Convolutional Neural Networks

F. Noé[1]

Deep Learning Classes, FU Berlin 2018

ConvNets
LeCun 1989

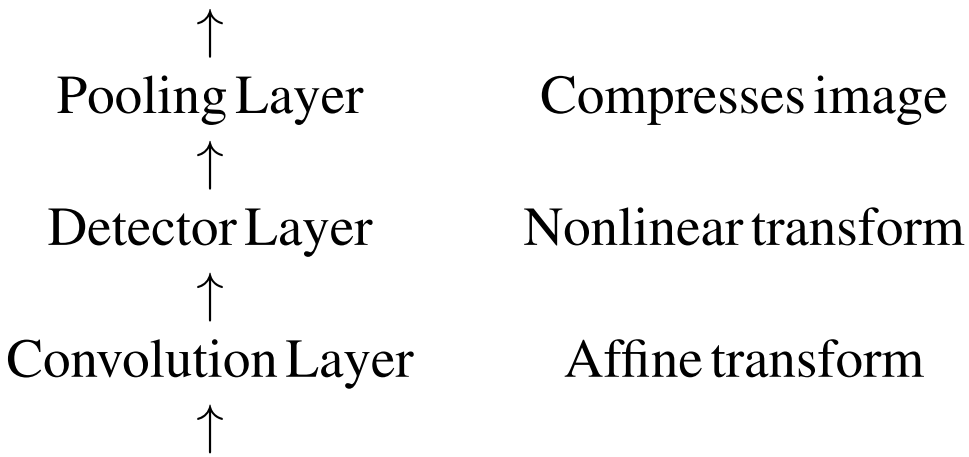
- Convolutional neural networks (ConvNets, CNNs), have a special network architecture that is suitable for exploiting invariances, e.g. translational invariance.
- Traditional CNNs are suited for data with a grid-like topology, e.g.: discretized time-series such as audio (1D), pixelated images (2D)
- In neural networks, convolutions are typically used in conjunction with a nonlinear transform (detector layer) and a pooling layer:

ConvNets
LeCun 1989

- Convolutional neural networks (ConvNets, CNNs), have a special network architecture that is suitable for exploiting invariances, e.g. translational invariance.
- Traditional CNNs are suited for data with a grid-like topology, e.g.: discretized time-series such as audio (1D), pixelated images (2D)
- In neural networks, convolutions are typically used in conjunction with a nonlinear transform (detector layer) and a pooling layer:

ConvNets
LeCun 1989

- Convolutional neural networks (ConvNets, CNNs), have a special network architecture that is suitable for exploiting invariances, e.g. translational invariance.
- Traditional CNNs are suited for data with a grid-like topology, e.g.: discretized time-series such as audio (1D), pixelated images (2D)
- In neural networks, convolutions are typically used in conjunction with a nonlinear transform (detector layer) and a pooling layer:



- Mathematical basis: convolution operation

$$y(b) = (x * w)(b) = \int_{a=-\infty}^{\infty} x(a)w(b-a)\,\mathrm{d}a$$

- Discrete convolution:

$$y_i = (\mathbf{x} * \mathbf{w})_i = \sum_j x_j w_{i-j} = \sum_j w_j x_{i-j}$$

  Here we have used that convolution is commutative:

$$x * w = w * x$$

- Here we call:
  - **x** input
  - **w** kernel
  - **y** output or feature map

- Mathematical basis: convolution operation

$$y(b) = (x * w)(b) = \int_{a=-\infty}^{\infty} x(a)w(b-a)\,\mathrm{d}a$$

- Discrete convolution:

$$y_i = (\mathbf{x} * \mathbf{w})_i = \sum_j x_j w_{i-j} = \sum_j w_j x_{i-j}$$

  Here we have used that convolution is commutative:

$$x * w = w * x$$

- Here we call:
  - **x** input
  - **w** kernel
  - **y** output or feature map

- Mathematical basis: convolution operation

$$y(b) = (x * w)(b) = \int_{a=-\infty}^{\infty} x(a)w(b-a)\,\mathrm{d}a$$

- Discrete convolution:

$$y_i = (\mathbf{x} * \mathbf{w})_i = \sum_j x_j w_{i-j} = \sum_j w_j x_{i-j}$$

  Here we have used that convolution is commutative:

$$x * w = w * x$$

- Here we call:
  - **x** input
  - **w** kernel
  - **y** output or feature map

- Convolution is a linear operation. To see that, let us use $\mathbf{x} \in \mathbb{R}^5$ and $\mathbf{w} \in \mathbb{R}^3$ as an example. As the j-sum can only run from 2 to 4, we get three equations for $\mathbf{y}$:

$$y_2 = w_1 x_1 + w_2 x_2 + w_3 x_3 = (w_1, w_2, w_3, 0, 0)^\top \mathbf{x}$$
$$y_3 = w_2 x_2 + w_3 x_3 + w_4 x_4 = (0, w_1, w_2, w_3, 0)^\top \mathbf{x}$$
$$y_4 = w_3 x_3 + w_4 x_4 + w_5 x_5 = (0, 0, w_1, w_2, w_3)^\top \mathbf{x}$$

- Convolving $\mathbf{x} \in \mathbb{R}^n$ with $\mathbf{w} \in \mathbb{R}^m$, $m \leq n$ can be written as linear operation

$$\mathbf{y} = \mathbf{W}\mathbf{x}$$

with $\mathbf{W} \in \mathbb{R}^{n-m+1 \times n}$ being a **Toeplitz matrix**:

$$\mathbf{W} = \begin{pmatrix} w_1 & \cdots & w_m & & \\ & \ddots & & \ddots & \\ & & w_1 & \cdots & w_m \end{pmatrix}$$

Here we have assumed that the index j in the convolution cannot go outside the indices of the input $\mathbf{x}$. As a result, the convolved output $\mathbf{y}$ will have a reduced dimension $n-m+1$. In practice, we often use *zero padding*.

- Convolution is a linear operation. To see that, let us use $\mathbf{x} \in \mathbb{R}^5$ and $\mathbf{w} \in \mathbb{R}^3$ as an example. As the j-sum can only run from 2 to 4, we get three equations for $\mathbf{y}$:

$$y_2 = w_1 x_1 + w_2 x_2 + w_3 x_3 = (w_1, w_2, w_3, 0, 0)^\top \mathbf{x}$$
$$y_3 = w_2 x_2 + w_3 x_3 + w_4 x_4 = (0, w_1, w_2, w_3, 0)^\top \mathbf{x}$$
$$y_4 = w_3 x_3 + w_4 x_4 + w_5 x_5 = (0, 0, w_1, w_2, w_3)^\top \mathbf{x}$$

- Convolving $\mathbf{x} \in \mathbb{R}^n$ with $\mathbf{w} \in \mathbb{R}^m$, $m \leq n$ can be written as linear operation

$$\mathbf{y} = \mathbf{W}\mathbf{x}$$

with $\mathbf{W} \in \mathbb{R}^{n-m+1 \times n}$ being a **Toeplitz matrix**:

$$\mathbf{W} = \begin{pmatrix} w_1 & \cdots & w_m & & \\ & \ddots & & \ddots & \\ & & w_1 & \cdots & w_m \end{pmatrix}$$

Here we have assumed that the index j in the convolution cannot go outside the indices of the input $\mathbf{x}$. As a result, the convolved output $\mathbf{y}$ will have a reduced dimension $n-m+1$. In practice, we often use *zero padding*.

ConvNets

- Multidimensional convolution, e.g. 2D:

$$Y_{ij} = (K * X)_{ij} = \sum_m \sum_n X_{i-m,j-n} K_{m,n}$$

- Many ML libraries use the cross-convolution function, which is almost identical to convolution, just with flipped indices:

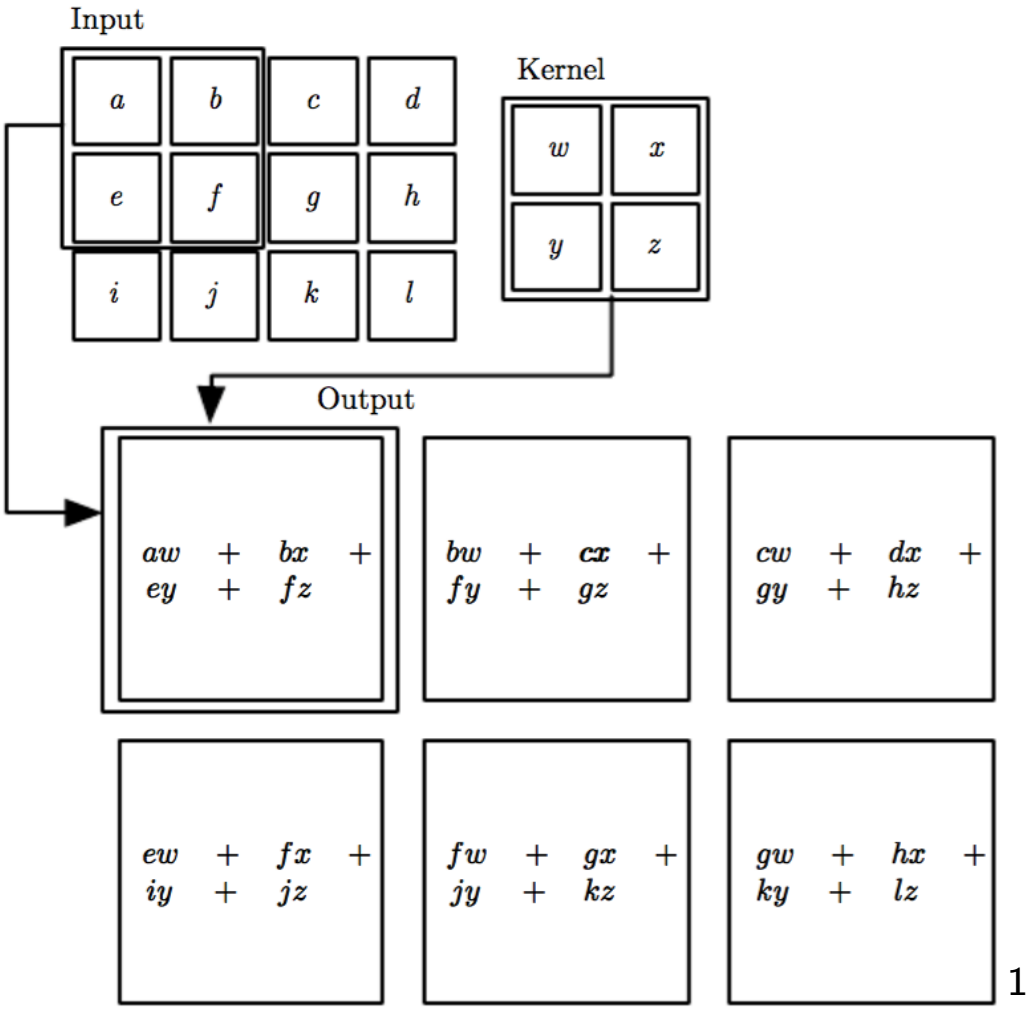
$$Y_{ij} = (X * K)_{ij} = \sum_m \sum_n X_{i+m,j+n} K_{m,n}$$

- Multidimensional convolution, e.g. 2D:

$$Y_{ij} = (K * X)_{ij} = \sum_m \sum_n X_{i-m,j-n} K_{m,n}$$

- Many ML libraries use the cross-convolution function, which is almost identical to convolution, just with flipped indices:

$$Y_{ij} = (X * K)_{ij} = \sum_m \sum_n X_{i+m,j+n} K_{m,n}$$

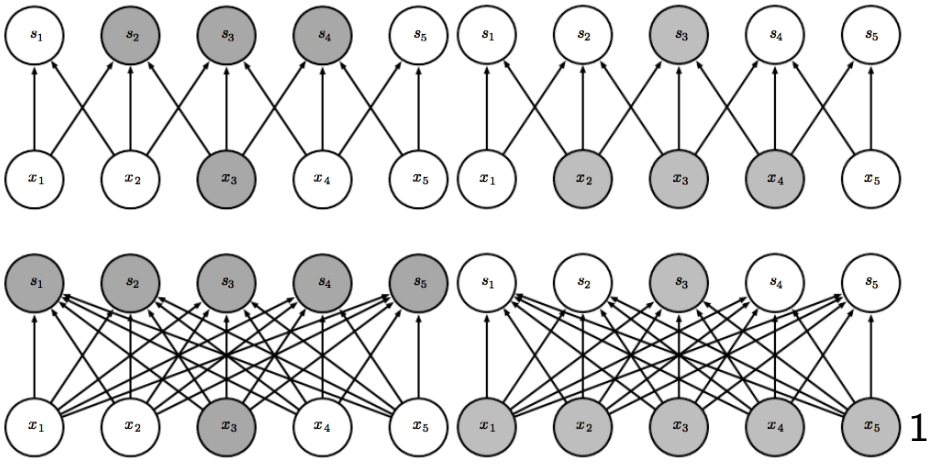


[1]

[1] From Goodfellow, Bengio and Courville: Deep Learning, MIT Press (2016)

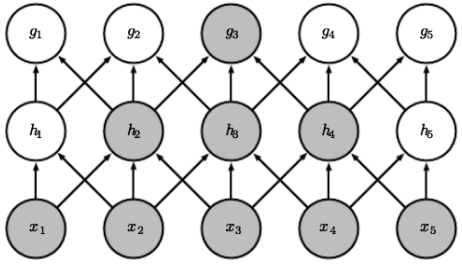
ConvNets

Motivation: **Sparse interactions**, parameter sharing, equivariance

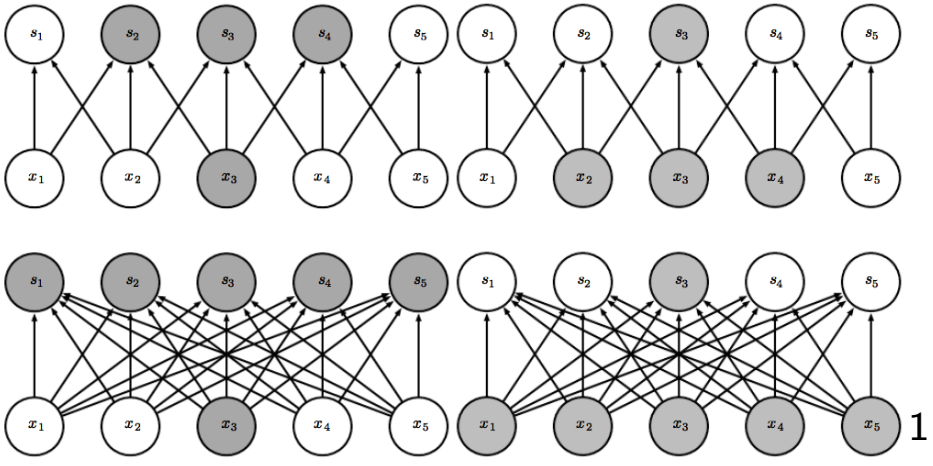


1

- **Sparse connectivity** – makes computations faster
- Each input dim. only affects some output dims.
- **Limited receptive field**: Each output only depends on some inputs.
- In deep CNNS, units of later layers can still receive the entire input.

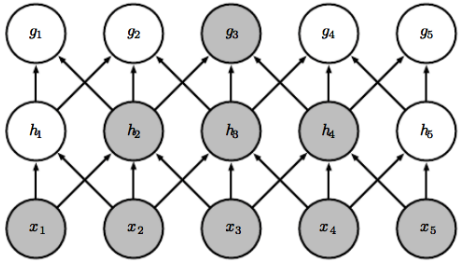


[1] From Goodfellow, Bengio and Courville: Deep Learning, MIT Press (2016)



[1]

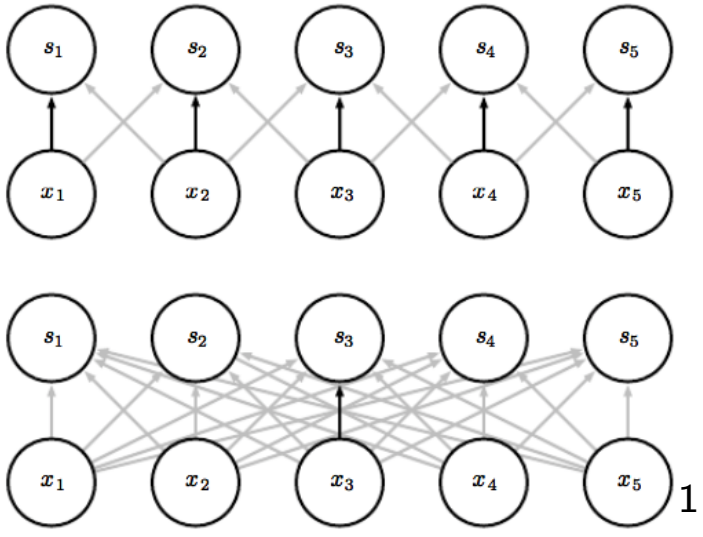
- **Sparse connectivity** – makes computations faster
- Each input dim. only affects some output dims.
- **Limited receptive field**: Each output only depends on some inputs.
- In deep CNNS, units of later layers can still receive the entire input.



[1] From Goodfellow, Bengio and Courville: Deep Learning, MIT Press (2016)

ConvNets

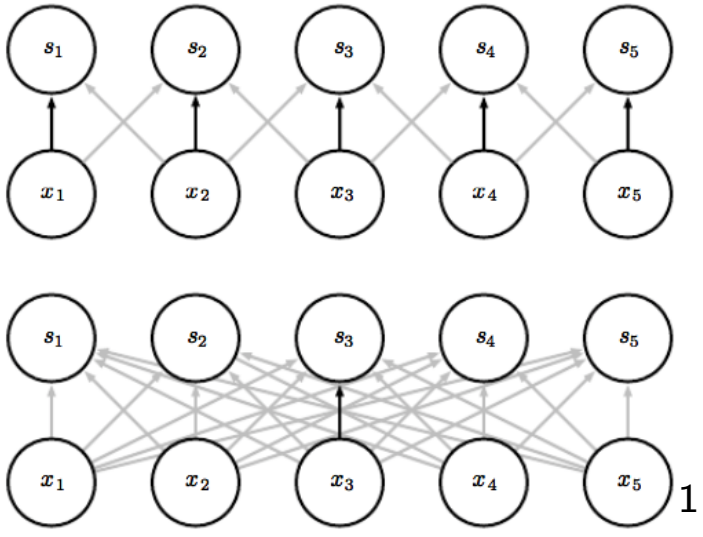
Motivation: Sparse interactions, **parameter sharing**, equivariance



[1]

- **Conv Layer**: Same parameters are used everywhere in the image. Reduces storage requirements and training problem.
- **Dense Layer**: No parameter sharing, many more degrees of freedom.
- Essential for image processing. Example: Input 512×512 image has 262144 pixels. A dense layer to an output of similar size would have 68×10^9 parameters.

[1] From Goodfellow, Bengio and Courville: Deep Learning, MIT Press (2016)

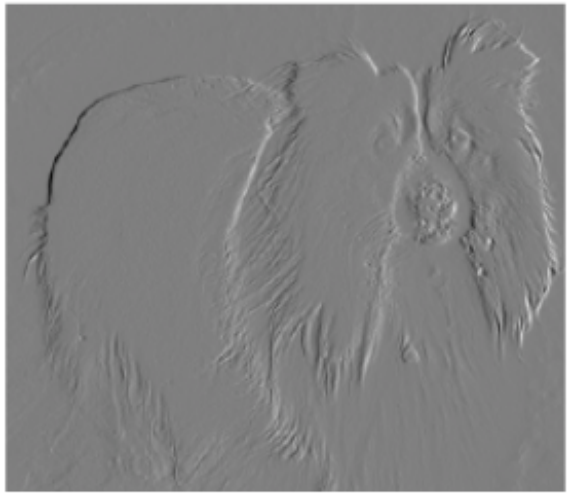


[1]

- **Conv Layer**: Same parameters are used everywhere in the image. Reduces storage requirements and training problem.
- **Dense Layer**: No parameter sharing, many more degrees of freedom.
- Essential for image processing. Example: Input 512×512 image has 262144 pixels. A dense layer to an output of similar size would have 68×10^9 parameters.

[1] From Goodfellow, Bengio and Courville: Deep Learning, MIT Press (2016)

Edge detection using $Y = ([-1\ 1] * X)$

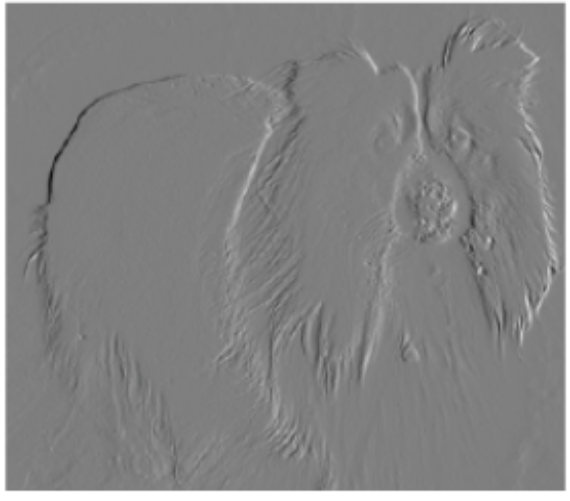


[1]

- **Conv Layer**:
  - Kernel K has 2 elements
  - Requires $319 \times 280 \times 3 = 267{,}960$ FLOPs (2 mult. + 1 add. per output)
- **Dense Layer**:
  - $320 \times 280 \times 319 \times 280 = 8.028 \times 10^9$ matrix entries
  - 15.056×10^9 FLOPs

[1] From Goodfellow, Bengio and Courville: Deep Learning, MIT Press (2016)

Edge detection using $Y = ([-1\ 1] * X)$

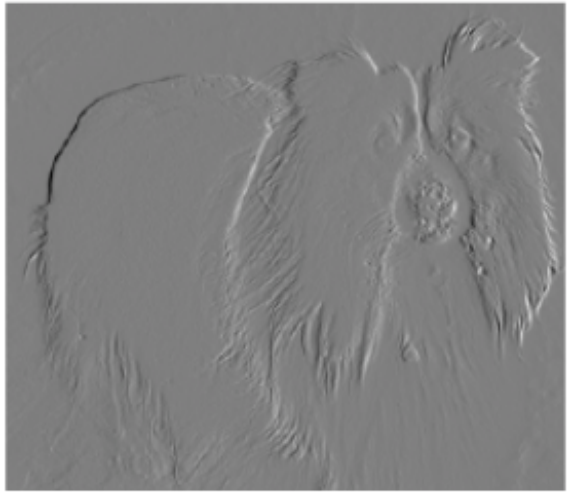


[1]

- **Conv Layer**:
  - Kernel K has 2 elements
  - Requires $319 \times 280 \times 3 = 267,960$ FLOPs (2 mult. + 1 add. per output)
- **Dense Layer**:
  - $320 \times 280 \times 319 \times 280 = 8.028 \times 10^9$ matrix entries
  - 15.056×10^9 FLOPs

[1] From Goodfellow, Bengio and Courville: Deep Learning, MIT Press (2016)

Edge detection using $Y = ([-1\ 1] * X)$



[1]

- **Conv Layer**:
  - Kernel K has 2 elements
  - Requires $319 \times 280 \times 3 = 267,960$ FLOPs (2 mult. + 1 add. per output)
- **Dense Layer**:
  - $320 \times 280 \times 319 \times 280 = 8.028 \times 10^9$ matrix entries
  - 15.056×10^9 FLOPs

[1] From Goodfellow, Bengio and Courville: Deep Learning, MIT Press (2016)

- Function f is equivariant to a function g if $f(g(x)) = g(f(x))$ - if the input changes, the output changes in the same way

$$\begin{array}{ccc} X & \xrightarrow{g} & X' \\ f\downarrow & & f\downarrow \\ Y & \xrightarrow{g} & Y' \end{array}$$

  - f is convolution with kernel $K - f : Y = K * X$
  - g is translation: $X' = g(X)$, $X'_{ij} = X_{i-u,j-v}$.
  - Equivariance: $g(K * X) = K * g(X)$
- **Note**: Equivariance does not automatically hold on the image boundary. Often one performs **zero-padding** in order to obtain equivariance everywhere.
- Convolution creates a 2-D map of where certain features appear in the input. If we move the object in the input, its representation will move the same amount in the output.
- Convolution is not intrinsically equivariant to other transformations, e.g. rotation or scaling.

- Function f is equivariant to a function g if $f(g(x)) = g(f(x))$ - if the input changes, the output changes in the same way

$$\begin{array}{ccc} X & \xrightarrow{g} & X' \\ f\downarrow & & f\downarrow \\ Y & \xrightarrow{g} & Y' \end{array}$$

  - f is convolution with kernel $K - f : Y = K * X$
  - g is translation: $X' = g(X)$, $X'_{ij} = X_{i-u,j-v}$.
  - Equivariance: $g(K * X) = K * g(X)$
- **Note**: Equivariance does not automatically hold on the image boundary. Often one performs **zero-padding** in order to obtain equivariance everywhere.
- Convolution creates a 2-D map of where certain features appear in the input. If we move the object in the input, its representation will move the same amount in the output.
- Convolution is not intrinsically equivariant to other transformations, e.g. rotation or scaling.

- Function f is equivariant to a function g if $f(g(x)) = g(f(x))$ - if the input changes, the output changes in the same way

$$\begin{array}{ccc} X & \xrightarrow{g} & X' \\ f\downarrow & & f\downarrow \\ Y & \xrightarrow{g} & Y' \end{array}$$

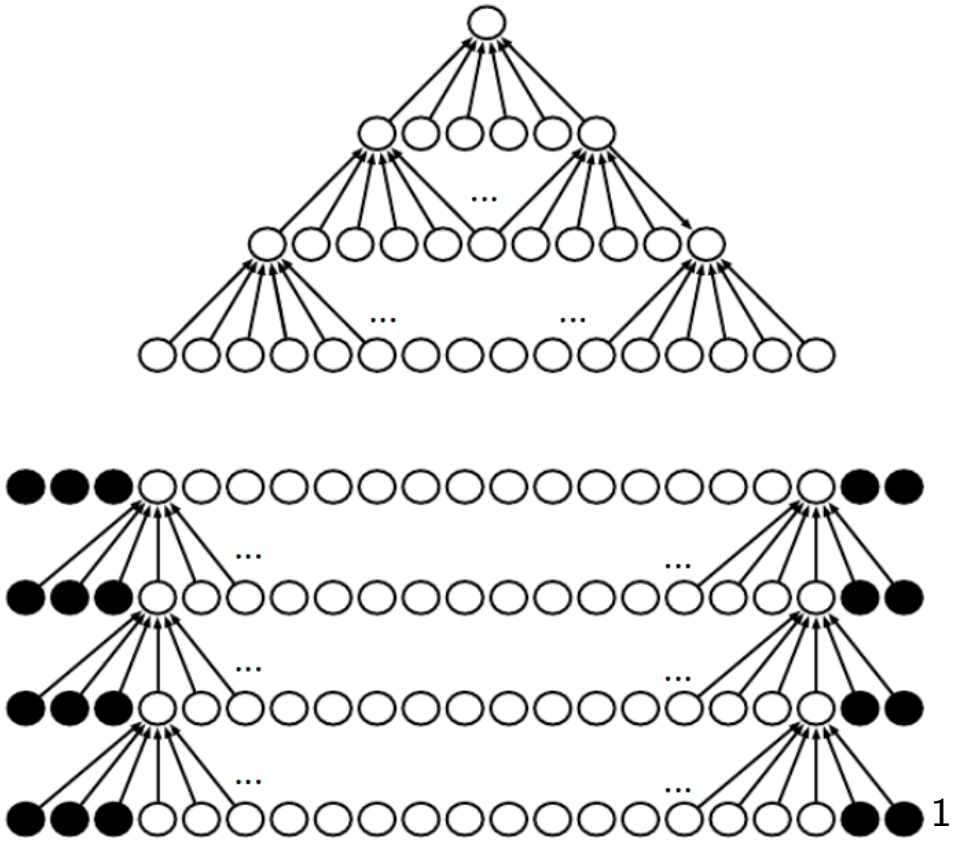
  - f is convolution with kernel $K - f : Y = K * X$
  - g is translation: $X' = g(X)$, $X'_{ij} = X_{i-u,j-v}$.
  - Equivariance: $g(K * X) = K * g(X)$
- **Note**: Equivariance does not automatically hold on the image boundary. Often one performs **zero-padding** in order to obtain equivariance everywhere.
- Convolution creates a 2-D map of where certain features appear in the input. If we move the object in the input, its representation will move the same amount in the output.
- Convolution is not intrinsically equivariant to other transformations, e.g. rotation or scaling.

- Function f is equivariant to a function g if $f(g(x)) = g(f(x))$ - if the input changes, the output changes in the same way

$$\begin{array}{ccc} X & \xrightarrow{g} & X' \\ f\downarrow & & f\downarrow \\ Y & \xrightarrow{g} & Y' \end{array}$$

  - f is convolution with kernel $K - f : Y = K * X$
  - g is translation: $X' = g(X)$, $X'_{ij} = X_{i-u,j-v}$.
  - Equivariance: $g(K * X) = K * g(X)$
- **Note**: Equivariance does not automatically hold on the image boundary. Often one performs **zero-padding** in order to obtain equivariance everywhere.
- Convolution creates a 2-D map of where certain features appear in the input. If we move the object in the input, its representation will move the same amount in the output.
- Convolution is not intrinsically equivariant to other transformations, e.g. rotation or scaling.

Zero padding



- Top: without zero padding, image size is reduced upon convolution.
- Bottom: with zero padding, image size stays constant.

[1] From Goodfellow, Bengio and Courville: Deep Learning, MIT Press (2016)

Zero padding

- **without zero padding**:
  - image size is reduced upon convolution.
  - Often called **valid** convolution in algebra packages
  - We are forced to choose between shrinking the spatial extent of the network rapidly or using small kernels—both scenarios that significantly limit the expressive power of the network.
- **with zero padding**:
  - image size stays constant. Arbitrarily deep convolutional networks can be used.
  - Allows us to control the kernel width and the size of the output independently.
  - **same** convolution: enough zeros are padded to make output size equal to input. Input pixels near the border influence fewer output pixels than the input pixels near the center, can result in underrespresentation of border pixels.
  - **full** convolution: enough zeroes are added for every pixel to be visited k times in each direction, resulting in an output image of width $m+k-1$. In this case, the output pixels near the border are a function of fewer pixels than the output pixels near the center.

Zero padding

- **without zero padding**:
  - image size is reduced upon convolution.
  - Often called **valid** convolution in algebra packages
  - We are forced to choose between shrinking the spatial extent of the network rapidly or using small kernels—both scenarios that significantly limit the expressive power of the network.
- **with zero padding**:
  - image size stays constant. Arbitrarily deep convolutional networks can be used.
  - Allows us to control the kernel width and the size of the output independently.
  - **same** convolution: enough zeros are padded to make output size equal to input. Input pixels near the border influence fewer output pixels than the input pixels near the center, can result in underrespresentation of border pixels.
  - **full** convolution: enough zeroes are added for every pixel to be visited k times in each direction, resulting in an output image of width $m+k-1$. In this case, the output pixels near the border are a function of fewer pixels than the output pixels near the center.

Zero padding

- **without zero padding**:
  - image size is reduced upon convolution.
  - Often called **valid** convolution in algebra packages
  - We are forced to choose between shrinking the spatial extent of the network rapidly or using small kernels—both scenarios that significantly limit the expressive power of the network.
- **with zero padding**:
  - image size stays constant. Arbitrarily deep convolutional networks can be used.
  - Allows us to control the kernel width and the size of the output independently.
  - **same** convolution: enough zeros are padded to make output size equal to input. Input pixels near the border influence fewer output pixels than the input pixels near the center, can result in underrespresentation of border pixels.
  - **full** convolution: enough zeroes are added for every pixel to be visited k times in each direction, resulting in an output image of width $m+k-1$. In this case, the output pixels near the border are a function of fewer pixels than the output pixels near the center.

Zero padding

- **without zero padding**:
  - image size is reduced upon convolution.
  - Often called **valid** convolution in algebra packages
  - We are forced to choose between shrinking the spatial extent of the network rapidly or using small kernels—both scenarios that significantly limit the expressive power of the network.
- **with zero padding**:
  - image size stays constant. Arbitrarily deep convolutional networks can be used.
  - Allows us to control the kernel width and the size of the output independently.
  - **same** convolution: enough zeros are padded to make output size equal to input. Input pixels near the border influence fewer output pixels than the input pixels near the center, can result in underrespresentation of border pixels.
  - **full** convolution: enough zeroes are added for every pixel to be visited k times in each direction, resulting in an output image of width $m+k-1$. In this case, the output pixels near the border are a function of fewer pixels than the output pixels near the center.

Zero padding

- **without zero padding**:
  - image size is reduced upon convolution.
  - Often called **valid** convolution in algebra packages
  - We are forced to choose between shrinking the spatial extent of the network rapidly or using small kernels—both scenarios that significantly limit the expressive power of the network.
- **with zero padding**:
  - image size stays constant. Arbitrarily deep convolutional networks can be used.
  - Allows us to control the kernel width and the size of the output independently.
  - **same** convolution: enough zeros are padded to make output size equal to input. Input pixels near the border influence fewer output pixels than the input pixels near the center, can result in underrespresentation of border pixels.
  - **full** convolution: enough zeroes are added for every pixel to be visited k times in each direction, resulting in an output image of width $m+k-1$. In this case, the output pixels near the border are a function of fewer pixels than the output pixels near the center.

Zero padding

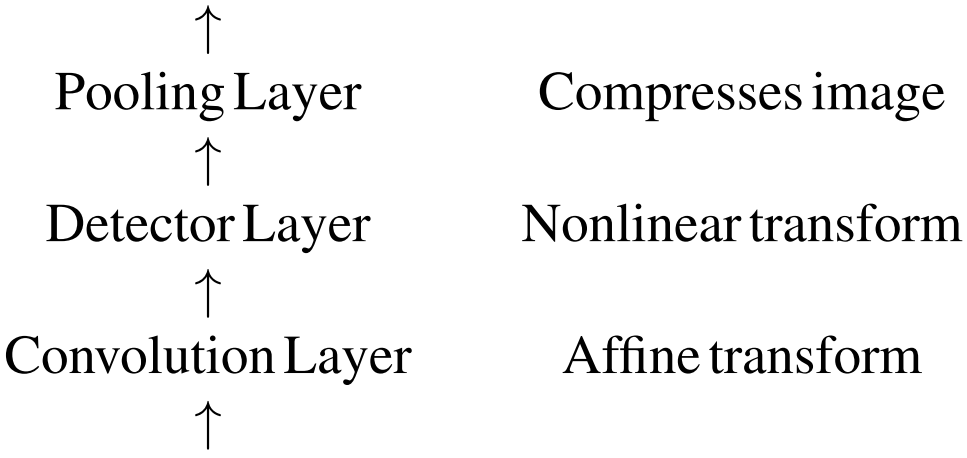
- **without zero padding**:
  - image size is reduced upon convolution.
  - Often called **valid** convolution in algebra packages
  - We are forced to choose between shrinking the spatial extent of the network rapidly or using small kernels—both scenarios that significantly limit the expressive power of the network.
- **with zero padding**:
  - image size stays constant. Arbitrarily deep convolutional networks can be used.
  - Allows us to control the kernel width and the size of the output independently.
  - **same** convolution: enough zeros are padded to make output size equal to input. Input pixels near the border influence fewer output pixels than the input pixels near the center, can result in underrespresentation of border pixels.
  - **full** convolution: enough zeroes are added for every pixel to be visited k times in each direction, resulting in an output image of width $m+k-1$. In this case, the output pixels near the border are a function of fewer pixels than the output pixels near the center.

Zero padding

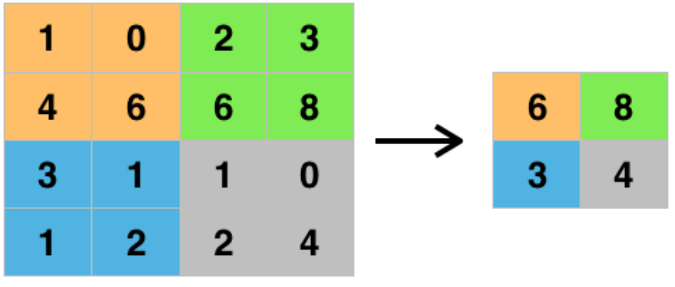
- **without zero padding**:
  - image size is reduced upon convolution.
  - Often called **valid** convolution in algebra packages
  - We are forced to choose between shrinking the spatial extent of the network rapidly or using small kernels—both scenarios that significantly limit the expressive power of the network.
- **with zero padding**:
  - image size stays constant. Arbitrarily deep convolutional networks can be used.
  - Allows us to control the kernel width and the size of the output independently.
  - **same** convolution: enough zeros are padded to make output size equal to input. Input pixels near the border influence fewer output pixels than the input pixels near the center, can result in underrespresentation of border pixels.
  - **full** convolution: enough zeroes are added for every pixel to be visited k times in each direction, resulting in an output image of width $m+k-1$. In this case, the output pixels near the border are a function of fewer pixels than the output pixels near the center.

Pooling

- Reminder:



- **Examples**:
  - **max pooling** of a rectangular neighborhood (Zhou and Chellappa, 1988), e.g. 2×2 *filter with stride 2*:



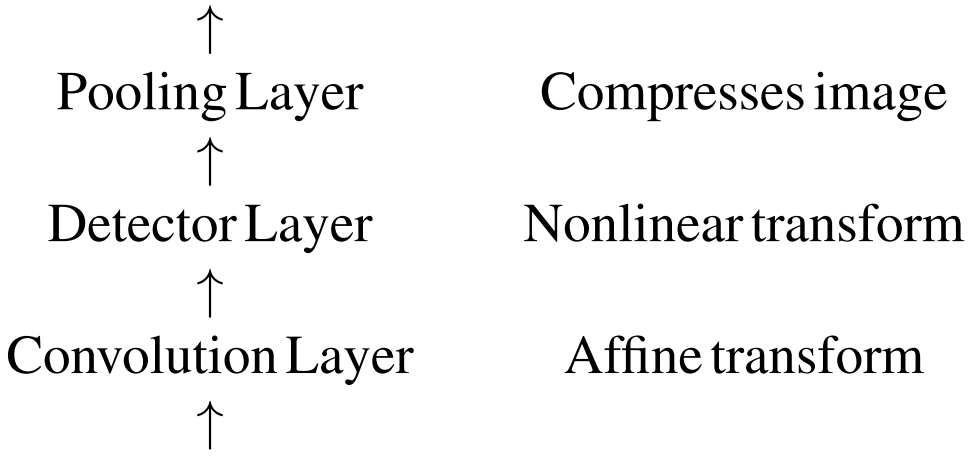
[1]

  - average of a rectangular neighborhood
  - L2 norm of a rectangular neighborhood
  - weighted average based on the distance from the central pixel.

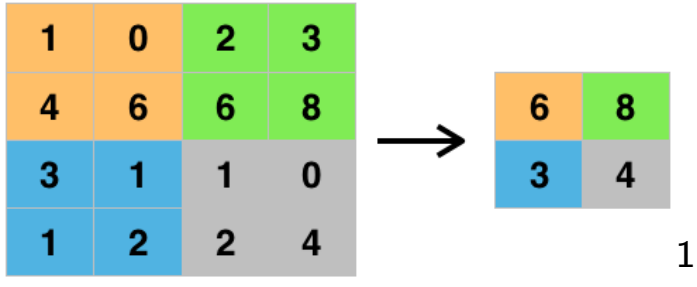
[1] From Goodfellow, Bengio and Courville: Deep Learning, MIT Press (2016)

Pooling

- Reminder:



- **Examples**:
  - **max pooling** of a rectangular neighborhood (Zhou and Chellappa, 1988), e.g. 2×2 *filter with stride 2*:



[1]

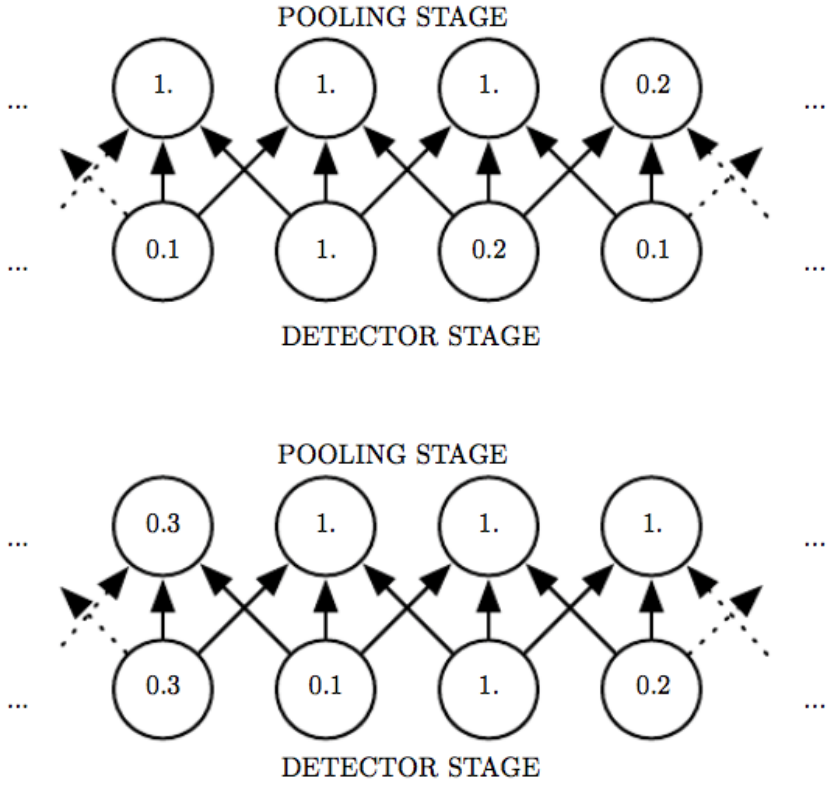
  - average of a rectangular neighborhood
  - L2 norm of a rectangular neighborhood
  - weighted average based on the distance from the central pixel.

[1] From Goodfellow, Bengio and Courville: Deep Learning, MIT Press (2016)

Pooling

- Using $\text{stride} > 1$ significantly reduces the size of the output $\rightarrow$ reduce memory and CPU requirements.
- Pooling useful for handling inputs of different sizes. Offsets can be varied such that the classification layer always receives the same number of summary statistics regardless of the input size.
- Example: final pooling layer of the network may be defined to output four sets of summary statistics, one for each quadrant of an image, regardless of the image size).
- Pooling over spatial regions makes the representation *approximately* invariant to small translations at the input (most pixels of Y do not change upon small translations of X) $\rightarrow$ useful if we want to detect whether a feature is present rather where it is.

Pooling



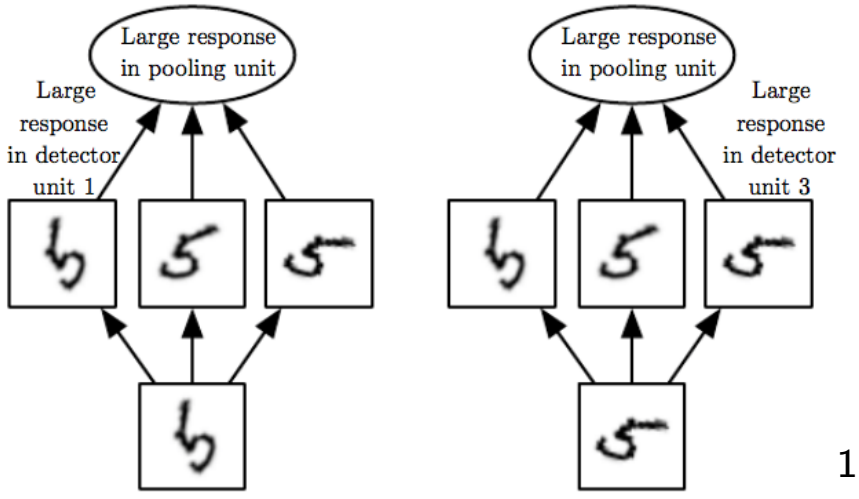
[1]

Example: each input pixel has changed, but only half of the output pixels have changed.

[1] From Goodfellow, Bengio and Courville: Deep Learning, MIT Press (2016)

Pooling

By pooling over separate convolution, the features can learn which transformations to become invariant to:



[1]

- Which pooling should I use? (Boureau et al., 2010).
- Dynamical pooling (Boureau et al., 2011).
- Adaptive pooling (Jia et al., 2012).

[1] From Goodfellow, Bengio and Courville: Deep Learning, MIT Press (2016)

Convolution as infinitely strong prior

- Reminder: **Prior distribution** – probability distribution over model parameters encoding assessment of likely models before having seen any data.
- **Weak prior**: high entropy, e.g. Gaussian with high variance allows the data to move the parameters more or less freely.
- **Strong prior**: low entropy, e.g. Gaussian with low variance. More strongly restricts the parameter values.
- **Infinitely strong prior**: places zero probability on some parameter values.
- **Convolutional layer**: similar to densely connected layer, but:
  - weights for one hidden unit must be identical to the weights of its neighbor, but shifted in space
  - enforcing that with zero probability on parameters outside receptive field.

Convolution as infinitely strong prior

- Reminder: **Prior distribution** – probability distribution over model parameters encoding assessment of likely models before having seen any data.
- **Weak prior**: high entropy, e.g. Gaussian with high variance allows the data to move the parameters more or less freely.
- **Strong prior**: low entropy, e.g. Gaussian with low variance. More strongly restricts the parameter values.
- **Infinitely strong prior**: places zero probability on some parameter values.
- **Convolutional layer**: similar to densely connected layer, but:
  - weights for one hidden unit must be identical to the weights of its neighbor, but shifted in space
  - enforcing that with zero probability on parameters outside receptive field.

Convolution as infinitely strong prior

- Reminder: **Prior distribution** – probability distribution over model parameters encoding assessment of likely models before having seen any data.
- **Weak prior**: high entropy, e.g. Gaussian with high variance allows the data to move the parameters more or less freely.
- **Strong prior**: low entropy, e.g. Gaussian with low variance. More strongly restricts the parameter values.
- **Infinitely strong prior**: places zero probability on some parameter values.
- **Convolutional layer**: similar to densely connected layer, but:
  - weights for one hidden unit must be identical to the weights of its neighbor, but shifted in space
  - enforcing that with zero probability on parameters outside receptive field.

Convolution as infinitely strong prior

- Reminder: **Prior distribution** – probability distribution over model parameters encoding assessment of likely models before having seen any data.
- **Weak prior**: high entropy, e.g. Gaussian with high variance allows the data to move the parameters more or less freely.
- **Strong prior**: low entropy, e.g. Gaussian with low variance. More strongly restricts the parameter values.
- **Infinitely strong prior**: places zero probability on some parameter values.
- **Convolutional layer**: similar to densely connected layer, but:
  - weights for one hidden unit must be identical to the weights of its neighbor, but shifted in space
  - enforcing that with zero probability on parameters outside receptive field.

Convolution as infinitely strong prior

- Reminder: **Prior distribution** – probability distribution over model parameters encoding assessment of likely models before having seen any data.
- **Weak prior**: high entropy, e.g. Gaussian with high variance allows the data to move the parameters more or less freely.
- **Strong prior**: low entropy, e.g. Gaussian with low variance. More strongly restricts the parameter values.
- **Infinitely strong prior**: places zero probability on some parameter values.
- **Convolutional layer**: similar to densely connected layer, but:
  - weights for one hidden unit must be identical to the weights of its neighbor, but shifted in space
  - enforcing that with zero probability on parameters outside receptive field.

Multi-channel convolution

- Input has usually multiple channels (e.g. RGB for images).
- Images: input X and output Y are $3d$ tensors $(\text{channel} \times i\,\text{pos} \times j\,\text{pos})$.
- Software: usually use batches and $4d$ tensors $(\text{batch} - \text{index} \times \text{channel} \times i\,\text{pos} \times j\,\text{pos})$
- For multi-channel convolution, linear operations are not guaranteed to be commutative, unless each operation has the same number of input and output channels.
- Multi-channel convolution: $4d$ Kernel $\mathbf{K}$ with $K_{i,j,k,l}$: connection strength between a unit in channel i of output and a unit in channel j of input, with offset of k rows and l columns between the output unit and the input unit.

$$Y_{i,j,k} = \sum_{l,m,n} X_{l,j+m-1,k+n-1} K_{i,l,m,n}$$

  where l, m, n sums over all valid indices.
- To reduce computational cost, it is possible to use a *stride* in the convolution operation.

Multi-channel convolution

- Input has usually multiple channels (e.g. RGB for images).
- Images: input X and output Y are $3d$ tensors $(\text{channel} \times i\,\text{pos} \times j\,\text{pos})$.
- Software: usually use batches and $4d$ tensors $(\text{batch} - \text{index} \times \text{channel} \times i\,\text{pos} \times j\,\text{pos})$
- For multi-channel convolution, linear operations are not guaranteed to be commutative, unless each operation has the same number of input and output channels.
- Multi-channel convolution: $4d$ Kernel $\mathbf{K}$ with $K_{i,j,k,l}$: connection strength between a unit in channel i of output and a unit in channel j of input, with offset of k rows and l columns between the output unit and the input unit.

$$Y_{i,j,k} = \sum_{l,m,n} X_{l,j+m-1,k+n-1} K_{i,l,m,n}$$

  where l, m, n sums over all valid indices.
- To reduce computational cost, it is possible to use a *stride* in the convolution operation.

Multi-channel convolution

- Input has usually multiple channels (e.g. RGB for images).
- Images: input X and output Y are $3d$ tensors $(\text{channel} \times i\,\text{pos} \times j\,\text{pos})$.
- Software: usually use batches and $4d$ tensors $(\text{batch} - \text{index} \times \text{channel} \times i\,\text{pos} \times j\,\text{pos})$
- For multi-channel convolution, linear operations are not guaranteed to be commutative, unless each operation has the same number of input and output channels.
- Multi-channel convolution: $4d$ Kernel $\mathbf{K}$ with $K_{i,j,k,l}$: connection strength between a unit in channel i of output and a unit in channel j of input, with offset of k rows and l columns between the output unit and the input unit.

$$Y_{i,j,k} = \sum_{l,m,n} X_{l,j+m-1,k+n-1} K_{i,l,m,n}$$

  where l, m, n sums over all valid indices.
- To reduce computational cost, it is possible to use a *stride* in the convolution operation.

Multi-channel convolution

- Input has usually multiple channels (e.g. RGB for images).
- Images: input X and output Y are $3d$ tensors $(\text{channel} \times i\,\text{pos} \times j\,\text{pos})$.
- Software: usually use batches and $4d$ tensors $(\text{batch} - \text{index} \times \text{channel} \times i\,\text{pos} \times j\,\text{pos})$
- For multi-channel convolution, linear operations are not guaranteed to be commutative, unless each operation has the same number of input and output channels.
- Multi-channel convolution: $4d$ Kernel $\mathbf{K}$ with $K_{i,j,k,l}$: connection strength between a unit in channel i of output and a unit in channel j of input, with offset of k rows and l columns between the output unit and the input unit.

$$Y_{i,j,k} = \sum_{l,m,n} X_{l,j+m-1,k+n-1} K_{i,l,m,n}$$

  where l, m, n sums over all valid indices.
- To reduce computational cost, it is possible to use a *stride* in the convolution operation.

Multi-channel convolution

- Input has usually multiple channels (e.g. RGB for images).
- Images: input X and output Y are $3d$ tensors $(\text{channel} \times i\,\text{pos} \times j\,\text{pos})$.
- Software: usually use batches and $4d$ tensors $(\text{batch} - \text{index} \times \text{channel} \times i\,\text{pos} \times j\,\text{pos})$
- For multi-channel convolution, linear operations are not guaranteed to be commutative, unless each operation has the same number of input and output channels.
- Multi-channel convolution: $4d$ Kernel $\mathbf{K}$ with $K_{i,j,k,l}$: connection strength between a unit in channel i of output and a unit in channel j of input, with offset of k rows and l columns between the output unit and the input unit.

$$Y_{i,j,k} = \sum_{l,m,n} X_{l,j+m-1,k+n-1} K_{i,l,m,n}$$

where l, m, n sums over all valid indices.

- To reduce computational cost, it is possible to use a *stride* in the convolution operation.

Multi-channel convolution

- Input has usually multiple channels (e.g. RGB for images).
- Images: input X and output Y are $3d$ tensors $(\text{channel} \times i\,\text{pos} \times j\,\text{pos})$.
- Software: usually use batches and $4d$ tensors $(\text{batch} - \text{index} \times \text{channel} \times i\,\text{pos} \times j\,\text{pos})$
- For multi-channel convolution, linear operations are not guaranteed to be commutative, unless each operation has the same number of input and output channels.
- Multi-channel convolution: $4d$ Kernel $\mathbf{K}$ with $K_{i,j,k,l}$: connection strength between a unit in channel i of output and a unit in channel j of input, with offset of k rows and l columns between the output unit and the input unit.

$$Y_{i,j,k} = \sum_{l,m,n} X_{l,j+m-1,k+n-1} K_{i,l,m,n}$$

  where l, m, n sums over all valid indices.
- To reduce computational cost, it is possible to use a *stride* in the convolution operation.

Locally connected layers

- Also called **unshared convolution** (LeCun, 1986, 1989)
- adjacency matrix in the graph of our MLP is the same, but every connection has its own weight, specified by a 6-D tensor $\mathbf{W}$ with indices: output channel, row, column i, j, k, input channel, row offset, column offset l, m, n:

$$Y_{i,j,k} = \sum_{l,m,n} X_{l,j+m-1,k+n-1} W_{i,j,k,l,m,n}$$

- useful when a feature is a function of a small region, but not independent of where it is in the picture.

Locally connected layers

- Also called **unshared convolution** (LeCun, 1986, 1989)
- adjacency matrix in the graph of our MLP is the same, but every connection has its own weight, specified by a 6-D tensor **W** with indices: output channel, row, column i, j, k, input channel, row offset, column offset l, m, n:

$$Y_{i,j,k} = \sum_{l,m,n} X_{l,j+m-1,k+n-1} W_{i,j,k,l,m,n}$$

- useful when a feature is a function of a small region, but not independent of where it is in the picture.

Locally connected layers

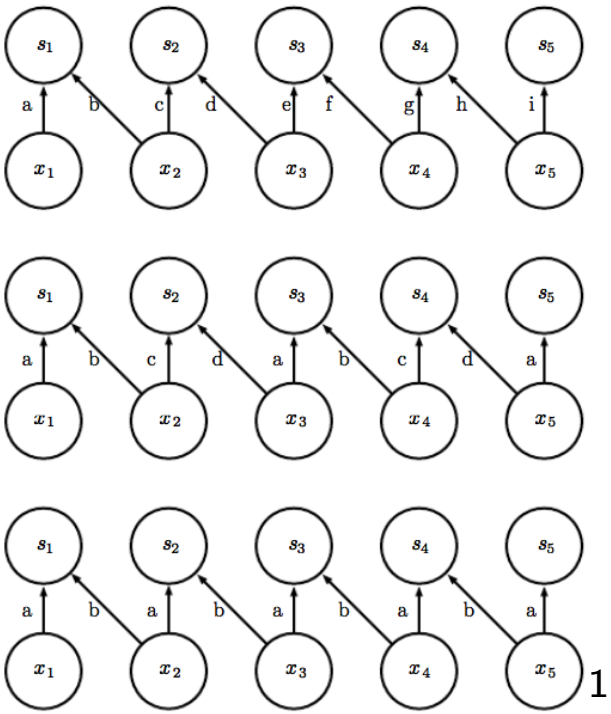
- Also called **unshared convolution** (LeCun, 1986, 1989)
- adjacency matrix in the graph of our MLP is the same, but every connection has its own weight, specified by a 6-D tensor **W** with indices: output channel, row, column i, j, k, input channel, row offset, column offset l, m, n:

$$Y_{i,j,k} = \sum_{l,m,n} X_{l,j+m-1,k+n-1} W_{i,j,k,l,m,n}$$

- useful when a feature is a function of a small region, but not independent of where it is in the picture.

Tiled convolution

- Compromise between a convolutional layer and a locally connected layer
- Rather than learning a separate set of weights at every spatial location, we learn a set of kernels that we rotate through as we move through space.
- Neighboring locations have different filters, as in locally connected layer, but memory requirements for parameters increase only by a factor of the size of this set of kernels.

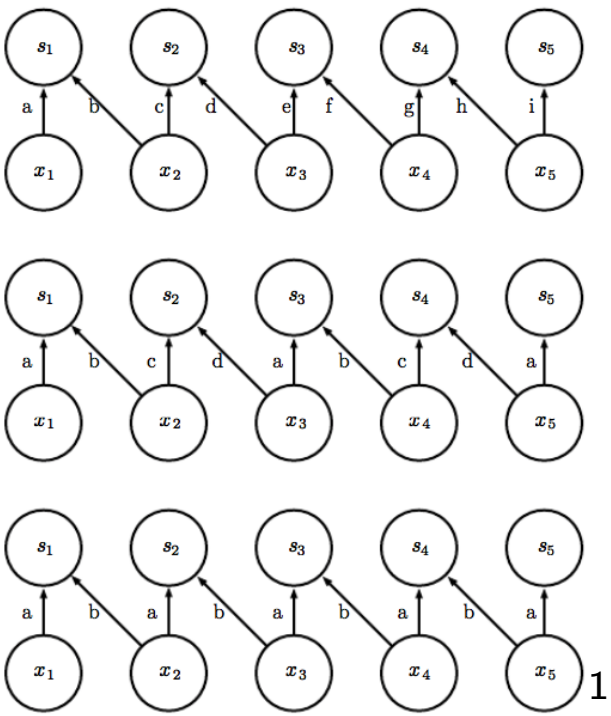


[1]

[1] From Goodfellow, Bengio and Courville: Deep Learning, MIT Press (2016)

Tiled convolution

- Compromise between a convolutional layer and a locally connected layer
- Rather than learning a separate set of weights at every spatial location, we learn a set of kernels that we rotate through as we move through space.
- Neighboring locations have different filters, as in locally connected layer, but memory requirements for parameters increase only by a factor of the size of this set of kernels.

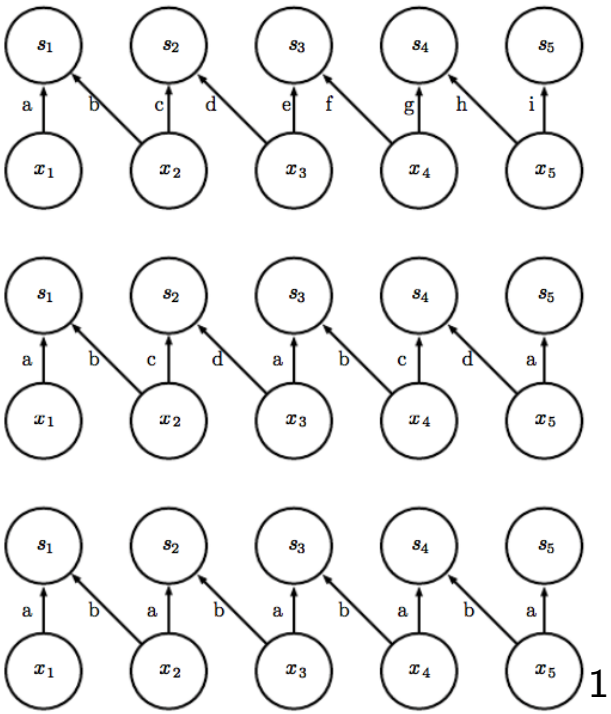


[1]

[1] From Goodfellow, Bengio and Courville: Deep Learning, MIT Press (2016)

Tiled convolution

- Compromise between a convolutional layer and a locally connected layer
- Rather than learning a separate set of weights at every spatial location, we learn a set of kernels that we rotate through as we move through space.
- Neighboring locations have different filters, as in locally connected layer, but memory requirements for parameters increase only by a factor of the size of this set of kernels.



[1]

[1] From Goodfellow, Bengio and Courville: Deep Learning, MIT Press (2016)

Examples

	Single channel	Multi-channel
$1d$	Audio waveform	
$2d$	Grayscale image	Color images
$3d$	Volumetric data (CT, MRI)	Color video

Efficient Convolution Algorithms

- Convolution can be implemented using fast Fourier transform:

$$K * X = \mathscr{F}^{-1}(\mathscr{F}(K) \odot \mathscr{F}(X))$$

  where $\odot$ is point-wise multiplication and $\mathscr{F}$, $\mathscr{F}^{-1}$ are forward and inverse Fourier transform. This approach is efficient for convolution of large images/kernels.

- When a d-dimensional kernel K can be expressed as the outer product of d vectors, e.g.:

$$K = \mathbf{u}\mathbf{v}^\top,$$

  then K is called separable. Convolution with K can then be composed by d one-dimensional convolutions with each of these vectors.
  A kernel with w elements and d dimensions requires generally $O(w^d)$ runtime and parameter storage space, but only requires $O(wd)$ runtime and parameter storage space when separable.

- Developing faster ways (algorithmic and hardware) to perform convolutions is an active area of reasearch.

Efficient Convolution Algorithms

- Convolution can be implemented using fast Fourier transform:

$$K * X = \mathscr{F}^{-1}(\mathscr{F}(K) \odot \mathscr{F}(X))$$

  where $\odot$ is point-wise multiplication and $\mathscr{F}$, $\mathscr{F}^{-1}$ are forward and inverse Fourier transform. This approach is efficient for convolution of large images/kernels.

- When a d-dimensional kernel K can be expressed as the outer product of d vectors, e.g.:

$$K = \mathbf{u}\mathbf{v}^{\top},$$

  then K is called separable. Convolution with K can then be composed by d one-dimensional convolutions with each of these vectors.
  A kernel with w elements and d dimensions requires generally $O(w^d)$ runtime and parameter storage space, but only requires $O(wd)$ runtime and parameter storage space when separable.

- Developing faster ways (algorithmic and hardware) to perform convolutions is an active area of reasearch.

Efficient Convolution Algorithms

- Convolution can be implemented using fast Fourier transform:

$$K * X = \mathscr{F}^{-1}(\mathscr{F}(K) \odot \mathscr{F}(X))$$

  where $\odot$ is point-wise multiplication and $\mathscr{F}$, $\mathscr{F}^{-1}$ are forward and inverse Fourier transform. This approach is efficient for convolution of large images/kernels.

- When a d-dimensional kernel K can be expressed as the outer product of d vectors, e.g.:

$$K = \mathbf{u}\mathbf{v}^{\top},$$

  then K is called separable. Convolution with K can then be composed by d one-dimensional convolutions with each of these vectors.
  A kernel with w elements and d dimensions requires generally $O(w^d)$ runtime and parameter storage space, but only requires $O(wd)$ runtime and parameter storage space when separable.

- Developing faster ways (algorithmic and hardware) to perform convolutions is an active area of reasearch.

Random and unsupervised features

- Most expensive part of ConvNet training is learning the features. Conv layers are usually much larger than the output layer due to pooling.
- ConvNet training cost can be reduced by using features that are not trained with supervised learning.
- Three strategies:
  1. **Random features**: often work surprisingly well in convolutional networks (Jarrett et al., 2009; Saxe et al., 2011; Pinto et al., 2011; Cox and Pinto, 2011).
  2. **Manually designed features**
  3. **Unsupervised features**:
     - Greedy layer-wise pretraining, to train the first layer in isolation, then extract all features from the first layer only once, then train the second layer in isolation given those features, and so on.
     - Convolutional deep belief network (Lee et al., 2009)
     - Coates et al. (2011): k-means clustering to small image patches, then use each learned centroid as a convolution kernel. Can train very large models. Full computational cost only at inference time.
     - Approach popular in 2007–2013, when labeled datasets were small and computational power was more limited.
     - Today, most convolutional networks are trained purely supervised.

Random and unsupervised features

- Most expensive part of ConvNet training is learning the features. Conv layers are usually much larger than the output layer due to pooling.
- ConvNet training cost can be reduced by using features that are not trained with supervised learning.
- Three strategies:
  1. **Random features**: often work surprisingly well in convolutional networks (Jarrett et al., 2009; Saxe et al., 2011; Pinto et al., 2011; Cox and Pinto, 2011).
  2. **Manually designed features**
  3. **Unsupervised features**:
     - Greedy layer-wise pretraining, to train the first layer in isolation, then extract all features from the first layer only once, then train the second layer in isolation given those features, and so on.
     - Convolutional deep belief network (Lee et al., 2009)
     - Coates et al. (2011): k-means clustering to small image patches, then use each learned centroid as a convolution kernel. Can train very large models. Full computational cost only at inference time.
     - Approach popular in 2007–2013, when labeled datasets were small and computational power was more limited.
     - Today, most convolutional networks are trained purely supervised.

Random and unsupervised features

- Most expensive part of ConvNet training is learning the features. Conv layers are usually much larger than the output layer due to pooling.
- ConvNet training cost can be reduced by using features that are not trained with supervised learning.
- Three strategies:
  1. **Random features**: often work surprisingly well in convolutional networks (Jarrett et al., 2009; Saxe et al., 2011; Pinto et al., 2011; Cox and Pinto, 2011).
  2. **Manually designed features**
  3. **Unsupervised features**:
     - Greedy layer-wise pretraining, to train the first layer in isolation, then extract all features from the first layer only once, then train the second layer in isolation given those features, and so on.
     - Convolutional deep belief network (Lee et al., 2009)
     - Coates et al. (2011): k-means clustering to small image patches, then use each learned centroid as a convolution kernel. Can train very large models. Full computational cost only at inference time.
     - Approach popular in 2007–2013, when labeled datasets were small and computational power was more limited.
     - Today, most convolutional networks are trained purely supervised.

Random and unsupervised features

- Most expensive part of ConvNet training is learning the features. Conv layers are usually much larger than the output layer due to pooling.
- ConvNet training cost can be reduced by using features that are not trained with supervised learning.
- Three strategies:
  1. **Random features**: often work surprisingly well in convolutional networks (Jarrett et al., 2009; Saxe et al., 2011; Pinto et al., 2011; Cox and Pinto, 2011).
  2. **Manually designed features**
  3. **Unsupervised features**:
     - Greedy layer-wise pretraining, to train the first layer in isolation, then extract all features from the first layer only once, then train the second layer in isolation given those features, and so on.
     - Convolutional deep belief network (Lee et al., 2009)
     - Coates et al. (2011): k-means clustering to small image patches, then use each learned centroid as a convolution kernel. Can train very large models. Full computational cost only at inference time.
     - Approach popular in 2007–2013, when labeled datasets were small and computational power was more limited.
     - Today, most convolutional networks are trained purely supervised.

Random and unsupervised features

- Most expensive part of ConvNet training is learning the features. Conv layers are usually much larger than the output layer due to pooling.
- ConvNet training cost can be reduced by using features that are not trained with supervised learning.
- Three strategies:
  1. **Random features**: often work surprisingly well in convolutional networks (Jarrett et al., 2009; Saxe et al., 2011; Pinto et al., 2011; Cox and Pinto, 2011).
  2. **Manually designed features**
  3. **Unsupervised features**:
     - Greedy layer-wise pretraining, to train the first layer in isolation, then extract all features from the first layer only once, then train the second layer in isolation given those features, and so on.
     - Convolutional deep belief network (Lee et al., 2009)
     - Coates et al. (2011): k-means clustering to small image patches, then use each learned centroid as a convolution kernel. Can train very large models. Full computational cost only at inference time.
     - Approach popular in 2007–2013, when labeled datasets were small and computational power was more limited.
     - Today, most convolutional networks are trained purely supervised.

ConvNets vs mammalian vision

- Pioneering neuroscientists: Hubel and Wiesel (1959, 1962, 1968):
  - Discrovered basic functionality of mammalian vision system by recording individual neuron activity in cats
  - Neurons in the early visual system respond mostly to specific patterns of light, such as precisely oriented bars.
- Primary visual cortex (V1): first brain area that performs advanced processing of visual input $\rightarrow$ inspires ConvNets
  - V1 is arranged in $2d$ spatial map, mirroring the image in the retina. $\rightarrow$ inspires $2d$ structure of ConvNets.
  - V1 **simple cells** respond approximately linearly to a small, spatially localized receptive field. $\rightarrow$ inspires ConvNet detector units
  - Most simple cells seem to perform convolutions whose weights are described by **Gabor functions**.

ConvNets vs mammalian vision

- Pioneering neuroscientists: Hubel and Wiesel (1959, 1962, 1968):
  - Discrovered basic functionality of mammalian vision system by recording individual neuron activity in cats
  - Neurons in the early visual system respond mostly to specific patterns of light, such as precisely oriented bars.
- Primary visual cortex (V1): first brain area that performs advanced processing of visual input $\rightarrow$ inspires ConvNets
  - V1 is arranged in $2d$ spatial map, mirroring the image in the retina. $\rightarrow$ inspires $2d$ structure of ConvNets.
  - V1 **simple cells** respond approximately linearly to a small, spatially localized receptive field. $\rightarrow$ inspires ConvNet detector units
  - Most simple cells seem to perform convolutions whose weights are described by **Gabor functions**.

ConvNets vs mammalian vision

- Pioneering neuroscientists: Hubel and Wiesel (1959, 1962, 1968):
  - Discrovered basic functionality of mammalian vision system by recording individual neuron activity in cats
  - Neurons in the early visual system respond mostly to specific patterns of light, such as precisely oriented bars.
- Primary visual cortex (V1): first brain area that performs advanced processing of visual input $\rightarrow$ inspires ConvNets
  - V1 is arranged in $2d$ spatial map, mirroring the image in the retina. $\rightarrow$ inspires $2d$ structure of ConvNets.
  - V1 **simple cells** respond approximately linearly to a small, spatially localized receptive field. $\rightarrow$ inspires ConvNet detector units
  - Most simple cells seem to perform convolutions whose weights are described by **Gabor functions**.

ConvNets vs mammalian vision

- Pioneering neuroscientists: Hubel and Wiesel (1959, 1962, 1968):
  - Discrovered basic functionality of mammalian vision system by recording individual neuron activity in cats
  - Neurons in the early visual system respond mostly to specific patterns of light, such as precisely oriented bars.
- Primary visual cortex (V1): first brain area that performs advanced processing of visual input $\rightarrow$ inspires ConvNets
  - V1 is arranged in $2d$ spatial map, mirroring the image in the retina. $\rightarrow$ inspires $2d$ structure of ConvNets.
  - V1 **simple cells** respond approximately linearly to a small, spatially localized receptive field. $\rightarrow$ inspires ConvNet detector units
  - Most simple cells seem to perform convolutions whose weights are described by **Gabor functions**.

ConvNets vs mammalian vision

- Pioneering neuroscientists: Hubel and Wiesel (1959, 1962, 1968):
  - Discrovered basic functionality of mammalian vision system by recording individual neuron activity in cats
  - Neurons in the early visual system respond mostly to specific patterns of light, such as precisely oriented bars.
- Primary visual cortex (V1): first brain area that performs advanced processing of visual input $\rightarrow$ inspires ConvNets
  - V1 is arranged in $2d$ spatial map, mirroring the image in the retina. $\rightarrow$ inspires $2d$ structure of ConvNets.
  - V1 **simple cells** respond approximately linearly to a small, spatially localized receptive field. $\rightarrow$ inspires ConvNet detector units
  - Most simple cells seem to perform convolutions whose weights are described by **Gabor functions**.

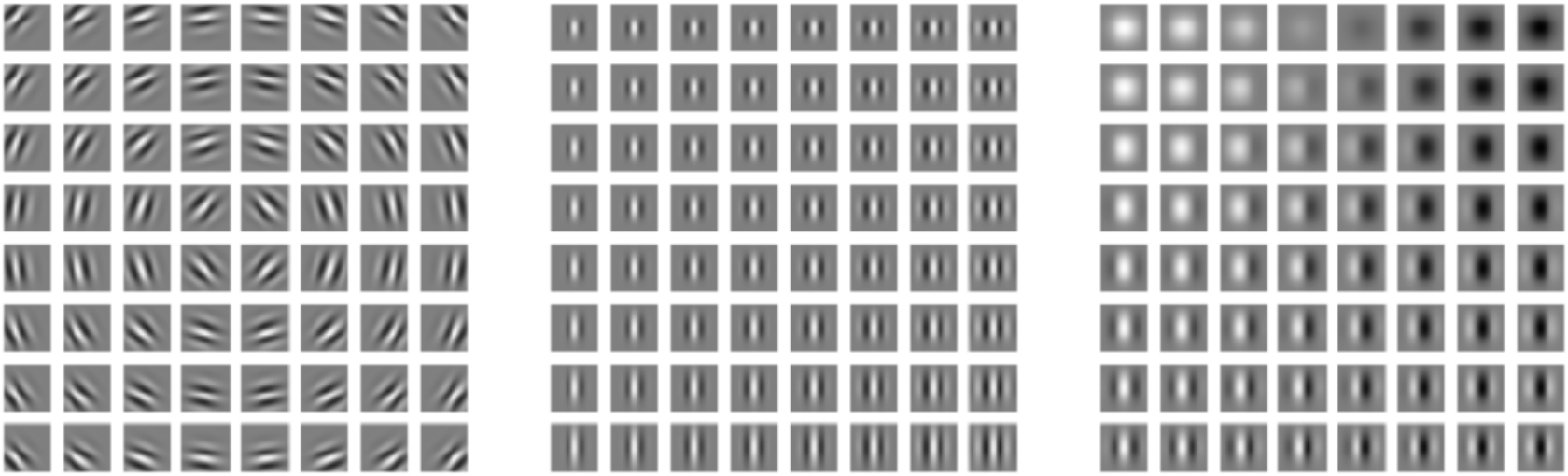
ConvNets vs mammalian vision

- Pioneering neuroscientists: Hubel and Wiesel (1959, 1962, 1968):
  - Discrovered basic functionality of mammalian vision system by recording individual neuron activity in cats
  - Neurons in the early visual system respond mostly to specific patterns of light, such as precisely oriented bars.
- Primary visual cortex (V1): first brain area that performs advanced processing of visual input $\rightarrow$ inspires ConvNets
  - V1 is arranged in $2d$ spatial map, mirroring the image in the retina. $\rightarrow$ inspires $2d$ structure of ConvNets.
  - V1 **simple cells** respond approximately linearly to a small, spatially localized receptive field. $\rightarrow$ inspires ConvNet detector units
  - Most simple cells seem to perform convolutions whose weights are described by **Gabor functions**.

ConvNets vs mammalian vision

Gabor functions:

$$w(x, y; \alpha, \beta_x, \beta_y, f, \phi, x_0, y_0, \tau) = \alpha \exp^{-\beta_x \bar{x}^2 - \beta_y \bar{y}^2} \cos(f\bar{x} + \phi)$$
$$\bar{x} = (x - x_0)\cos(\tau) + (y - y_0)\sin(\tau)$$
$$\bar{y} = (y - y_0)\cos(\tau) - (x - x_0)\sin(\tau)$$



→ many learning algorithms, include ConvNets, learn Gabor-like features when applied to natural images

ConvNets vs mammalian vision

- V1 **complex cells** respond to features as simple cells, but invariant to small position shifts
  $\rightarrow$ inspires pooling units.
  invariant to some changes in lighting
  $\rightarrow$ inspires cross-channel pooling.
- **Assumption**: basic strategy of detection and pooling is repeated in deeper brain layers.
- **Grandmother cells** (concept): a neuron that activates when a person sees an image of their grandmother, regardless of position in the image, close-up or full-body short, brightly lit or in shadows etc.
  $\rightarrow$ have been shown to exist in the medial temporal lobe in the human brain (Quiroga et al., 2005).
- **Halle Berry neuron** (concept): an individual neuron that is activated by the concept of Halle Berry (seeing a photo or a drawing of Halle Berry, or reading text "Halle Berry")

ConvNets vs mammalian vision

- V1 **complex cells** respond to features as simple cells, but invariant to small position shifts
  $\rightarrow$ inspires pooling units.
  invariant to some changes in lighting
  $\rightarrow$ inspires cross-channel pooling.
- **Assumption**: basic strategy of detection and pooling is repeated in deeper brain layers.
- **Grandmother cells** (concept): a neuron that activates when a person sees an image of their grandmother, regardless of position in the image, close-up or full-body short, brightly lit or in shadows etc.
  $\rightarrow$ have been shown to exist in the medial temporal lobe in the human brain (Quiroga et al., 2005).
- **Halle Berry neuron** (concept): an individual neuron that is activated by the concept of Halle Berry (seeing a photo or a drawing of Halle Berry, or reading text "Halle Berry")

ConvNets vs mammalian vision

- V1 **complex cells** respond to features as simple cells, but invariant to small position shifts
  $\rightarrow$ inspires pooling units.
  invariant to some changes in lighting
  $\rightarrow$ inspires cross-channel pooling.
- **Assumption**: basic strategy of detection and pooling is repeated in deeper brain layers.
- **Grandmother cells** (concept): a neuron that activates when a person sees an image of their grandmother, regardless of position in the image, close-up or full-body short, brightly lit or in shadows etc.
  $\rightarrow$ have been shown to exist in the medial temporal lobe in the human brain (Quiroga et al., 2005).
- **Halle Berry neuron** (concept): an individual neuron that is activated by the concept of Halle Berry (seeing a photo or a drawing of Halle Berry, or reading text "Halle Berry")

ConvNets vs mammalian vision

- V1 **complex cells** respond to features as simple cells, but invariant to small position shifts
  $\rightarrow$ inspires pooling units.
  invariant to some changes in lighting
  $\rightarrow$ inspires cross-channel pooling.
- **Assumption**: basic strategy of detection and pooling is repeated in deeper brain layers.
- **Grandmother cells** (concept): a neuron that activates when a person sees an image of their grandmother, regardless of position in the image, close-up or full-body short, brightly lit or in shadows etc.
  $\rightarrow$ have been shown to exist in the medial temporal lobe in the human brain (Quiroga et al., 2005).
- **Halle Berry neuron** (concept): an individual neuron that is activated by the concept of Halle Berry (seeing a photo or a drawing of Halle Berry, or reading text "Halle Berry")

ConvNets vs mammalian vision: Differences

- Human eye mostly low resolution, except for a tiny patch, with a size of a thumbnail held at arms length (**fovea**). Perception to see an entire scene in high resolution is an illusion stitched together from glimpses of small areas.
- Visual models with foveation mechanisms have been developed but so far have not become the dominant approach (Larochelle and Hinton, 2010; Denil et al., 2012)
- The human visual system is integrated with many other senses, such as hearing, and factors like our moods and thoughts. Convolutional networks so far are purely visual.
- Human visual system does much more than just recognize objects. It is able to understand entire scenes including many objects and relationships between objects (see Capsule networks)
- Simple brain areas like V1 are heavily impacted by feedback from higher levels.
- Brains cells are much more complicated and diverse than artificial neurons

ConvNets vs mammalian vision: Differences

- Human eye mostly low resolution, except for a tiny patch, with a size of a thumbnail held at arms length (**fovea**). Perception to see an entire scene in high resolution is an illusion stitched together from glimpses of small areas.
- Visual models with foveation mechanisms have been developed but so far have not become the dominant approach (Larochelle and Hinton, 2010; Denil et al., 2012)
- The human visual system is integrated with many other senses, such as hearing, and factors like our moods and thoughts. Convolutional networks so far are purely visual.
- Human visual system does much more than just recognize objects. It is able to understand entire scenes including many objects and relationships between objects (see Capsule networks)
- Simple brain areas like V1 are heavily impacted by feedback from higher levels.
- Brains cells are much more complicated and diverse than artificial neurons

ConvNets vs mammalian vision: Differences

- Human eye mostly low resolution, except for a tiny patch, with a size of a thumbnail held at arms length (**fovea**). Perception to see an entire scene in high resolution is an illusion stitched together from glimpses of small areas.
- Visual models with foveation mechanisms have been developed but so far have not become the dominant approach (Larochelle and Hinton, 2010; Denil et al., 2012)
- The human visual system is integrated with many other senses, such as hearing, and factors like our moods and thoughts. Convolutional networks so far are purely visual.
- Human visual system does much more than just recognize objects. It is able to understand entire scenes including many objects and relationships between objects (see Capsule networks)
- Simple brain areas like V1 are heavily impacted by feedback from higher levels.
- Brains cells are much more complicated and diverse than artificial neurons

ConvNets vs mammalian vision: Differences

- Human eye mostly low resolution, except for a tiny patch, with a size of a thumbnail held at arms length (**fovea**). Perception to see an entire scene in high resolution is an illusion stitched together from glimpses of small areas.
- Visual models with foveation mechanisms have been developed but so far have not become the dominant approach (Larochelle and Hinton, 2010; Denil et al., 2012)
- The human visual system is integrated with many other senses, such as hearing, and factors like our moods and thoughts. Convolutional networks so far are purely visual.
- Human visual system does much more than just recognize objects. It is able to understand entire scenes including many objects and relationships between objects (see Capsule networks)
- Simple brain areas like V1 are heavily impacted by feedback from higher levels.
- Brains cells are much more complicated and diverse than artificial neurons

ConvNets vs mammalian vision: Differences

- Human eye mostly low resolution, except for a tiny patch, with a size of a thumbnail held at arms length (**fovea**). Perception to see an entire scene in high resolution is an illusion stitched together from glimpses of small areas.
- Visual models with foveation mechanisms have been developed but so far have not become the dominant approach (Larochelle and Hinton, 2010; Denil et al., 2012)
- The human visual system is integrated with many other senses, such as hearing, and factors like our moods and thoughts. Convolutional networks so far are purely visual.
- Human visual system does much more than just recognize objects. It is able to understand entire scenes including many objects and relationships between objects (see Capsule networks)
- Simple brain areas like V1 are heavily impacted by feedback from higher levels.
- Brains cells are much more complicated and diverse than artificial neurons

ConvNets vs mammalian vision: Differences

- Human eye mostly low resolution, except for a tiny patch, with a size of a thumbnail held at arms length (**fovea**). Perception to see an entire scene in high resolution is an illusion stitched together from glimpses of small areas.
- Visual models with foveation mechanisms have been developed but so far have not become the dominant approach (Larochelle and Hinton, 2010; Denil et al., 2012)
- The human visual system is integrated with many other senses, such as hearing, and factors like our moods and thoughts. Convolutional networks so far are purely visual.
- Human visual system does much more than just recognize objects. It is able to understand entire scenes including many objects and relationships between objects (see Capsule networks)
- Simple brain areas like V1 are heavily impacted by feedback from higher levels.
- Brains cells are much more complicated and diverse than artificial neurons

Random and unsupervised features

- Neural network research group at AT&T developed a convolutional network for reading checks (LeCun et al., 1998b). By the end of the 1990s, this system deployed by NEC was reading over 10% of all the checks in the US.
- Several OCR and handwriting recognition systems based on convolutional nets were deployed by Microsoft (Simard et al., 2003).
- Current intensity of commercial interest in deep learning began when Krizhevsky et al. (2012) won the ImageNet object recognition challenge
- Convolutional nets were some of the first working deep networks trained with back-propagation

Random and unsupervised features

- Neural network research group at AT&T developed a convolutional network for reading checks (LeCun et al., 1998b). By the end of the 1990s, this system deployed by NEC was reading over 10% of all the checks in the US.
- Several OCR and handwriting recognition systems based on convolutional nets were deployed by Microsoft (Simard et al., 2003).
- Current intensity of commercial interest in deep learning began when Krizhevsky et al. (2012) won the ImageNet object recognition challenge
- Convolutional nets were some of the first working deep networks trained with back-propagation

Random and unsupervised features

- Neural network research group at AT&T developed a convolutional network for reading checks (LeCun et al., 1998b). By the end of the 1990s, this system deployed by NEC was reading over 10% of all the checks in the US.
- Several OCR and handwriting recognition systems based on convolutional nets were deployed by Microsoft (Simard et al., 2003).
- Current intensity of commercial interest in deep learning began when Krizhevsky et al. (2012) won the ImageNet object recognition challenge
- Convolutional nets were some of the first working deep networks trained with back-propagation

Random and unsupervised features

- Neural network research group at AT&T developed a convolutional network for reading checks (LeCun et al., 1998b). By the end of the 1990s, this system deployed by NEC was reading over 10% of all the checks in the US.
- Several OCR and handwriting recognition systems based on convolutional nets were deployed by Microsoft (Simard et al., 2003).
- Current intensity of commercial interest in deep learning began when Krizhevsky et al. (2012) won the ImageNet object recognition challenge
- Convolutional nets were some of the first working deep networks trained with back-propagation