CEE 616: Probabilistic Machine Learning

M3 Deep Neural Networks: L3c: Neural Networks for Images

Jimi Oke

UMass Amherst

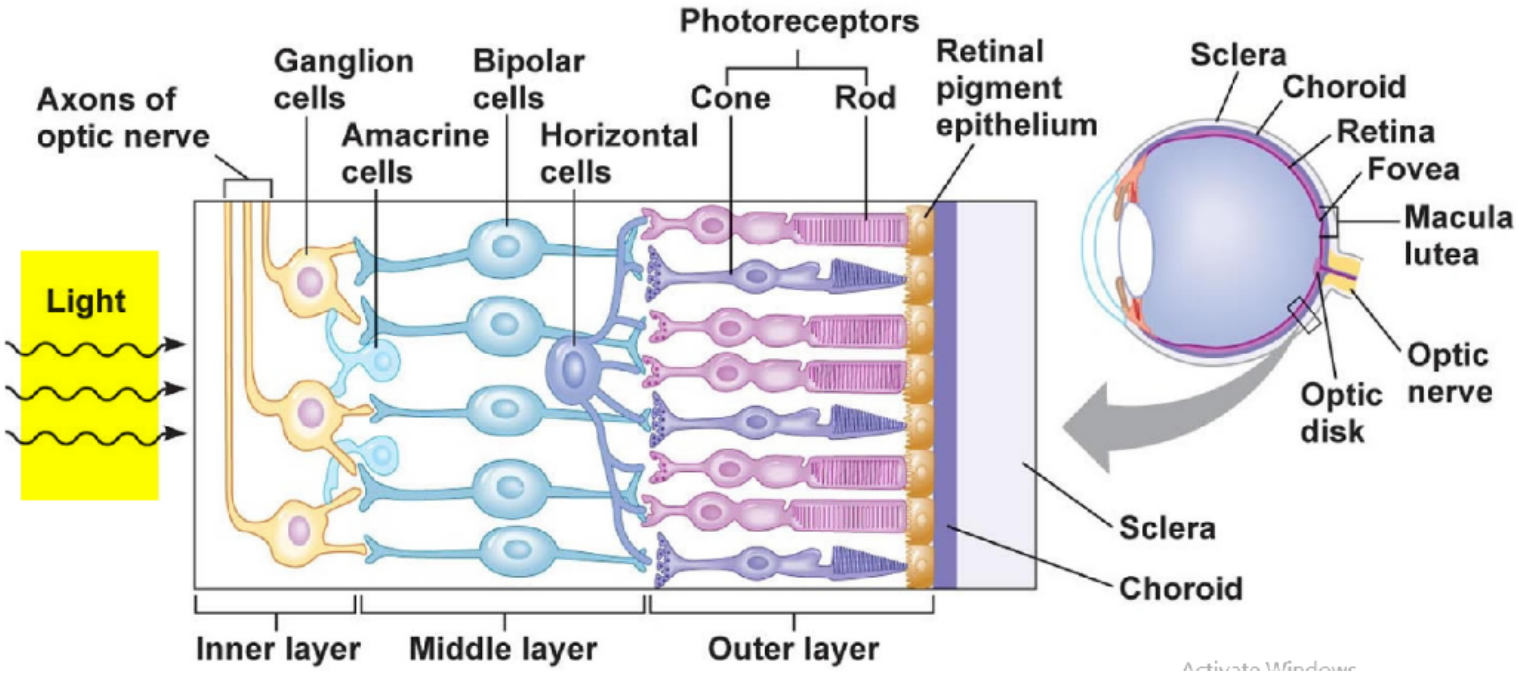
College of Engineering

Thu, Oct 23, 2025

Outline

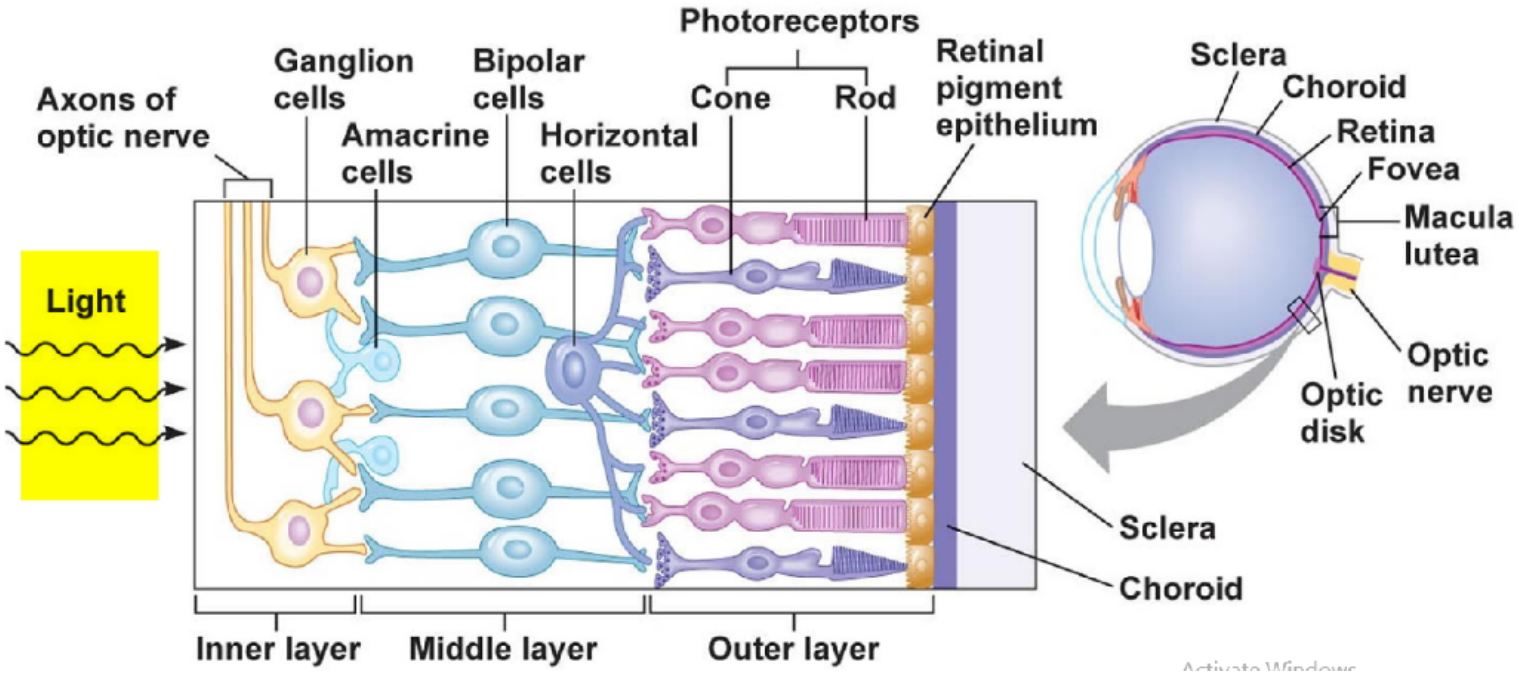
The convolutional neural network (CNN)

The convolutional neural network (CNN)



Source: `https://s3mn.mnimgs.com/img/shared/content_ck_images/ck_5ab694d5e73f4.png`

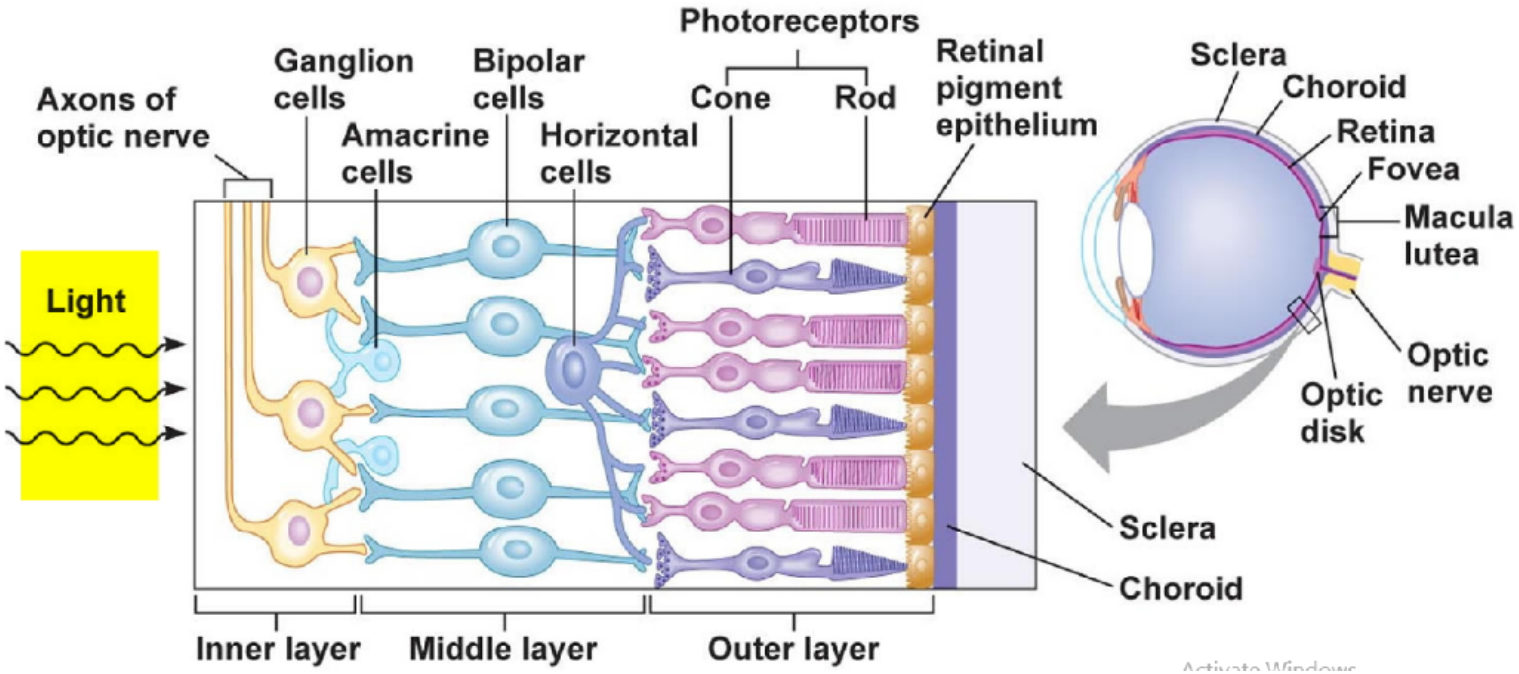
The convolutional neural network (CNN)



Source: `https://s3mn.mnimgs.com/img/shared/content_ck_images/ck_5ab694d5e73f4.png`

- Motivated by the image recognition process of the brain's visual cortex.

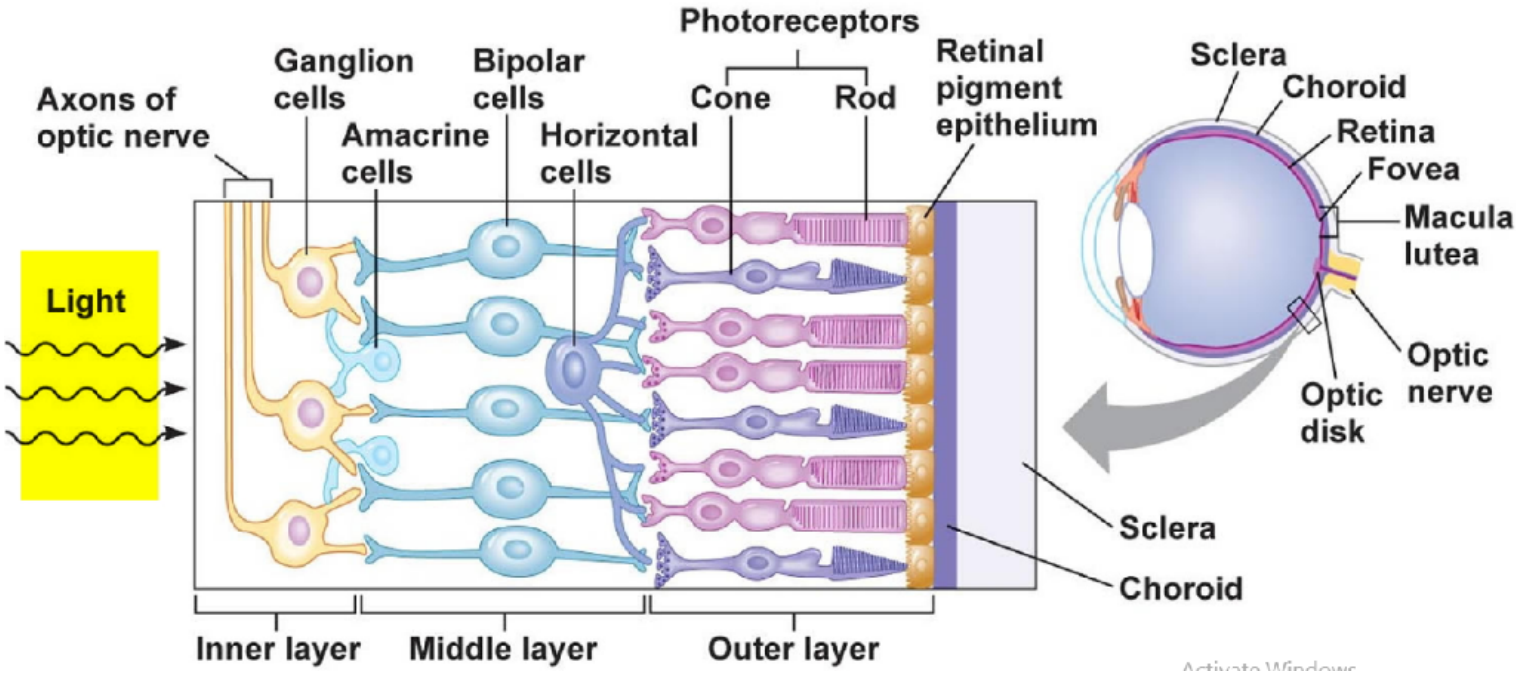
The convolutional neural network (CNN)



Source: `https://s3mn.mnimgs.com/img/shared/content_ck_images/ck_5ab694d5e73f4.png`

- Motivated by the image recognition process of the brain's visual cortex.
- Groundbreaking study showed that *local receptive fields* activate neurons in the visual cortex. (Hubel & Wiesel, 1958; 1959)

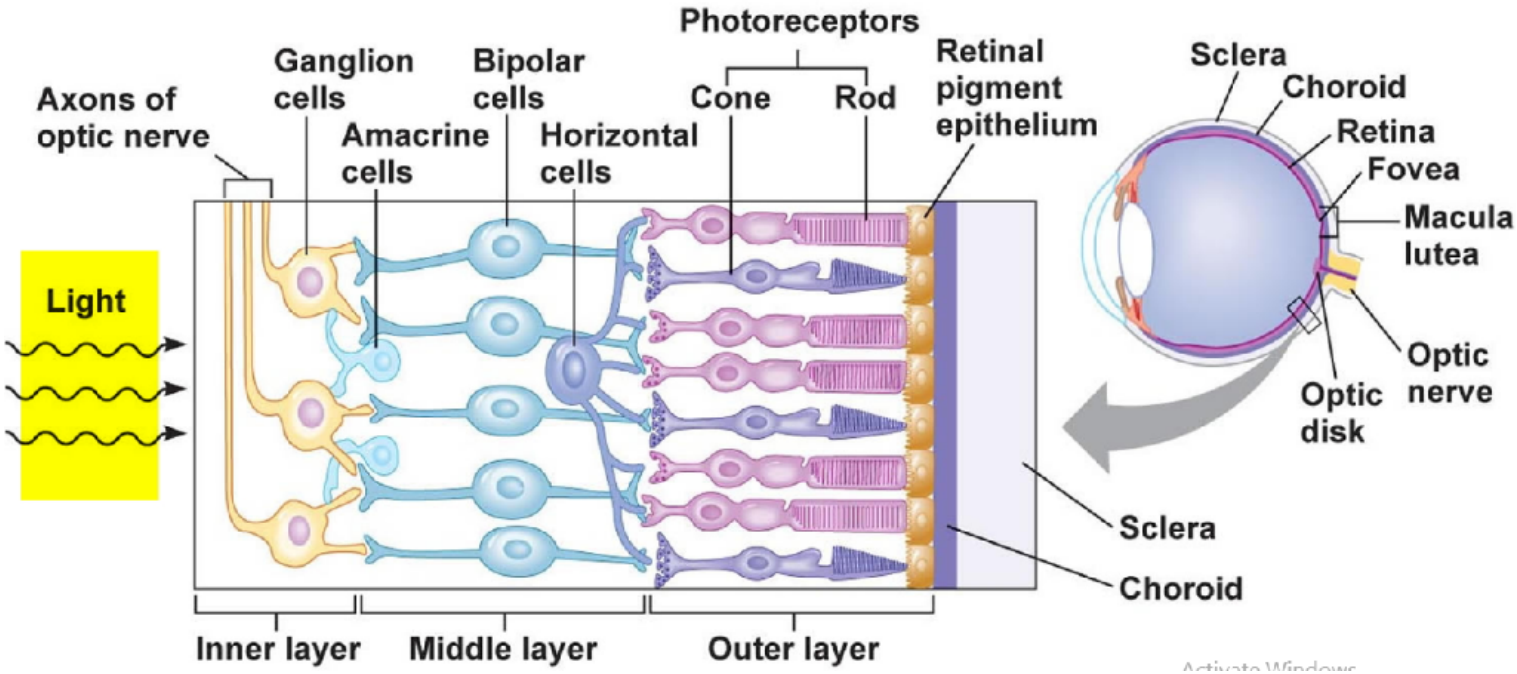
The convolutional neural network (CNN)



Source: `https://s3mn.mnimgs.com/img/shared/content_ck_images/ck_5ab694d5e73f4.png`

- Motivated by the image recognition process of the brain's visual cortex.
- Groundbreaking study showed that *local receptive fields* activate neurons in the visual cortex. (Hubel & Wiesel, 1958; 1959)
- Earliest neural network for image recognitron introduced (Fukushima, 1980)

The convolutional neural network (CNN)



Source: `https://s3mn.mnimgs.com/img/shared/content_ck_images/ck_5ab694d5e73f4.png`

- Motivated by the image recognition process of the brain's visual cortex.
- Groundbreaking study showed that *local receptive fields* activate neurons in the visual cortex. (Hubel & Wiesel, 1958; 1959)
- Earliest neural network for image recognitron introduced (Fukushima, 1980)
- Milestone: introduction of *LeNet-5* architecture for handwritten digit recognition (Yann LeCun et al., 1998)

Building blocks of a CNN

Building blocks of a CNN

- **Input layer:** the image to be classified

Building blocks of a CNN

- **Input layer:** the image to be classified
- **Convolutional layer:** represents the action of a filter transmitting signals (features) from various portions (receptive fields) of the preceding layer. The size of the receptive field is specified by the *convolutional kernel*. Each layer can have multiple feature maps representing different filters.

Building blocks of a CNN

- **Input layer:** the image to be classified
- **Convolutional layer:** represents the action of a filter transmitting signals (features) from various portions (receptive fields) of the preceding layer. The size of the receptive field is specified by the *convolutional kernel*. Each layer can have multiple feature maps representing different filters.
- **Pooling layer:** subsamples signals from preceding layer to reduce dimensionality and extract dominant features (subsample space determined by kernel size)

Building blocks of a CNN

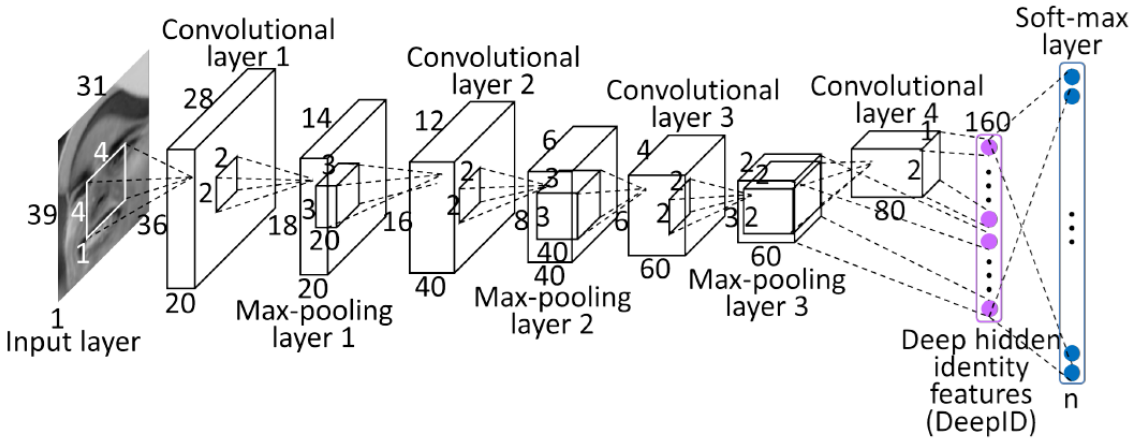
- **Input layer:** the image to be classified
- **Convolutional layer:** represents the action of a filter transmitting signals (features) from various portions (receptive fields) of the preceding layer. The size of the receptive field is specified by the *convolutional kernel*. Each layer can have multiple feature maps representing different filters.
- **Pooling layer:** subsamples signals from preceding layer to reduce dimensionality and extract dominant features (subsample space determined by kernel size)
- **Dense layer:** neuron outputs are flattened and fully connected

Building blocks of a CNN

- **Input layer:** the image to be classified
- **Convolutional layer:** represents the action of a filter transmitting signals (features) from various portions (receptive fields) of the preceding layer. The size of the receptive field is specified by the *convolutional kernel*. Each layer can have multiple feature maps representing different filters.
- **Pooling layer:** subsamples signals from preceding layer to reduce dimensionality and extract dominant features (subsample space determined by kernel size)
- **Dense layer:** neuron outputs are flattened and fully connected
- **Output layer:** neurons equal to number of classes; with softmax activation

Building blocks of a CNN

- **Input layer:** the image to be classified
- **Convolutional layer:** represents the action of a filter transmitting signals (features) from various portions (receptive fields) of the preceding layer. The size of the receptive field is specified by the *convolutional kernel*. Each layer can have multiple feature maps representing different filters.
- **Pooling layer:** subsamples signals from preceding layer to reduce dimensionality and extract dominant features (subsample space determined by kernel size)
- **Dense layer:** neuron outputs are flattened and fully connected
- **Output layer:** neurons equal to number of classes; with softmax activation



Source: https://neurdiness.wordpress.com/2018/05/17/deep-convolutional-neural-networks-as-models-of-the-visual-system-qa/

Convolution

Convolution

A convolution is a mathematical operation that computes the overlap of a function g as it shifts across a function f.

Convolution

A convolution is a mathematical operation that computes the overlap of a function g as it shifts across a function f.

Thus, the convolution of f and g in the interval $[0, t]$ is given by:

Convolution

A convolution is a mathematical operation that computes the overlap of a function g as it shifts across a function f.

Thus, the convolution of f and g in the interval $[0, t]$ is given by:

$$(f \circledast g)(t) =$$

Convolution

A convolution is a mathematical operation that computes the overlap of a function g as it shifts across a function f.

Thus, the convolution of f and g in the interval $[0, t]$ is given by:

$$(f \circledast g)(t) = \int f(\tau) g(t - \tau) d\tau \tag{1}$$

Convolution

A convolution is a mathematical operation that computes the overlap of a function g as it shifts across a function f.

Thus, the convolution of f and g in the interval $[0, t]$ is given by:

$$(f \circledast g)(t) = \int f(\tau) g(t - \tau) d\tau \tag{1}$$

Watch this animation for a better understanding:
`https://youtu.be/C1N55M1VD2o`

In CNN terminology:

Convolution

A convolution is a mathematical operation that computes the overlap of a function g as it shifts across a function f.

Thus, the convolution of f and g in the interval $[0, t]$ is given by:

$$(f \circledast g)(t) = \int f(\tau) g(t - \tau) d\tau \tag{1}$$

Watch this animation for a better understanding:
`https://youtu.be/C1N55M1VD2o`

In CNN terminology:

- The function f is the **input**
- The function g is the **kernel**
- The convolution (or output) is called the **feature map**

Convolution (cont.)

Convolution (cont.)

The convolution operation is commutative:

Convolution (cont.)

The convolution operation is commutative:

$$(f \circledast g)(t) =$$

Convolution (cont.)

The convolution operation is commutative:

$$(f \circledast g)(t) = (g \circledast f)(t) \tag{2}$$

Convolution (cont.)

The convolution operation is commutative:

$$(f \circledast g)(t) = (g \circledast f)(t) \tag{2}$$

In CNNs, convolutions are typically

Convolution (cont.)

The convolution operation is commutative:

$$(f \circledast g)(t) = (g \circledast f)(t) \tag{2}$$

In CNNs, convolutions are typically

- performed on a 2D image I (input)

Convolution (cont.)

The convolution operation is commutative:

$$(f \circledast g)(t) = (g \circledast f)(t) \tag{2}$$

In CNNs, convolutions are typically

- performed on a 2D image I (input)
- using a 2D kernel K (dimensions $M \times N$)

Convolution (cont.)

The convolution operation is commutative:

$$(f \circledast g)(t) = (g \circledast f)(t) \tag{2}$$

In CNNs, convolutions are typically

- performed on a 2D image I (input)
- using a 2D kernel K (dimensions $M \times N$)

Thus, the discrete 2D convolution is given by:

Convolution (cont.)

The convolution operation is commutative:

$$(f \circledast g)(t) = (g \circledast f)(t) \tag{2}$$

In CNNs, convolutions are typically

- performed on a 2D image I (input)
- using a 2D kernel K (dimensions $M \times N$)

Thus, the discrete 2D convolution is given by:

$$(I \circledast K)(i, j) =$$

Convolution (cont.)

The convolution operation is commutative:

$$(f \circledast g)(t) = (g \circledast f)(t) \tag{2}$$

In CNNs, convolutions are typically

- performed on a 2D image I (input)
- using a 2D kernel K (dimensions $M \times N$)

Thus, the discrete 2D convolution is given by:

$$(I \circledast K)(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(m,n) K(i-m, j-n) \tag{3}$$

Convolution (cont.)

The convolution operation is commutative:

$$(f \circledast g)(t) = (g \circledast f)(t) \tag{2}$$

In CNNs, convolutions are typically

- performed on a 2D image I (input)
- using a 2D kernel K (dimensions $M \times N$)

Thus, the discrete 2D convolution is given by:

$$(I \circledast K)(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(m,n) K(i-m, j-n) \tag{3}$$

By the commutative property, we can write

Convolution (cont.)

The convolution operation is commutative:

$$(f \circledast g)(t) = (g \circledast f)(t) \tag{2}$$

In CNNs, convolutions are typically

- performed on a 2D image I (input)
- using a 2D kernel K (dimensions $M \times N$)

Thus, the discrete 2D convolution is given by:

$$(I \circledast K)(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(m,n) K(i-m, j-n) \tag{3}$$

By the commutative property, we can write

$$(K \circledast I)(i,j) =$$

Convolution (cont.)

The convolution operation is commutative:

$$(f \circledast g)(t) = (g \circledast f)(t) \tag{2}$$

In CNNs, convolutions are typically

- performed on a 2D image I (input)
- using a 2D kernel K (dimensions $M \times N$)

Thus, the discrete 2D convolution is given by:

$$(I \circledast K)(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(m,n) K(i-m, j-n) \tag{3}$$

By the commutative property, we can write

$$(K \circledast I)(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(i-m, j-n) K(m,n) \tag{4}$$

Convolution (cont.)

The convolution operation is commutative:

$$(f \circledast g)(t) = (g \circledast f)(t) \tag{2}$$

In CNNs, convolutions are typically

- performed on a 2D image I (input)
- using a 2D kernel K (dimensions $M \times N$)

Thus, the discrete 2D convolution is given by:

$$(I \circledast K)(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(m,n) K(i-m, j-n) \tag{3}$$

By the commutative property, we can write

$$(K \circledast I)(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(i-m, j-n) K(m,n) \tag{4}$$

which is more straightforward to compute (the kernel is *flipped* relative to the input)

Convolution and cross-correlation

In reality, NN libraries typically compute the **cross-correlation**:

Convolution and cross-correlation

In reality, NN libraries typically compute the **cross-correlation**:

$$(K * I)(i, j) =$$

Convolution and cross-correlation

In reality, NN libraries typically compute the **cross-correlation**:

$$(K * I)(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(i+m, j+n) K(m,n) \tag{5}$$

Convolution and cross-correlation

In reality, NN libraries typically compute the **cross-correlation**:

$$(K * I)(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(i+m, j+n) K(m,n) \tag{5}$$

which is equivalent to convolution performed without *kernel flipping*.

Convolution and cross-correlation

In reality, NN libraries typically compute the **cross-correlation**:

$$(K * I)(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(i+m, j+n) K(m,n) \tag{5}$$

which is equivalent to convolution performed without *kernel flipping*.

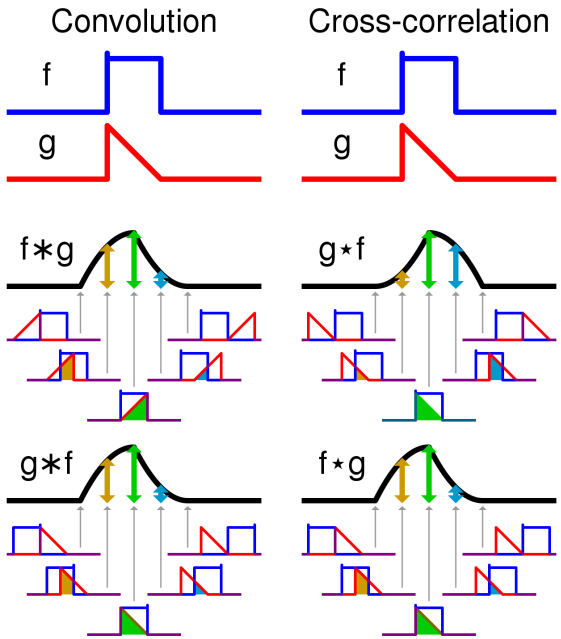


Figure: Convolution versus cross-correlation. Source: https://en.wikipedia.org/wiki/Convolution

Convolution and cross-correlation

In reality, NN libraries typically compute the **cross-correlation**:

$$(K * I)(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(i+m, j+n) K(m,n) \tag{5}$$

which is equivalent to convolution performed without *kernel flipping*.

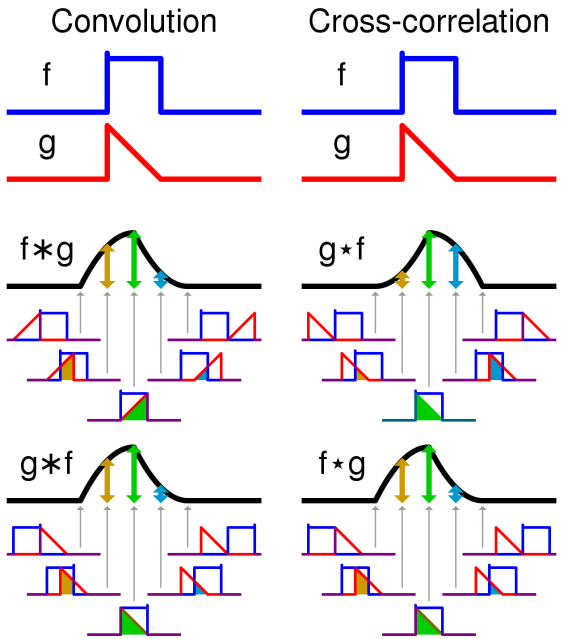


Figure: Convolution versus cross-correlation. Source: https://en.wikipedia.org/wiki/Convolution

Convolution and cross-correlation

In reality, NN libraries typically compute the **cross-correlation**:

$$(K * I)(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(i+m, j+n) K(m,n) \tag{5}$$

which is equivalent to convolution performed without *kernel flipping*.

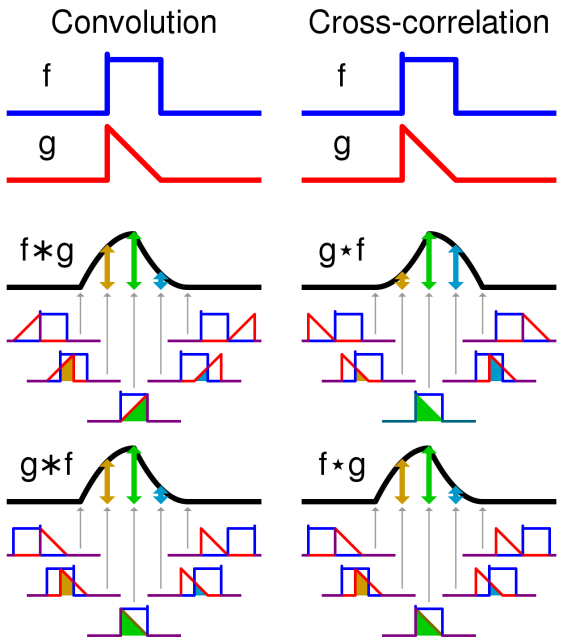


Figure: Convolution versus cross-correlation. Source: https://en.wikipedia.org/wiki/Convolution

If the kernel is symmetric (which it is in practice), then cross-correlation and

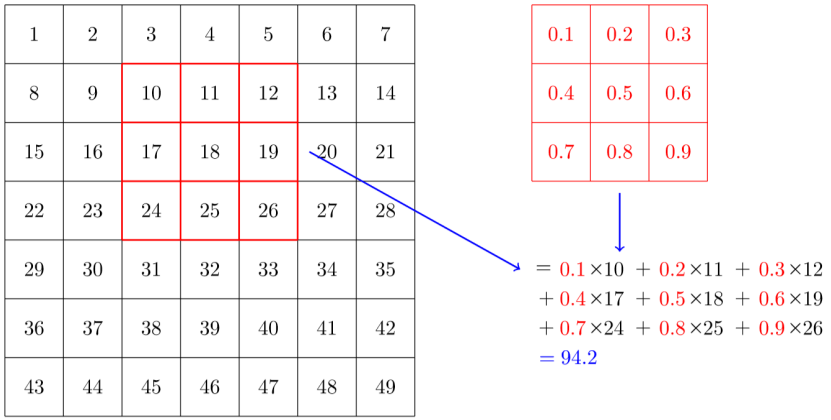
Convolution example

Convolution example

Here, we compute $(K * I)(2, 3)$ with $M = 3$ and $N = 3$ (3×3 kernel)

Convolution example

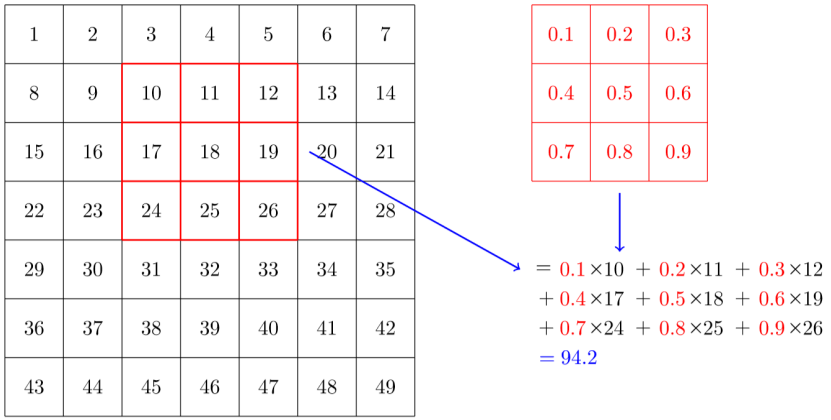
Here, we compute $(K * I)(2, 3)$ with $M = 3$ and $N = 3$ (3×3 kernel)



Source: `https://sgugger.github.io/convolution-in-depth.html`

Convolution example

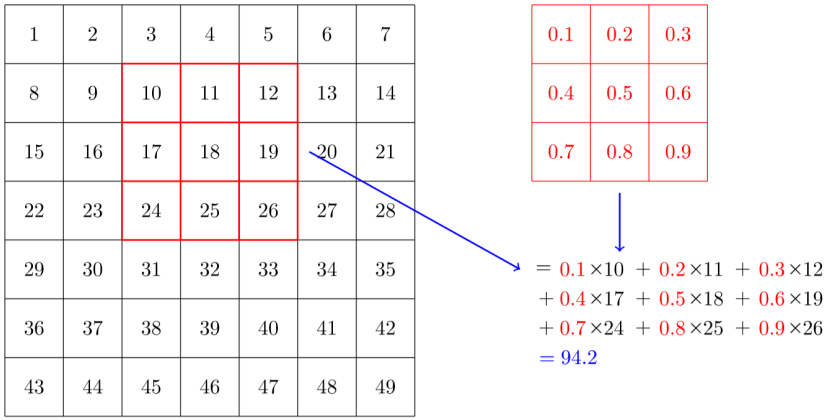
Here, we compute $(K * I)(2, 3)$ with $M = 3$ and $N = 3$ (3×3 kernel)



Source: `https://sgugger.github.io/convolution-in-depth.html`

Convolution example

Here, we compute $(K * I)(2, 3)$ with $M = 3$ and $N = 3$ (3×3 kernel)

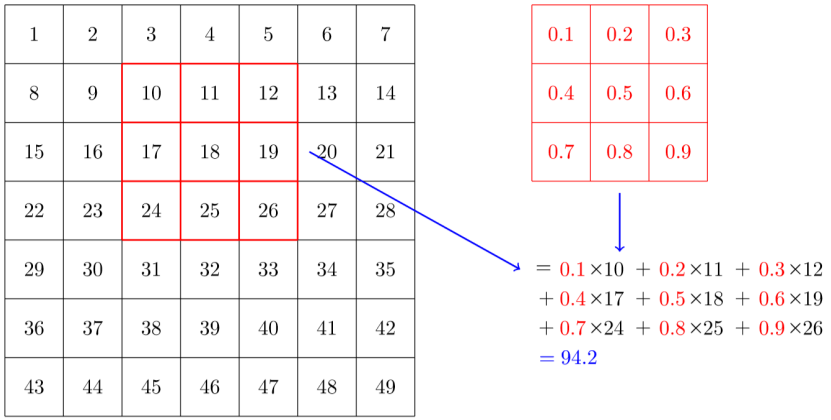


Source: `https://sgugger.github.io/convolution-in-depth.html`

$$(K * I)(2, 3) \quad =$$

Convolution example

Here, we compute $(K * I)(2, 3)$ with $M = 3$ and $N = 3$ (3×3 kernel)

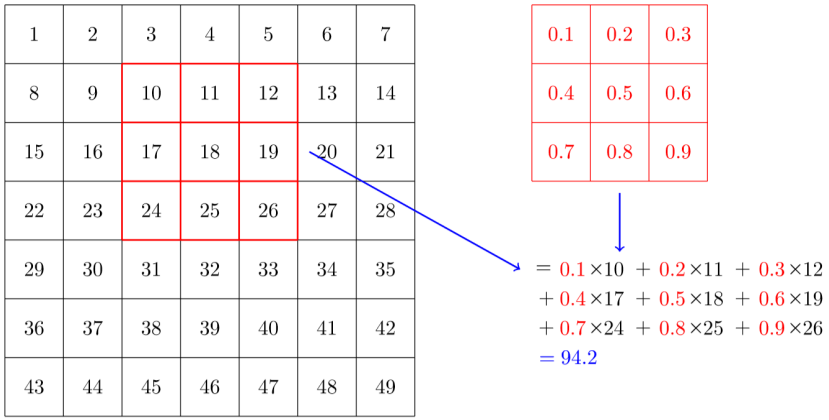


Source: `https://sgugger.github.io/convolution-in-depth.html`

$$(K * I)(2, 3) = \sum_{m=0}^{2} \sum_{n=0}^{2} I(2 + m, 3 + n) K(m, n)$$

Convolution example

Here, we compute $(K * I)(2, 3)$ with $M = 3$ and $N = 3$ (3×3 kernel)



Source: `https://sgugger.github.io/convolution-in-depth.html`

$$
\begin{aligned}
(K * I)(2,3) &= \sum_{m=0}^{2} \sum_{n=0}^{2} I(2+m, 3+n) K(m,n) \\
&= \sum_{m=0}^{2} \Big[I(2+m,3)K(m,0) \\
&\qquad + I(2+m,4)K(m,1) \\
&\qquad + I(2+m,5)K(m,2) \Big]
\end{aligned}
$$

Zero-padding

Zero-padding

- Convolving an $f_h \times f_w$ filter over an image of size $x_h \times x_w$ produces an output of size $(x_h - f_h + 1) \times (x_w - f_w + 1)$ (**valid convolution**)

Zero-padding

- Convolving an $f_h \times f_w$ filter over an image of size $x_h \times x_w$ produces an output of size $(x_h - f_h + 1) \times (x_w - f_w + 1)$ (**valid convolution**)
- But this reduces the spatial dimensions of the image at each layer

Zero-padding

- Convolving an $f_h \times f_w$ filter over an image of size $x_h \times x_w$ produces an output of size $(x_h - f_h + 1) \times (x_w - f_w + 1)$ (**valid convolution**)
- But this reduces the spatial dimensions of the image at each layer
- To preserve spatial dimensions, **zero-padding** is used: adding zeros around the border of the input image (**same convolution**)

Zero-padding

- Convolving an $f_h \times f_w$ filter over an image of size $x_h \times x_w$ produces an output of size $(x_h - f_h + 1) \times (x_w - f_w + 1)$ (**valid convolution**)
- But this reduces the spatial dimensions of the image at each layer
- To preserve spatial dimensions, **zero-padding** is used: adding zeros around the border of the input image (**same convolution**)
- Thus, for an input of size $x_h \times x_w$, adding padding of size p_h and p_w results in an output of size $(x_h - f_h + 2p_h + 1) \times (x_w - f_w + 2p_w + 1)$

Zero-padding

- Convolving an $f_h \times f_w$ filter over an image of size $x_h \times x_w$ produces an output of size $(x_h - f_h + 1) \times (x_w - f_w + 1)$ (**valid convolution**)
- But this reduces the spatial dimensions of the image at each layer
- To preserve spatial dimensions, **zero-padding** is used: adding zeros around the border of the input image (**same convolution**)
- Thus, for an input of size $x_h \times x_w$, adding padding of size p_h and p_w results in an output of size $(x_h - f_h + 2p_h + 1) \times (x_w - f_w + 2p_w + 1)$

Striding

Striding

Striding refers to the step size s with which the kernel moves across the input image.

Striding

Striding refers to the step size s with which the kernel moves across the input image.

- Striding is used to reduce redundancy and thus speed up computation.

Striding

Striding refers to the step size s with which the kernel moves across the input image.

- Striding is used to reduce redundancy and thus speed up computation.
- A stride of 1 means the kernel is applied to every possible position.

Striding

Striding refers to the step size s with which the kernel moves across the input image.

- Striding is used to reduce redundancy and thus speed up computation.
- A stride of 1 means the kernel is applied to every possible position.
- A stride of 2 means the kernel is applied every other position, effectively downsampling the feature map.

Striding

Striding refers to the step size s with which the kernel moves across the input image.

- Striding is used to reduce redundancy and thus speed up computation.
- A stride of 1 means the kernel is applied to every possible position.
- A stride of 2 means the kernel is applied every other position, effectively downsampling the feature map.

Striding

Striding refers to the step size s with which the kernel moves across the input image.

- Striding is used to reduce redundancy and thus speed up computation.
- A stride of 1 means the kernel is applied to every possible position.
- A stride of 2 means the kernel is applied every other position, effectively downsampling the feature map.

Thus, given an input of size $x_h \times x_w$ and kernel size of $f_h \times f_w$, if we use a zero-padding of p_h and p_w and strides of s_h and s_w, then the output has size:

Striding

Striding refers to the step size s with which the kernel moves across the input image.

- Striding is used to reduce redundancy and thus speed up computation.
- A stride of 1 means the kernel is applied to every possible position.
- A stride of 2 means the kernel is applied every other position, effectively downsampling the feature map.

Thus, given an input of size $x_h \times x_w$ and kernel size of $f_h \times f_w$, if we use a zero-padding of p_h and p_w and strides of s_h and s_w, then the output has size:

$$\lfloor \frac{x_h - f_h + 2p_h + s_h}{s_h} \rfloor \times \lfloor \frac{x_w - f_w + 2p_w + s_w}{s_w} \rfloor \tag{6}$$

Example structure of a CNN

Example structure of a CNN

Convolutional layers enable the encoding of spatial relationships between neurons (pixels)

Example structure of a CNN

Convolutional layers enable the encoding of spatial relationships between neurons (pixels)

- Weight sharing reduces number of parameters (i.e. same kernel/filter used for each feature map)

Example structure of a CNN

Convolutional layers enable the encoding of spatial relationships between neurons (pixels)

- Weight sharing reduces number of parameters (i.e. same kernel/filter used for each feature map)
- Complexity handled by increasing feature maps

Example structure of a CNN

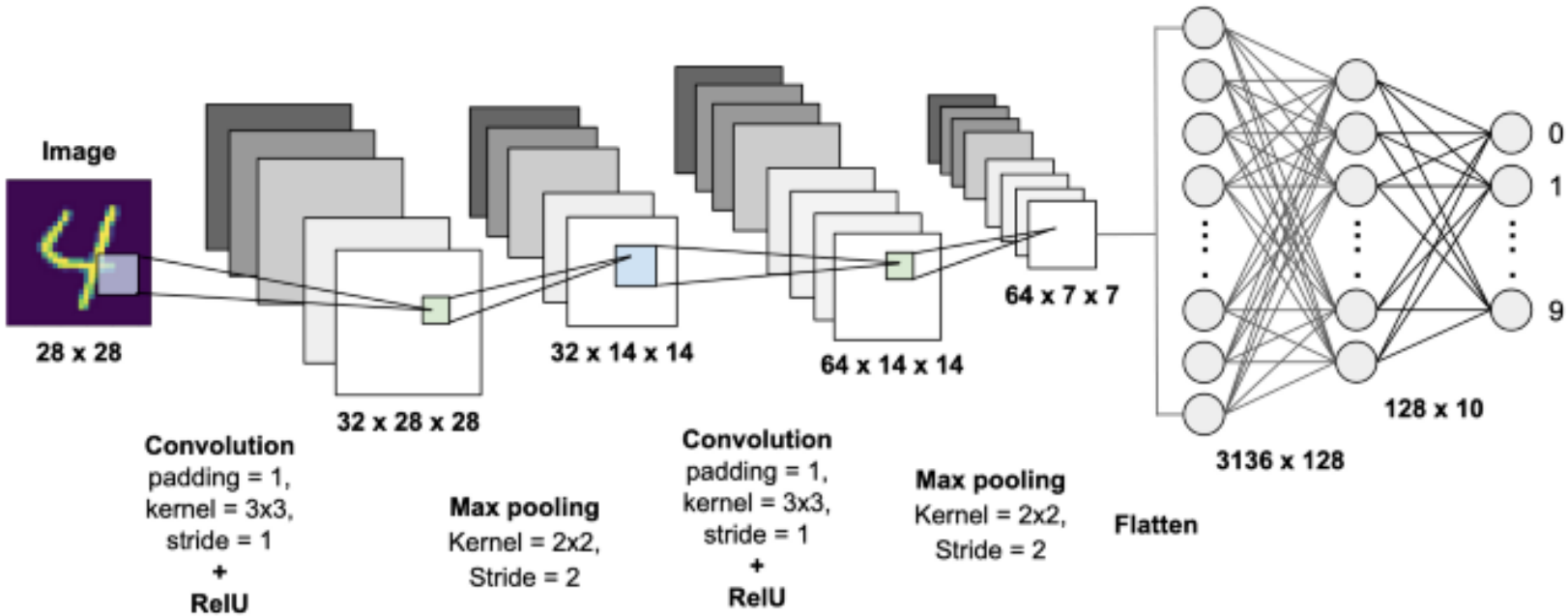
Convolutional layers enable the encoding of spatial relationships between neurons (pixels)

- Weight sharing reduces number of parameters (i.e. same kernel/filter used for each feature map)
- Complexity handled by increasing feature maps
- Multiple convolutional layers can be stacked to add depth

Example structure of a CNN

Convolutional layers enable the encoding of spatial relationships between neurons (pixels)

- Weight sharing reduces number of parameters (i.e. same kernel/filter used for each feature map)
- Complexity handled by increasing feature maps
- Multiple convolutional layers can be stacked to add depth



Source:

Convolutional layer

Convolutional layer

The convolutional layer consists of several filters (kernels), each corresponding to a feature map.

- Input: tensor of size $h_1 \times w_1 \times d_1$
  - For the first convolutional layer: $d_1 = 1$ (single-channel/grayscale image); $D_1 = 3$ (color/RGB image)

Convolutional layer

The convolutional layer consists of several filters (kernels), each corresponding to a feature map.

- Input: tensor of size $h_1 \times w_1 \times d_1$
  - For the first convolutional layer: $d_1 = 1$ (single-channel/grayscale image); $D_1 = 3$ (color/RGB image)
- Hyperparameters:

Convolutional layer

The convolutional layer consists of several filters (kernels), each corresponding to a feature map.

- Input: tensor of size $h_1 \times w_1 \times d_1$
  - For the first convolutional layer: $d_1 = 1$ (single-channel/grayscale image); $D_1 = 3$ (color/RGB image)
- Hyperparameters:
  - number of filters F
  - kernel size (local receptive field) f_h, f_w
  - stride length s
  - amount of zero padding p (typically same on both sides)

Convolutional layer

The convolutional layer consists of several filters (kernels), each corresponding to a feature map.

- Input: tensor of size $h_1 \times w_1 \times d_1$
  - For the first convolutional layer: $d_1 = 1$ (single-channel/grayscale image); $D_1 = 3$ (color/RGB image)
- Hyperparameters:
  - number of filters F
  - kernel size (local receptive field) f_h, f_w
  - stride length s
  - amount of zero padding p (typically same on both sides)
- Output: tensor of size $h_2 \times w_2 \times d_2$:

Convolutional layer

The convolutional layer consists of several filters (kernels), each corresponding to a feature map.

- Input: tensor of size $h_1 \times w_1 \times d_1$
  - For the first convolutional layer: $d_1 = 1$ (single-channel/grayscale image); $D_1 = 3$ (color/RGB image)
- Hyperparameters:
  - number of filters F
  - kernel size (local receptive field) f_h, f_w
  - stride length s
  - amount of zero padding p (typically same on both sides)
- Output: tensor of size $h_2 \times w_2 \times d_2$:
  - $h_2 = (h_1 - f_h + 2p + s)/s$
  - $w_2 = (w_1 - f_w + 2p + s)/s$
  - $d_2 = F$

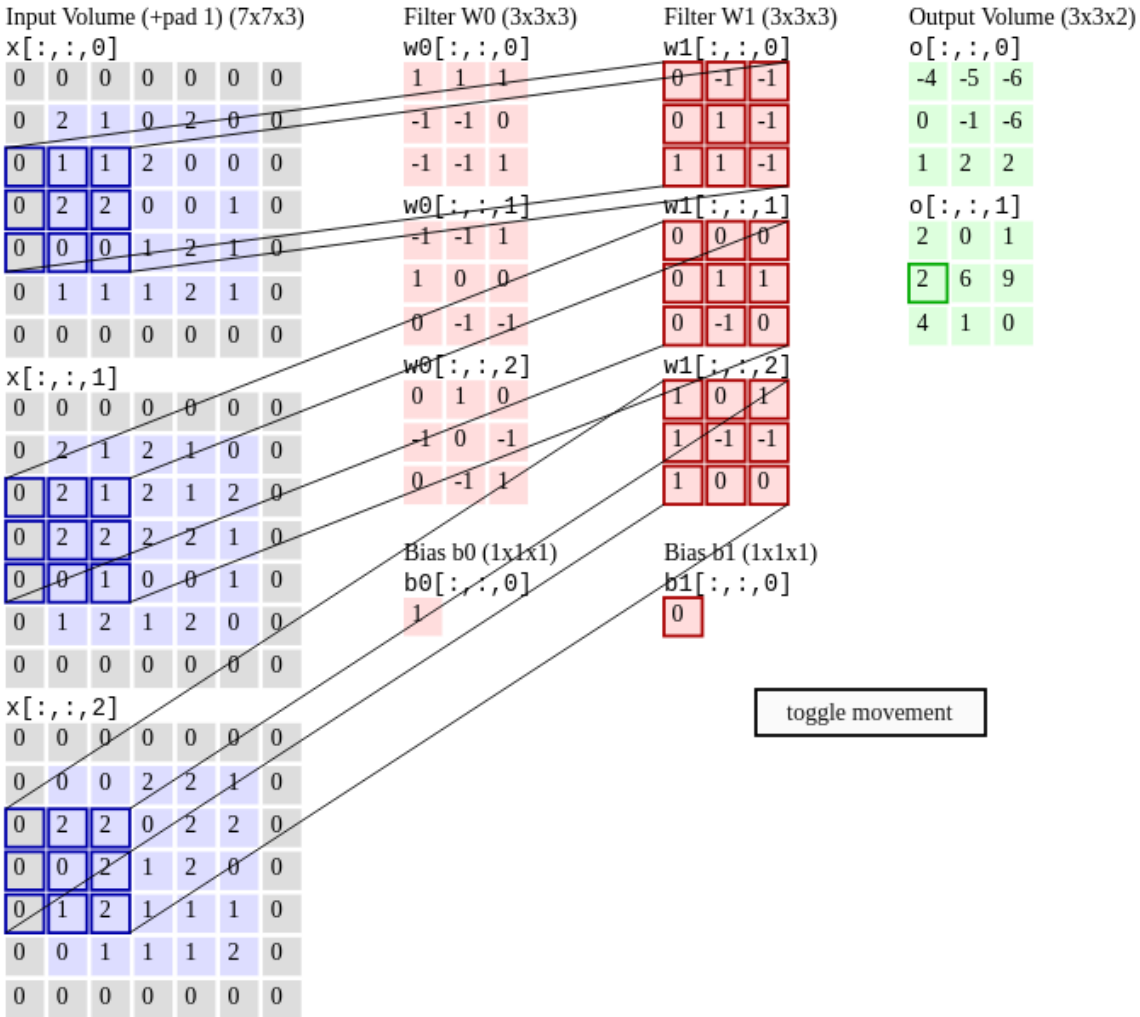
Example: Convolutional layer operation

Example: Convolutional layer operation

$d_1 = 3$ (3-channel input) and $F = 2$ (2 filters).

Example: Convolutional layer operation

$d_1 = 3$ (3-channel input) and $F = 2$ (2 filters).



Source: `https://cs231n.github.io/convolutional-networks/` (animated)

Pooling layer

Pooling layer

Pooling spatially downsamples the output of a convolutional layer via a summary statistic of nearby values at a given location

Pooling layer

Pooling spatially downsamples the output of a convolutional layer via a summary statistic of nearby values at a given location

- Introduced to ensure shift/translation invariance in a trained network

Pooling layer

Pooling spatially downsamples the output of a convolutional layer via a summary statistic of nearby values at a given location

- Introduced to ensure shift/translation invariance in a trained network
- Leads to robust performance

Pooling layer

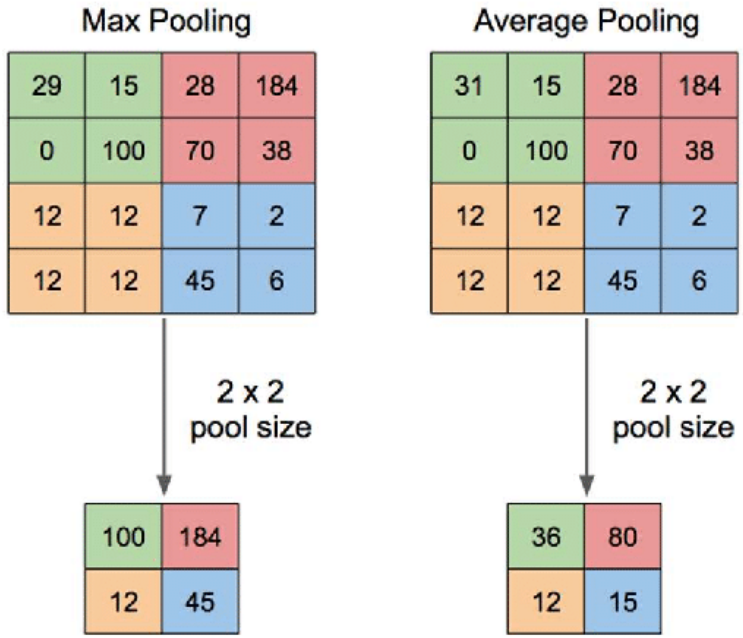
Pooling spatially downsamples the output of a convolutional layer via a summary statistic of nearby values at a given location

- Introduced to ensure shift/translation invariance in a trained network
- Leads to robust performance
- Typically, **max** pooling or **average** pooling is used

Pooling layer

Pooling spatially downsamples the output of a convolutional layer via a summary statistic of nearby values at a given location

- Introduced to ensure shift/translation invariance in a trained network
- Leads to robust performance
- Typically, **max** pooling or **average** pooling is used



Training hyperparameters in a CNN

Several decisions must be made in selecting hyperparameters for training a CNN.

- Number of convolutional layers and feature maps in each layer

Training hyperparameters in a CNN

Several decisions must be made in selecting hyperparameters for training a CNN.

- Number of convolutional layers and feature maps in each layer
- Convolutional kernel size

Training hyperparameters in a CNN

Several decisions must be made in selecting hyperparameters for training a CNN.

- Number of convolutional layers and feature maps in each layer
- Convolutional kernel size
- Stride length (spacing of filters)

Training hyperparameters in a CNN

Several decisions must be made in selecting hyperparameters for training a CNN.

- Number of convolutional layers and feature maps in each layer
- Convolutional kernel size
- Stride length (spacing of filters)
- Choice of pooling function (max, average, etc)

Training hyperparameters in a CNN

Several decisions must be made in selecting hyperparameters for training a CNN.

- Number of convolutional layers and feature maps in each layer
- Convolutional kernel size
- Stride length (spacing of filters)
- Choice of pooling function (max, average, etc)
- Number of dense layers

Training hyperparameters in a CNN

Several decisions must be made in selecting hyperparameters for training a CNN.

- Number of convolutional layers and feature maps in each layer
- Convolutional kernel size
- Stride length (spacing of filters)
- Choice of pooling function (max, average, etc)
- Number of dense layers
- Activation function in each layer (ReLU, tanh, etc)

Normalization layers

Normalization layers

Normalization layers are often used to improve training performance and stability.

Normalization layers

Normalization layers are often used to improve training performance and stability.

- Batch Normalization (BN): normalizes activations within a mini-batch

$$\tilde{\boldsymbol{z}}_n = \boldsymbol{\gamma} \odot \hat{\boldsymbol{z}}_n + \boldsymbol{\beta}, \quad \hat{\boldsymbol{z}}_n = \frac{\boldsymbol{z}_n - \boldsymbol{\mu}_{\text{batch}}}{\sqrt{\boldsymbol{\sigma}^2_{\text{batch}} + \epsilon}}$$

- Layer Normalization (LN): normalizes across features for each training example
- Instance Normalization (IN): normalizes across spatial dimensions for each channel

CNN architectures for classification

CNN architectures for classification

Various high-performing deep architectures have been developed in recent years that can be adapted for other problems:

CNN architectures for classification

Various high-performing deep architectures have been developed in recent years that can be adapted for other problems:

- LeNet

CNN architectures for classification

Various high-performing deep architectures have been developed in recent years that can be adapted for other problems:

- LeNet
- AlexNet

CNN architectures for classification

Various high-performing deep architectures have been developed in recent years that can be adapted for other problems:

- LeNet
- AlexNet
- VGGNet

CNN architectures for classification

Various high-performing deep architectures have been developed in recent years that can be adapted for other problems:

- LeNet
- AlexNet
- VGGNet
- ResNet

CNN architectures for classification

Various high-performing deep architectures have been developed in recent years that can be adapted for other problems:

- LeNet
- AlexNet
- VGGNet
- ResNet
- Inception

LeNet-5

LeNet-5

- Created by Yann LeCun in 1998

LeNet-5

- Created by Yann LeCun in 1998
- 5 convolutional layers

LeNet-5

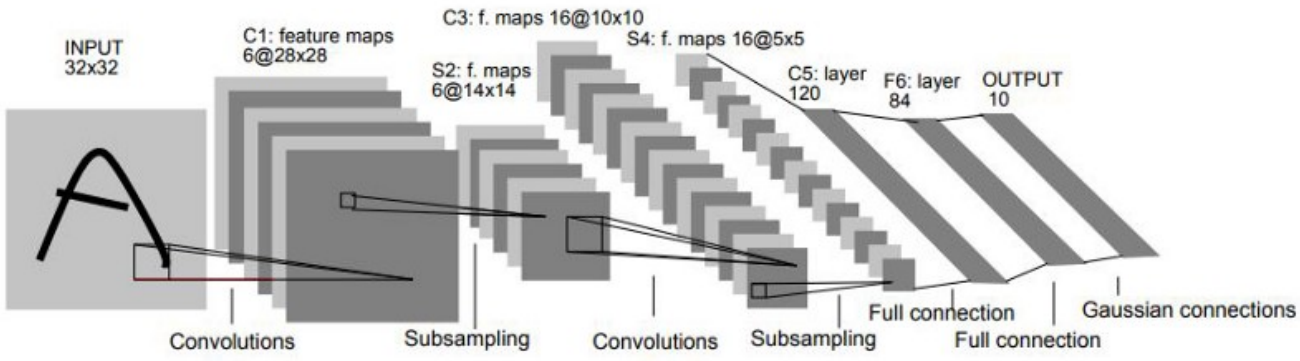
- Created by Yann LeCun in 1998
- 5 convolutional layers
- Developed to classify handwritten digits (technology adopted by USPS and banks)

LeNet-5

- Created by Yann LeCun in 1998
- 5 convolutional layers
- Developed to classify handwritten digits (technology adopted by USPS and banks)

LeNet-5

- Created by Yann LeCun in 1998
- 5 convolutional layers
- Developed to classify handwritten digits (technology adopted by USPS and banks)



Source: https://www.datasciencecentral.com/lenet-5-a-classic-cnn-architecture/

AlexNet

AlexNet

- ReLU activations instead of tanh

AlexNet

- ReLU activations instead of tanh
- Dropout (instead of weight decay)

AlexNet

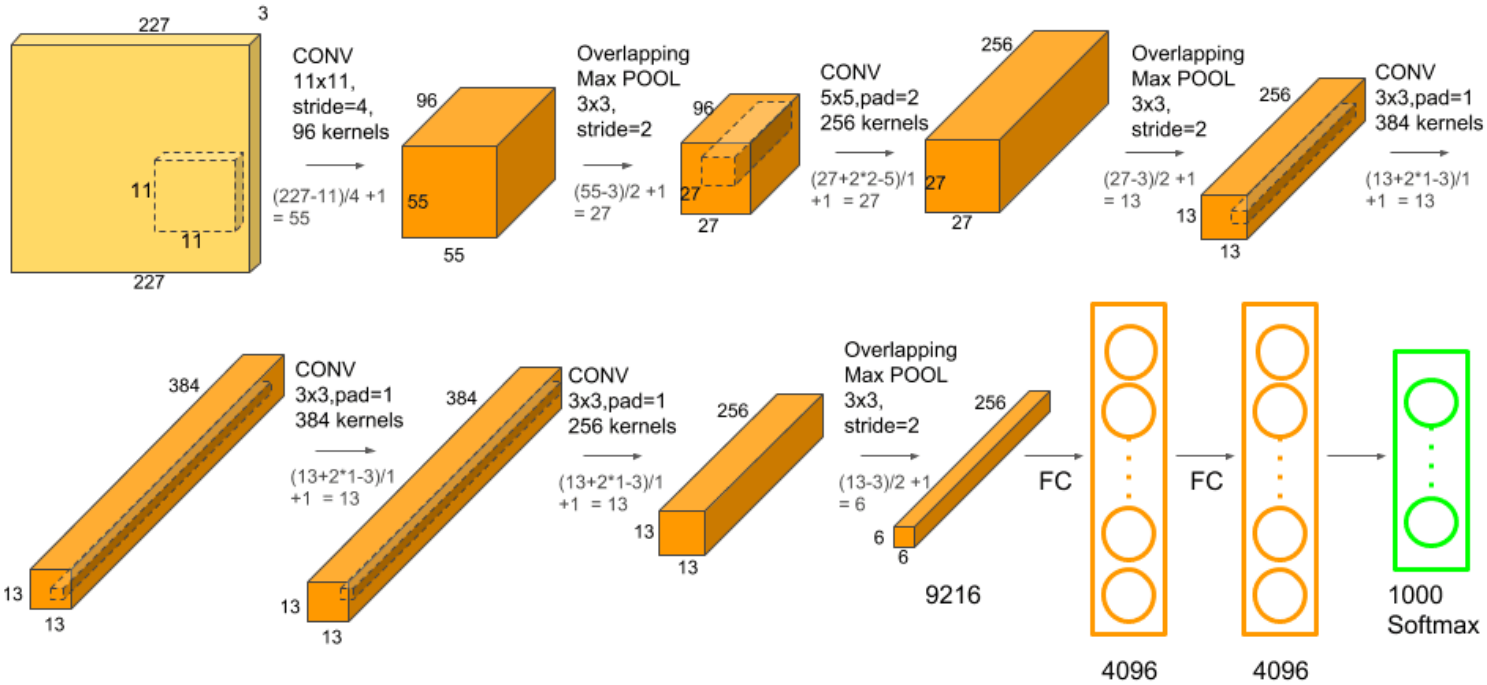
- ReLU activations instead of tanh
- Dropout (instead of weight decay)
- Stacking of convolutional layers for larger receptive fields

AlexNet

- ReLU activations instead of tanh
- Dropout (instead of weight decay)
- Stacking of convolutional layers for larger receptive fields

AlexNet

- ReLU activations instead of tanh
- Dropout (instead of weight decay)
- Stacking of convolutional layers for larger receptive fields



Source: https://neurohive.io/en/popular-networks/alexnet-imagenet-classification-with-deep-convolutional-neural-networks/

VGGNet

VGGNet

- Developed by Visual Geometry Group at Oxford

VGGNet

- Developed by Visual Geometry Group at Oxford
- Smaller receptive fields than AlexNet

VGGNet

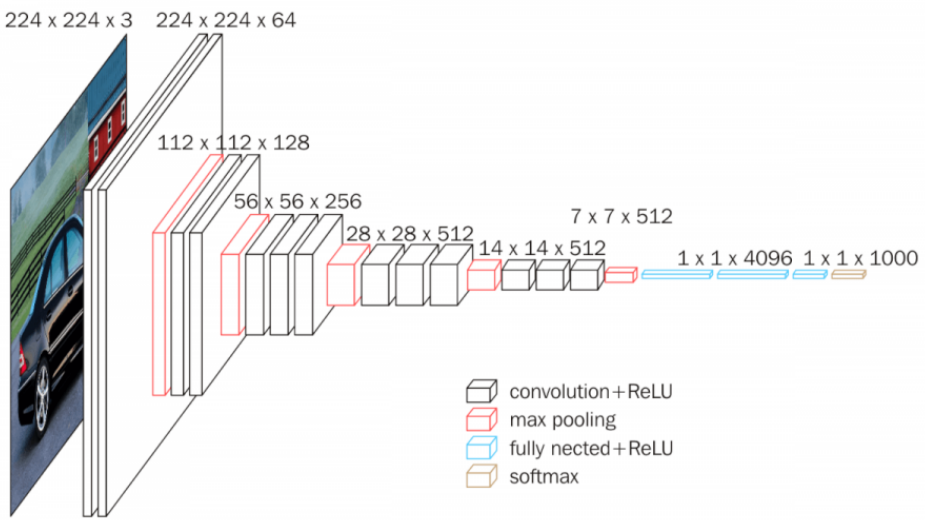
- Developed by Visual Geometry Group at Oxford
- Smaller receptive fields than AlexNet
- 16-19 layers deep

VGGNet

- Developed by Visual Geometry Group at Oxford
- Smaller receptive fields than AlexNet
- 16-19 layers deep

VGGNet

- Developed by Visual Geometry Group at Oxford
- Smaller receptive fields than AlexNet
- 16-19 layers deep



Source: `https://www.kaggle.com/code/blurredmachine/vggnet-16-architecture-a-complete-guide`

ResNet

ResNet

- Residual blocks: training nonlinear layer $\mathcal{F}$ to fit residual:

ResNet

- Residual blocks: training nonlinear layer $\mathcal{F}$ to fit residual:

$$\boldsymbol{x}_{\ell+1} = \varphi(\boldsymbol{x}_\ell + \mathcal{F}(\boldsymbol{x}_\ell)) \tag{7}$$

ResNet

- Residual blocks: training nonlinear layer $\mathcal{F}$ to fit residual:

$$\boldsymbol{x}_{\ell+1} = \varphi(\boldsymbol{x}_\ell + \mathcal{F}(\boldsymbol{x}_\ell)) \tag{7}$$

instead of total output:

ResNet

- Residual blocks: training nonlinear layer $\mathcal{F}$ to fit residual:

$$\boldsymbol{x}_{\ell+1} = \varphi(\boldsymbol{x}_\ell + \mathcal{F}(\boldsymbol{x}_\ell)) \tag{7}$$

instead of total output:

$$\boldsymbol{x}_{\ell+1} = \mathcal{F}(\boldsymbol{x}_\ell) \tag{8}$$

ResNet

- Residual blocks: training nonlinear layer $\mathcal{F}$ to fit residual:

$$\boldsymbol{x}_{\ell+1} = \varphi(\boldsymbol{x}_\ell + \mathcal{F}(\boldsymbol{x}_\ell)) \tag{7}$$

  instead of total output:

$$\boldsymbol{x}_{\ell+1} = \mathcal{F}(\boldsymbol{x}_\ell) \tag{8}$$

- The introduction of skip connections allow for very deep networks (e.g. up to 1001 layers)

ResNet

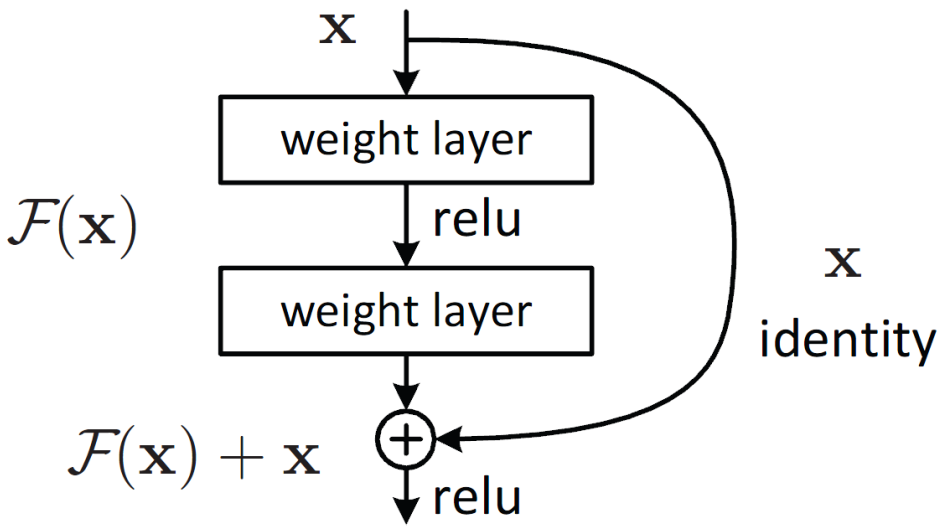
- Residual blocks: training nonlinear layer $\mathcal{F}$ to fit residual:

$$\boldsymbol{x}_{\ell+1} = \varphi(\boldsymbol{x}_\ell + \mathcal{F}(\boldsymbol{x}_\ell)) \tag{7}$$

instead of total output:

$$\boldsymbol{x}_{\ell+1} = \mathcal{F}(\boldsymbol{x}_\ell) \tag{8}$$

- The introduction of skip connections allow for very deep networks (e.g. up to 1001 layers)



Source: `https://paperswithcode.com/method/residual-connection`

Inception (GoogLeNet)

Inception (GoogLeNet)

- First introduction of inception module (v1)

Inception (GoogLeNet)

- First introduction of inception module (v1)
- Allows for multiple filter sizes at the same level

Inception (GoogLeNet)

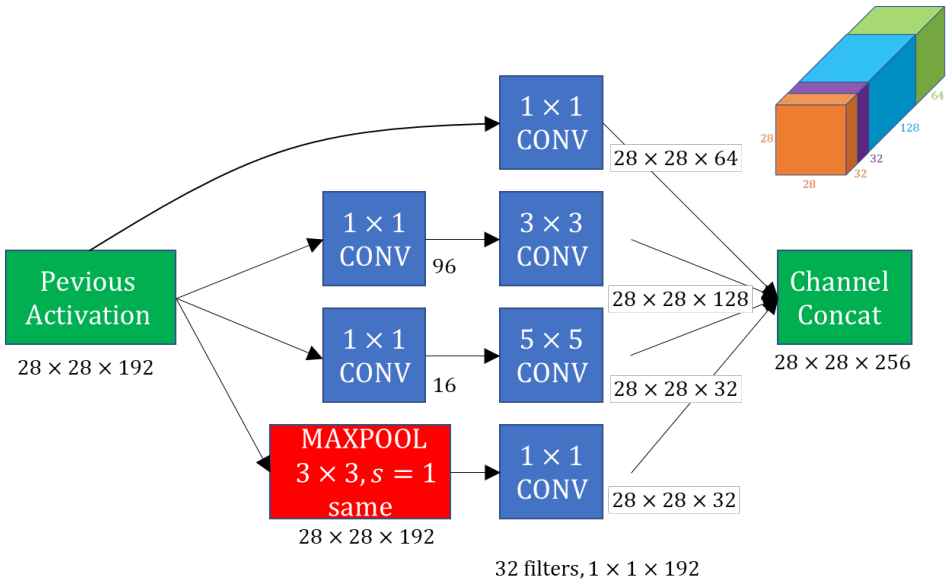
- First introduction of inception module (v1)
- Allows for multiple filter sizes at the same level
- 1×1 convolutions reduce dimensionality (channels)

Inception (GoogLeNet)

- First introduction of inception module (v1)
- Allows for multiple filter sizes at the same level
- 1×1 convolutions reduce dimensionality (channels)

Inception (GoogLeNet)

- First introduction of inception module (v1)
- Allows for multiple filter sizes at the same level
- 1×1 convolutions reduce dimensionality (channels)



Source: `https://datahacker.rs/building-inception-network/`

Image tagging

Image tagging

- Assign multiple labels to single image (multi-label prediction; each tag predicted independently)

Image tagging

- Assign multiple labels to single image (multi-label prediction; each tag predicted independently)

Image tagging

- Assign multiple labels to single image (multi-label prediction; each tag predicted independently)
- Output space: $\mathcal{Y} = \{0, 1\}^C$, where C is the number of tag types

Image tagging

- Assign multiple labels to single image (multi-label prediction; each tag predicted independently)
- Output space: $\mathcal{Y} = \{0, 1\}^C$, where C is the number of tag types

Image tagging

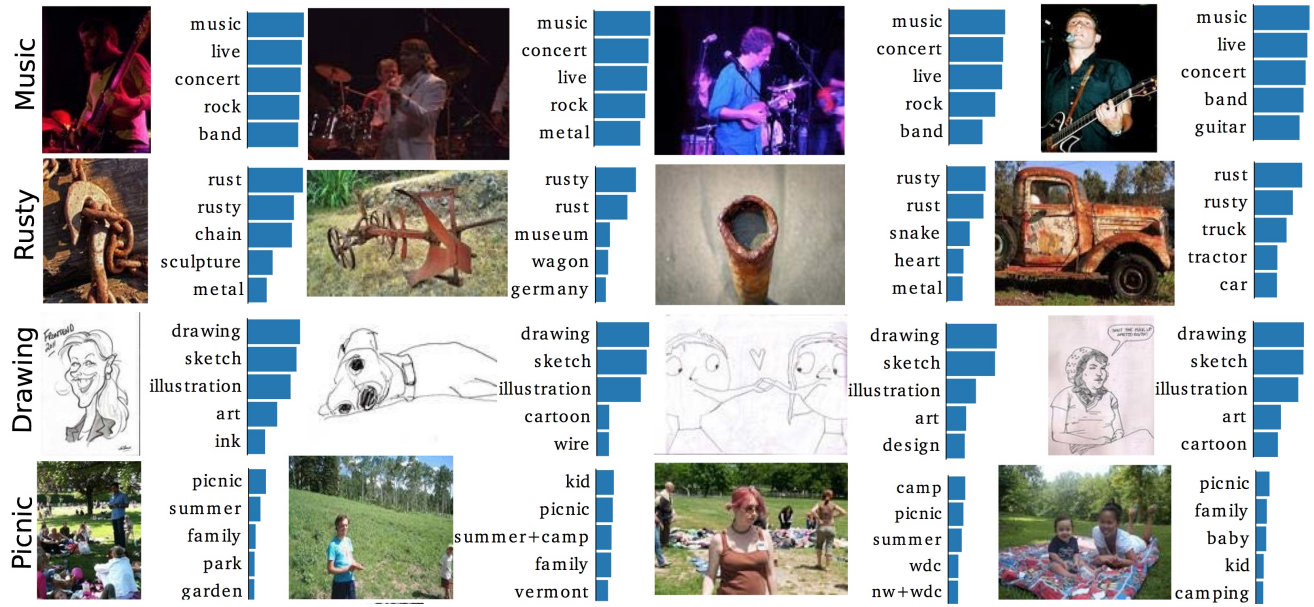
- Assign multiple labels to single image (multi-label prediction; each tag predicted independently)
- Output space: $\mathcal{Y} = \{0, 1\}^C$, where C is the number of tag types
- Final layer of CNN has logistic sigmoid units (activations) instead of softmax

Image tagging

- Assign multiple labels to single image (multi-label prediction; each tag predicted independently)
- Output space: $\mathcal{Y} = \{0, 1\}^C$, where C is the number of tag types
- Final layer of CNN has logistic sigmoid units (activations) instead of softmax

Image tagging

- Assign multiple labels to single image (multi-label prediction; each tag predicted independently)
- Output space: $\mathcal{Y} = \{0, 1\}^C$, where C is the number of tag types
- Final layer of CNN has logistic sigmoid units (activations) instead of softmax



Source: `https://izadinia.github.io/files/DeepTagging-mmcommons.pdf`

Object detection

Object detection

- Variable number of objects of interest in a given image (open world problem)

Object detection

- Variable number of objects of interest in a given image (open world problem)
- Returns set of labeled bounding boxes around objects of interest

Object detection

- Variable number of objects of interest in a given image (open world problem)
- Returns set of labeled bounding boxes around objects of interest
- Popular models for object detection:

Object detection

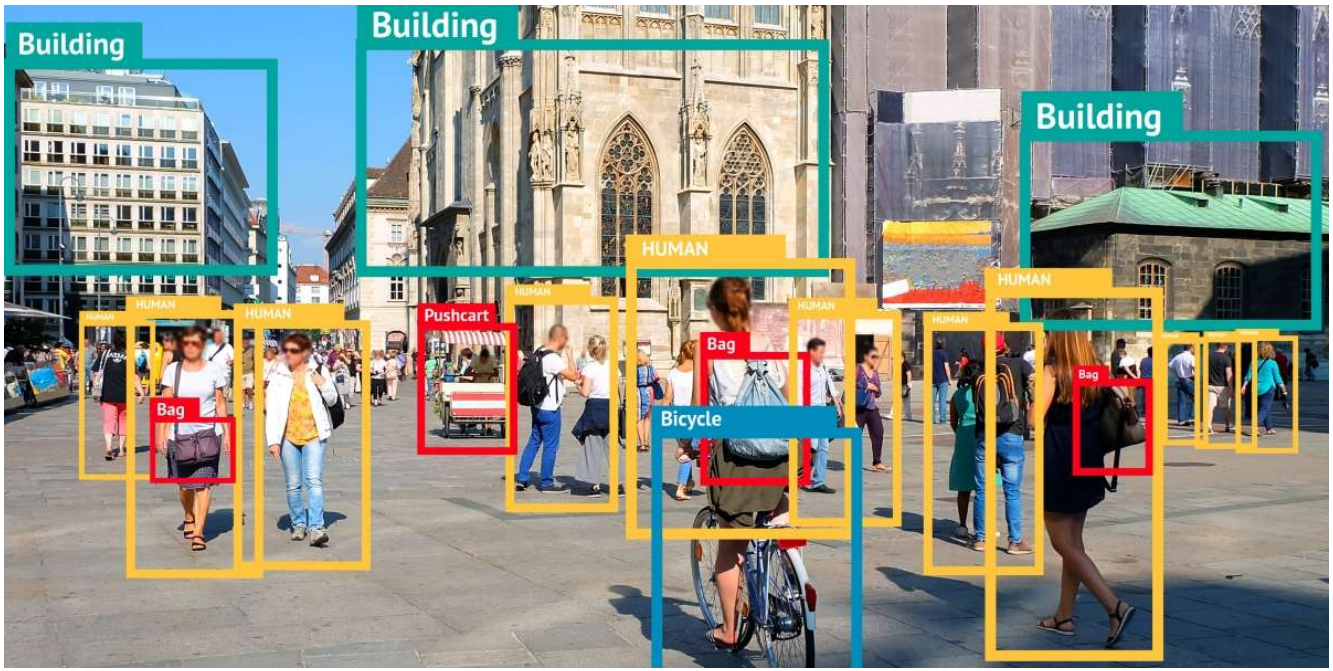
- Variable number of objects of interest in a given image (open world problem)
- Returns set of labeled bounding boxes around objects of interest
- Popular models for object detection:
  - YOLO (You Only Look Once)

Object detection

- Variable number of objects of interest in a given image (open world problem)
- Returns set of labeled bounding boxes around objects of interest
- Popular models for object detection:
  - YOLO (You Only Look Once)
  - SSD (Single shot detector)

Object detection

- Variable number of objects of interest in a given image (open world problem)
- Returns set of labeled bounding boxes around objects of interest
- Popular models for object detection:
  - YOLO (You Only Look Once)
  - SSD (Single shot detector)



Source: https://aigeekprogrammer.com/yolo-fast-object-detection-and-classification/

Semantic segmentation

Semantic segmentation

- Predicts a class label for each pixel rather than an image, e.g. $y_i \in \{1, \ldots, c\}$ where 1≡sky, 2≡sheep, 3≡grass, etc

Semantic segmentation

- Predicts a class label for each pixel rather than an image, e.g. $y_i \in \{1, \ldots, c\}$ where 1≡sky, 2≡sheep, 3≡grass, etc
- Achieved via encoder-decoder network (captures high-level features in 2D bottleneck)

Semantic segmentation

- Predicts a class label for each pixel rather than an image, e.g. $y_i \in \{1, \ldots, c\}$ where 1≡sky, 2≡sheep, 3≡grass, etc
- Achieved via encoder-decoder network (captures high-level features in 2D bottleneck)
- Popular model: U-Net

Semantic segmentation

- Predicts a class label for each pixel rather than an image, e.g. $y_i \in \{1, \ldots, c\}$ where 1≡sky, 2≡sheep, 3≡grass, etc
- Achieved via encoder-decoder network (captures high-level features in 2D bottleneck)
- Popular model: U-Net

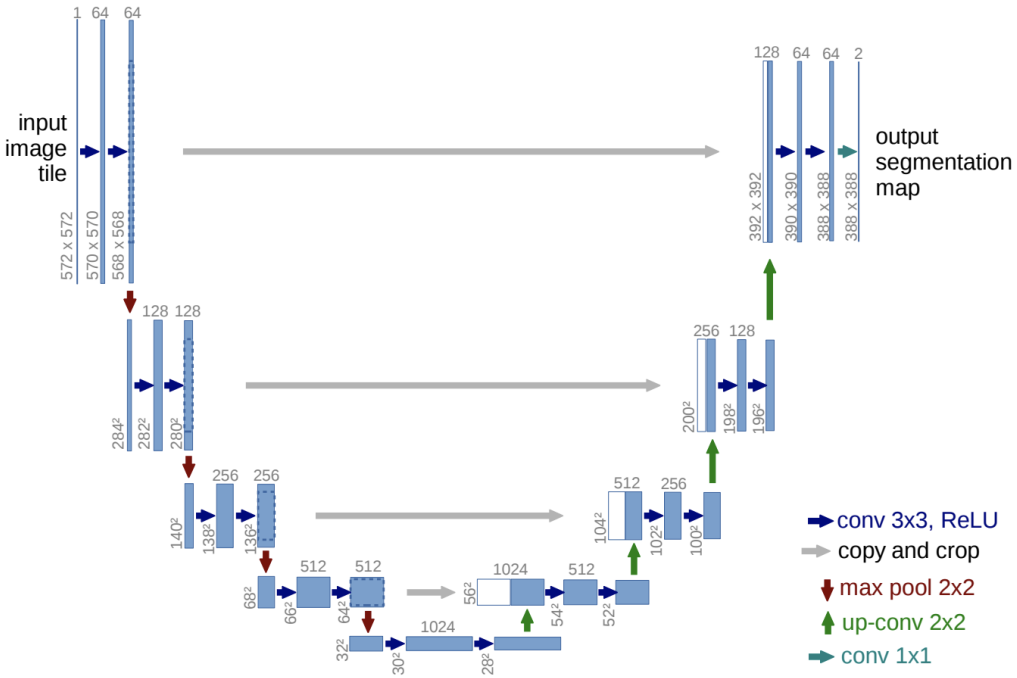
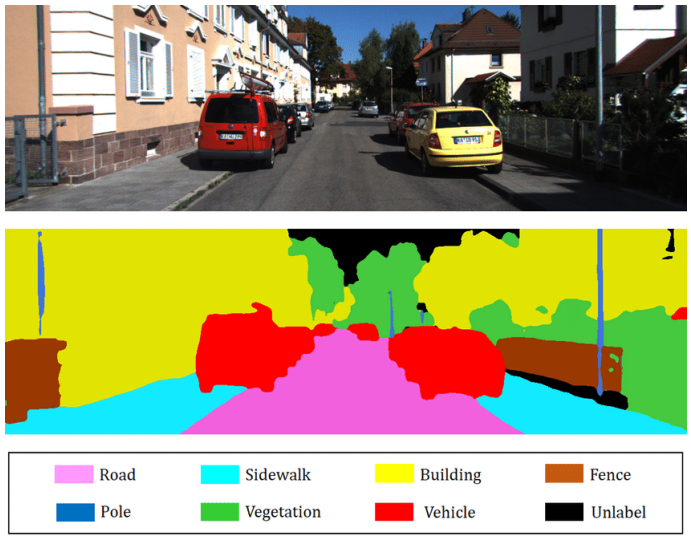


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.



Source:

`https://towardsai.net/p/l/machine-learning-7`

Instance segmentation

Instance segmentation

- Combines object detection and semantic segmentation

Instance segmentation

- Combines object detection and semantic segmentation
- For object instance, a *2d shape mask* is predicted instead of the bounding box

Instance segmentation

- Combines object detection and semantic segmentation
- For object instance, a *2d shape mask* is predicted instead of the bounding box
- Achieved by applying **semantic segmentation** to each detected box in order to label each pixel as foreground or background

Instance segmentation

- Combines object detection and semantic segmentation
- For object instance, a *2d shape mask* is predicted instead of the bounding box
- Achieved by applying **semantic segmentation** to each detected box in order to label each pixel as foreground or background
- Popular models:

Instance segmentation

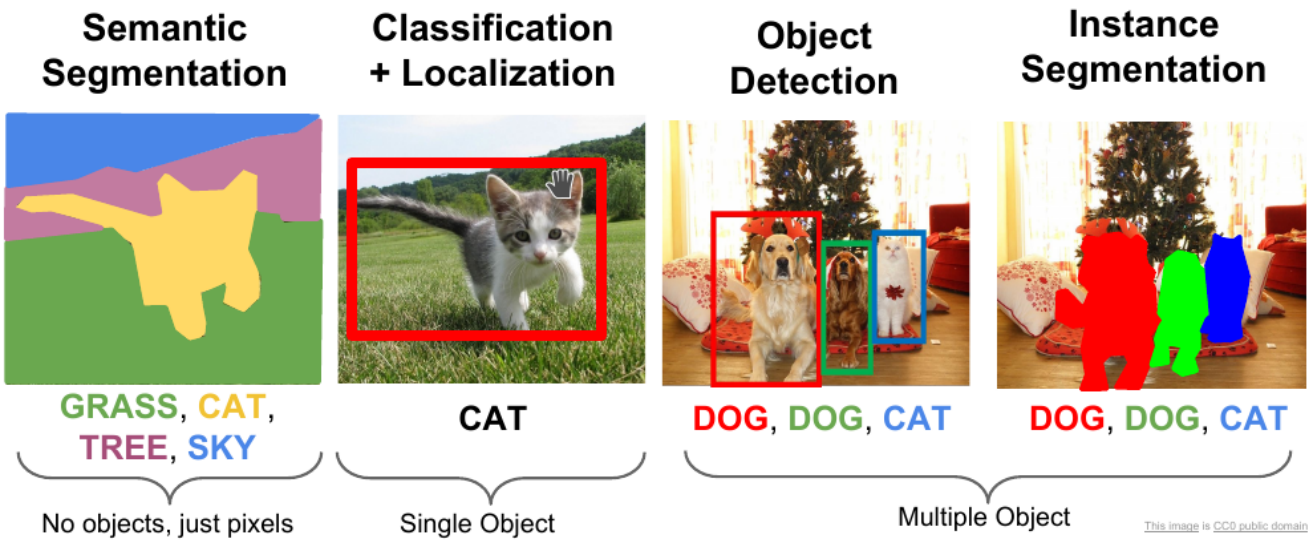
- Combines object detection and semantic segmentation
- For object instance, a *2d shape mask* is predicted instead of the bounding box
- Achieved by applying **semantic segmentation** to each detected box in order to label each pixel as foreground or background
- Popular models:
  - Mask R-CNN

Instance segmentation

- Combines object detection and semantic segmentation
- For object instance, a *2d shape mask* is predicted instead of the bounding box
- Achieved by applying **semantic segmentation** to each detected box in order to label each pixel as foreground or background
- Popular models:
  - Mask R-CNN
  - Faster R-CNN

Instance segmentation

- Combines object detection and semantic segmentation
- For object instance, a *2d shape mask* is predicted instead of the bounding box
- Achieved by applying **semantic segmentation** to each detected box in order to label each pixel as foreground or background
- Popular models:
  - Mask R-CNN
  - Faster R-CNN



Source: `https://kharshit.github.io/blog/2019/08/23/quick-intro-to-instance-segmentation`

Human pose estimation

Human pose estimation

- Predicts location of fixed set of skeletal keypoints (in 2D or 3D)

Human pose estimation

- Predicts location of fixed set of skeletal keypoints (in 2D or 3D)
- Popular models:

Human pose estimation

- Predicts location of fixed set of skeletal keypoints (in 2D or 3D)
- Popular models:
  - PersonLab

Human pose estimation

- Predicts location of fixed set of skeletal keypoints (in 2D or 3D)
- Popular models:
  - PersonLab
  - OpenPose

Human pose estimation

- Predicts location of fixed set of skeletal keypoints (in 2D or 3D)
- Popular models:
  - PersonLab
  - OpenPose

Source: `https://frl.nyu.edu/tricking-openpose/`

Summary

- Basic convolutional neural networks for image classification consist of:
  - input layer
  - convolutional layers (multiple filters=feature maps)
  - pooling layers
  - dense/fully-connected layers
  - output layer
- Visualize and experiment with a CNN for handwritten digit recognition: `https://www.cs.ryerson.ca/~aharley/vis/conv/`
- Reading: **PMLI** 14; **DL** 9