Course Notes: Deep Learning for Visual Computing

Peter Wonka

August 30, 2021

Contents

1 Convolutional Neural Network Building Blocks 4

1.1 Overview 5

1.2 Example Network 7

1.3 Convolution 9

1.4 Convolution to Smooth a Function 12

1.5 Discrete Convolution in 1D 13

1.6 2D Convolution 14

1.7 Discrete 2D Convolution 15

1.8 Discrete Convolution on Images: Graphical Examples 16

1.9 Discrete Convolution: Numerical Example 18

1.10 Stride 20

1.11 Padding 22

1.12 Sparse Filters - Dilation 25

1.13 2D RGB Image (Rank 3 Tensor) Convolution Example 26

1.14 Groups 28

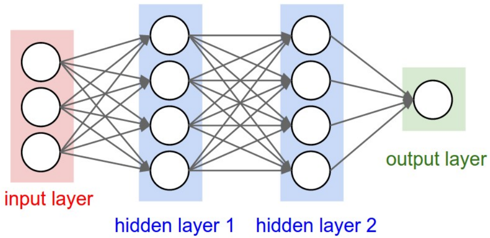
1.15 Convolution with Multiple Filters 29
1.16 Convolution for a Batch of Tensors 32
1.17 Convolution for Tensors of Different Rank 33
1.18 Note on 1×1 Convolution 34
1.19 Summary Basic Conv Layer 35
1.20 Example Settings 37
1.21 Non-symmetric Treatment of Width and Height 38
1.22 Visualizing Convolutions 39
1.23 Receptive Field 40
1.24 Max Pooling 41
1.25 Basic Max Pooling Parameters 44
1.26 Classical Convolutional Network Architecture 46
1.27 Fully Connected vs. Convolutional Layers 47
1.28 Filter Visualization Example 49

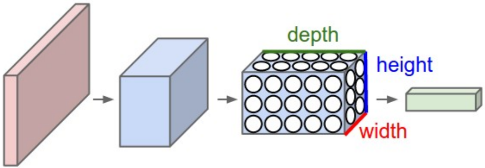
1 Convolutional Neural Network Building Blocks

1.1 Overview

- **Convolutional Neural Networks** or **CNNs** or **ConvNets** use a combination of different building blocks
  - convolutional layers
  - fully connected layers
  - non-linear activations
  - pooling layers
  - normalization layers
  - ...
  - for a good overview see the PyTorch documentation
- The name **CNN** because convolutional layers are dominant
- CNNs typically assume images (videos) as input
  - Fully connected networks conceptually arrange neurons as vectors

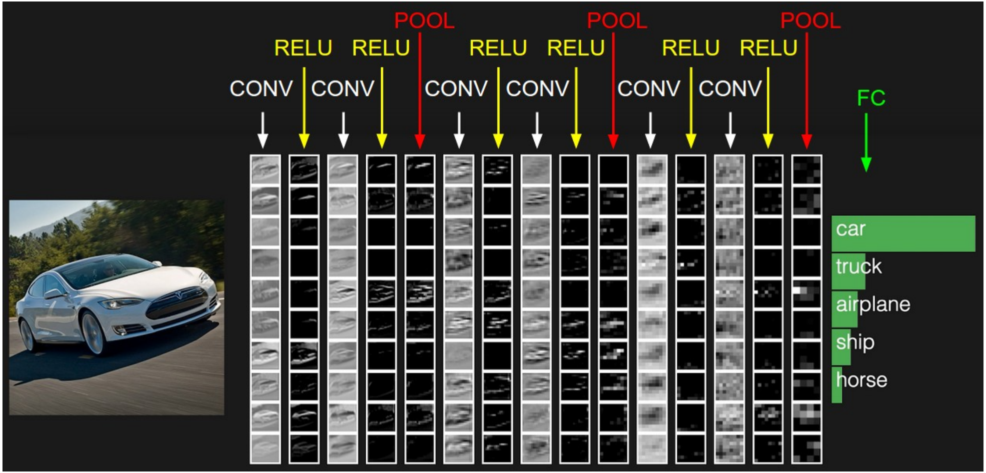
- CNNs arrange neurons as rank 3 tensors (3D arrays) $\in \mathbb{R}^{width \times height \times depth}$





1.2 Example Network

- Live Web Version
- 17 layers total, softmax layer at the end


POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

1.3 Convolution

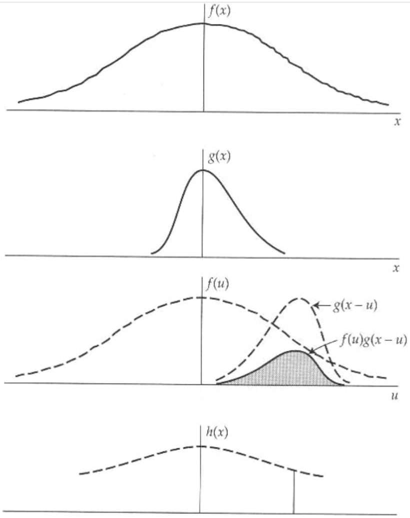
- Convolution takes two functions $f(x)$ and $g(x)$ as input and produces a function $h(x)$ as output
  - $$h(x) = f(x) * g(x) = \int_{-\inf}^{+inf} f(u) g(x-u) \, \mathrm{d}u \tag{1.1}$$
- Warning:
  - $g(x)$ is flipped. Some people use the term **correlation** to define a version of convolution where the filter is not getting flipped.
  - In vision (and other applications) people often use the term convolution, but implement correlation where filters are not flipped.

- Note:

$$f(x) * g(x) = g(x) * f(x) \tag{1.2}$$

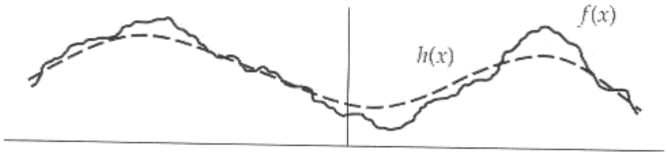
$$f * (g * h) = (f * g) * h \tag{1.3}$$

$$f * (g + h) = f * g + f * h \tag{1.4}$$


f(x)
x
g(x)
x
f(u)
g(x − u)
f(u)g(x − u)
u
h(x)

1.4 Convolution to Smooth a Function

- Convolution is often used to smooth a function
- $h(x) = f(x) * g(x)$
  - $h(x)$ - smooth output function
  - $f(x)$ - input function that should be smoothed
  - $g(x)$ - filter that defines the smoothing operation



1.5 Discrete Convolution in 1D

- For functions of a discrete variable x, i.e. arrays of numbers, the definition is:

$$h[x] = f[x] * g[x] = \sum_{k=-\inf}^{+inf} f(k)g(x-k) \tag{1.5}$$

1.6 2D Convolution

- For functions of two variables x and y (for example continuous images), the definition is:

$$h(x,y) = f(x,y) * g(x,y) = \tag{1.6}$$

$$\int_{u_1=-\inf}^{+inf} \int_{u_2=-\inf}^{+inf} f(u_1,u_2) g(x-u_1, y-u_2) \, \mathrm{d}u_1 \mathrm{d}u_2 \tag{1.7}$$

1.7 Discrete 2D Convolution

- For discrete functions of two variables x and y (for example discrete images), the definition is:

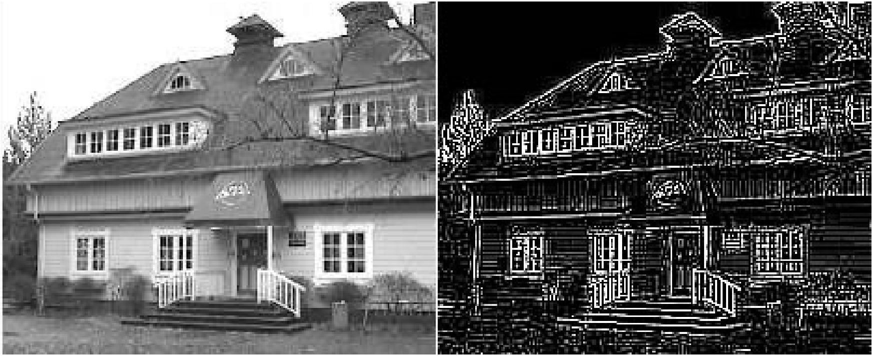
$$h[x,y] = f[x,y] * g[x,y] = \sum_{k_1=-\inf}^{+inf} \sum_{k_2=-\inf}^{+inf} f[k_1,k_2]g[x-k_1,y-k_2] \tag{1.8}$$

1.8 Discrete Convolution on Images: Graphical Examples

- Convolution Examples: blurring and edge detection
- Filter:

-1	-1	-1
-1	8	-1
-1	-1	-1

- Input and Result:



- Typically one function is the image and the other function is a filter

1.9 Discrete Convolution: Numerical Example

- I input image 7×7
- K convolution filter 3×3
- Output is a 5×5 image
- For the green output the red filter is aligned with the blue submatrix in image I
  - $13 = 1 \times 2 + 1 \times 3 + 3 \times 1 + 5 \times 1$
- x are values that are not shown

$$\underset{I}{\begin{pmatrix} 0 & 2 & 1 & 3 & 5 & 2 & 9 \\ 3 & 1 & 4 & 1 & 1 & 2 & 5 \\ 2 & 2 & 1 & 3 & 5 & 2 & 9 \\ 0 & 2 & 1 & 4 & 5 & 2 & 8 \\ 1 & 2 & 4 & 7 & 1 & 2 & 6 \\ 1 & 2 & 1 & 3 & 4 & 4 & 7 \\ 2 & 2 & 4 & 5 & 3 & 2 & 8 \end{pmatrix}} * \underset{K}{\begin{pmatrix} 0 & 0 & 2 \\ 3 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}} = \underset{I * K}{\begin{pmatrix} x & x & x & x & x \\ x & x & 13 & 20 & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{pmatrix}} \tag{1.9}$$

1.10 Stride

- We can subsample the output. We use the term **stride** to denote the subsampling amount.
  - $stride = 1$ is default, no subsampling, move filter one pixel each time
  - $stride = 2$, skip every second pixel, move filter 2 pixels each time
- We show an example with stride 2
- The orange values in I show possible placements of the top left corner of the filter

$$
\underbrace{\begin{pmatrix}
0 & 2 & 1 & 3 & 5 & 2 & 9 \\
3 & 1 & 4 & 1 & 1 & 2 & 5 \\
2 & 2 & 1 & 3 & 5 & 2 & 9 \\
0 & 2 & 1 & 4 & 5 & 2 & 8 \\
1 & 2 & 4 & 7 & 1 & 2 & 6 \\
1 & 2 & 1 & 3 & 4 & 4 & 7 \\
2 & 2 & 4 & 5 & 3 & 2 & 8
\end{pmatrix}}_{I}
*
\underbrace{\begin{pmatrix}
0 & 0 & 2 \\
3 & 1 & 0 \\
0 & 0 & 1
\end{pmatrix}}_{K}
=
\underbrace{\begin{pmatrix}
x & x & x \\
x & x & 41 \\
x & x & x
\end{pmatrix}}_{I * K}
\tag{1.10}
$$

1.11 Padding

- Boundary problem: in the examples before, the output image is (a bit) smaller than the input image depending on the filter size
- **Padding** can be used to keep the spatial resolution of the output image (tensor) the same as the spatial resolution of the input (tensor)
  - What value should be used for padding?
    - Typically 0, but other options exist
    - Copy the value from the nearest pixel
    - Copy the value from the other side (left - right, top - bottom)
  - How large should the padding region be?
    - Typically proportional to the filter size
    - a 3×3 filter uses a padding of 1
    - a 5×5 filter uses a padding of 2

    - a 7×7 filter uses a padding of 3
- Terms:
  - **Same padding** means padding parameter is set such that input and output tensor of convolution have the same size
  - **Valid padding** means no padding
- In this example padding of 1 is shown for a 3×3 filter
  - The extra 0s at the boundary are shown in orange
  - The extra values in the output are shown as orange x
  - The original image and the output have the same resolution

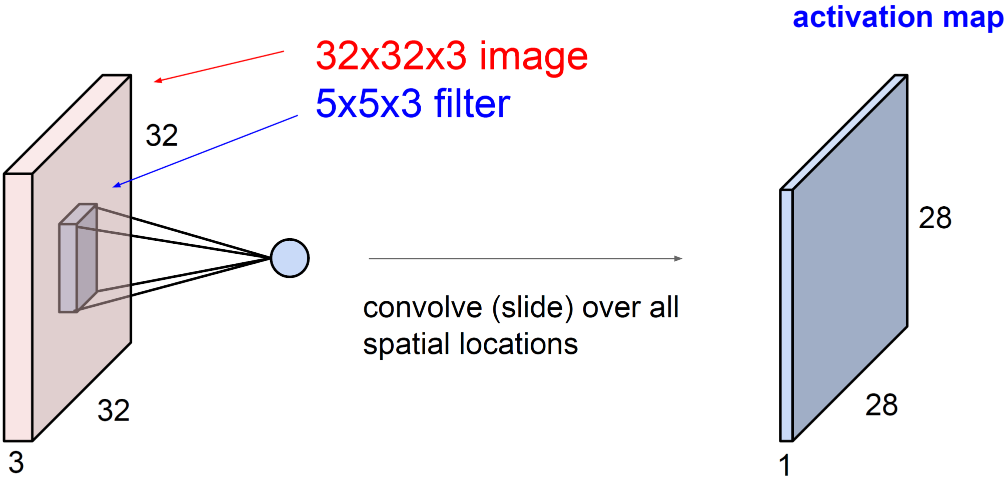
$$
\underbrace{\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 2 & 1 & 3 & 5 & 2 & 9 & 0 \\
0 & 3 & 1 & 4 & 1 & 1 & 2 & 5 & 0 \\
0 & 2 & 2 & 1 & 3 & 5 & 2 & 9 & 0 \\
0 & 0 & 2 & 1 & 4 & 5 & 2 & 8 & 0 \\
0 & 1 & 2 & 4 & 7 & 1 & 2 & 6 & 0 \\
0 & 1 & 2 & 1 & 3 & 4 & 4 & 7 & 0 \\
0 & 2 & 2 & 4 & 5 & 3 & 2 & 8 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}}_{I}
*
\underbrace{\begin{pmatrix}
0 & 0 & 2 \\
3 & 1 & 0 \\
0 & 0 & 1
\end{pmatrix}}_{K}
=
\underbrace{\begin{pmatrix}
x & x & x & x & x & x & x \\
x & x & x & x & x & x & x \\
x & x & x & 13 & 20 & x & x \\
x & x & x & x & x & x & x \\
x & x & x & x & x & x & x \\
x & x & x & x & x & x & x \\
x & x & x & x & x & x & x
\end{pmatrix}}_{I * K}
\tag{1.11}
$$

1.12 Sparse Filters - Dilation

- Filters can contain a structured sparsity pattern
  - can be used for a multi-resolution effect
- The values can only be on orange locations marked x for $dilation = 2$

$$\begin{pmatrix} x & 0 & x & 0 & x \\ 0 & 0 & 0 & 0 & 0 \\ x & 0 & x & 0 & x \\ 0 & 0 & 0 & 0 & 0 \\ x & 0 & x & 0 & x \end{pmatrix} \tag{1.12}$$

1.13 2D RGB Image (Rank 3 Tensor) Convolution Example

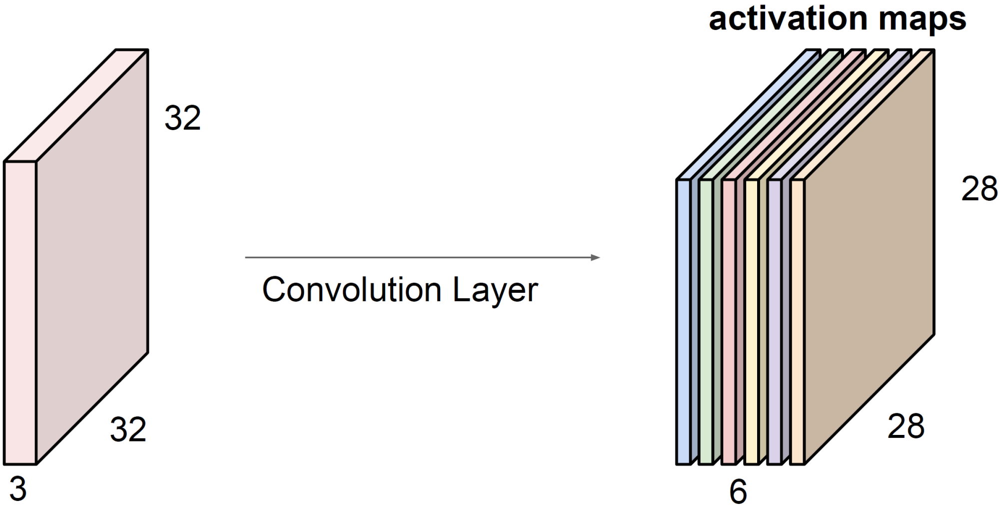


- Filter has $5 \times 5 \times 3 + 1$ parameters (1 for the bias term)
  - We need 76 values to obtain 28×28 values in the next layer
  - A fully connected / linear layer would need $(32 \times 32 \times 3 + 1) \times 28 \times 28$
  - Filter covers the full depth (all channels) of the input tensor
- One filter defines one output **Activation Map** = one output channel
- Alternate design:
  - One can break down the computation as a separate convolution per channel, where each channel has a channel specific convolution filter
  - After computing a channel specific intermediate activation map, all intermediate activation maps are summed up to give the final activation map
  - Instead of summing up, once could simply use all intermediate activation maps (problem: too many output activation maps)

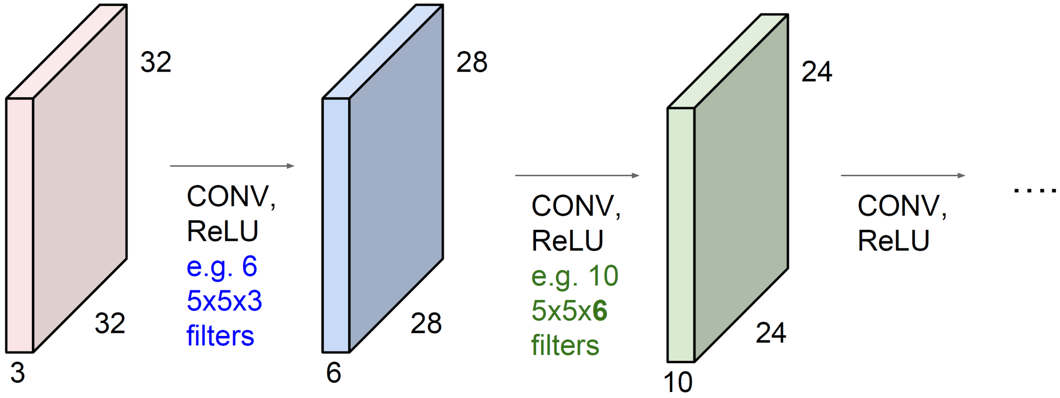
1.14 Groups

- A filter can be applied only to a subset of the input channels
  - e.g. some filters only apply to the first half of the input channels, some filters only to the second half

1.15 Convolution with Multiple Filters



- k filters (e.g. $k = 6$) provide k activation maps
- A convolutional layer typically consists of multiple filters
- The number of channels in the output tensor is equal to the number of convolution filters used
- Example: with filters of spatial extent 5×5, we need $6 \times (5 \times 5 \times 3 + 1)$ values
- After a conv layer, We typically use a non-linear activation function (e.g. ReLU) on each output value
- We also want to interleave convolutional layers with ReLU layers and stack multiple layers:


32
32
3
CONV,
ReLU
e.g. 6
5x5x3
filters
28
28
6
CONV,
ReLU
e.g. 10
5x5x6
filters
24
24
10
CONV,
ReLU
....

1.16 Convolution for a Batch of Tensors

```
torch.nn.Conv2d(in_channels: int, out_channels: int, kernel_size: Union[
    int, Tuple[int, int]], stride: Union[int, Tuple[int, int]] = 1,
    padding: Union[int, Tuple[int, int]] = 0, dilation: Union[int, Tuple[
    int, int]] = 1, groups: int = 1, bias: bool = True, padding_mode: str
    = 'zeros')
```

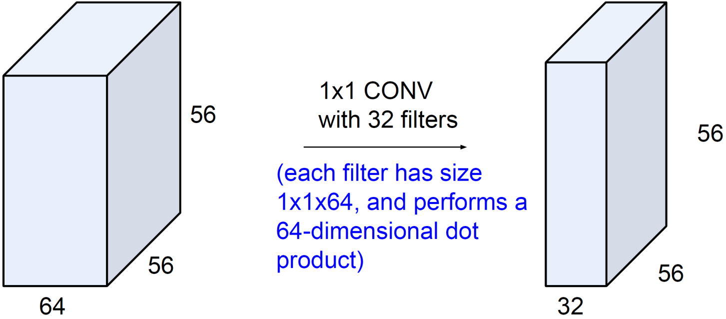
- Convolution is computed for a batch of inputs, e.g. images
- The convolution layer takes a rank-4 tensor $(N, W_{in}, H_{in}, C_{in})$ as input and computes a rank-4 tensor $(N, W_{out}, H_{out}, C_{out})$ as output
- PyTorch uses the channel first convention

1.17 Convolution for Tensors of Different Rank

- 1D convolution nn.Conv1D
- 2D convolution nn.Conv2D
- 3D convolution nn.Conv3D, filter size $F_1 \times F_2 \times F_3 \times C_{in}$

1.18 Note on 1×1 Convolution

- Similar to traditional dimension reduction: compare to SVD, PCA
- Identical to adding a linear layer on each pixel separately



1.19 Summary Basic Conv Layer

- Input: tensor of size $W_{in} \times H_{in} \times C_{in}$
- Hyperparameters
  - Number of filters K
  - Size of the filter F
  - Stride S
  - Amount of zero padding P
- Output: tensor of size $W_{out} \times H_{out} \times C_{out}$
  - $W_{out} = (W_{in} - F + 2P)/S + 1$
  - $H_{out} = (H_{in} - F + 2P)/S + 1$
  - $C_{out} = K$
- Number of parameters: $F \times F \times C_{in} + 1$ per filter (1 for the bias)

- Total number: $(F \times F \times C_{in} + 1) \times K$

1.20 Example Settings

- Filter size F is small, e.g. 3×3, 5×5, 7×7
  - Using subsequent convolutions with small F is more common than one convolution with large F
- Same padding setting $P = (F - 1)/2$
- Use powers of 2 for the number of channels C_{in}, C_{out}, e.g. 32, 64, ..., 1024
- $F = 3, P = 1, S = 1$: standard 3×3 convolution
- $K = 3, P = 1, S = 2$: downsampling combined with 3×3 convolution

1.21 Non-symmetric Treatment of Width and Height

- Parameters like filter size, stride, padding, dilation are typically the same for the width and height dimension
- Values can also be set separately
- For example, filter size can be 1×7 or 7×1 to look for horizontal or vertical features

1.22 Visualizing Convolutions

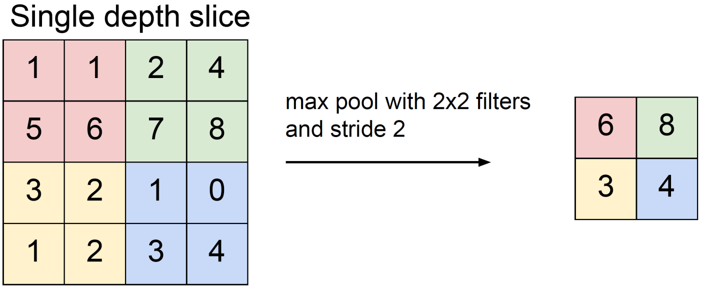
- CS231n Convolution Demo
- Filter Pattern Visualization
  - Corresponding pdf
- Convolution Arithmetic

1.23 Receptive Field

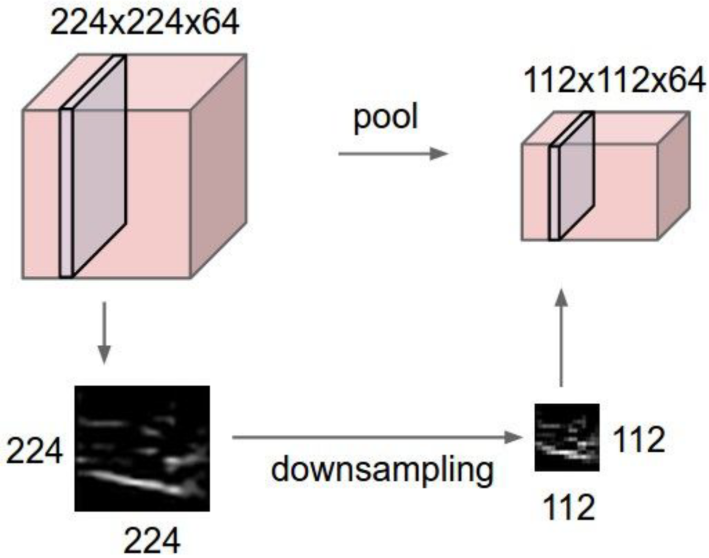
- The **receptive field** of a neuron (e.g. value in a tensor) is the area of the original input that can influence the value of the neuron
- Alternatively, receptive field can also be determined for an arbitrary later, e.g. the previous layer

1.24 Max Pooling

- We can use other aggregation functions instead of convolution, e.g. max, min, or average



- A pooling operation typically operates on one input channel to produce one output channel


224x224x64
pool
112x112x64
224
224
downsampling
112
112

- Different pooling operation exists, e.g. max pooling, max unpooling, fractional max pooling, average pooling, ...
- Max pooling also has parameters stride, padding, dilation, ...

1.25 Basic Max Pooling Parameters

- Accepts a tensor of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  - filter size F
  - stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- introduces zero parameters since it computes a fixed function of the input
- not common to pad the input using zero-padding
- Two common settings:

- $F = 3, S = 2$ (called overlapping-pooling)
- $F = 2, S = 2$

1.26 Classical Convolutional Network Architecture

- Repeat several blocks of [Conv, ReLU, Pool]
- Flatten: convert tensor to vector
- Repeast several blocks of [FC, ReLU]
- FC

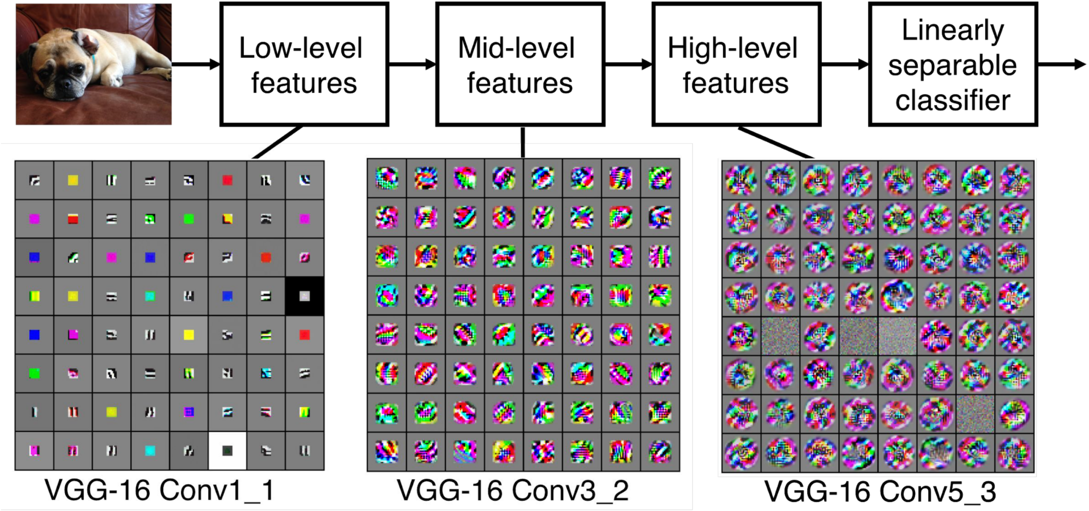
1.27 Fully Connected vs. Convolutional Layers

- Any CONV layer can be converted to an FC layer
  - e.g. as a large weight matrix with mainly 0s
  - needs weight sharing
- FC layers can be converted to CONV layer
  - set the filter size equal to the size of the input volume
  - e.g. input volume of size $7 \times 7 \times 512$, 4096 neurons as output
    - use a CONV layer with filter size = 7, padding = 0, stride = 1, number of filters = 4096
- Case Study
  - Network accepts $224 \times 224 \times 3$ images as input
  - Sequence of CONV and pool layers to yield a $7 \times 7 \times 512$ intermediate output

    - One FC layer to give a $1 \times 1 \times 4096$ output
    - One FC layer to give a $1 \times 1 \times 4096$ output
    - One FC layer to give final $1 \times 1 \times 1000$ output (1000 classes in imagenet)
- Each of the 3 FC layers can be converted to CONV layers
    - The network can now be applied to larger inputs
    - E.g. an image of size $384 \times 384 \times 3$ as input
        - intermediate volume of $12 \times 12 \times 512$
        - final volume of $6 \times 6 \times 1000$
        - equivalent to applying the original network $6 \times 6 = 36$ times to $224 \times 224 \times 3$ crops of the input with stride 32
    - More efficient

1.28 Filter Visualization Example

Cifar-10 Visualization


Low-level features
Mid-level features
High-level features
Linearly separable classifier
VGG-16 Conv1_1
VGG-16 Conv3_2
VGG-16 Conv5_3