Faculty of Sciences of Tetouan

Introduction to Deep Learning

6- Architectures (Part 1): Convolutional Neural Networks (CNNs)

Prof. Monir EL ANNAS

Networks for images

Problems with fully-connected networks:

1. Size
    - High-resolution RGB images yield very high-dimensional inputs
    - 224×224 RGB image $= 150,528$ dimensions
2. Nearby pixels statistically related
    - But could permute pixels and relearn and get same results with FC
3. Should be stable under transformations
    - Don't want to re-learn appearance at different parts of image

Convolutional networks:

- Parameters only look at local image patches
- Share parameters across image

Convolutional networks

1D convolution operation

- Input vector $\mathbf{x}$:

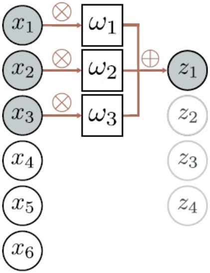
$$\mathbf{x} = [x_1, x_2, \ldots, x_I]$$

- Output is weighted sum of neighbors:

$$z_i = \omega_1 x_{i-1} + \omega_2 x_i + \omega_3 x_{i+1}$$

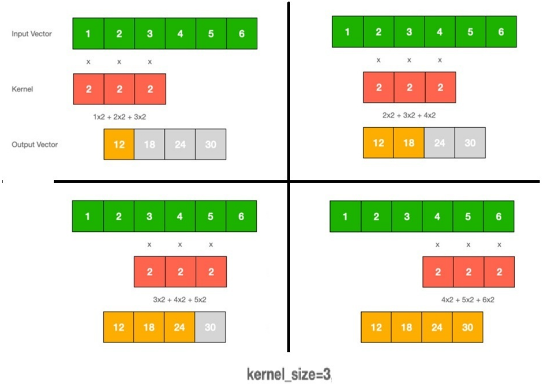
- Convolutional kernel or filter:

$$\boldsymbol{\omega} = [\omega_1, \omega_2, \omega_3]^T$$



Convolutional networks

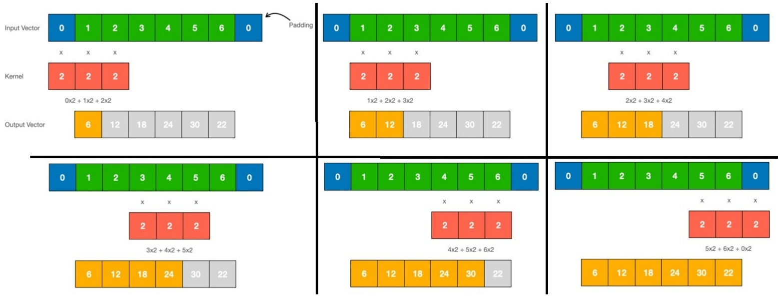
1D Convolution: Operation



- **Kernel size** = weight a different number of inputs for each output
    - Combine information from a larger area.
    - A Kernel size 3 uses 3 parameters

Convolutional networks

1D Convolution: Padding

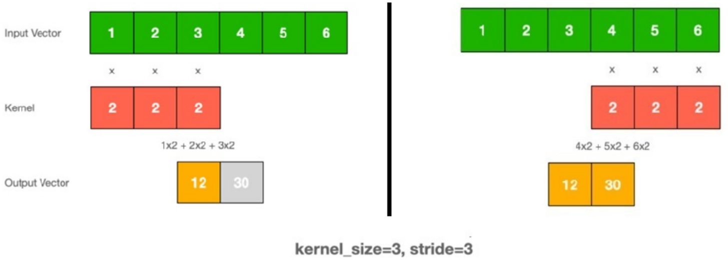


kernel_size=3, padding=1

- **Padding**= pad the edges of the inputs with new values and proceed as usual.
  - Zero padding assumes the input is zero outside its valid range.
  - A padding of 1 means adding 1 extra pixel around the border of the input.

Convolutional networks

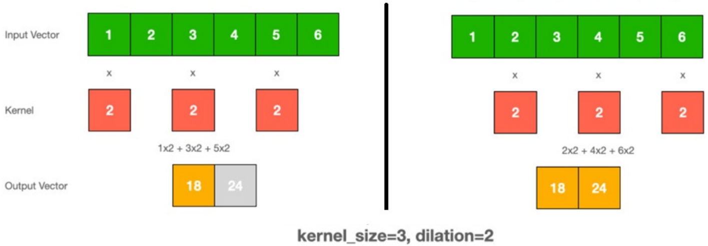
1D Convolution: Stride



- **Stride** = shift by k positions for each output
  - Decreases size of output relative to input

Convolutional networks

1D Convolution: Dilation



- **Dilation** = "inflate" the kernel by inserting spaces between the kernel elements.
  - Dilation = 2 means there is one gap (zero-weighted space) between each kernel element when sliding over the input.

Convolutional Networks

1D Convolution: Output Dimensions

For a 1D convolution, the output dimensions (length) are calculated as follows:

- Output Length:

$$L_{\text{out}} = \left\lfloor \frac{L_{\text{in}} + 2P_l - D(K_l - 1) - 1}{S_l} + 1 \right\rfloor$$

Where:
- L_{in} is the input length.
- K_l is the length of the kernel (filter).
- P_l is the padding applied to the length.
- S_l is the stride (step size) in the length direction.
- D is the dilation rate. The dilation rate controls the spacing between kernel elements.

Convolutional networks

1D convolution: Convolutional layers

- Convolutional network:

$$
\begin{aligned}
h_i &= a\left[\beta + \omega_1 x_{i-1} + \omega_2 x_i + \omega_3 x_{i+1}\right] \\
&= a\left[\beta + \sum_{j=1}^{3} \omega_j x_{i+j-2}\right]
\end{aligned}
$$

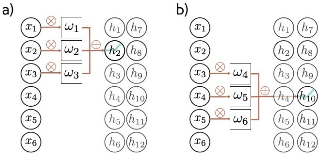
- Fully connected network:

$$
h_i = a\left[\beta_i + \sum_{j=1}^{D} \omega_{ij} x_j\right]
$$

Convolutional networks

1D convolution : Feature maps

- The convolutional operation averages together the inputs
- Plus passes through ReLU function
- Has to lose information
- Solution:
    - Apply several convolutions and stack them in channels
    - Sometimes also called feature maps

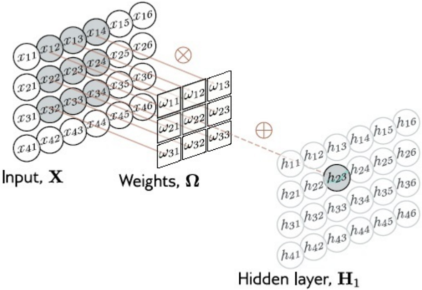


Convolutional networks

2D convolution: Operation

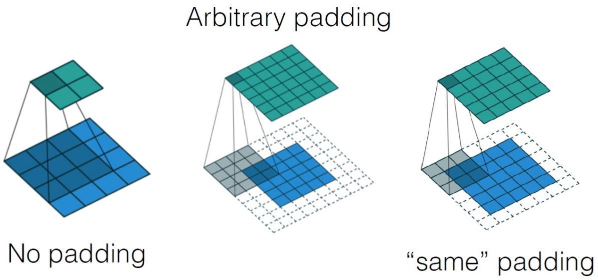
- The convolutional kernel is now a 2D object. A 3×3 kernel $\Omega \in \mathbb{R}^{3\times 3}$ applied to a 2D input comprising of elements x_{ij} computes a single layer of hidden units h_{ij} as:

$$h_{ij} = a\left(\beta + \sum_{m=1}^{3}\sum_{n=1}^{3} \omega_{mn} x_{i+m-2,j+n-2}\right)$$



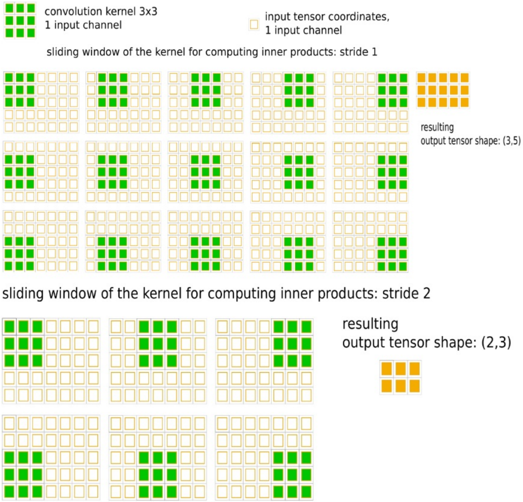
Convolutional networks

2D convolution: Padding



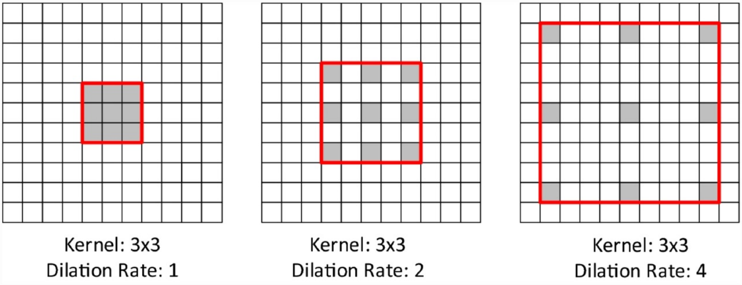
Convolutional networks

2D convolution: Striding



Convolutional networks

2D convolution: Dilated Convolutions



- A dilation factor of 2 corresponds to putting a 0 between every pair of elements in the filter.
- A dilation factor of 4 corresponds to putting three 0s between every pair of elements in the filter.

Convolutional Networks

2D Convolution: Output Dimensions

For a 2D convolution, the output dimensions (height and width) are calculated as follows:

- Output Height:

$$H_{\text{out}} = \left\lfloor \frac{H_{\text{in}} + 2P_h - D_h(K_h - 1) - 1}{S_h} + 1 \right\rfloor$$

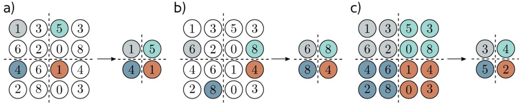
- Output Width:

$$W_{\text{out}} = \left\lfloor \frac{W_{\text{in}} + 2P_w - D_w(K_w - 1) - 1}{S_w} + 1 \right\rfloor$$

Where (for height and width):
- $H_{\text{in}}, W_{\text{in}}$ are the input height and width.
- K_h, K_w are the kernel (filter) height and width.
- P_h, P_w are the padding applied to the height and width.
- S_h, S_w are the strides (step sizes) in the height and width directions.
- D_h, D_w are the dilation rates. Dilation controls the spacing between kernel elements.

Convolutional networks

Downsampling

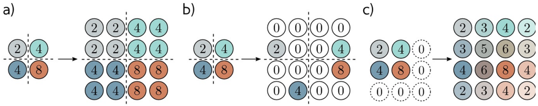


Sample every other position (equivalent to stride two)

Max pooling (partial invariance to translation)

Mean pooling

Upsampling



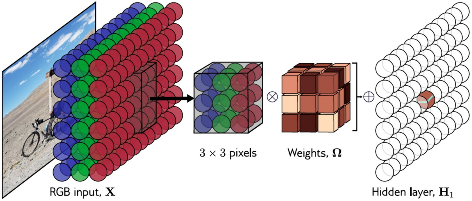
Duplicate

Max-upsampling

Bilinear interpolation

Convolutional networks

2D convolution with multiple channels



0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2

0	1	1
0	1	0
1	-1	1

Kernel Channel #3

$$308 \quad + \quad -498 \quad + \quad 164 \quad + 1 = -25$$

Bias = 1

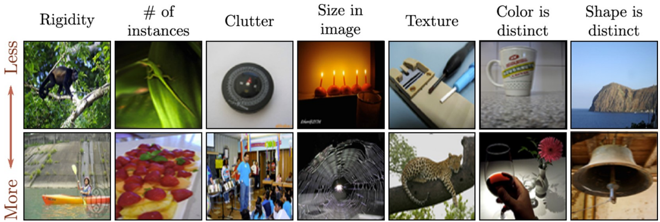
Output

-25				...
				...
				...
				...
...	...	...	...	...

Convolutional networks Applications

Image classification

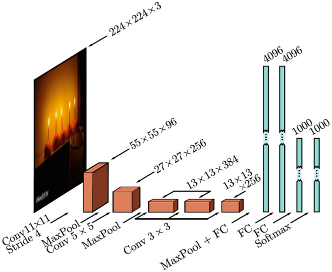
- ImageNet dataset:
  - 224 x 224 images
  - 1,281,167 training images, 50,000 validation images, and 100,000 test images
  - 1000 classes



Convolutional networks Applications

Image classification

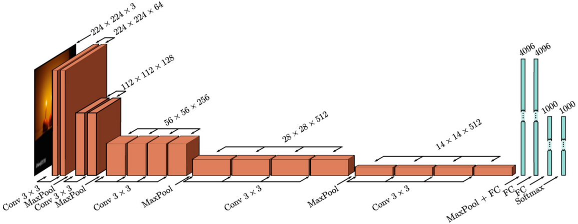
- AlexNet:
  - The network maps a 224×224 color image to a 1000-dimensional vector representing class probabilities
  - The network first convolves with 11×11 kernels and stride 4 to create 96 channels.
  - It decreases the resolution again using a max pool operation and applies a 5×5 convolutional layer.
  - Another max pooling layer follows, and three 3×3 convolutional layers are applied.
  - After a final max pooling operation, the result is vectorized and passed through three fully connected (FC) layers and finally the softmax layer.
  - 60 million parameters
  - This system achieved a 16.4% top-5 error rate and a 38.1% top-1 error rate.



Convolutional networks Applications

Image classification

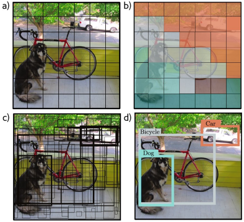
- VGG network:
  - This network consists of a series of convolutional layers and max pooling operations, in which the spatial scale of the representation gradually decreases, but the number of channels gradually increases.
  - The hidden layer after the last convolutional operation is resized to a 1D vector and three fully connected layers follow.
  - The network outputs 1000 activations corresponding to the class labels that are passed through a softmax function to create class probabilities.
  - 144 million parameters
  - This system achieved a 6.8% top-5 error rate, 23.7% top-1 error rate



Convolutional networks Applications

Object detection

- You Only Look Once (YOLO):
  - The input image is reshaped to 448×448 and divided into a regular 7×7 grid
  - The system predicts the most likely class at each grid cell.
  - It also predicts two bounding boxes per cell, and a confidence value (represented by thickness of line).
  - During inference, the most likely bounding boxes are retained, and boxes with lower confidence values that belong to the same object are suppressed.
  - Momentum, weight decay, dropout, and data augmentation
  - Heuristic at the end to threshold and decide final boxes



Convolutional networks Applications

Semantic segmentation

- Semantic segmentation example:
  - The input is a 224×224 image, which is passed through a version of the VGG network and eventually transformed into a representation of size 4096 using a fully connected layer. This contains information about the entire image
  - This is then reformed into a representation of size 7×7 using another fully connected layer, and the image is upsampled and deconvolved (transposed convolutions without upsampling) in a mirror image of the VGG network.
  - The output is a 224×224×21 representation that gives the output probabilities for the 21 classes at each position.