» Convolutional Networks

- *ConvNets*, *CNNs* $\rightarrow$ 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.
  - https://www.coursera.org/learn/convolutional-neural-networks/
  - Stanford CS231 course https://cs231n.github.io/

Example of 3x3 2D convolution for image processing: https://setosa.io/ev/image-kernels/

» Convolutional Layer

- 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:

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

Input

*

1	0	-1
1	0	-1
1	0	-1

Kernel

=

Output

Note: Kernel in CNN and kernel in SVM are different concepts

» Convolutional Layer

1^{1}	2^{0}	3^{-1}	4	5
1^{1}	3^{0}	2^{-1}	3	10
3^{1}	2^{0}	1^{-1}	4	5
6	1	1	2	2
3	2	1	5	4

Input

*

1	0	-1
1	0	-1
1	0	-1

Kernel

=

-1		

Output

* $$1\times 1+1\times 1+3\times 1+2\times 0+3\times 0+2\times 0+3\times(-1)+2\times(-1)+1\times(-1)=5-6=-1$$

» Convolutional Layer

1	2^{1}	3^{0}	4^{-1}	5
1	3^{1}	2^{0}	3^{-1}	10
3	2^{1}	1^{0}	4^{-1}	5
6	1	1	2	2
3	2	1	5	4

Input

*

1	0	-1
1	0	-1
1	0	-1

Kernel

=

-1	-4	

Output

* $2\times1+3\times1+2\times1+3\times0+2\times0+1\times0+4\times(-1)+3\times(-1)+4\times(-1) = 7-11 = -4$

» Convolutional Layer

1	2	3^{1}	4^{0}	5^{-1}
1	3	2^{1}	3^{0}	10^{-1}
3	2	1^{1}	4^{0}	5^{-1}
6	1	1	2	2
3	2	1	5	4

Input

*

1	0	-1
1	0	-1
1	0	-1

Kernel

=

-1	-4	-14

Output

* $3\times1+2\times1+1\times1+4\times0+3\times0+4\times0+5\times(-1)+10\times(-1)+5\times(-1) = 6-25 = -14$

» Convolutional Layer

$$
\underbrace{\begin{array}{|c|c|c|c|c|}\hline 1 & 2 & 3 & 4 & 5 \\ \hline 1^{1} & 3^{0} & 2^{-1} & 3 & 10 \\ \hline 3^{1} & 2^{0} & 1^{-1} & 4 & 5 \\ \hline 6^{1} & 1^{0} & 1^{-1} & 2 & 2 \\ \hline 3 & 2 & 1 & 5 & 4 \\ \hline \end{array}}_{\text{Input}}
*
\underbrace{\begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline \end{array}}_{\text{Kernel}}
=
\underbrace{\begin{array}{|c|c|c|}\hline -1 & -4 & -14 \\ \hline 6 & & \\ \hline & & \\ \hline \end{array}}_{\text{Output}}
$$

* $1\times1+3\times1+6\times1+3\times0+2\times0+1\times0+2\times(-1)+1\times(-1)+1\times(-1) = 10-4 = 6$
* Observe that applying 3×3 kernel to a 5×5 matrix gives a 3×3 matrix

» Convolutional Layer

Input

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

*

Kernel

1	0	-1
1	0	-1
1	0	-1

=

Output

-1	-4	-14
6	-3	-13
9	-6	-8

» Example: Edge Detection

* Suppose we want to detect vertical edges in image …
* Try kernel:

1	0	-1
1	0	-1
1	0	-1

Note: Edge detection, smoothing filters, and other image processing have been used for decades (i.e., way before CNNs) even downsampling can be implemented as a convolution!

» Example: Edge Detection

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

*

1	0	-1
1	0	-1
1	0	-1

=

» Example: Edge Detection

$$
\begin{array}{|c|c|c|c|c|c|}
\hline
10^{1} & 10^{0} & 10^{-1} & 0 & 0 & 0 \\
\hline
10^{1} & 10^{0} & 10^{-1} & 0 & 0 & 0 \\
\hline
10^{1} & 10^{0} & 10^{-1} & 0 & 0 & 0 \\
\hline
10 & 10 & 10 & 0 & 0 & 0 \\
\hline
10 & 10 & 10 & 0 & 0 & 0 \\
\hline
\end{array}
*
\begin{array}{|c|c|c|}
\hline
1 & 0 & -1 \\
\hline
1 & 0 & -1 \\
\hline
1 & 0 & -1 \\
\hline
\end{array}
=
\begin{array}{|c|c|c|c|}
\hline
0 & & & \\
\hline
 & & & \\
\hline
 & & & \\
\hline
\end{array}
$$

* $10\times1+10\times1+10\times1+10\times0+10\times0+10\times0+10\times(-1)+10\times(-1)+10\times(-1)=0$

» Example: Edge Detection

10	10 [1]	10 [0]	0 [-1]	0	0
10	10 [1]	10 [0]	0 [-1]	0	0
10	10 [1]	10 [0]	0 [-1]	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

1	0	-1
1	0	-1
1	0	-1

=

0	30		

* $10 \times 1 + 10 \times 1 + 10 \times 1 + 10 \times 0 + 10 \times 0 + 10 \times 0 + 10 \times (-1) + 0 \times (-1) + 0 \times (-1) = 30$

» Example: Edge Detection

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

*

1	0	-1
1	0	-1
1	0	-1

=

0	30	30	0
0	30	30	0
0	30	30	0

* Observe that non-zero values in output highlight the edge

» Example: Edge Detection

Dark→light vs light→dark edges

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

*

1	0	-1
1	0	-1
1	0	-1

=

0	-30	-30	0
0	-30	-30	0
0	-30	-30	0

- Sign of output depends on whether transition is dark→light or light→dark
- To detect horizontal edges use kernel:

1	1	1
0	0	0
-1	-1	-1

- Similarly 45° angled edges, 70° etc
- Can use kernel larger than 3×3, but 3×3 is v common in ConvNets.

» Learning Kernels

First key idea: use convolution/kernels for extracting features (e.g., edges)
Second key idea: learn kernels

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

$$\begin{bmatrix} 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \end{bmatrix} * \begin{bmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ x_5 & x_6 & x_7 & x_8 \\ x_9 & x_{10} & x_{11} & x_{12} \end{bmatrix}$$

* 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 θ 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 3×3 kernel to 5×5 matrix gives 3×3 matrix $\rightarrow$ output is *smaller* than input
- Often want to keep output same size as input $\rightarrow$ 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

*

1	0	-1
1	0	-1
1	0	-1

Kernel

=

- Add extra rows and columns of zeros to pad original 5×5 input matrix out to 7×7. Apply kernel to this padded matrix to obtain 5×5 output i.e. output same size as original 5×5 input.
- Some terminology:
  - *Valid* convolution: apply kernel directly to input $\rightarrow$ output is smaller than input
  - *Same* convolution: pad original input then apply kernel $\rightarrow$ output is same size as input

» Strided Convolutions

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

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 3 & 2 & 3 & 10 \\ 3 & 2 & 1 & 4 & 5 \\ 6 & 1 & 1 & 2 & 2 \\ 3 & 2 & 1 & 5 & 4 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} -1 & \\ & \end{bmatrix}$$

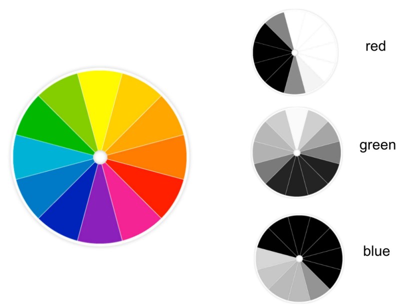
$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 3 & 2 & 3 & 10 \\ 3 & 2 & 1 & 4 & 5 \\ 6 & 1 & 1 & 2 & 2 \\ 3 & 2 & 1 & 5 & 4 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} -1 & -14 \\ & \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 3 & 2 & 3 & 10 \\ 3 & 2 & 1 & 4 & 5 \\ 6 & 1 & 1 & 2 & 2 \\ 3 & 2 & 1 & 5 & 4 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} -1 & -14 \\ 9 & \end{bmatrix}$$

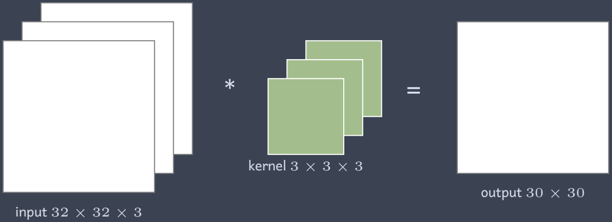
- Observe that increasing stride reduces the size of the output matrix

» Multiple Channels

* 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.

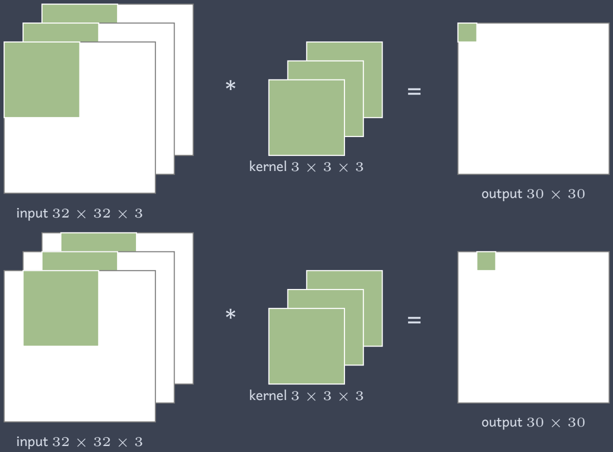


» Multiple Channels



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

» Multiple Channels



* Apply each 3×3 kernel to its corresponding 32×32 input channel. This gives three 30×30 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 30×30 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.

» Multiple Channels

More detailed example:

* Three 3×3 kernels:

channel 1

1	0	-1
1	0	-1
1	0	-1

channel 2

2	0	-2
2	0	-2
2	0	-2

channel 3

3	0	-3
3	0	-3
3	0	-3

* Three input channels:

channel 1

1	2	3	4
1	3	2	3
3	2	1	4
6	1	1	2

channel 2

4	3	1	2
2	6	1	2
1	3	1	2
3	2	1	0

channel 3

7	2	3	4
6	3	2	3
5	2	1	4
1	5	6	4

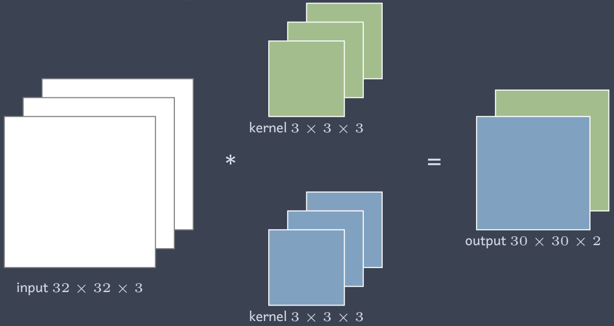
* Output is obtained by applying channel *i* kernel to channel *i* input then summing. E.g. element (1,1) of output is

$$
\begin{aligned}
&1 \times 1 + 1 \times 1 + 3 \times 1 + 2 \times 0 + 3 \times 0 + 2 \times 0 + 3 \times (-1) + 2 \times (-1) + 1 \times (-1) \\
&+4 \times 2 + 2 \times 2 + 1 \times 2 + 3 \times 0 + 6 \times 0 + 3 \times 0 + 1 \times (-2) + 1 \times (-2) + 1 \times (-2) \\
&+7 \times 3 + 6 \times 3 + 5 \times 3 + 2 \times 0 + 3 \times 0 + 2 \times 0 + 3 \times (-3) + 2 \times (-3) + 1 \times (-3) \\
&= -1 + 8 + 36 = 43
\end{aligned}
$$

* 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:



- Note: number of output channels can be larger/smaller/same as number of input channels
- Kernel weights w, input a. After convolution output is $w * a$. Here a and $a * w$ 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 (it depends only on the kernel, not on the input -> scalable)
  - One $3 \times 3 \times 3$ kernel for each output channel, so $27 \times 2 = 54$ weights.