Convolutional Neural Networks

Based on slides by Gilles Louppe, Fei-Fei Li, and Justin Johnson
Let us consider the first layer of a MLP taking images as input. What are the problems with this architecture?
Issues

- Too many parameters: 100 × 784 + 100.
  - What if images are 640 × 480 × 3?
  - What if the first layer counts 1000 units?
- Spatial organization of the input is destroyed.
- The network is not invariant to transformations (e.g., translation).
Weight Sharing

Instead, let us only keep a **sparse** set of connections, where all weights having the same color are **shared**.

- The resulting operation can be seen as **shifting** the same weight triplet (kernel).
- The set of inputs seen by each unit is its **receptive field**.

$\Rightarrow$ This is a **1D convolution**, which can be generalized to more dimensions.
Convolutions

For one-dimensional tensors, given an input vector $\mathbf{x} \in \mathbb{R}^W$ and a convolutional kernel $\mathbf{u} \in \mathbb{R}^w$, the discrete convolution $\mathbf{u} \star \mathbf{x}$ is a vector of size $W - w + 1$ such that

$$(\mathbf{u} \star \mathbf{x})[i] = \sum_{m=0}^{w-1} u_m x_{m+i}.$$ 

- Technically, $\star$ denotes the cross-correlation operator.
- However, most machine learning libraries call it convolution.
Convolutions
Convolutions generalize to multi-dimensional tensors:

- In its most usual form, a convolution takes as input a 3D tensor $x \in \mathbb{R}^{C \times H \times W}$, called the input feature map.

- A kernel $u \in \mathbb{R}^{C \times h \times w}$ slides across the input feature map, along its height and width. The size $h \times w$ is the size of the receptive field.

- At each location, the element-wise product between the kernel and the input elements it overlaps is computed and the results are summed up.
Convolutions
Convolutions

- The final output $o$ is a 2D tensor of size $(H - h + 1) \times (W - w + 1)$ called the output feature map and such that:

$$o_{j,i} = b_{j,i} + \sum_{c=0}^{C-1} (u_c \ast x_c)[j,i] = b_{j,i} + \sum_{c=0}^{C-1} \sum_{n=0}^{h-1} \sum_{m=0}^{w-1} u_{c,n,m} x_{c,n+j,m+i}$$

where $u$ and $b$ are shared parameters to learn.

- $D$ convolutions can be applied in the same way to produce a $D \times (H - h + 1) \times (W - w + 1)$ feature map, where $D$ is the depth.
Convolutions

A 32x32x3 image is convolved with a 5x5x3 filter. Convolving the filter with the image involves sliding the filter over the image spatially, computing dot products.
**Convolutions**

- A 32x32x3 image
- A 5x5x3 filter

*Filters always extend the full depth of the input volume*

*Convolve* the filter with the image, i.e., "slide over the image spatially, computing dot products"
Convolutions

- A 32x32x3 image
- A 5x5x3 filter $w$

1 number:
The result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. $5 \times 5 \times 3 = 75$-dimensional dot product + bias)

$$w^T x + b$$
Convolutions

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map
Convolutions

let’t add one more filter, the green one
Convolutions

with 6 separate filters, we’ll get 6 separate activation maps
Convolution as a Matrix Multiplication: the Toeplitz Matrix

Any \( n \times n \) matrix \( A \) of the form

\[
A = \begin{bmatrix}
    a_0 & a_{-1} & a_{-2} & \cdots & \cdots & a_{-(n-1)} \\
    a_1 & a_0 & a_{-1} & \ddots & & \\
    a_2 & a_1 & \ddots & \ddots & \ddots & \\
    \vdots & \ddots & \ddots & \ddots & \ddots & \\
    \vdots & & \ddots & \ddots & a_{-1} & a_{-2} \\
    a_{n-1} & \cdots & \cdots & a_2 & a_1 & a_0 \\
\end{bmatrix}
\]

is a Toeplitz matrix. If the \( i,j \) element of \( A \) is denoted \( A_{i,j} \), then we have

\[
A_{i,j} = A_{i+1,j+1} = a_{i-j}.
\]

A Toeplitz matrix is not necessarily square.

1D Convolution as a Matrix Multiplication

\[ y = h \ast x = \begin{bmatrix} h_1 & 0 & \ldots & 0 & 0 \\ h_2 & h_1 & \ldots & \vdots & \vdots \\ h_3 & h_2 & \ldots & 0 & 0 \\ \vdots & \vdots & \ldots & h_1 & 0 \\ h_{m-1} & \vdots & \ldots & h_2 & h_1 \\ h_m & h_{m-1} & \vdots & \vdots & h_2 \\ 0 & h_m & \ldots & h_{m-2} & \vdots \\ 0 & 0 & \ldots & h_{m-1} & h_{m-2} \\ \vdots & \vdots & \vdots & h_m & h_{m-1} \\ 0 & 0 & 0 & \ldots & h_m \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} \]

\[ y^T = [h_1 \ h_2 \ h_3 \ \ldots \ h_{m-1} \ h_m] \begin{bmatrix} x_1 & x_2 & x_3 & \ldots & x_n & 0 & 0 & 0 & \ldots & 0 \\ 0 & x_1 & x_2 & x_3 & \ldots & x_n & 0 & 0 & \ldots & 0 \\ 0 & 0 & x_1 & x_2 & x_3 & \ldots & x_n & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & 0 \\ 0 & \ldots & 0 & 0 & x_1 & \ldots & x_{n-2} & x_{n-1} & x_n & \vdots \\ 0 & \ldots & 0 & 0 & 0 & x_1 & \ldots & x_{n-2} & x_{n-1} & x_n \end{bmatrix} \]
2D Convolution as a Matrix Multiplication

\[
\mathbf{u} \ast \mathbf{x} = \begin{pmatrix} 1 & 4 & 1 \\ 1 & 4 & 3 \\ 3 & 3 & 1 \end{pmatrix} \ast \begin{pmatrix} 4 & 5 & 8 & 7 \\ 1 & 8 & 8 & 8 \\ 3 & 6 & 6 & 4 \\ 6 & 5 & 7 & 8 \end{pmatrix} = \begin{pmatrix} 122 & 148 \\ 126 & 134 \end{pmatrix}
\]

The convolution operation can be equivalently re-expressed as a single matrix multiplication:

- the convolutional kernel \( \mathbf{u} \) is rearranged as a sparse Toeplitz circulant matrix, called the convolution matrix:

\[
\mathbf{U} = \begin{pmatrix} 1 & 4 & 1 & 0 & 1 & 4 & 3 & 0 & 3 & 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 & 0 & 1 & 4 & 3 & 0 & 3 & 3 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 4 & 1 & 0 & 1 & 4 & 3 & 0 & 3 & 3 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 4 & 1 & 0 & 1 & 4 & 3 & 0 & 3 & 3 & 1 \end{pmatrix}
\]

- the input \( \mathbf{x} \) is flattened row by row, from top to bottom:

\[
\mathbf{v}(\mathbf{x}) = (4 \ 5 \ 8 \ 7 \ 1 \ 8 \ 8 \ 8 \ 3 \ 6 \ 6 \ 4 \ 6 \ 5 \ 7 \ 8)^T
\]

Then,

\[
\mathbf{Uv}(\mathbf{x}) = (122 \ 148 \ 126 \ 134)^T
\]

which we can reshape to a \( 2 \times 2 \) matrix to obtain \( \mathbf{u} \ast \mathbf{x} \).
2D Convolution as a Matrix Multiplication

The same procedure generalizes to $\mathbf{x} \in \mathbb{R}^{H \times W}$ and convolutional kernel $\mathbf{u} \in \mathbb{R}^{h \times w}$, such that:

- the convolutional kernel is rearranged as a sparse Toeplitz circulant matrix $\mathbf{U}$ of shape $(H - h + 1)(W - w + 1) \times HW$ where

- the input $\mathbf{x}$ is flattened into a column vector $\mathbf{v}(\mathbf{x})$ of shape $HW \times 1$;

- the output feature map $\mathbf{u} \ast \mathbf{x}$ is obtained by reshaping the $(H - h + 1)(W - w + 1) \times 1$ column vector $\mathbf{U} \mathbf{v}(\mathbf{x})$ as a $(H - h + 1) \times (W - w + 1)$ matrix.

Therefore, a convolutional layer is a special case of a fully connected layer:

$$\mathbf{h} = \mathbf{u} \ast \mathbf{x} \Leftrightarrow \mathbf{v}(\mathbf{h}) = \mathbf{U} \mathbf{v}(\mathbf{x}) \Leftrightarrow \mathbf{v}(\mathbf{h}) = \mathbf{W}^T \mathbf{v}(\mathbf{x})$$
Computational Graph

\[ x \xrightarrow{\times} h \]

\[ x \xrightarrow{\text{flatten}} \xrightarrow{\text{matmul}} \xrightarrow{\text{reshape}} h \]
Many Photoshop Effects are just Convolutions
Many Photoshop Effects are just Convolutions

Gaussian Convolution:

blurs image to represent average (smoothing)
Many Photoshop Effects are just Convolutions

"Emboss" filter:

\[
\begin{bmatrix}
-2 & -1 & 0 \\
-1 & 0 & 1 \\
0 & 1 & 2 \\
\end{bmatrix}
\]
Strides

- The **stride** specifies the size of the step for the convolution operator.
- This parameter reduces the size of the output map.
With Stride 1

7x7 input (spatially) assume 3x3 filter

=> 5x5 output
7x7 input (spatially)
assume 3x3 filter
applied with stride 2
=> 3x3 output!
With Stride 3

7x7 input (spatially) assume 3x3 filter applied with stride 3?

doesn’t fit! cannot apply 3x3 filter on 7x7 input with stride 3.
Output Size

<table>
<thead>
<tr>
<th>N</th>
<th>F</th>
<th>N</th>
</tr>
</thead>
</table>

Output size:
\((N - F) / \text{stride} + 1\)

e.g. \(N = 7, F = 3\):
- \(\text{stride 1} \Rightarrow (7 - 3)/1 + 1 = 5\)
- \(\text{stride 2} \Rightarrow (7 - 3)/2 + 1 = 3\)
- \(\text{stride 3} \Rightarrow (7 - 3)/3 + 1 = 2.33\)
Padding

- **Padding** specifies whether the input volume is padded artificially around its border.
- This parameter is useful to keep spatial dimensions constant across filters.
- Zero-padding is the default mode.
Padding

What happens with stride 3?

e.g. input 7x7
3x3 filter, applied with stride 1
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

What happen with stride 3?
Sizes

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters $K$,
  - their spatial extent $F$,
  - the stride $S$,
  - the amount of zero padding $P$.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P) / S + 1$
  - $H_2 = (H_1 - F + 2P) / S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and $K$ biases.
- In the output volume, the $d$-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the $d$-th filter over the input volume with a stride of $S$, and then offset by $d$-th bias.

Common settings:

- $F = 3$, $S = 1$, $P = 1$
- $F = 5$, $S = 1$, $P = 2$
- $F = 5$, $S = 2$, $P = ?$ (whatever fits)
- $F = 1$, $S = 1$, $P = 0$
Equivariance

A function $f$ is **equivariant** to $g$ if $f(g(x)) = g(f(x))$.

- Parameter sharing used in a convolutional layer causes the layer to be equivariant to translation.
- That is, if $g$ is any function that translates the input, the convolution function is equivariant to $g$.

*If an object moves in the input image, its representation will move the same amount in the output.*

- Convolution is not equivariant to other operations such as change in scale or rotation.
Pooling

When the input volume is large, pooling layers can be used to reduce the input dimension while preserving its global structure, in a way similar to a down-scaling operation.

Consider a pooling area of size $h \times w$ and a 3D input tensor $x \in \mathbb{R}^{C \times (rh) \times (sw)}$.

- Max-pooling produces a tensor $o \in \mathbb{R}^{C \times r \times s}$ such that

$$o_{c,j,i} = \max_{n<h, m<w} x_{c,j+n,si+m}.$$

- Average pooling produces a tensor $o \in \mathbb{R}^{C \times r \times s}$ such that

$$o_{c,j,i} = \frac{1}{hw} \sum_{n=0}^{h-1} \sum_{m=0}^{w-1} x_{c,j+n,si+m}.$$

Pooling is very similar in its formulation to convolution.
Pooling
Pooling

Single depth slice

<table>
<thead>
<tr>
<th>x</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

max pool with 2x2 filters and stride 2

<table>
<thead>
<tr>
<th>y</th>
<th>6</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
Invariance

A function $f$ is invariant to $g$ if $f(g(x)) = f(x)$.

- Pooling layers can be used for building inner activations that are (slightly) invariant to small translations of the input.
- Invariance to local translation is helpful if we care more about the presence of a pattern rather than its exact position.
Putting things together: Layer Patterns

A convolutional network can often be defined as a composition of convolutional layers (\texttt{CONV}), pooling layers (\texttt{POOL}), linear rectifiers (\texttt{RELU}) and fully connected layers (\texttt{FC}).
Architectures

The most common convolutional network architecture follows the pattern:

\[
\text{INPUT} \rightarrow [[[\text{CONV} \rightarrow \text{RELU}] \ast N \rightarrow \text{POOL}?] \ast M \rightarrow [\text{FC} \rightarrow \text{RELU}] \ast K \rightarrow \text{FC}
\]

where:

- \( \ast \) indicates repetition;
- \text{POOL}? indicates an optional pooling layer;
- \( N \geq 0 \) (and usually \( N \leq 3 \)), \( M \geq 0 \), \( K \geq 0 \) (and usually \( K < 3 \));
- the last fully connected layer holds the output (e.g., the class scores).
Common Architectures

- **INPUT → FC**, which implements a linear classifier ($N = M = K = 0$).
- **INPUT → [FC → RELU]*$K$ → FC**, which implements a $K$-layer MLP.
- **INPUT → CONV → RELU → FC**.
- **INPUT → [CONV → RELU → POOL]*2 → FC → RELU → FC**.
- **INPUT → [[CONV → RELU]*2 → POOL]*3 → [FC → RELU]*2 → FC**.
Common Architectures

**LeNet-5 (LeCun et al, 1998)**

- First convolutional network to use backpropagation.
- Applied to character recognition.

![Diagram of LeNet-5 architecture](image)

> Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.
Common Architectures

AlexNet (Krizhevsky et al, 2012)

- 16.4% top-5 error on ILSVRC’12, outperformed all by 10%.
- Implementation on two GPUs, because of memory constraints.

Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network’s input is 150,528-dimensional, and the number of neurons in the network’s remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.
Common Architectures

VGG (Simonyan and Zisserman, 2014)

- 7.3% top-5 error on ILSVRC'14.
- Depth increased up to 19 layers, kernel sizes reduced to 3.
Common Architectures

ResNet (He et al, 2015)

- Even deeper models (34, 50, 101 and 152 layers)
- Skip connections.
- Resnet-50 vs. VGG:
  - 5.25% top-5 error vs. 7.1%
  - 25M vs. 138M parameters
  - 3.8B Flops vs. 15.3B Flops
  - Fully convolutional until the last layer

Figure 2. Residual learning: a building block.
Deeper is better

Finding the optimal neural network architecture remains an active area of research
Practical Tricks

• Pre-trained models
  – Training a model on natural images, from scratch, takes days or weeks.
  – Many models trained on ImageNet are publicly available for download. These models can be used as feature extractors for smart initialization.

• Transfer learning models
  – Take a pre-trained network, remove the last layers(), and then treat the rest of the network as a fixed feature extractor
  – Train a model from these features on a new task
  – Often better than handcrafted feature extraction for natural images, or better than training from data of the new task only

• Fine-tuning models
  – Same as for transfer learning, but also fine-tune the weights of the pre-trained network by continuing back-propagation
  – All or only some of the layers can be tuned
Practical Tricks

In the case of models pre-trained on ImageNet, this often works even when input images for the new task are not photographs of objects or animals, such as biomedical images, satellite images or paintings.

Tissues examined under an optical microscope

Figure 2. Feature extraction from pre-trained convolutional neural networks
CNN for Natural Language Processing

In the morning, the <e1>President</e1> traveled to <e2>Detroit</e2>

position embeddings matrix

word embedding matrix

Look-up tables

Convolutional layer with multiple window sizes for filters

Max pooling

Fully connected layer with dropout and softmax output

entity 1

entity 2
Convolutional networks can be inspected by looking for input images $\mathbf{x}$ that maximize the activation $\mathbf{h}_{\ell,d}(\mathbf{x})$ of a chosen convolutional kernel $\mathbf{u}$ at layer $\ell$ and index $d$ in the layer filter bank.

Such images can be found by gradient ascent on the input space:

$$\mathcal{L}_{\ell,d}(\mathbf{x}) = \| \mathbf{h}_{\ell,d}(\mathbf{x}) \|_2$$

$$\mathbf{x}_0 \sim U[0, 1]^{C \times H \times W}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \gamma \nabla_{\mathbf{x}} \mathcal{L}_{\ell,d}(\mathbf{x}_t)$$
A little insight

VGG-16, convolutional layer 1-1, a few of the 64 filters
A little insight

VGG-16, convolutional layer 2-1, a few of the 128 filters
A little insight

VGG-16, convolutional layer 3-1, a few of the 256 filters
A little insight

VGG-16, convolutional layer 4-1, a few of the 512 filters
A little insight

VGG-16, convolutional layer 5-1, a few of the 512 filters
A little insight

- The first layers appear to encode direction and color.
- The direction and color filters get combined into grid and spot textures.
- These textures gradually get combined into increasingly complex patterns.

In other words, the network appears to learn a hierarchical composition of patterns.

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]