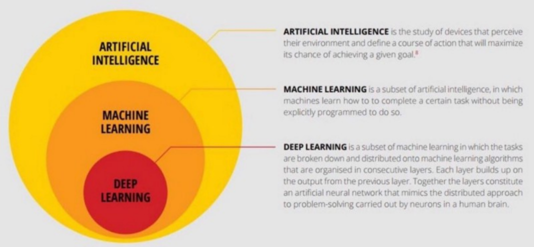
Deep learning

April 2021

Outline

What is deep learning?

- a sub-field of machine learning dealing with algorithms inspired by the structure and function of the brain called artificial neural networks



- it mirrors the functioning of our brains
- similar to how nervous system structured where each neuron connected each other and passing information

Deep Convolutional Neural Networks

(ConvNets) are deep artificial neural networks used:

- to classify images (e.g. name what they see)
- cluster them by similarity (photo search)
- perform object recognition within scenes

algorithms that can identify faces, individuals, street signs, tumors, perform optical character recognition (OCR).

Deep Convolutional Neural Networks

- pre-processing required in a ConvNet is much lower as compared to other classification algorithms
- architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain - Visual Cortex
- a ConvNet captures the spatial and temporal dependencies in an image

Tensors

An n^{th} - rank tensor in m-dimensional space is a mathematical object that has n indices and m^n components and obeys certain transformation rules.

- generalizations of scalars, vectors, and matrices to an arbitrary number of indices.

-used in physics such as elasticity, fluid mechanics, and general relativity.

Notations for tensors

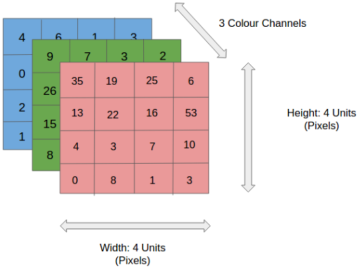
- $a_{ijk\ldots}$, $a^{ijk\ldots}$, $a_i^{jk\ldots}$, etc., may have an arbitrary number of indices
- a tensor (rank $r+s$) may be of mixed type (r,s): r "contravariant" (upper) indices and s "covariant" (lower) indices.

- the positions of the slots in which contravariant and covariant indices are placed are significant!

$$a_{\mu\nu}^{\lambda} \neq a_{\mu}^{\nu\lambda}$$

Images as tensors

ConvNets ingest and process images as tensors.



- reduce the images $\longrightarrow$ form easy to process (without losing critical features)

Convolution operation

an operation on two functions $x(t)$ (**input**) and $w(t)$ (**kernel**) of a real - valued argument

$$s(t) = \int x(a)w(t-a)da$$

notation:

$$s(t) = (x \circledast w)(t)$$

if t is discreet the integral turns into a sum

$$s(t) = (x \circledast w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

Convolution operation

in practice we have two tensors:

- the input - a multidimensional array of data
- the kernel - a multidimensional array of parameters (adapted by the learning algorithm)

- stored separately $\longrightarrow$ that these functions are zero everywhere but in the finite set of points

- we can implement the infinite summation as a summation over a finite number of array elements

- convolutions can be over more than one axis at a time

$$S(i,j) = (I \circledast K)(i,j) = \sum_m \sum_n I(m,n) K(i-m, j-n)$$

Using a kernel

we want apply a filter over the image

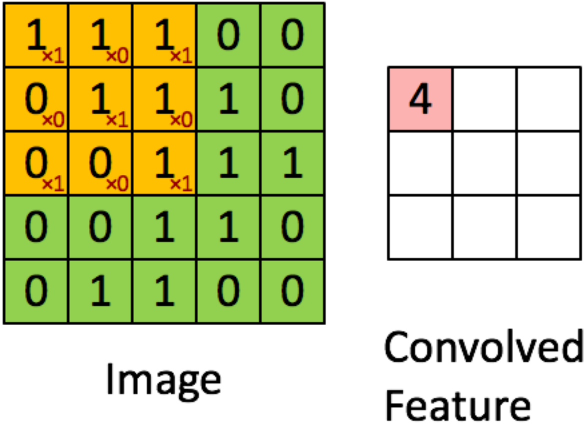
AIM: **extract the high-level features** (edges, color, gradient orientation)

Example: *Image Dimensions = 5 (Height) x 5 (Breadth) x 1 (Number of channels, eg. RGB)*

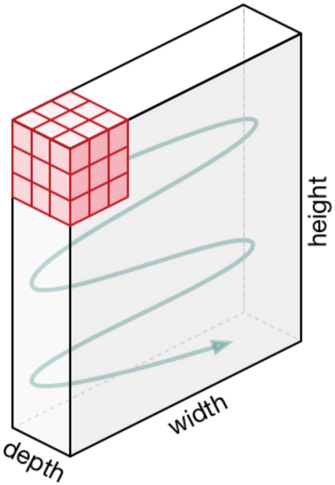
Kernel / filter:

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

Using a kernel



Using a Kernel - movement of the Kernel



- kernel shifts
- every time performing a matrix multiplication operation between K and the portion P of the image over which the kernel is hovering

Using a kernel - 3 channels

0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

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

Kernel Channel #1

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

Kernel Channel #2

0	1	1
0	1	0
1	-1	1

Kernel Channel #3

$$308 \quad + \quad -498 \quad + \quad 164 \quad + 1 = -25$$

Bias = 1

Output

-25				...
				...
				...
				...
...	...	...	...	...

Layers of a convolutional network

Typical three stages:

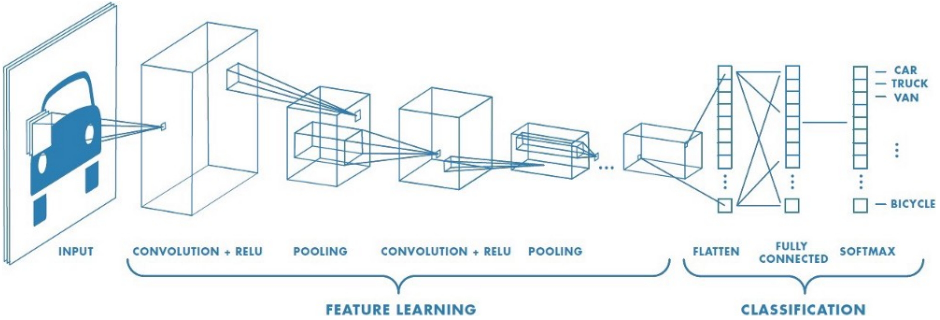
- first: several convolutions
- second: a nonlinear activation function (ex: rectified linear activation function) - **detector stage**
- third: **pooling** function to modify the output

Examples: max pooling (Zhou and Chellappa, 1988 - maximum output within a rectangular neighborhood), the average of a rectangular neighborhood, L^2 norm of a rectangular neighborhood, a weighted average based on the distance from the central pixel.

Training the network

- similar with ANN
- after computing the error, a gradient descent method is applied to all layers

Feature learning



Feature learning

Applying this filter to an image will result in a feature map that only contains vertical lines. It is a vertical line detector.

0.0	1.0	0.0
0.0	1.0	0.0
0.0	1.0	0.0

Table: A 3x3 element filter for detecting vertical lines

Dragging this filter systematically across pixel values in an image can only highlight vertical line pixels.

Feature learning

0.0	0.0	0.0
1.0	1.0	1.0
0.0	0.0	0.0

Table: A horizontal line detector

- combining the filters' results (combining both feature maps, will result in all of the lines in an image being highlighted)
- a suite of tens or even hundreds of other small filters can be designed to detect other features in the image
- the values of the filter are weights to be learned during the training of the network

Feature learning

training under gradient descent

- the network will learn what types of features to extract from the input
- the extracted features will be those that minimize the loss function (e. g. extract features that are the most useful for classifying images as dogs or cats)

Gradient descent on Convolution Layer

- the input $X = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,1} & x_{3,2} & x_{3,3} \end{bmatrix}$
- the filter $F = \begin{bmatrix} F_{1,1} & F_{1,2} \\ F_{2,1} & F_{2,2} \end{bmatrix}$
- the output is $O = X \circledast F$
- $\frac{\partial E}{\partial O}$ is the gradient of loss from previous layer

Gradient descent on Convolution Layer

After we apply the convolution on X and F we have:

$$O_{1,1} = x_{1,1} * F_{1,1} + x_{1,2} * F_{1,2} + x_{2,1} * F_{2,1} + x_{2,2} * F_{2,2}$$

$$O_{1,2} = x_{1,2} * F_{1,1} + x_{1,3} * F_{1,2} + x_{2,2} * F_{2,1} + x_{2,3} * F_{2,2}$$

$$O_{2,1} = x_{2,1} * F_{1,1} + x_{2,2} * F_{1,2} + x_{3,1} * F_{2,1} + x_{3,2} * F_{2,2}$$

$$O_{2,2} = x_{2,2} * F_{1,1} + x_{2,3} * F_{1,2} + x_{3,2} * F_{2,1} + x_{3,3} * F_{2,2}$$

Gradient descent on Convolution Layer

we compute the partial derivative of O with respect of F

$$\frac{\partial O_{1,1}}{\partial F_{1,1}} = x_{1,1},\ \frac{\partial O_{1,1}}{\partial F_{1,2}} = x_{1,2},\ \frac{\partial O_{1,1}}{\partial F_{2,1}} = x_{2,1},\ \frac{\partial O_{1,1}}{\partial F_{2,2}} = x_{2,2}$$

similar we compute for $O_{1,2}$, $O_{2,1}$, and $O_{2,2}$

the gradient to update the filter will be

$$\frac{\partial E}{\partial F} = \frac{\partial E}{\partial O} * \frac{\partial O}{\partial F} \tag{1}$$

Gradient descent on Convolution Layer

if we expand Equation 1

$$
\begin{aligned}
\frac{\partial E}{\partial F_{1,1}} &= \frac{\partial E}{\partial O_{1,1}} * \frac{\partial O_{1,1}}{\partial F_{1,1}} + \frac{\partial E}{\partial O_{1,2}} * \frac{\partial O_{1,2}}{\partial F_{1,1}} \\
&+ \frac{\partial E}{\partial O_{2,1}} * \frac{\partial O_{2,1}}{\partial F_{1,1}} + \frac{\partial E}{\partial O_{2,2}} * \frac{\partial O_{2,2}}{\partial F_{1,1}} \\
&= \frac{\partial E}{\partial O_{1,1}} * x_{1,1} + \frac{\partial E}{\partial O_{1,2}} * x_{1,2} + \frac{\partial E}{\partial O_{2,1}} * x_{2,1} \\
&+ \frac{\partial E}{\partial O_{2,2}} * x_{2,2}
\end{aligned}
$$

Gradient descent on Convolution Layer

we observe that this is actually a convolution product between the input and the loss gradient

$$\begin{bmatrix} \frac{\partial E}{\partial F_{1,1}} & \frac{\partial E}{\partial F_{1,2}} \\ \frac{\partial E}{\partial F_{2,1}} & \frac{\partial E}{\partial F_{2,2}} \end{bmatrix} = X \circledast \begin{bmatrix} \frac{\partial E}{\partial O_{1,1}} & \frac{\partial E}{\partial O_{1,2}} \\ \frac{\partial E}{\partial O_{2,1}} & \frac{\partial E}{\partial O_{2,2}} \end{bmatrix} \tag{2}$$

Gradient descent on Convolution Layer

similar we get for the derivative of E with respect of X

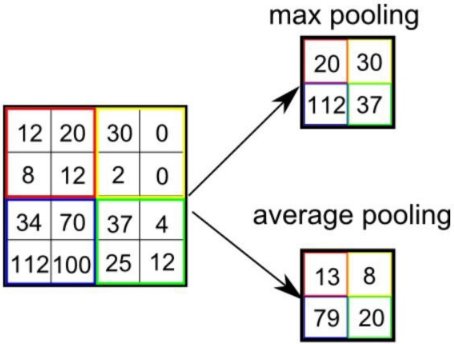
$$\begin{bmatrix} \frac{\partial E}{\partial x_{1,1}} & \frac{\partial E}{\partial x_{1,2}} & \frac{\partial E}{\partial x_{1,3}} \\ \frac{\partial E}{\partial x_{2,1}} & \frac{\partial E}{\partial x_{2,2}} & \frac{\partial E}{\partial x_{2,3}} \\ \frac{\partial E}{\partial x_{3,1}} & \frac{\partial E}{\partial x_{3,2}} & \frac{\partial E}{\partial x_{3,3}} \end{bmatrix} = \begin{bmatrix} F_{2,2} & F_{2,1} \\ F_{1,2} & F_{1,1} \end{bmatrix} \circledast \begin{bmatrix} \frac{\partial E}{\partial O_{1,1}} & \frac{\partial E}{\partial O_{1,2}} \\ \frac{\partial E}{\partial O_{2,1}} & \frac{\partial E}{\partial O_{2,2}} \end{bmatrix}$$

observe that matrix F is flipped over 180^o degrees in this formula.

Pooling

helps to make the representation approximately invariant to small translations of the input
- useful property if we care more about whether some feature is present than exactly where it is
- essential for handling inputs of varying size
- Some guidance as to which kinds of pooling one should use in various situations : Boureau et al., 2010.

Pooling



- Max Pooling - returns the maximum value from the portion of the image covered by the Kernel
- Average Pooling returns the average of all the values from the portion of the image covered by the Kernel

Training the Pooling layer

in an *NxN* pooling block, forward propagation of error is reduced to a single value - value of the “winning unit”