ELEC-C5220
Lecture 3:
Spoken digit recognition with Convolution Neural Networks

Machine learning in information technology

Aalto University
School of Electrical Engineering

Lauri Juvela

25.1.2024

Lecture 3 - content

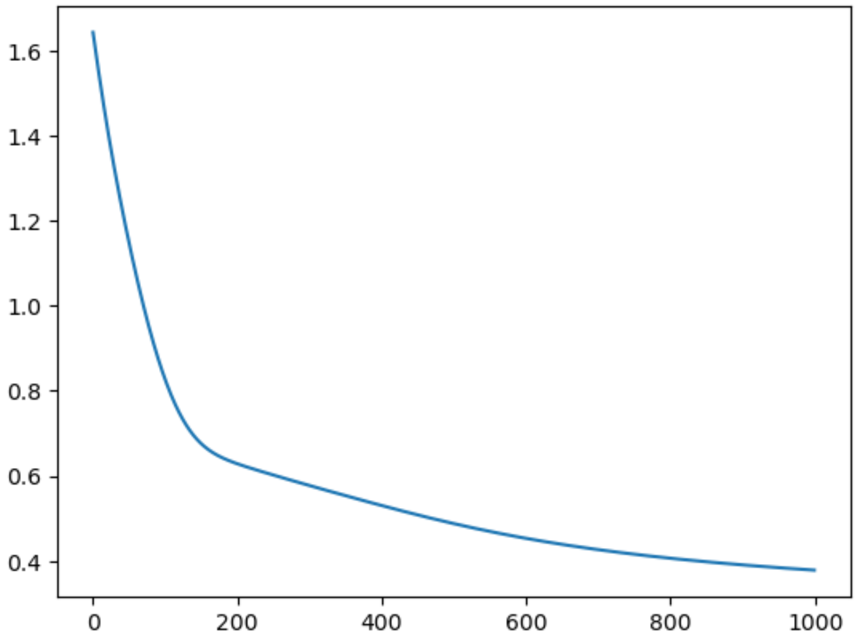
- Filtering revisited
- Convolution for audio
- Convolution for images
- Convolution layers
- Residual networks
- Pooling
- Convolution Neural Networks (CNNs)

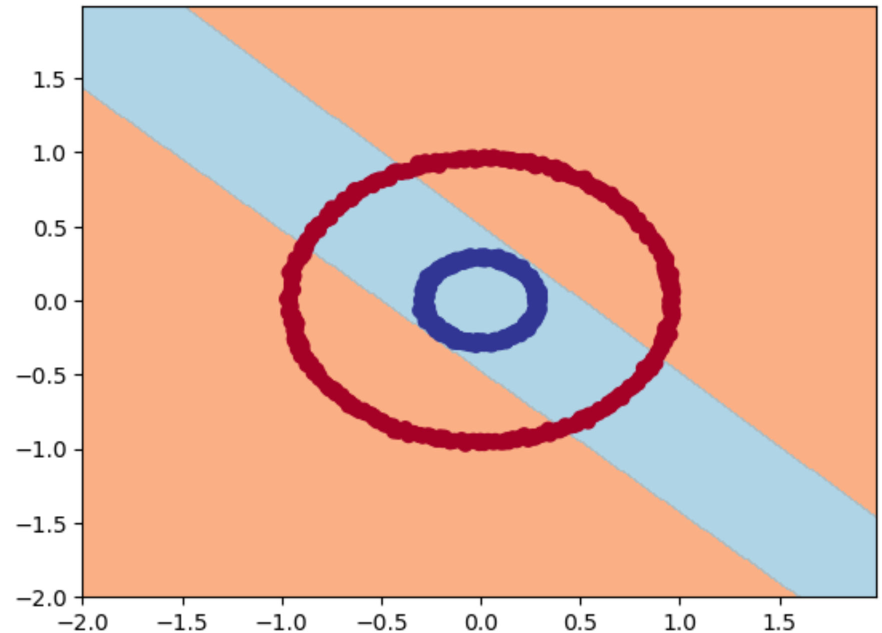
Exercise 01 – Analysis

- **Average score was high**
- **Submissions by deadline: 70**
- **Main lessons**
  - Remember to add a non-linearity between layers
  - Remember to normalise output distribution (keep the sigmoid)
  - It it doesn’t work, try changing the hyperparameters (network depth, hidden size, learning rate, etc.)

Exercise 01 – Analysis

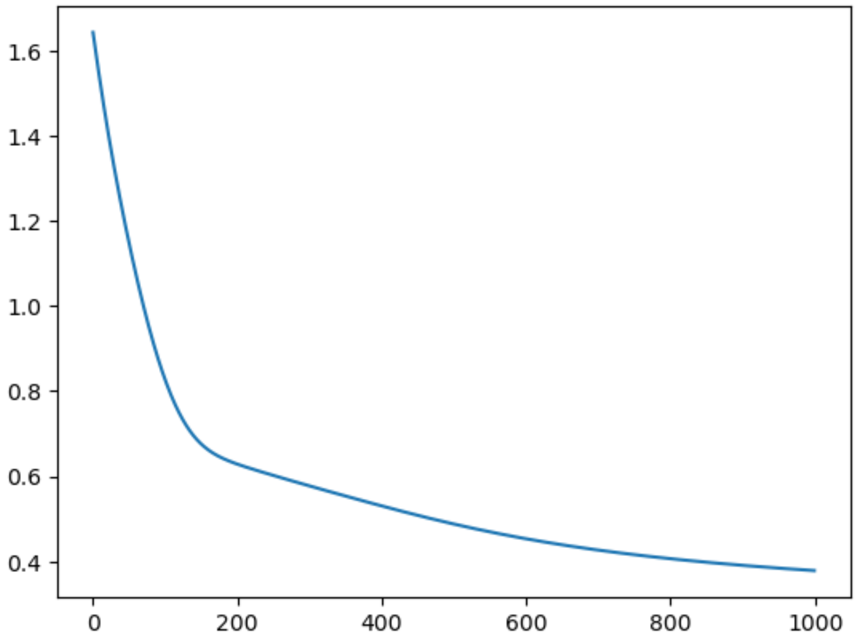
- **What went wrong here?**

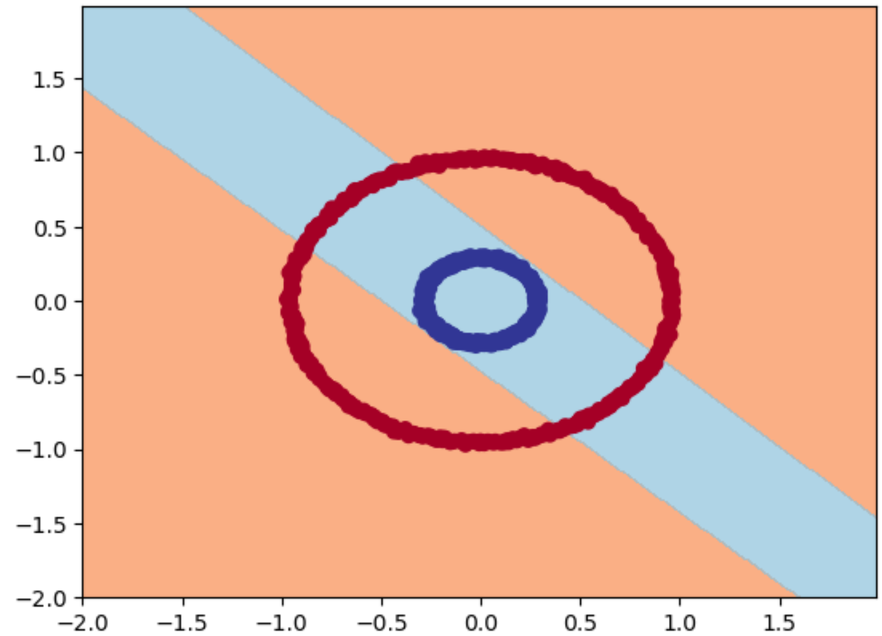




Exercise 01 – Analysis

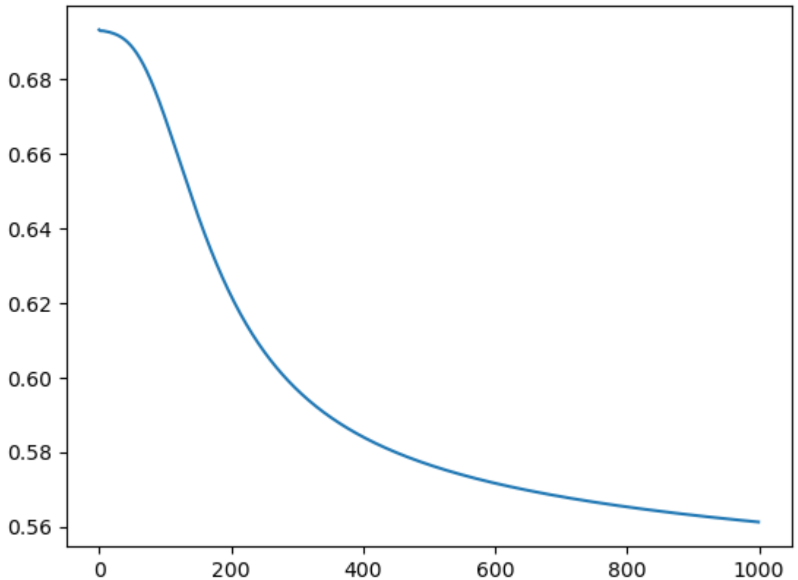
- What went wrong here?
- Hidden size 3 and sigmoid activation function

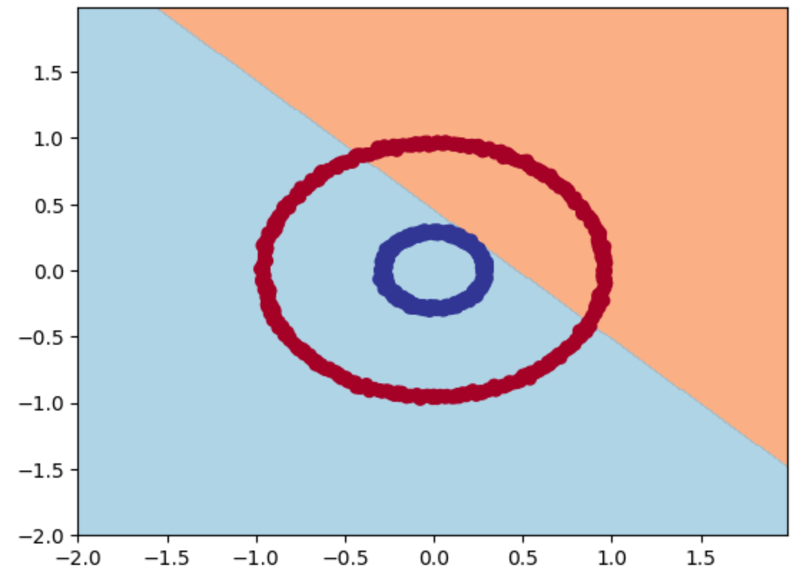




Exercise 01 – Analysis

- Hidden size 1 with sigmoid activation function





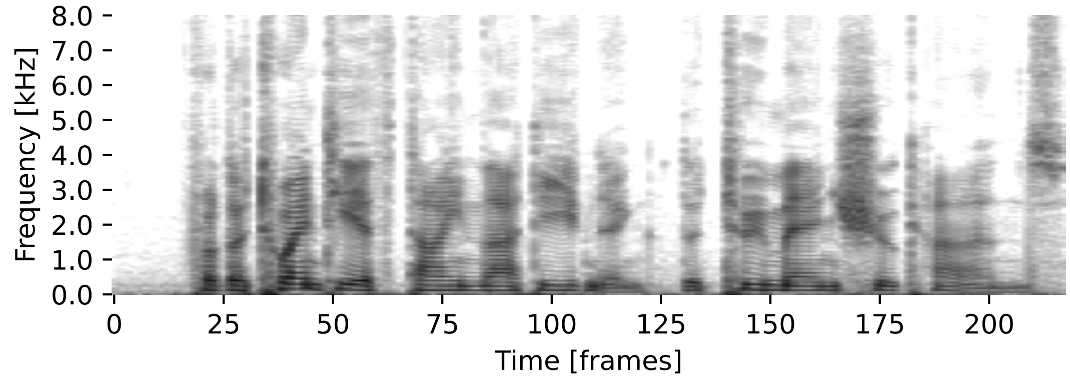
What will be in the Exercise next week?

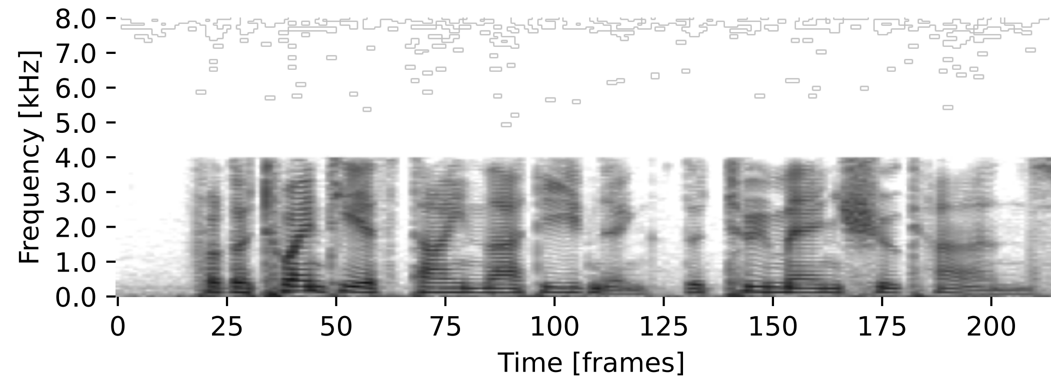
- MNIST handwritten digit recognition with CNNs
- Spoken digit recognition with spectrograms and CNNs
- No filter design
- No manually implementation of convolution layers

Filtering revisited

- **Filters are convenient to design and implement in the frequency domain**

$$Y(z) = A(z)X(z)$$





Filtering and convolution

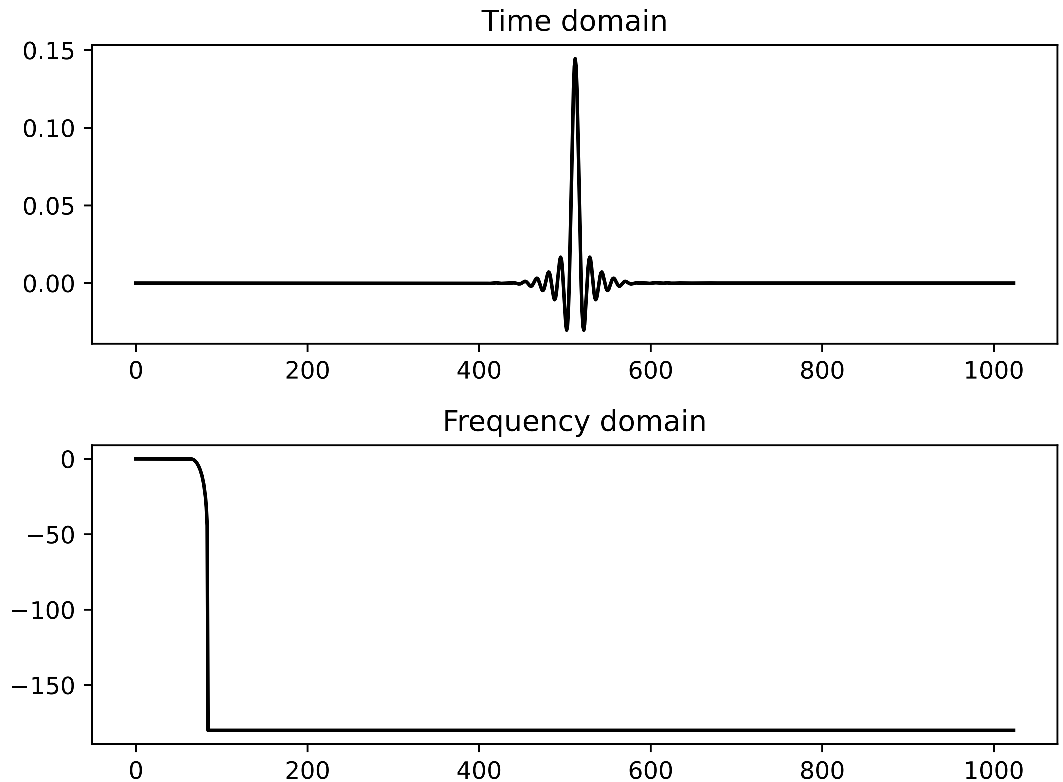
- Frequency domain implementation requires FFTs and zero-padding
- For short filters, it is often more efficient to implement the filter in time domain using convolution
- Typically, filter order P < 100
- This type of filter is called Finite Impulse Response (FIR) filter

$$Y(z) = A(z)X(z)$$

$$y_n = \sum_{i=0}^{P} a_i x_{n-i}$$

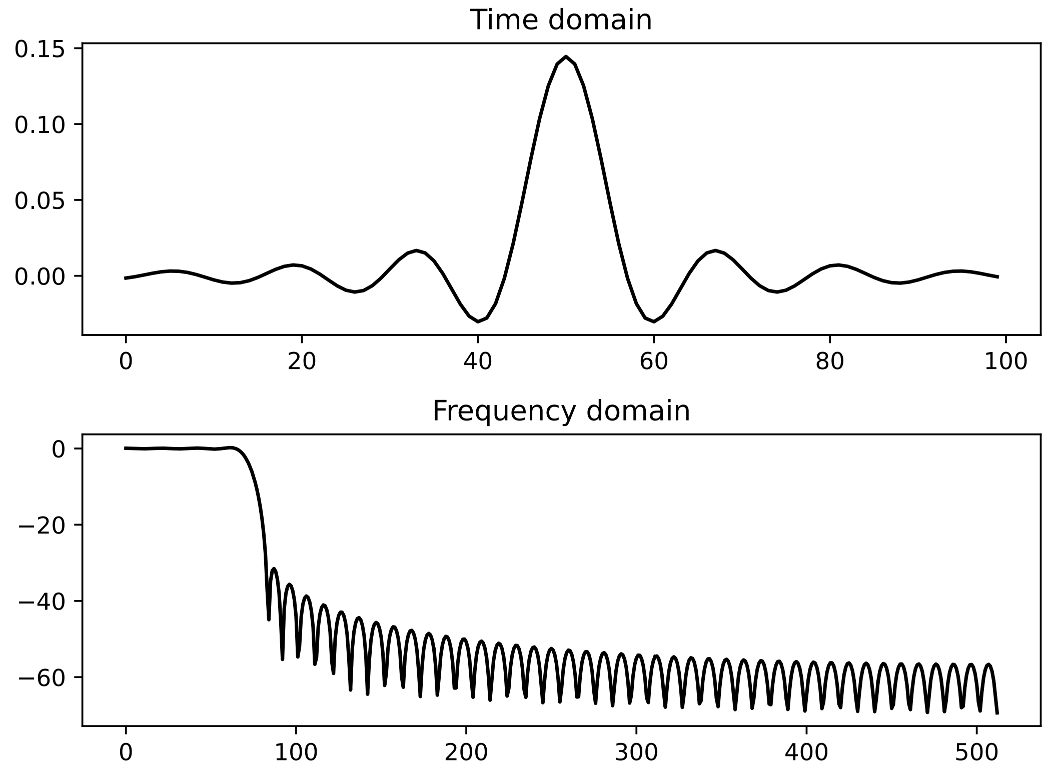
Low-pass filter design

- **Draw the prototype filter response in frequency domain**
- **Inverse Fourier transform gives the filter impulse respose**
- **In deep learning, the impulse response is called “convolution kernel”**



Filter design involves trade-offs

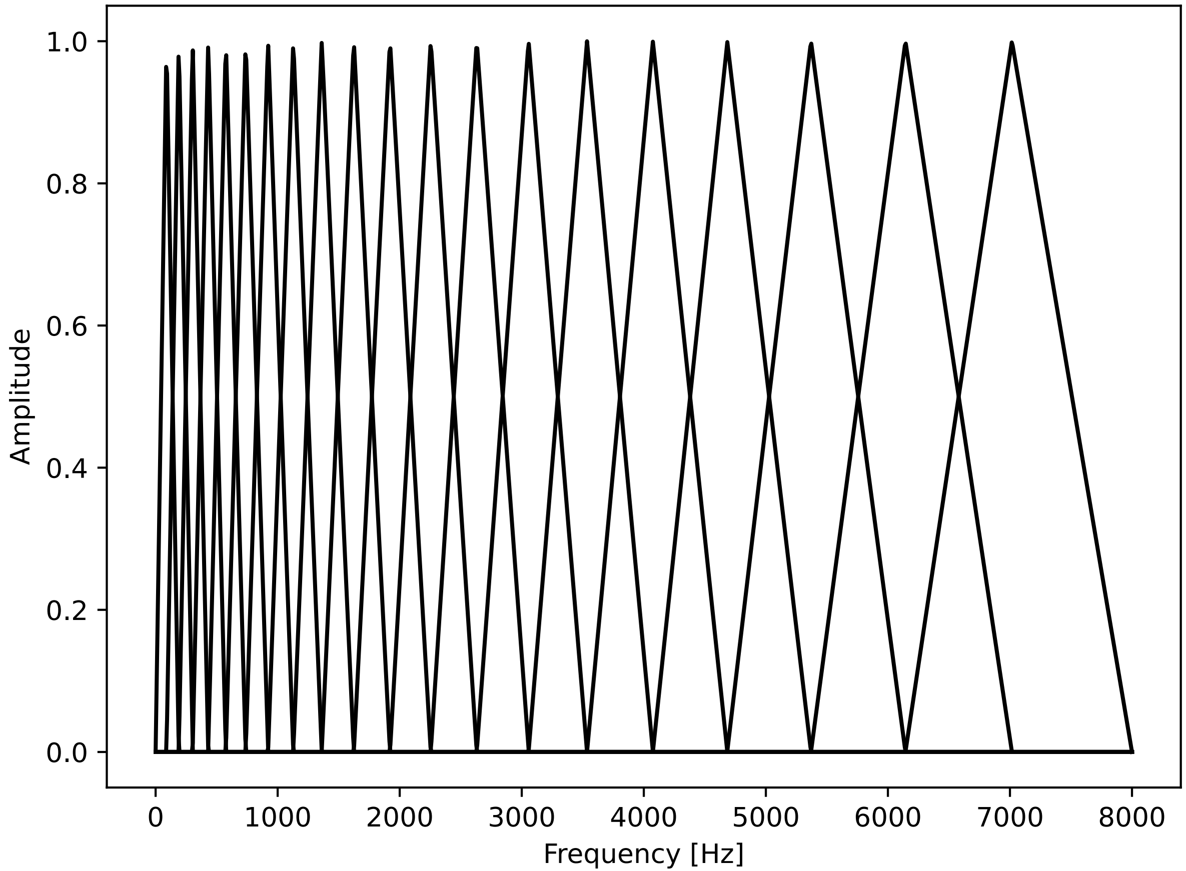
- **Truncate the impulse response to reduce computation cost and latency**
- **Truncation in time causes ripples and side-lobes in the frequency response**
- **Time-frequency uncertainty principle**



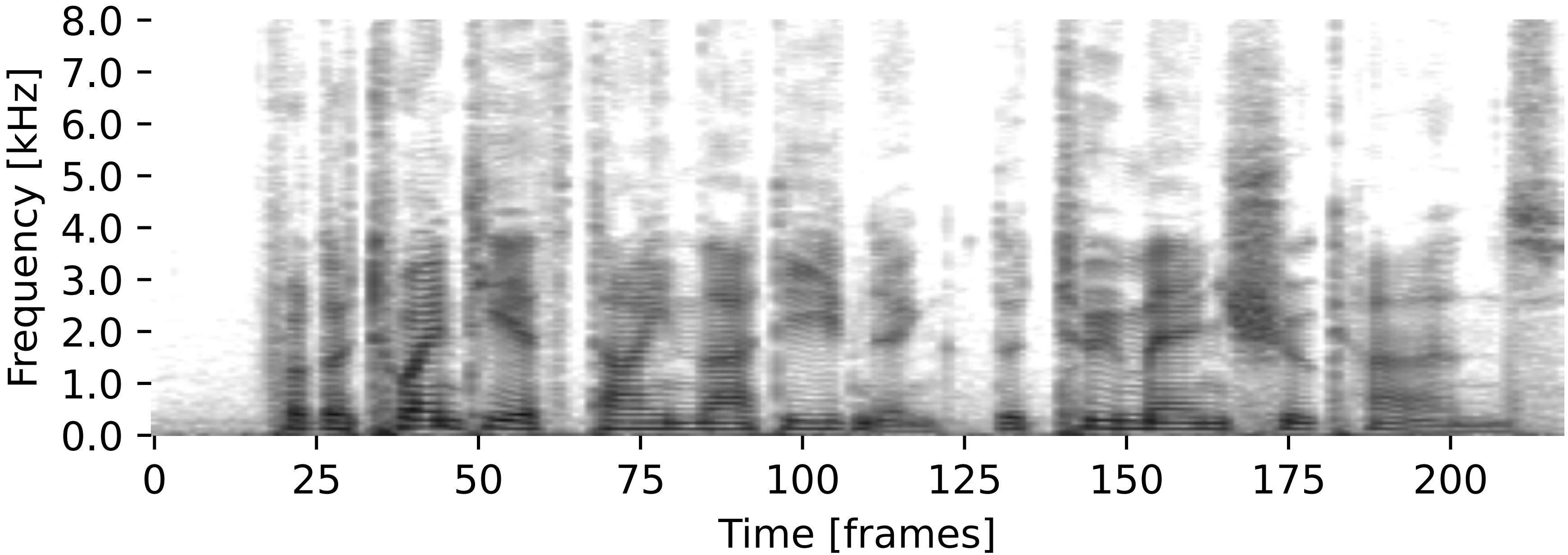
Filters for feature extraction

- Various filters are useful for different detection tasks
- Low pass filters can be used for energy estimation
- Band-pass filters can be used for Fourier analysis
- High-pass filters can be used for edge detection
- Etc…

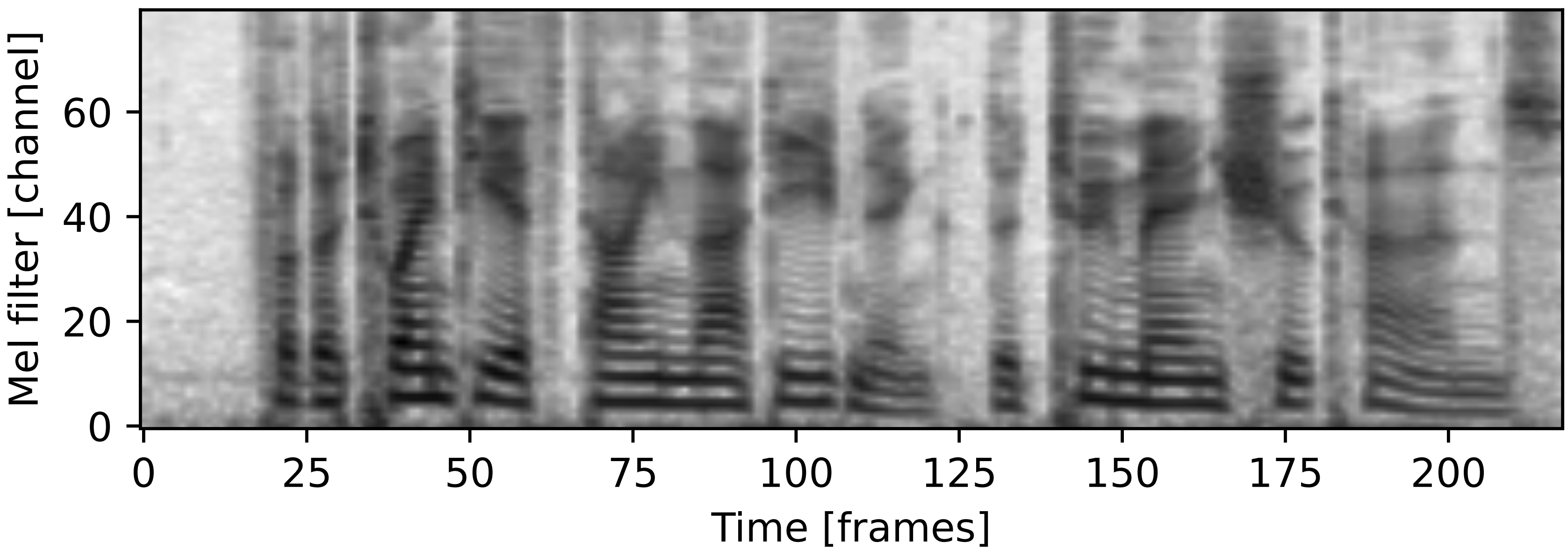
Mel-frequency filterbank



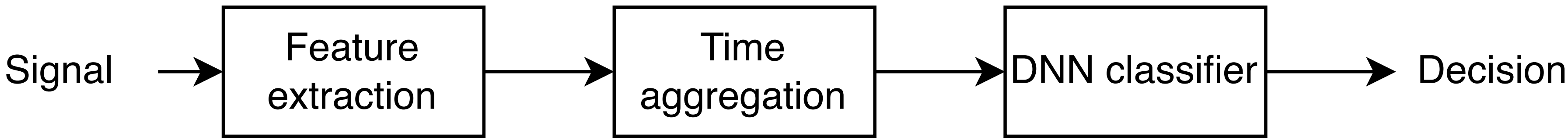
Spectrogram (is also a filter bank)



Mel-spectrogram

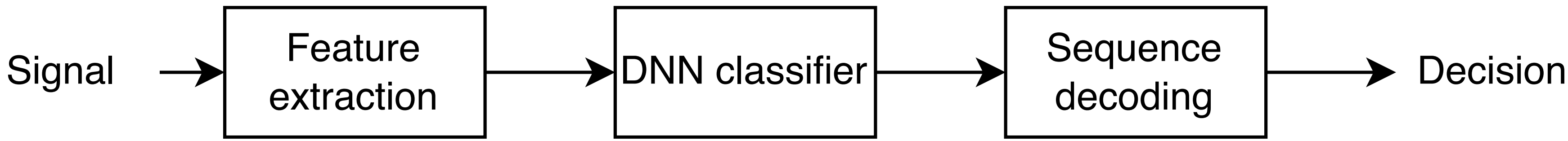


Feature-based classification



- **Step 1: Extract features,**
  - For example, with filterbanks
- **Step 2: Aggregate features over time**
  - For example, long term average spectrograms,
- **Step 3: Build a DNN classifier on time-aggregated feature vectors**

Feature-based classification



- **Step 1: Extract features,**
  - for example, with filterbanks
- **Step 2: Build a DNN classifier on sequence time elements individually**
- **Step 3: Decode probability sequence to a single decision (using Viterbi search or**

Towards convolution neural networks

- **Feature design is hard, including filterbanks**
- **We don’t know what kind of filters or filterbanks provide optimal features for our task**
- **Step one: let’s make the front-end filter parameters part of a neural network and optimise them jointly**
- **Step two: let’s make the whole network out of filters to capture long-term temporal dependencies**

Convolution

$$y_n = \sum_{i=0}^{P} a_i x_{n-i}$$

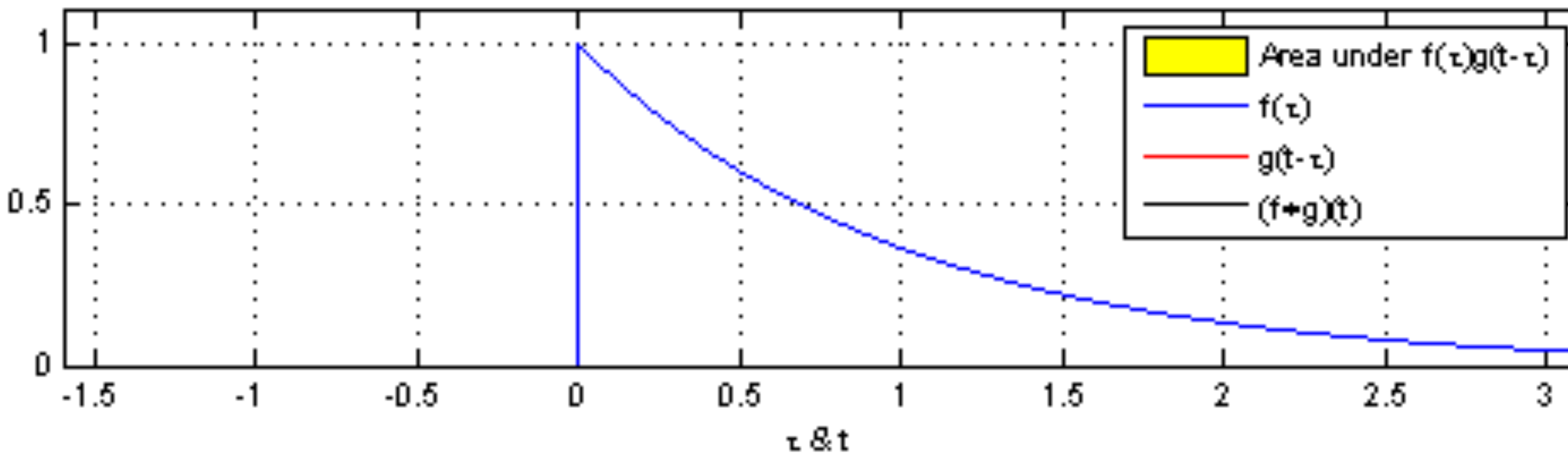
$$Y(z) = A(z)X(z)$$

$$(f * g)(t) = \int f(\tau)g(t-\tau)\mathrm{d}\tau$$

$$(f * g)[n] = \sum_i f[i]g[n-i]$$

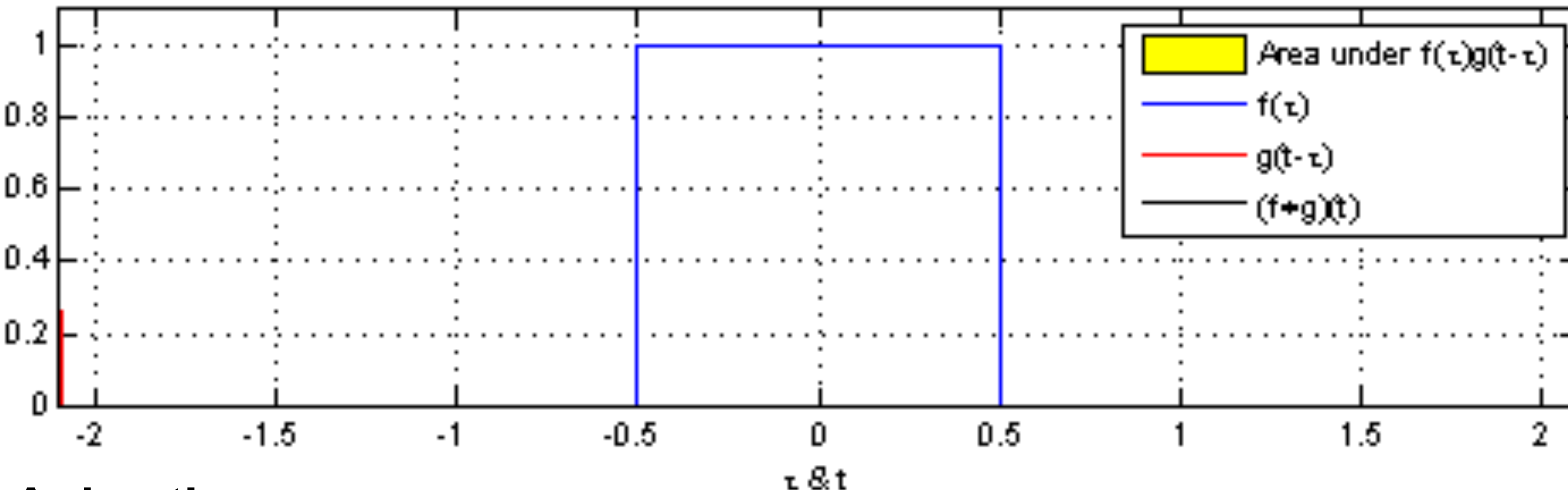
$$(f * g)(t) \leftrightarrow F(\omega)G(\omega)$$

Convolution animated



Animation source:
https://en.wikipedia.org/wiki/Convolution#Visual_explanation

Convolution animated



Animation source:
https://en.wikipedia.org/wiki/Convolution#Visual_explanation

Convolution and cross-correlation

- What deep learning toolkits call convolution is actually cross-correlation!
- Convolution time-reverses the filter coefficients
- Cross-correlation is otherwise the same, but the filters are the same

Convolution and cross-correlation

$$(f * g)(t) = \int f(\tau) g(t - \tau) \mathrm{d}\tau \qquad (f \star g)(t) = \int f(\tau) g(t + \tau) \mathrm{d}\tau$$

$$(f * g)[n] = \sum_i f[i] g[n - i] \qquad (f \star g)[n] = \sum_i f[i] g[n + i]$$

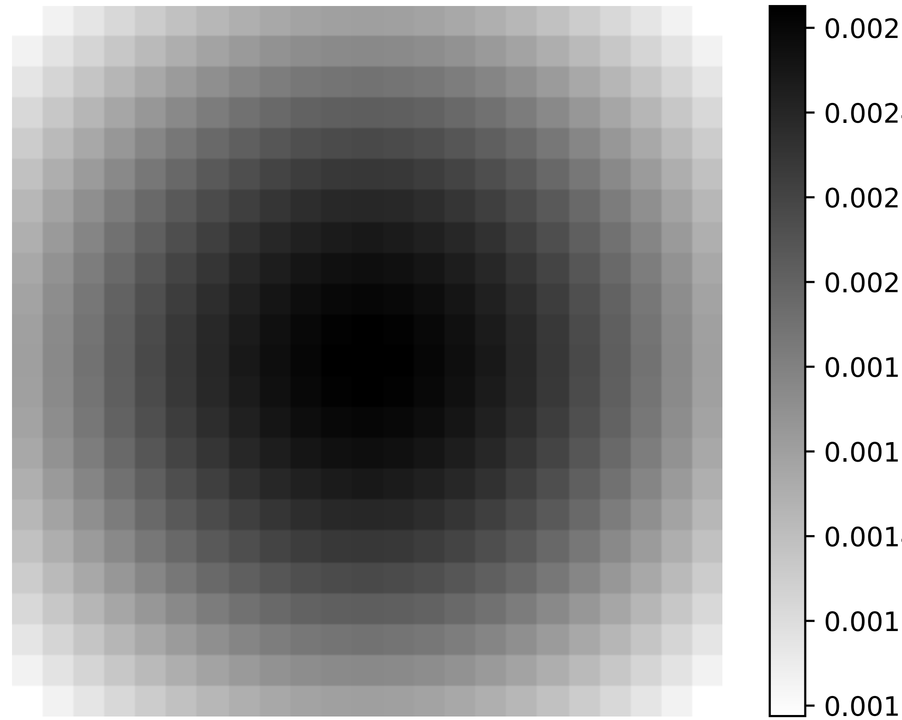
$$(f * g)(t) \leftrightarrow F(\omega) G(\omega) \qquad (f \star g)(t) \leftrightarrow F(\omega) G^*(\omega)$$

$$y_n = \sum_{i=0}^{P} a_i x_{n-i} \qquad y_n = \sum_{i=0}^{P} a_i x_{n+i}$$

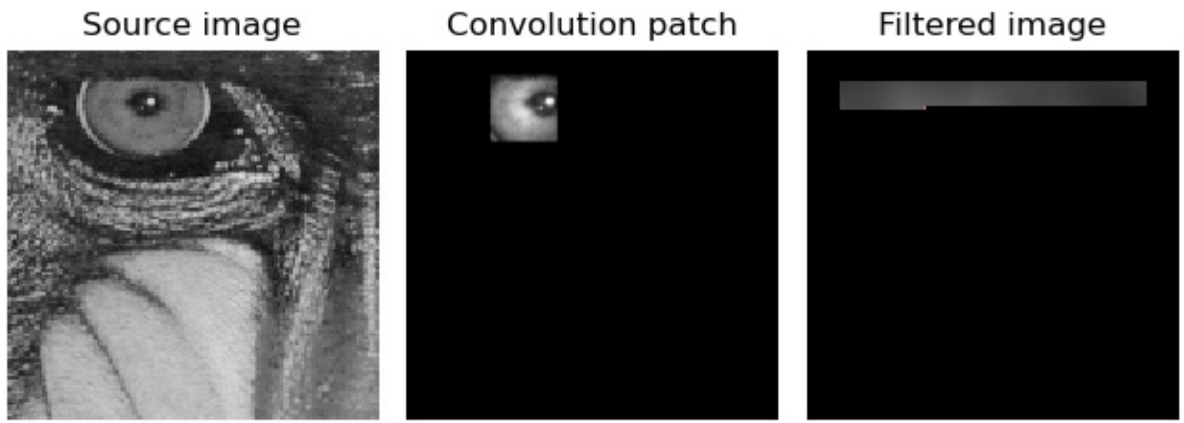
$$Y(z) = A(z) X(z) \qquad Y(z) = A^*(z) X(z)$$

Convolution for images

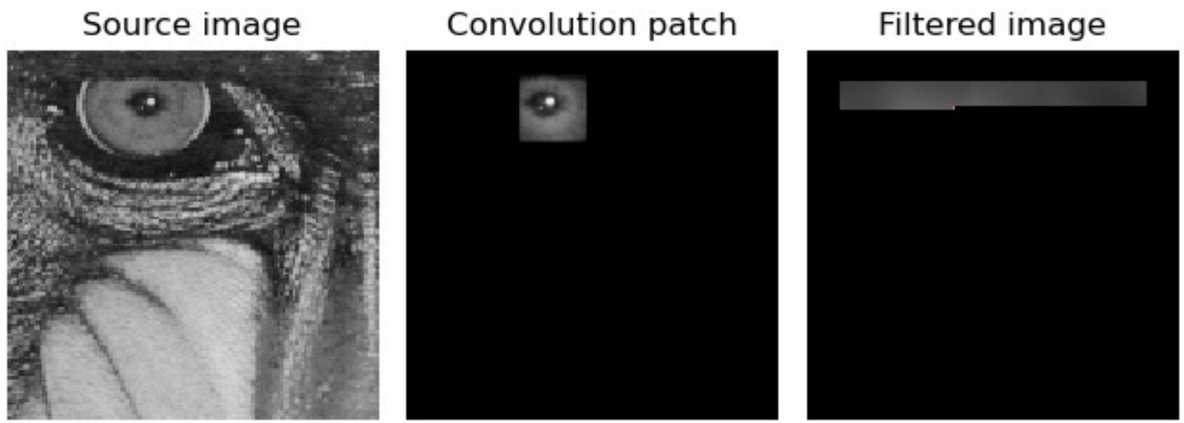
- **Let's apply a Gaussian blur low-pass filter**
- **Filter kernel on the right: top view of a 2D Gaussian bell shape**
- **For each pixel:**
    - Choose a patch around the pixel
    - Multiply with kernel
    - Sum over patch



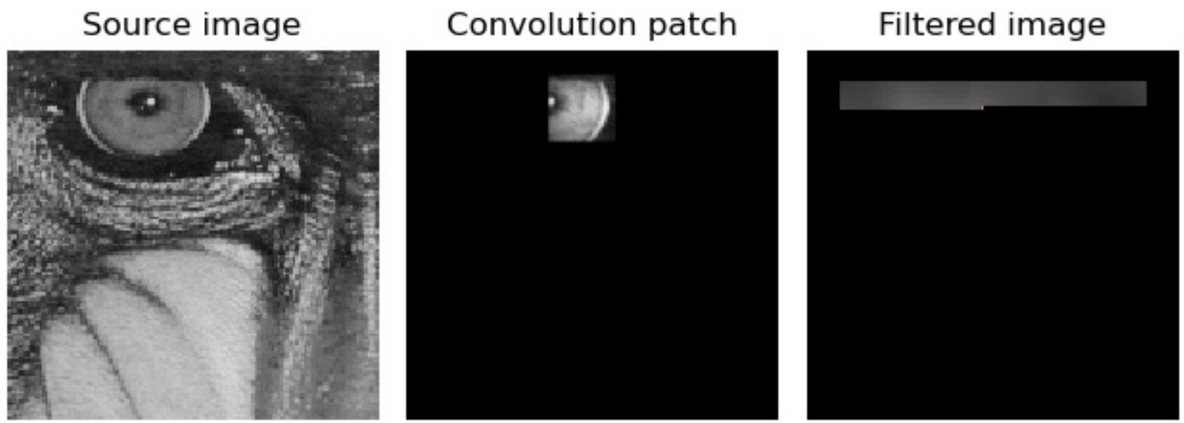
Convolution for images



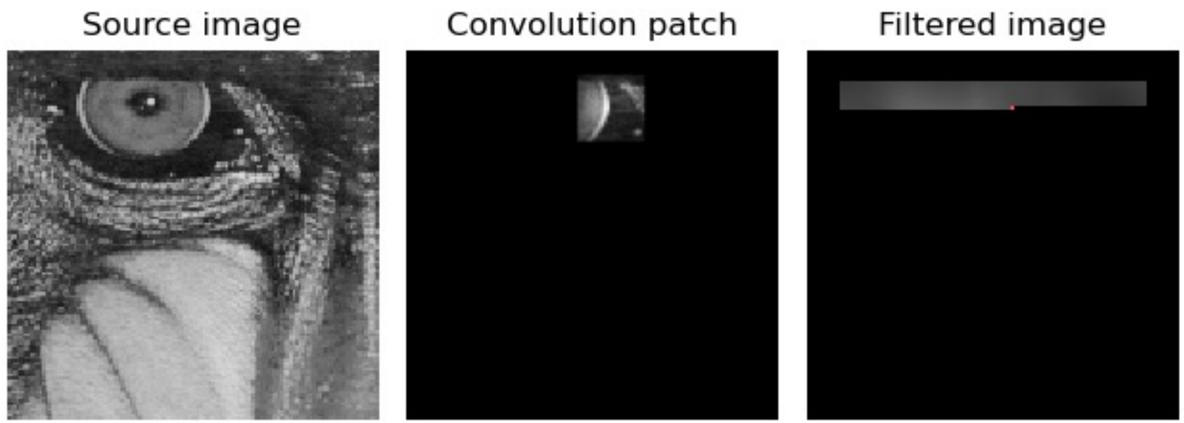
Convolution for images



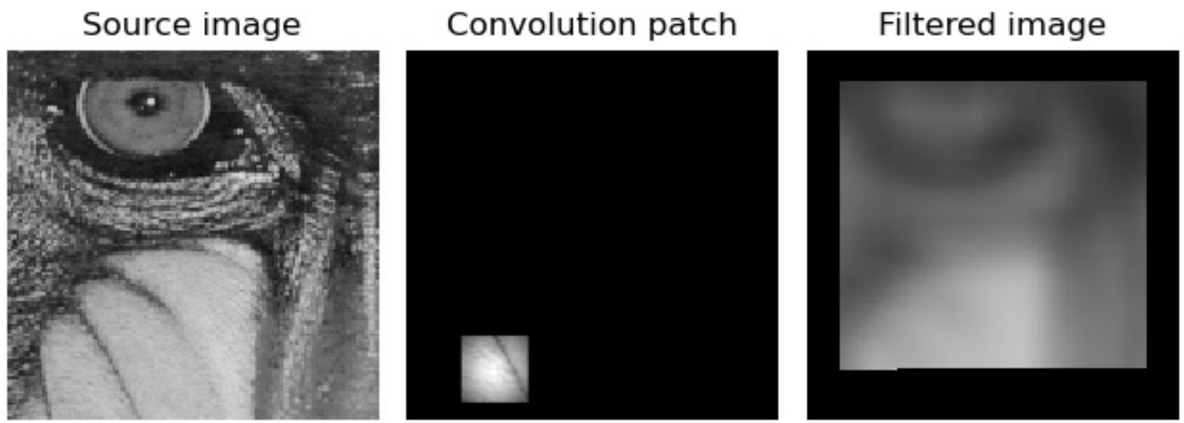
Convolution for images



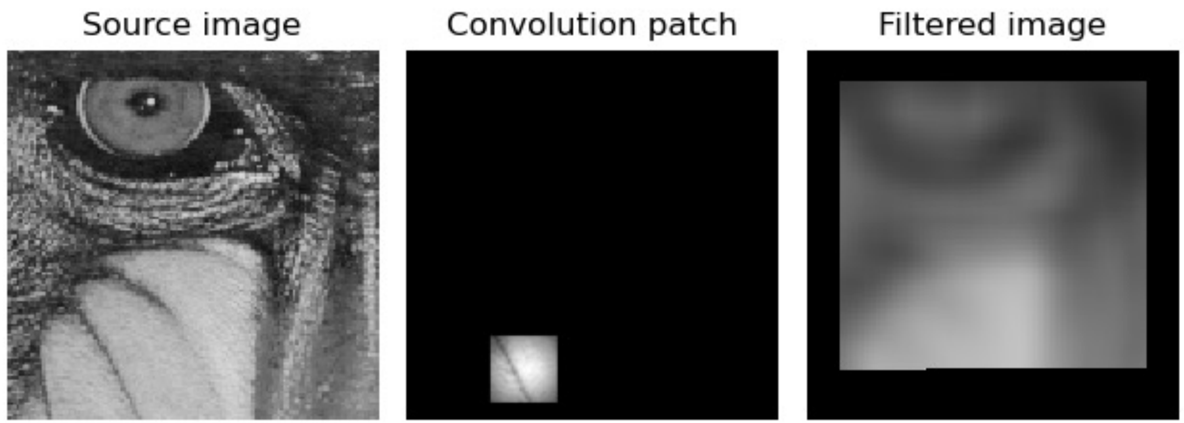
Convolution for images



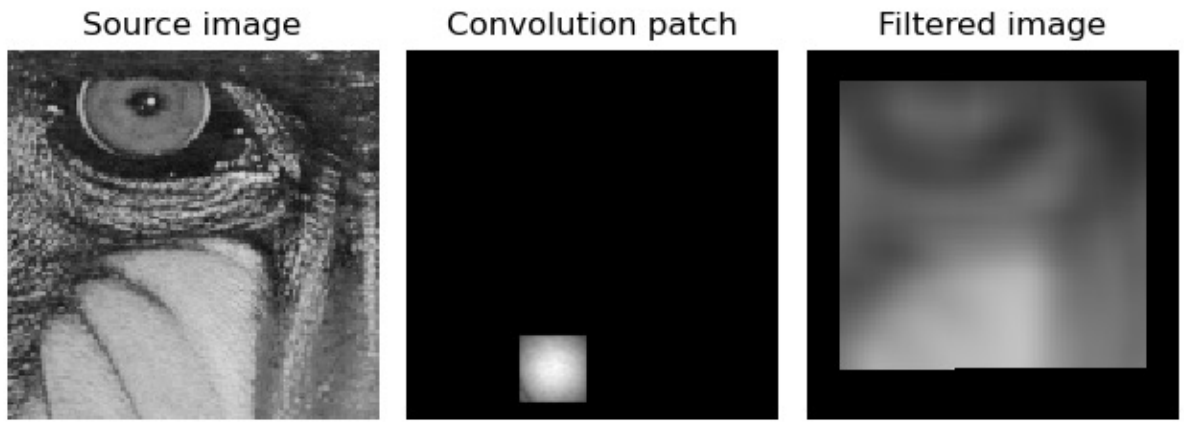
Convolution for images



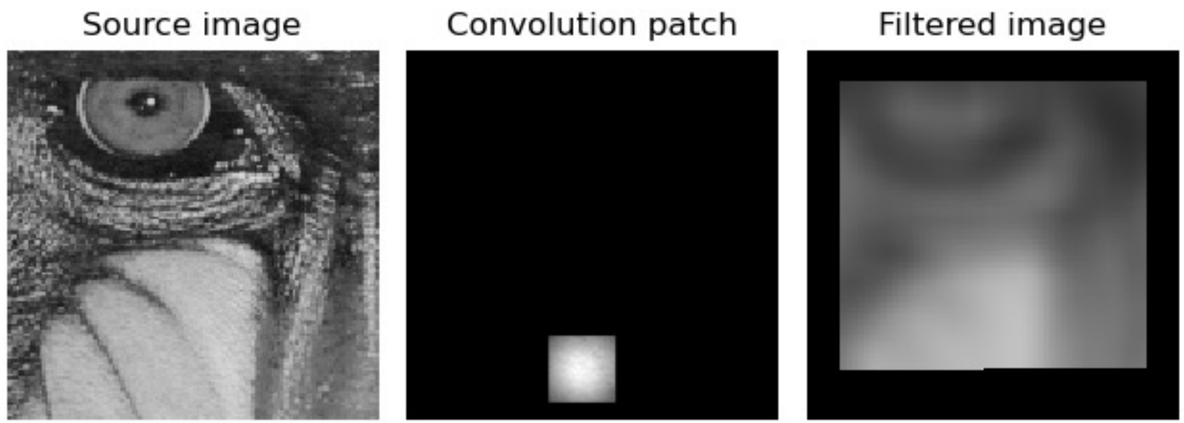
Convolution for images



Convolution for images



Convolution for images



Convolution layers in PyTorch

CONV1D

CLASS torch.nn.Conv1d(*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 1D convolution over an input signal composed of several input planes.

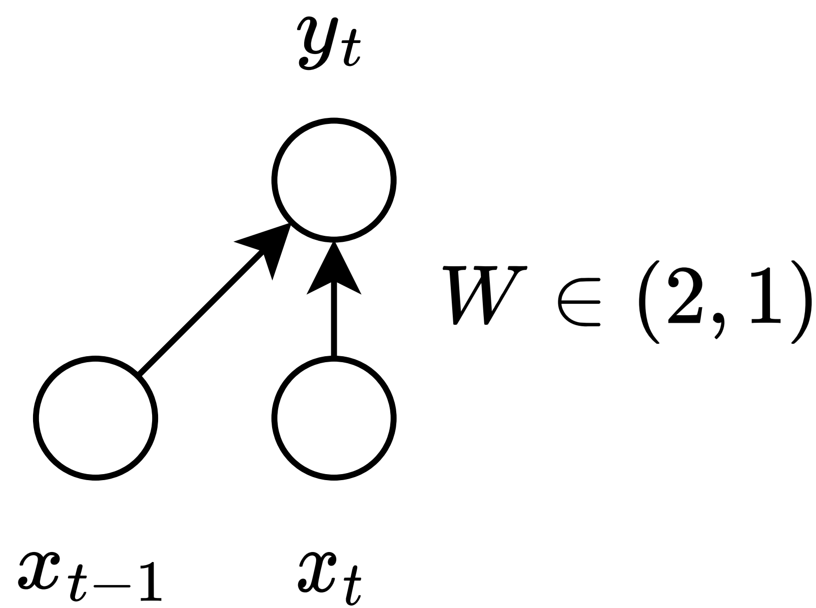
In the simplest case, the output value of the layer with input size (N, C_{in}, L) and output $(N, C_{\text{out}}, L_{\text{out}})$ can be precisely described as:

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

where $\star$ is the valid cross-correlation operator, N is a batch size, C denotes a number of channels, L is a length of signal sequence.

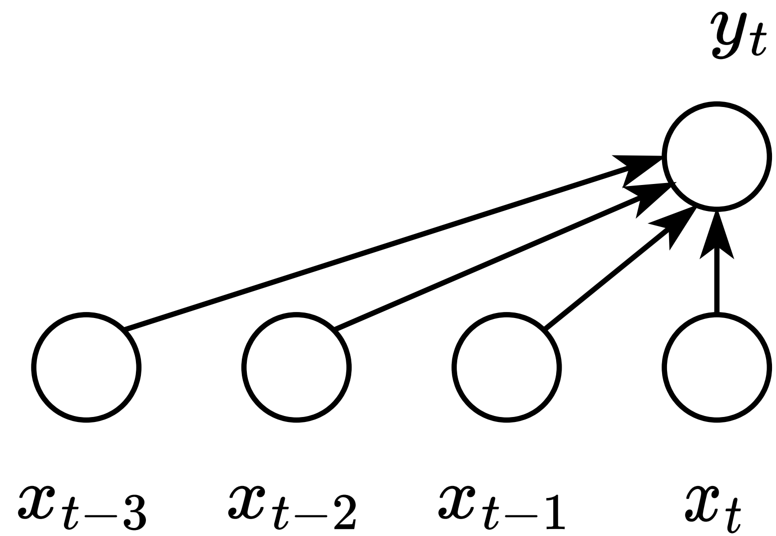
Minimal convolution net

- At each time-step, the output depends on the input values at current and previous time-steps
- Same dependency for all time values: weight sharing across time



Convolution is filtering

- **Input dimension – 4 time steps**
- **Output dimension – 1 time step**

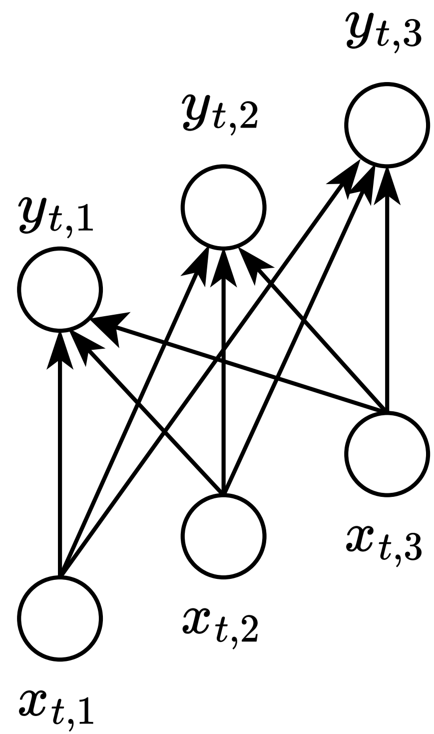


$$W \in (4, 1)$$

$$y_t = \sum_{i=0}^{3} W_{i,0} x_{t-i}$$

Convolution is fully connected

- Channels in CNNs are fully connected
- Fully connected DNNs are a special case of Convolution networks
- Kernel width = 1
- Input dim. = input channels
- Output dim. = output channels

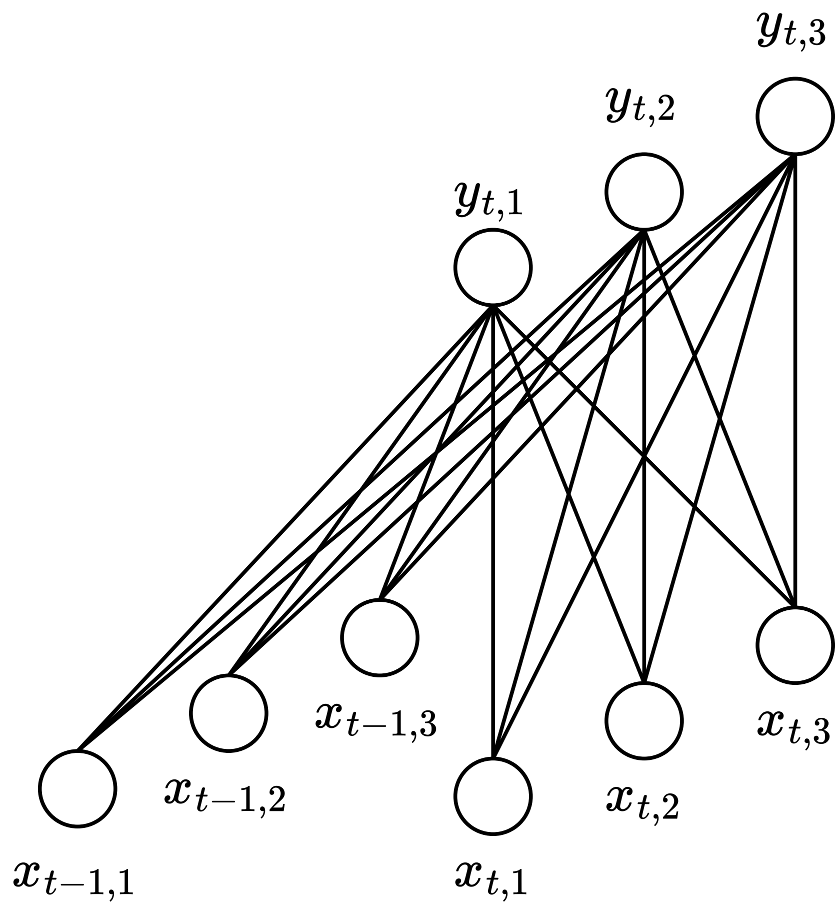


$W \in (3, 3)$

$$y_{t,j} = \sum_{i=1}^{3} W_{j,i} x_{t,i}$$

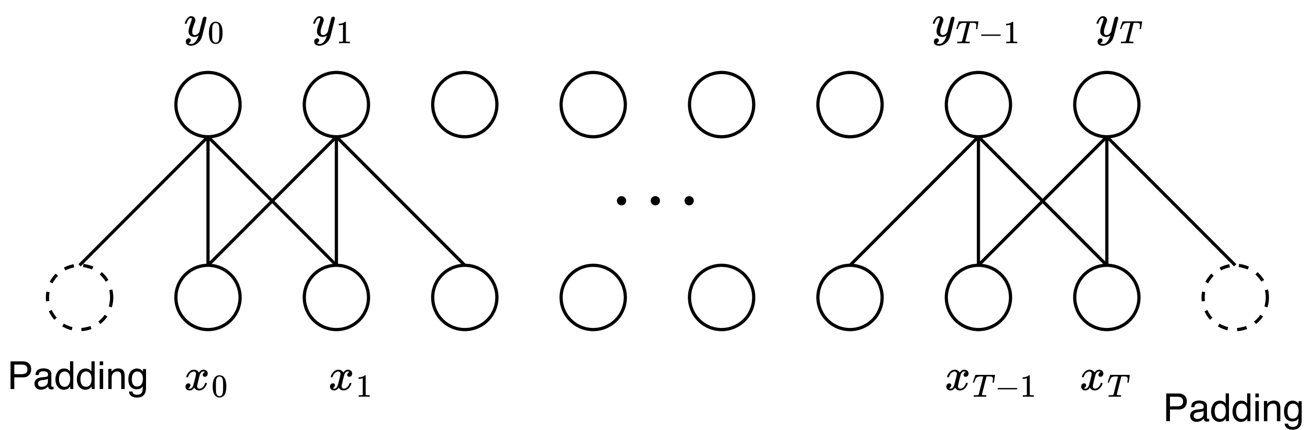
Convolution layer

- **Fully connected over channels**
- **Fully connected over kernel width in time**
- **Slide the connectivity pattern over the input to compute output values for the whole sequence**



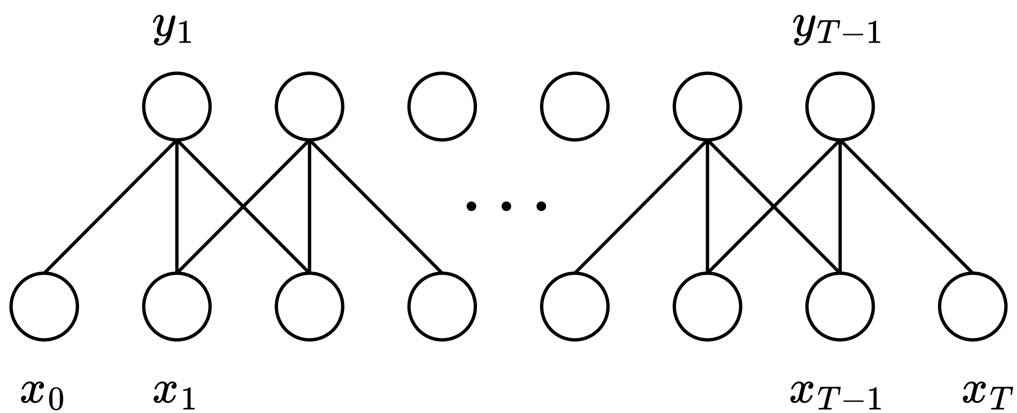
$W \in (3, 3, 2)$

Padding and valid convolution



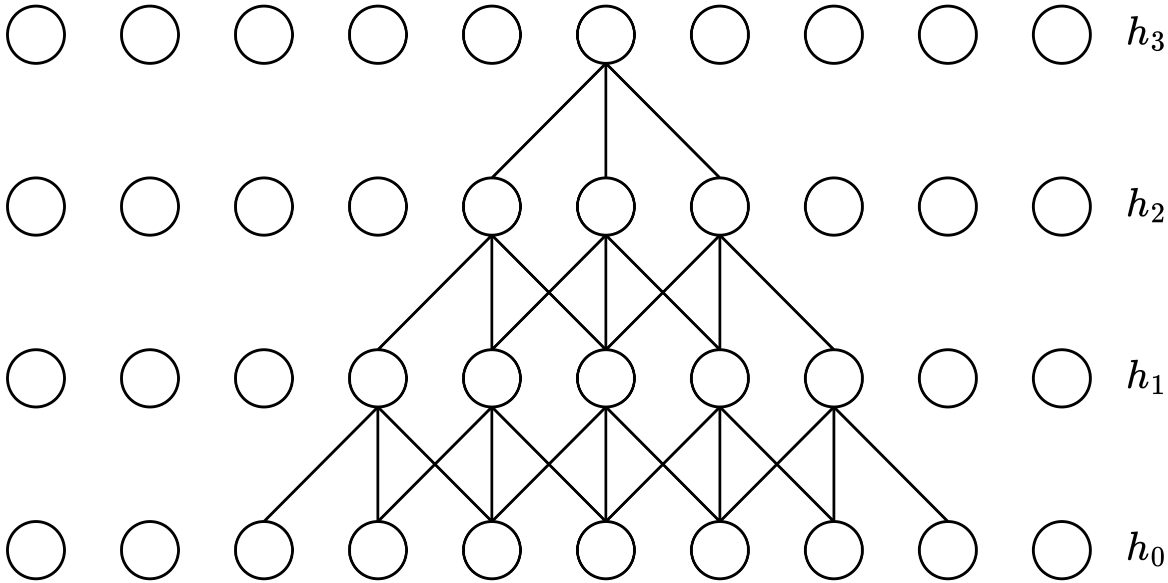
- **"Same" padding mode zero-pads input so that the convolution output has the same number of timesteps as the input**
- **In this example, the filter width is 3**

Padding and valid convolution



- “Valid” convolution does not zero pad and invalid values are dropped at the output edges
- In this example, the filter width is 3, which drops 1 sample at both ends
- This can be useful for spatial reduction

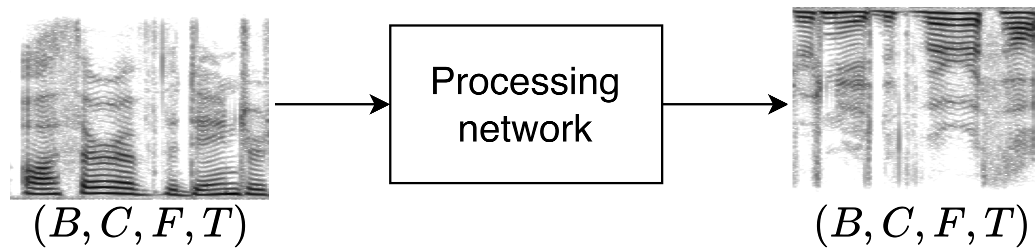
Receptive field



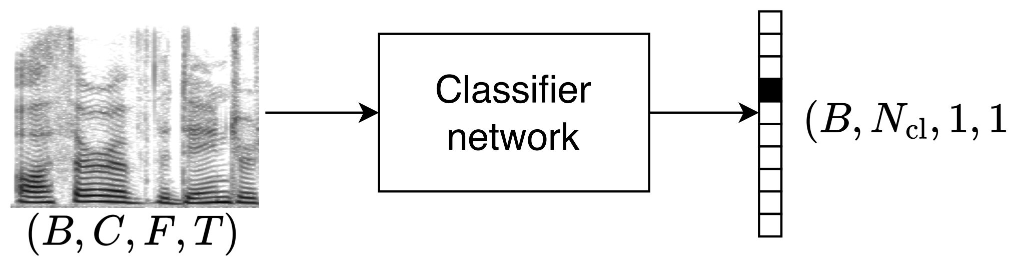
- **The number of input nodes (i.e., time-steps) an output sees depends on the filter widths and network depth**

Task defines input and output shapes

- **Processing tasks often have the same size for input and output**



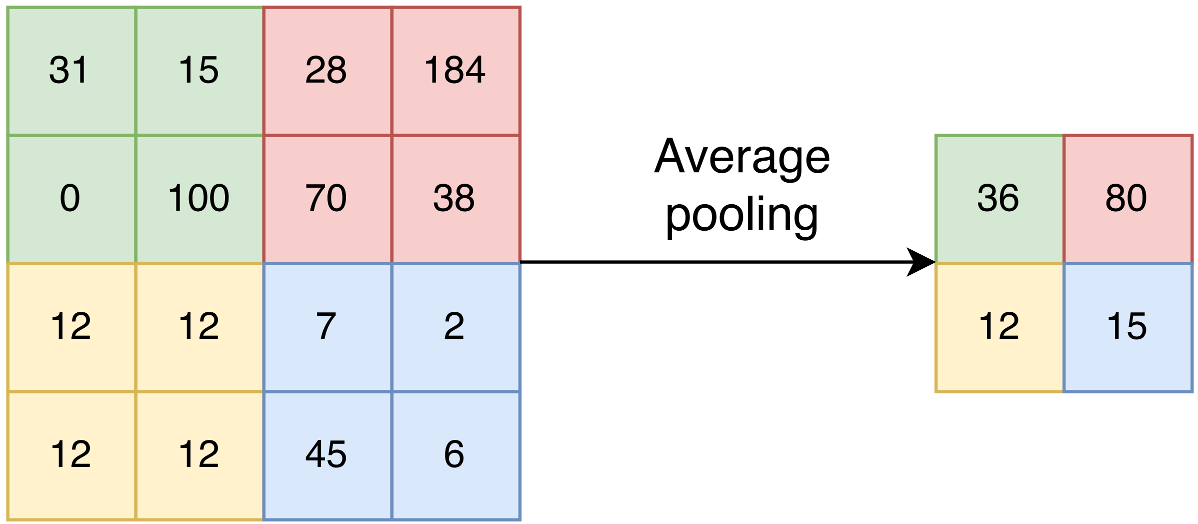
- **Classification tasks require dimensionality reduction on spatial or time-frequency axes**



Spatial dimensionality reduction by pooling

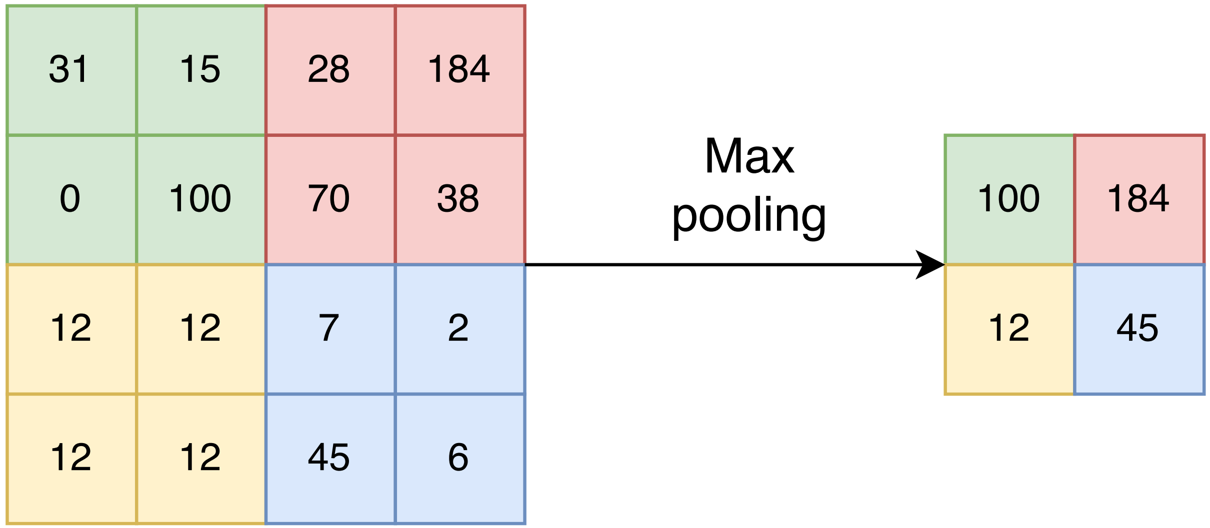
- **Option 1: No pooling, flatten and apply fully connected layer**
  - Pros: simple concept
  - Cons: sequence length must be fixed
- **Option 2: Global pooling**
  - Pros: works for any sequence length
  - Cons: all pooling is done at the same time, poor temporal resolution
- **Option 3: Pool and downsample with sliding windows**
  - Pros: works for arbitrary sequence lenghts, good resolution
  - Cons: more design choices to make

Average pooling



- **Sliding window size (2, 2)**
- **Stride determines the downsampling factor, (2,2) in this case**
- **Output is the average of values within a window**

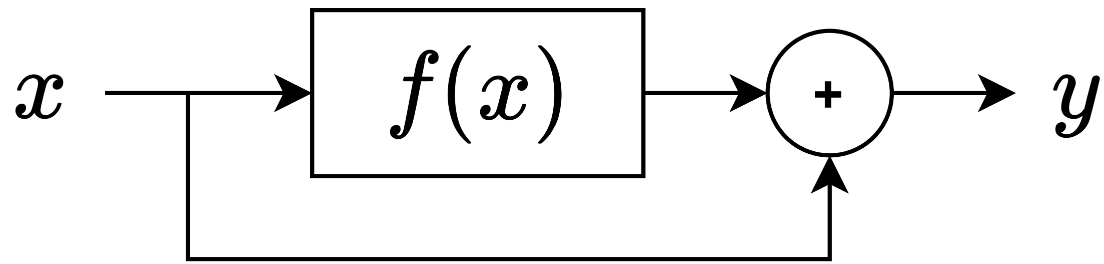
Max pooling



- Sliding window size (2, 2)
- Stride determines the downsampling factor, (2,2) in this case
- Output is the maximum of value within a window

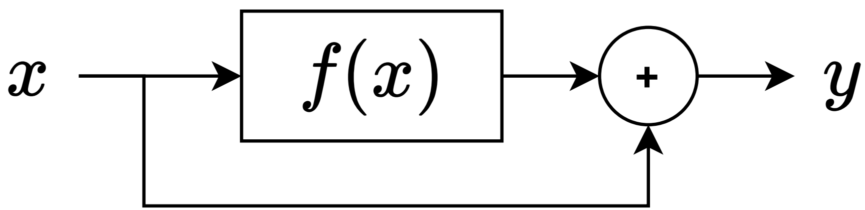
Residual connections

- Deep and narrow networks are often more parameter-efficient
- Deep nets suffer from vanishing gradients
- Residual connections help in training very deep networks



$$y(x) = f(x) + x$$

Residual connection adds a direct shortcut for gradients



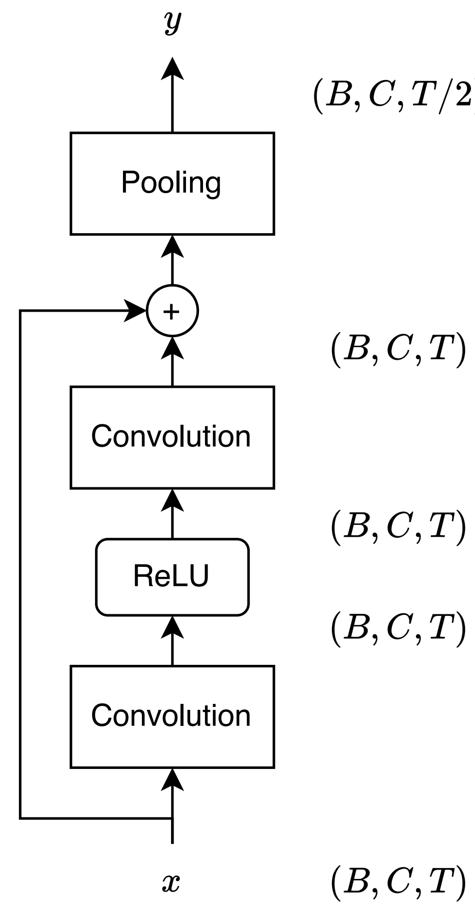
$$y(x) = f(x) + x$$

$$\frac{\partial y(x)}{\partial x} = \frac{\partial f(x)}{\partial x} + 1$$

$$\frac{\partial L}{\partial x} = \frac{\partial y(x)}{\partial x}\frac{L}{\partial y} = \frac{\partial f(x)}{\partial x}\frac{\partial L}{\partial y} + \frac{\partial L}{\partial y}$$

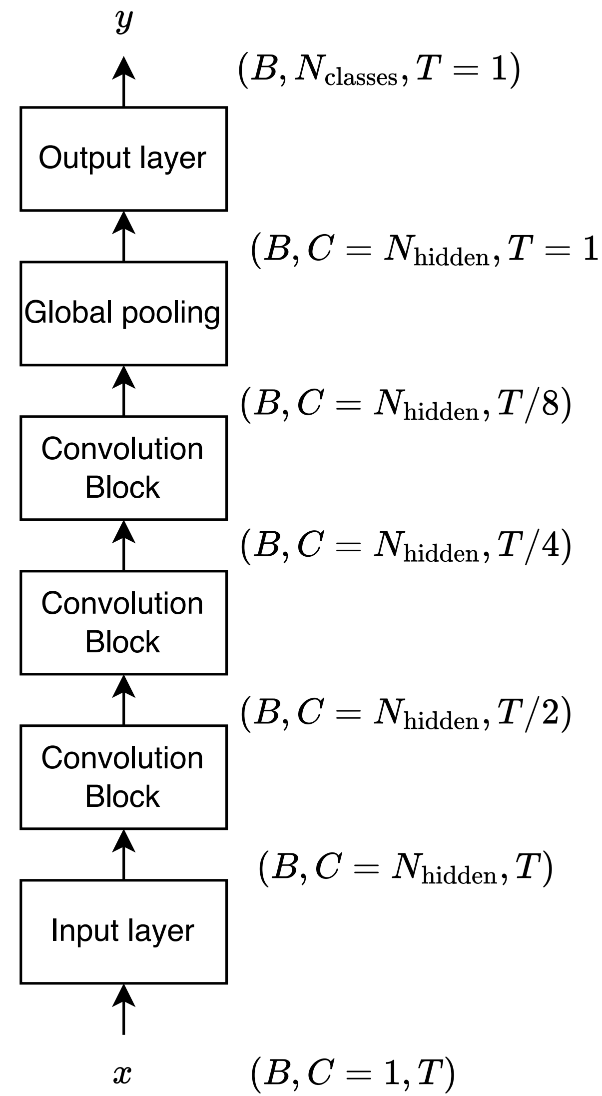
Convolution Block

- **Typical convolution layers (aka Blocks) contain**
    - Convolutions
    - Activations (ReLU)
    - Residual connections
    - Pooling (Max or Avg.)



CNN classifier model

- **Input layer embeds the data to hidden dimension**
- **Convolution layers learn representations and gradually downsample the input**
- **Global pooling deals with whatever sequence length remains**
- **Output layer projects to number of classes**



End of Lecture 3

- Questions?