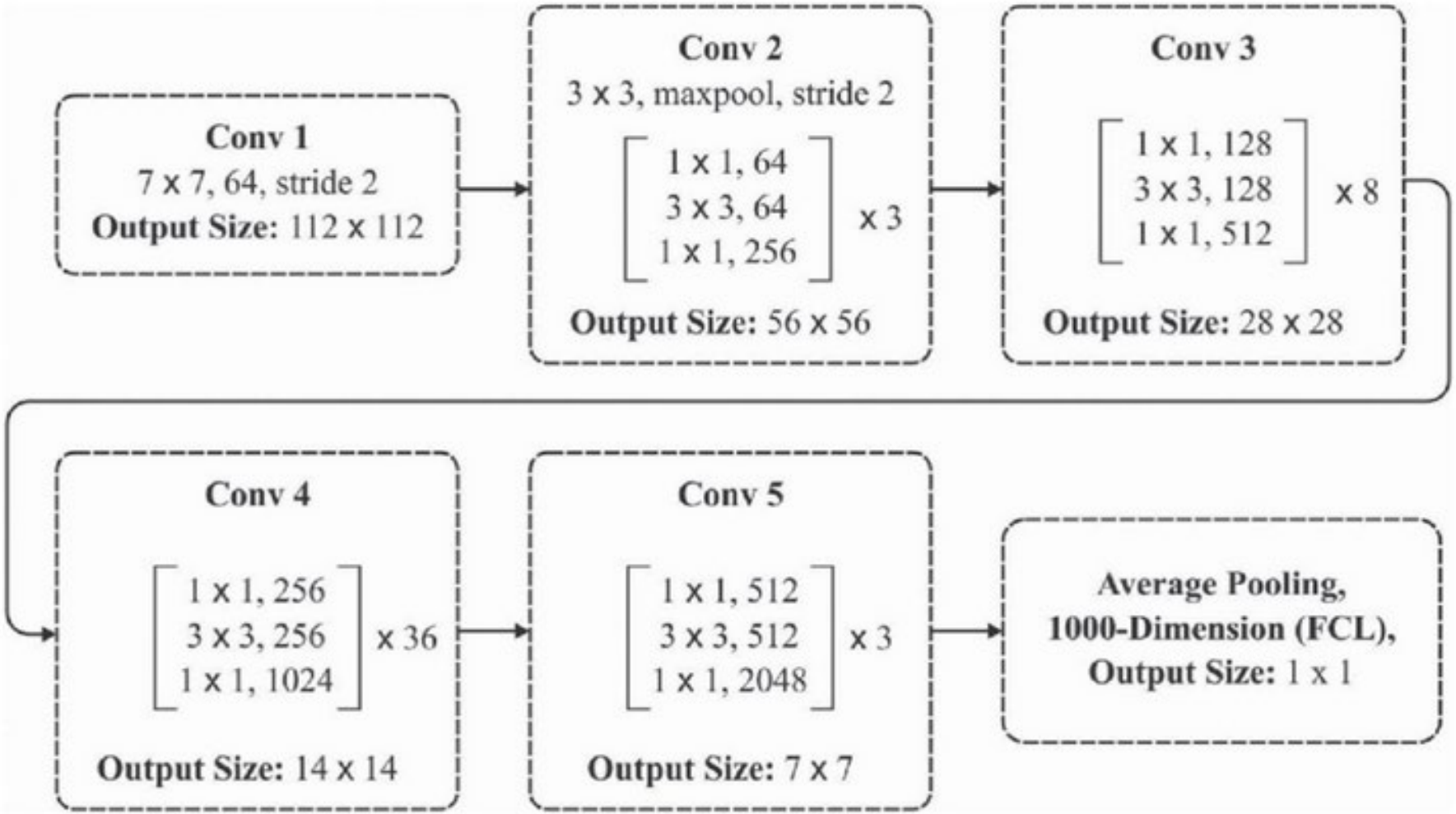
However, in order to be able to apply the addition,
x and $g(x)$ need to have the same shape. Resnet uses
1D convolution to make the number of channels the same.





K. He, et. al, Deep Residual Learning for Image Recognition, 2015

Resnet 152


Conv 1
7 x 7, 64, stride 2
Output Size: 112 x 112
Conv 2
3 x 3, maxpool, stride 2
1 x 1, 64
3 x 3, 64
1 x 1, 256
x 3
Output Size: 56 x 56
Conv 3
1 x 1, 128
3 x 3, 128
1 x 1, 512
x 8
Output Size: 28 x 28
Conv 4
1 x 1, 256
3 x 3, 256
1 x 1, 1024
x 36
Output Size: 14 x 14
Conv 5
1 x 1, 512
3 x 3, 512
1 x 1, 2048
x 3
Output Size: 7 x 7
Average Pooling,
1000-Dimension (FCL),
Output Size: 1 x 1