ConvNets, CNNs → have driven change in image processing since around 2012

Usually Deep learning = ConvNets

Also increasingly being used for text analytics/natural language processing, and other places where input data is a sequence.

ConvNets consist of input and output layers plus multiple hidden layers, e.g. 10-100 layers. Hence “deep” learning since MLPs (“shallow” networks) usually have just one hidden layer.

Each layer takes output of previous layer as its input.
  - Main types of layer: Convolutional, pooling, fully connected

Some good resources online (also plenty of terrible ones) e.g.
  - Stanford CS231 course https://cs231n.github.io/
Nodes in a convolutional layer use a *kernel* or *filter*

Basic primitive: take a matrix as input, apply kernel to it (*convolve* the matrix and kernel) and produce a matrix as output

Conventional to use * to denote convolution, try not to mix up with multiplication

Example:

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
1 & 3 & 2 & 3 \\
3 & 2 & 1 & 4 \\
6 & 1 & 1 & 2 \\
3 & 2 & 1 & 5 \\
\end{array}
\times
\begin{array}{ccc}
1 & 0 & -1 \\
1 & 0 & -1 \\
1 & 0 & -1 \\
\end{array}
= 
\begin{array}{ccc}
\text{Output} \\
\end{array}
\]
Convolutional Layer

Input

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>-1</td>
<td>4</td>
</tr>
</tbody>
</table>

Kernel

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>-1</th>
</tr>
</thead>
</table>

\[
* 1 \times 1 + 1 \times 1 + 3 \times 1 + 2 \times 0 + 3 \times 0 + 2 \times 0 + 3 \times (-1) + 2 \times (-1) + 1 \times (-1) = 5 - 6 = -1
\]
Convolutional Layer

\[
\begin{array}{c|cccc}
1 & 2 & 1 & 3 & 0 \\
1 & 3 & 1 & 2 & 0 \\
3 & 2 & 1 & 0 & 4 \\
6 & 1 & 1 & 2 & 2 \\
3 & 2 & 1 & 5 & 4 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
1 & 0 & -1 & \\
1 & 0 & -1 & \\
1 & 0 & -1 & \\
\end{array}
\]

\[
\begin{array}{c|c|c}
-1 & -4 & \\
\end{array}
\]

\[
\ast \ 2 \times 1 + 3 \times 1 + 2 \times 1 + 3 \times 0 + 2 \times 0 + 1 \times 0 + 4 \times (-1) + 3 \times (-1) + 4 \times (-1) = 7 - 11 = -4
\]
**Convolutional Layer**

<table>
<thead>
<tr>
<th>Input</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>5</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>10</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{Kernel} \times \begin{bmatrix}
1 & 0 & -1 \\
1 & 0 & -1 \\
1 & 0 & -1 \\
\end{bmatrix} = \begin{bmatrix}
-1 & -4 & -19 \\
\end{bmatrix}
\]

\[
\ast 3 \times 1 + 2 \times 1 + 1 \times 1 + 4 \times 0 + 3 \times 0 + 4 \times 0 + 5 \times (-1) + 10 \times (-1) + 5 \times (-1) = 6 - 25 = -19
\]
Convolutional Layer

\[
\begin{array}{ccccc}
1 & 2 & 3 & 4 & 5 \\
1 & 3 & 2 & \cdot & 3 & 10 \\
3 & 2 & 1 & \cdot & 4 & 5 \\
6 & 1 & 1 & \cdot & 2 & 2 \\
3 & 2 & 1 & \cdot & 5 & 4 \\
\end{array}
\]

\[
\begin{array}{ccc}
1 & 0 & -1 \\
1 & 0 & -1 \\
\cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot \\
\end{array}
\]

\[
\begin{array}{c}
-1 \\
-4 \\
-19 \\
6 \\
\cdot \\
\cdot \\
\end{array}
\]

* 1 × 1 + 3 × 1 + 6 × 1 + 3 × 0 + 2 × 0 + 1 × 0 + 2 × (-1) + 1 × (-1) + 1 × (-1) = 10 − 4 = 6

* Observe that applying 3 × 3 kernel to a 5 × 5 matrix gives a 3 × 3 matrix
Convolutional Layer

\[
\begin{array}{ccccc}
1 & 2 & 3 & 4 & 5 \\
1 & 3 & 2 & 3 & 10 \\
3 & 2 & 1 & 4 & 5 \\
6 & 1 & 1 & 2 & 2 \\
3 & 2 & 1 & 5 & 4 \\
\end{array}
\times
\begin{array}{ccc}
1 & 0 & -1 \\
1 & 0 & -1 \\
1 & 0 & -1 \\
\end{array}
= 
\begin{array}{ccc}
-1 & -4 & -19 \\
6 & -3 & -13 \\
9 & -6 & -8 \\
\end{array}
\]
Example: Edge Detection

Suppose we want to detect vertical edges in image ...

Try kernel:

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
</tr>
</tbody>
</table>
Example: Edge Detection

\[
\begin{bmatrix}
10 & 10 & 10 & 0 & 0 & 0 \\
10 & 10 & 10 & 0 & 0 & 0 \\
10 & 10 & 10 & 0 & 0 & 0 \\
10 & 10 & 10 & 0 & 0 & 0 \\
10 & 10 & 10 & 0 & 0 & 0 \\
\end{bmatrix}
\times
\begin{bmatrix}
1 & 0 & -1 \\
1 & 0 & -1 \\
1 & 0 & -1 \\
\end{bmatrix}
= \text{resulting matrix}
\]
Example: Edge Detection

\[
\begin{bmatrix}
10 & 10 & 10 & 0 & 0 & 0 \\
10 & 10 & 10 & 0 & 0 & 0 \\
10 & 10 & 10 & 0 & 0 & 0 \\
10 & 10 & 10 & 0 & 0 & 0 \\
10 & 10 & 10 & 0 & 0 & 0 \\
10 & 10 & 10 & 0 & 0 & 0 \\
\end{bmatrix}
\times
\begin{bmatrix}
1 & 0 & -1 \\
1 & 0 & -1 \\
1 & 0 & -1 \\
\end{bmatrix}
= 
\begin{bmatrix}
0 \\
0 \\
0 \\
\end{bmatrix}
\]

\[
10 \times 1 + 10 \times 1 + 10 \times 1 + 10 \times 0 + 10 \times 0 + 10 \times 0 + 10 \times (-1) + 10 \times (-1) + 10 \times (-1) = 0
\]
### Example: Edge Detection

<table>
<thead>
<tr>
<th>10</th>
<th>10</th>
<th>10</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
10 \times 1 + 10 \times 1 + 10 \times 1 + 10 \times 0 + 10 \times 0 + 10 \times 0 + 10 \times (-1) + 0 \times (-1) + 0 \times (-1) = 30
\]
**Example: Edge Detection**

Observe that non-zero values in output highlight the edge.
Example: Edge Detection

Dark $\rightarrow$ light vs light $\rightarrow$ dark edges

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>10</th>
<th>10</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

$\begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$

$\begin{bmatrix} 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \end{bmatrix}$

$*$ Sign of output depends on whether transition is dark $\rightarrow$ light or light $\rightarrow$ dark

$*$ To detect horizontal edges use kernel:

$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

$*$ Similarly 45° angled edges, 70° etc

$*$ Can use kernel larger than $3 \times 3$, but $3 \times 3$ is very common in ConvNets.
Learning Kernels

* Rather than hand-crafting kernels, *learn* them!

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>10</th>
<th>10</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

Output from convolution node is a matrix, we need to map this to a scalar output/prediction $\rightarrow$ reshape/flatten matrix into a vector $x = [x_1, x_2, \ldots, x_{12}]$, then use this as input to one of our usual models E.g.

* Linear model $\hat{y} = \theta^T x$

* NB: Can add more convolution & other layers before map to output $\rightarrow$ will come back go this soon!

* Use training data to learn the unknown (i) output parameters $\theta$ and (ii) kernel weights $w = [w_1, w_2, \ldots, w_9]$:  
  1. Define cost function  
  2. Use gradient descent $\rightarrow$ typically use stochastic gradient descent variant.

* Back in familiar territory: a model mapping from input (matrix of pixel values) to predicted output, model has unknown parameters, learn these using a cost function+training data.
Padding

* Applying $3 \times 3$ kernel to $5 \times 5$ matrix gives $3 \times 3$ matrix → output is smaller than input
* Often want to keep output same size as input → use padding

```
0 0 0 0 0 0 0
0 1 2 3 4 5 0
0 1 3 2 3 10 0
0 3 2 1 4 5 0
0 6 1 1 2 2 0
0 3 2 1 5 4 0
0 0 0 0 0 0 0
```

* Add extra rows and columns of zeros to pad original $5 \times 5$ input matrix out to $7 \times 7$. Apply kernel to this padded matrix to obtain $5 \times 5$ output i.e. output same size as original $5 \times 5$ input.

* Some terminology:
  * **Valid** convolution: apply kernel directly to input → output is smaller than input
  * **Same** convolution: pad original input then apply kernel → output is same size as input
Strided Convolutions

* In previous examples we moved kernel along by one column/row at each step → we used a stride of 1
* Can also use larger strides e.g. stride 2

Observe that increasing stride reduces the size of the output matrix
* Gray-scale images are described by a single matrix, each element of matrix specifying shade of corresponding pixel
* Colour images are described by *three* matrices.
  * E.g. RGB image has one matrix specifying red intensity of each pixel, one specifying green and one specifying blue.
Multiple Channels

- $32 \times 32 \times 3$ input has three *channels*, each channel is a $32 \times 32$ matrix of values
  - The $32 \times 32 \times 3$ stack of three $32 \times 32$ matrices is called a *tensor*
- We define a separate $3 \times 3$ kernel for each channel, so overall have a $3 \times 3 \times 3$ kernel
- How to calculate output?
**Multiple Channels**

* Apply each $3 \times 3$ kernel to its corresponding $32 \times 32$ input channel. This gives three $30 \times 30$ output matrices.

* Now add element $(1,1)$ of each of the three matrices together to get element $(1,1)$ of final output. Repeat for all elements $(i,j), i = 1, \ldots, 30, j = 1, \ldots, 30 \rightarrow$ end result is a single $30 \times 30$ matrix as output.

* Note: number of channels of kernel *must* match number of input channels. E.g. if have 3 input channels then need 3 kernel channels.
More detailed example:

- Three $3 \times 3$ kernels:
  - channel 1
    
    \[
    \begin{bmatrix}
    1 & 0 & -1 \\
    1 & 0 & -1 \\
    1 & 0 & -1 \\
    \end{bmatrix}
    \]
  - channel 2
    
    \[
    \begin{bmatrix}
    2 & 0 & -2 \\
    2 & 0 & -2 \\
    2 & 0 & -2 \\
    \end{bmatrix}
    \]
  - channel 3
    
    \[
    \begin{bmatrix}
    3 & 0 & -3 \\
    3 & 0 & -3 \\
    3 & 0 & -3 \\
    \end{bmatrix}
    \]

- Three input channels:
  - channel 1
    
    \[
    \begin{bmatrix}
    1 & 2 & 3 & 4 \\
    1 & 3 & 2 & 3 \\
    3 & 2 & 1 & 4 \\
    6 & 1 & 1 & 2 \\
    \end{bmatrix}
    \]
  - channel 2
    
    \[
    \begin{bmatrix}
    4 & 3 & 1 & 2 \\
    2 & 6 & 1 & 2 \\
    1 & 3 & 1 & 2 \\
    3 & 2 & 1 & 0 \\
    \end{bmatrix}
    \]
  - channel 3
    
    \[
    \begin{bmatrix}
    7 & 2 & 3 & 4 \\
    6 & 3 & 2 & 3 \\
    5 & 2 & 1 & 4 \\
    1 & 5 & 6 & 4 \\
    \end{bmatrix}
    \]

- Output is obtained by applying channel $i$ kernel to channel $i$ input then summing. E.g. element $(1,1)$ of output is

\[
1 \times 1 + 1 \times 1 + 3 \times 1 + 2 \times 0 + 3 \times 0 + 2 \times 0 + 3 \times (-1) + 2 \times (-1) + 1 \times (-1) \\
+ 4 \times 2 + 2 \times 2 + 1 \times 2 + 3 \times 0 + 6 \times 0 + 3 \times 0 + 1 \times (-2) + 1 \times (-2) + 1 \times (-2) \\
+ 7 \times 3 + 6 \times 3 + 5 \times 3 + 2 \times 0 + 3 \times 0 + 2 \times 0 + 3 \times (-3) + 2 \times (-3) + 1 \times (-3) \\
= -1 + 8 + 36 = 43
\]

- Now shift all kernels by one column and repeat to get element $(1,2)$ of output, and so on.
Multiple Filters

- We can apply several filters to the same input and stack their outputs together.
- E.g. Apply two $3 \times 3 \times 3$ kernels to a $32 \times 32 \times 3$ input to get a $30 \times 30 \times 2$ output:

  ![Diagram](image.png)

- Note: number of output channels can be larger/smaller/same as number of input channels.
- Kernel weights $w$, input $a$. After convolution output is $w \ast a$. Here $a$ and $a \ast w$ are both tensors i.e. a stack of matrices.
- What is number of parameters $w$ in this setup?
  - Each $3 \times 3 \times 3$ kernel has 27 weights.
  - One $3 \times 3 \times 3$ kernel for each output channel, so $27 \times 2 = 54$ weights.