



CMPE 258, Deep Learning

Convolution Neural Network

March 15, 2018

DMH 149A

Taehee Jeong

Ph.D., Data Scientist

Architecture for Exam_1

Top five students

initialization	Hidden layer 1	Hidden layer 2	Hidden layer 3	activation 1	activation 2	activation 3	regularization
He-like	256	32		relu	sigmoid		L1
He	268	150	100	relu	relu	sigmoid	dropout + L2
He + Xavier	100	40	10	relu	relu	sigmoid	L2
random	50	25		sigmoid	sigmoid		L2
random	75	75		relu	relu		L2

Assignment_4

Deep Neural Network with tensorflow

Due day is Sunday, March 18th.

There is penalty for late submission or re-submission after due day.

In Assignment_3 and mid-term exam, we used pandas and numpy.

For Assignment_4, we will use Tensorflow.

Please build Deep Neural Network with two hidden layers.

For activation function, please use relu or elu for hidden layers, sigmoid for output layer.

For weight initialization, please use xavier initialization.

For optimization, please use adam optimization.

For regularization, please use dropout.

As the final output, please plot train accuracy and test accuracy with probability (0.1 ~ 0.9) of dropout.

Final deadline for Assignment_1,2,3

March 18th.

Not accept after the final deadline.

Group Project

Proposal date: 4/12, 4/24

Presentation date : 5/8, 5/10

Report (including code) due date : 5/6

Number of members : 1 to 4

Content: DNN, CNN, RNN related

Platform : Pandas, Numpy, tensorflow, keras (please discuss with me for others)

Grading policy:

- Content : 40 pts

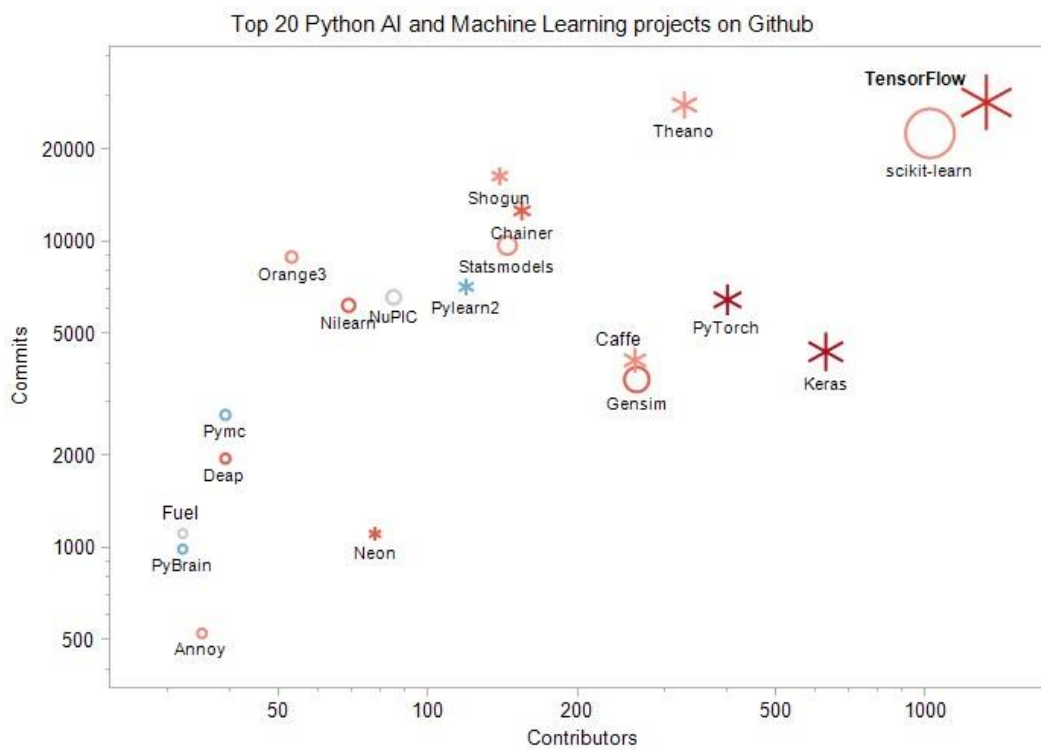
- ; Creativity in data collection, Neural network architecture / algorithm, application (same quality as a conference paper)

- Presentation : 20 pts

- Report : 20 pts

- Code : 20 pts

Deep Learning platforms



Top 20 Python AI and Machine Learning Open Source Projects
By Ilan Reinstein, KDnuggets

Last lecture

- Parameter update:
Momentum, Adagrad, Rmsprop, Adam
- Learning rate decay:
 $1/t$, exponential decay, step decay

Parameter update

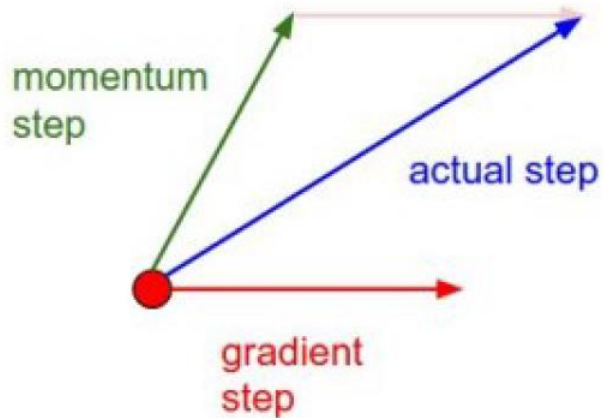
Faster optimization

- Gradient Descent with Momentum
- Nesterov momentum : not cover
- Adagrad
- Rmsprop
- Adam

Momentum update

Gradient descent update

$$x := x - \text{learning rate} * dx$$



Momentum update

$$v = \beta * v - \text{learning rate} * dx$$

$$x := x + v$$

v : velocity

β : momentum

Adagrad

Adaptive learning rate method

$$v := v + dx^{**2}$$

$$x := x - \text{learning_rate} * dx / (\text{sqrt}(v) + \text{eps})$$

v : sum of squared gradients

Normalized the parameter update step, element-wise.

Weights with high gradients will have effective learning rate reduces,

Weights with low gradients will have effective learning rate increases.

Gradient descent with momentum

$$V_{dW} = 0, V_{db} = 0$$

For every iteration t:

Compute V_{dW} , V_{db} using dW and db

$$V_{dW} := \beta_1 \cdot V_{dW} + (1 - \beta_1) \cdot dW$$

$$V_{db} := \beta_1 \cdot V_{db} + (1 - \beta_1) \cdot db$$

$$W := W - \alpha \cdot V_{dW}$$

$$b := b - \alpha \cdot V_{db}$$

RMS prop

Root Mean square prop

For every iteration t:

Compute S_{dW} , S_{db} using dW and db

$$S_{dW} := \beta_2 \cdot S_{dW} + (1 - \beta_2) \cdot dW^2$$

$$S_{db} := \beta_2 \cdot S_{db} + (1 - \beta_2) \cdot db^2$$

$$S_{dW} \neq 0, S_{db} \neq 0, \epsilon = 10^{-8}$$

$$W := W - \alpha \cdot \frac{dW}{\sqrt{S_{dW} + \epsilon}}$$

$$b := b - \alpha \cdot \frac{db}{\sqrt{S_{db} + \epsilon}}$$

Adam Optimization

Adaption moment estimation

For every iteration t: $V_{dW} = 0, V_{db} = 0, S_{dW} = 0, S_{db} = 0, \epsilon = 10^{-8}$

Compute $V_{dW}, V_{db}, S_{dW}, S_{db}$ using dW and db

$$V_{dW} := \beta_1 \cdot V_{dW} + (1 - \beta_1) \cdot dW$$

$$V_{db} := \beta_1 \cdot V_{db} + (1 - \beta_1) \cdot db$$

$$S_{dW} := \beta_2 \cdot S_{dW} + (1 - \beta_2) \cdot dW^2$$

$$S_{db} := \beta_2 \cdot S_{db} + (1 - \beta_2) \cdot db^2$$

$$W := W - \alpha \cdot V_{dW}$$

$$b := b - \alpha \cdot V_{db}$$

Adam Optimization

Bias correction

$$V_{dW}^{cor} = \frac{V_{dW}}{1 - \beta_1^t}$$

$$V_{db}^{cor} = \frac{V_{db}}{1 - \beta_1^t}$$

$$S_{dW}^{cor} = \frac{S_{dW}}{1 - \beta_2^t}$$

$$S_{db}^{cor} = \frac{S_{db}}{1 - \beta_2^t}$$

$$W := W - \alpha \cdot \frac{V_{dW}^{cor}}{\sqrt{S_{dW}^{cor} + \epsilon}}$$

$$b := b - \alpha \cdot \frac{V_{db}^{cor}}{\sqrt{S_{db}^{cor} + \epsilon}}$$

Adam optimization

Hyper parameters choices

α : need to be tune

β_1 : 0.9 (momentum)

β_2 : 0.999 (RMS prop)

ϵ : 10^{-8}

Learning rate decay

Slowing decreasing α

If learning rate is large, parameters decay too aggressively and the parameters are unable to reach the local minimum.

1 epoch : 1 pass through data set, iteration number

- 1/t decay
- Exponentially decay
- Step decay: discrete staircase
- Manual decay

1/t decay

$$\alpha = \frac{\alpha_0}{1 + \text{decay-rate} \times \text{epoch-num}}$$

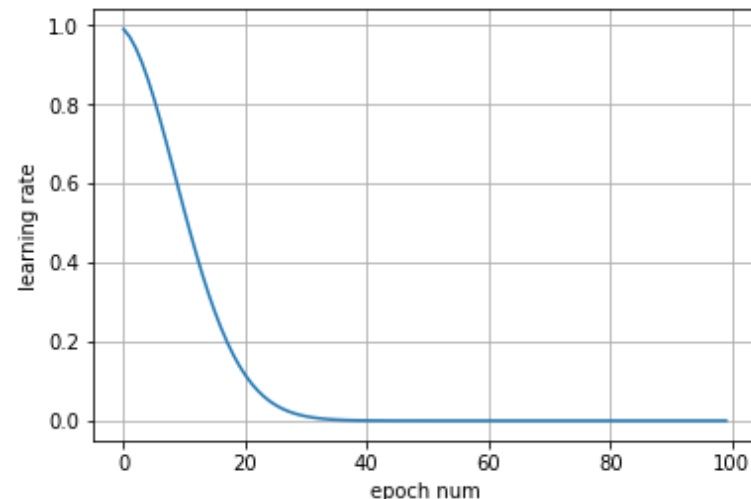
$$\alpha = \frac{\alpha_0}{1 + kt}$$

$$\alpha_0 = 1$$

decay rate = 0.1



decay rate = 0.01

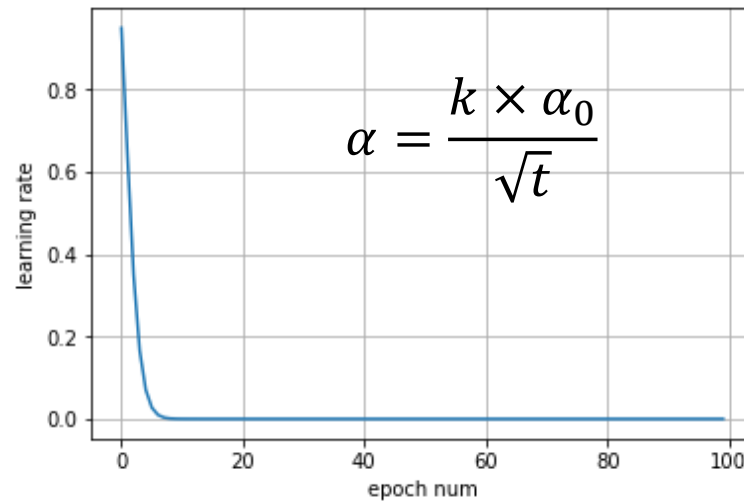


1/t decay

$$\alpha = \frac{k \times \alpha_0}{\sqrt{\text{epoch} - \text{num}}}$$

$$\alpha_0 = 1$$

$$k = 0.95$$

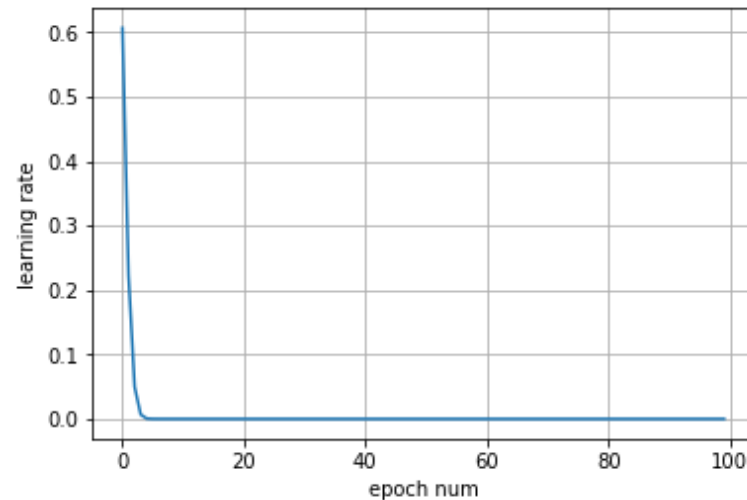


Exponential decay

$$\alpha = \alpha_0 \cdot \exp(-kt)$$

$$\alpha_0 = 1$$

$$k = 0.5$$

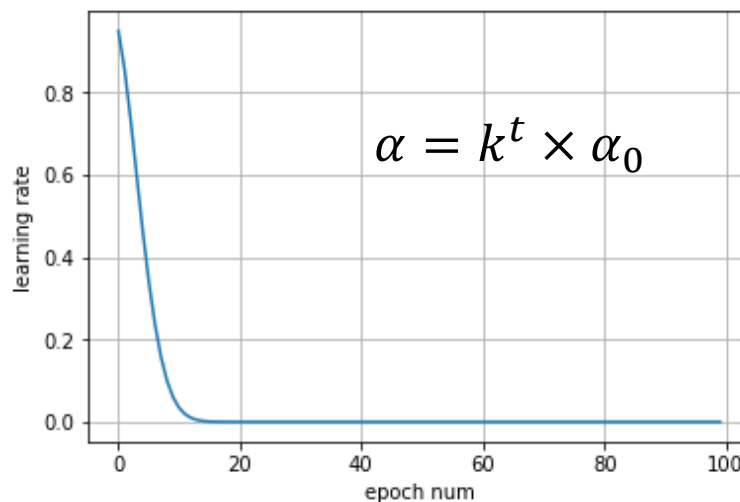


Exponentially decay

$$\alpha = k^{\text{epoch_num}} \times \alpha_0$$

$$\alpha_0 = 1$$

$$k = 0.95$$



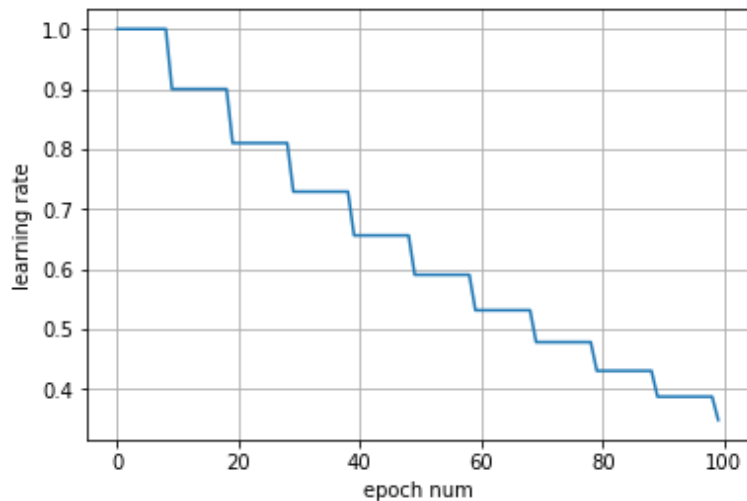
Discrete staircase

$\alpha = k \times \alpha$ with every few epochs

$$\alpha_0 = 1$$

$$k = 0.9$$

every 10 epochs



Learning rate decay

Annealing learning rate

Hyper parameter k: [0.01, 0.1, 0.5, 0.9, 0.95, 0.99]

$$\alpha = \frac{\alpha_0}{1 + kt}$$

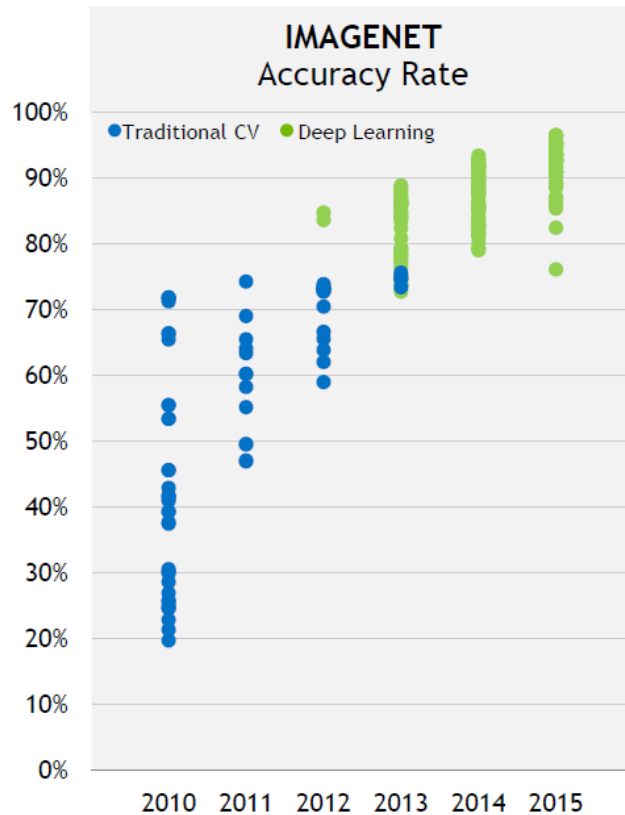
$$\alpha = \frac{k \times \alpha_0}{\sqrt{t}}$$

$$\alpha = \alpha_0 \cdot \exp(-kt)$$

$$\alpha = k^t \times \alpha_0$$

$$\alpha = k \times \alpha \text{ with every } t$$

Image classification

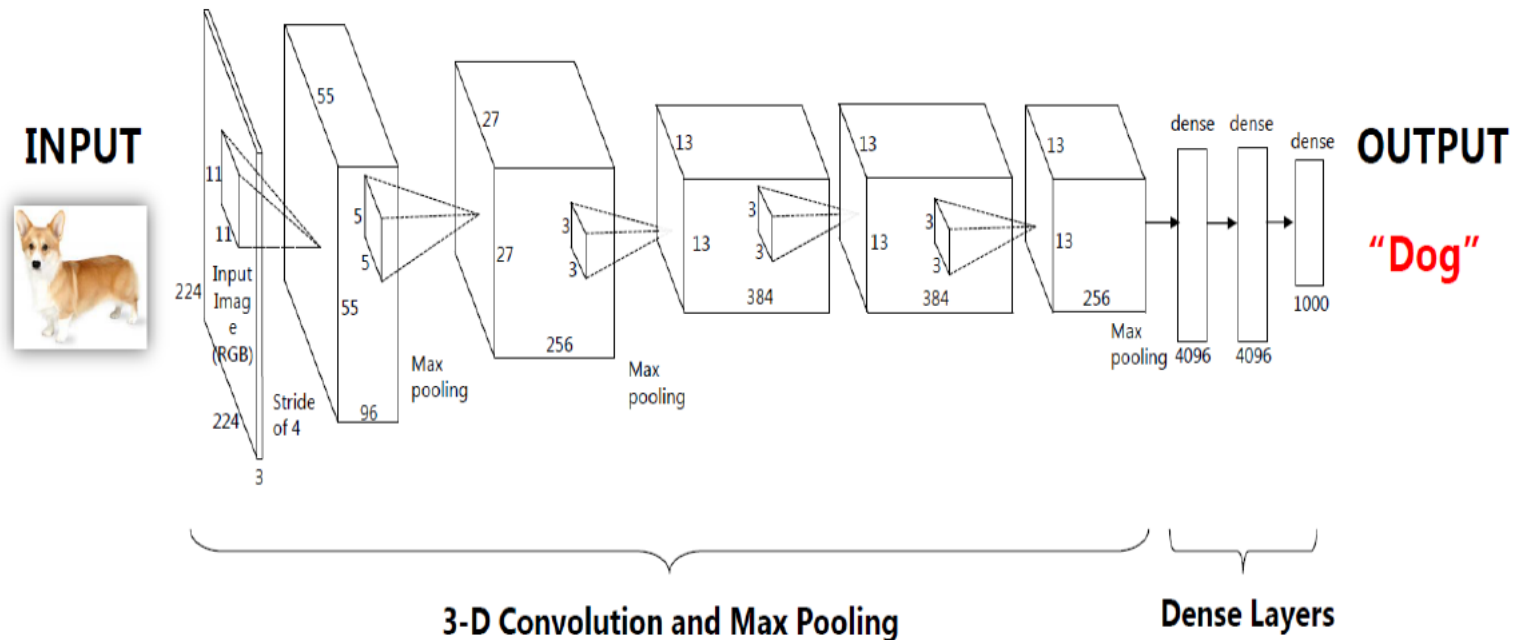


Top 5 error rate for image classification in ILSVRC
ImageNet challenge
From 26% to 3% in six years

Human accuracy : 95 ~ 97 %

Image-Net Large-Scale Visual Recognition Challenge (ILSVRC)
<http://image-net.org/challenges/LSVRC/>
<https://www.slideshare.net/NVIDIA/nvidia-ces-2016-press-conference>

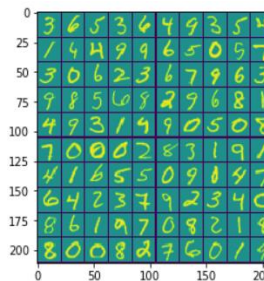
Image classification using Convolution Neural Network



Ralph Wittig, Power-Efficient Machine Learning using FPGAs on Power systems, OpenPOWERSummit

Digital representation of an image

- Grayscale image is a matrix of pixels (**picture elements**)
- Dimensions of this matrix are called image resolution (e.g. 20 x 20)
- Each pixel stores its brightness (or **intensity**) ranging from 0 to 255, 0 intensity corresponds to black color
- Color images store pixel intensities for 3 channels: red, green and blue



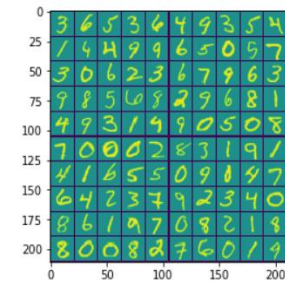
	0	1	2	3	4	5	6	7	8	9	...	391	392	393	394	395	396	397	398	399	y
0	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0 5
1	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0 9
2	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0 7
3	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0 6
4	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0 5

Intro to deep learning, national research university

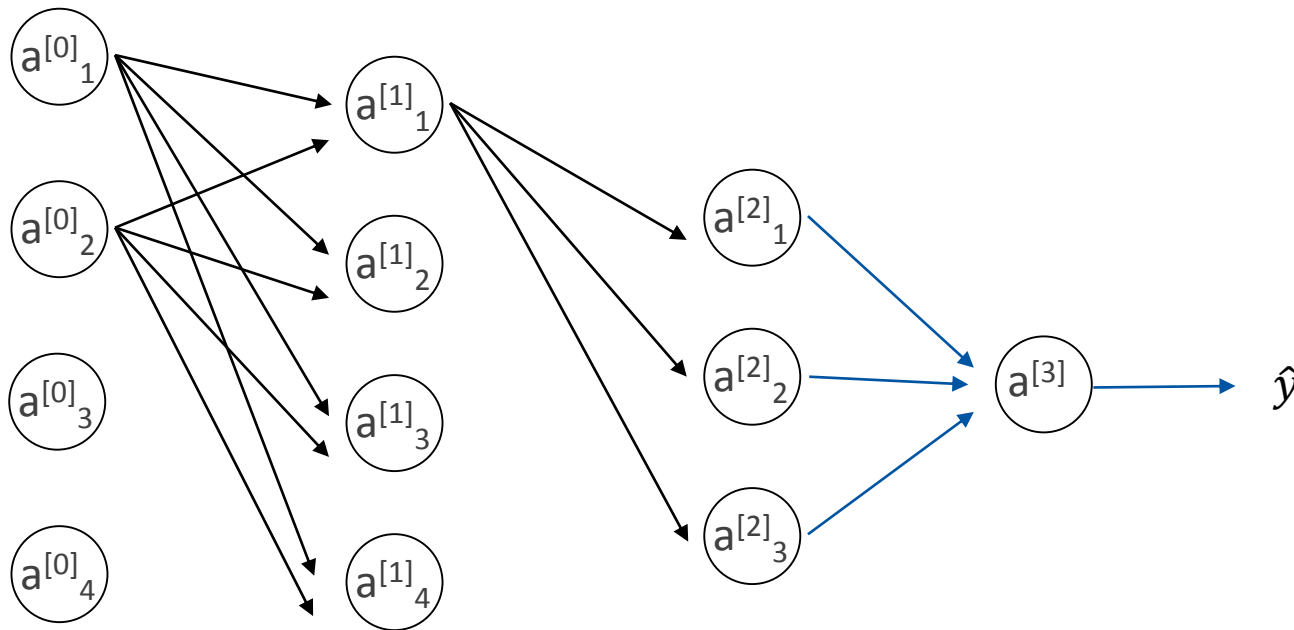
Normalize input pixels

$$x_{norm} = \frac{x}{255} - 0.5$$

	0	1	2	3	4	5	6	7	8	9	...	391	392	393	394	395	396	397	398	399	y
0	0	0	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0 5
1	0	0	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0 9
2	0	0	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0 7
3	0	0	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0 6
4	0	0	0	0	0	0	0	0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0 5



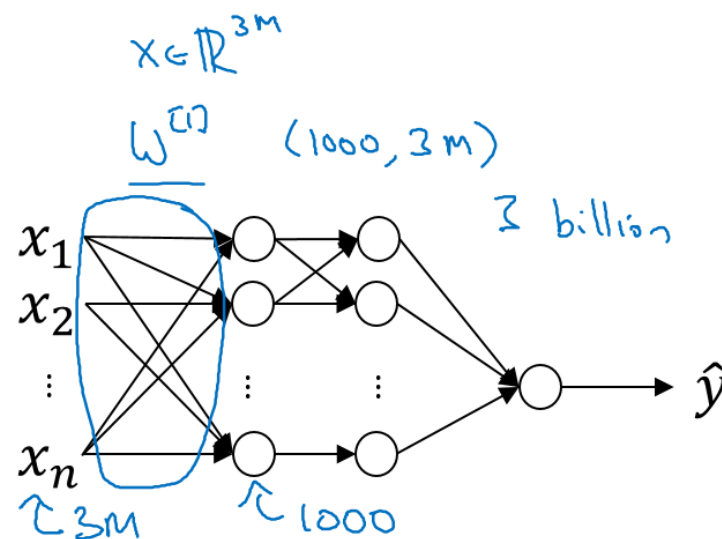
Why not Deep Neural Network?



Why not Deep Neural Network?



$$1000 \times 1000 \times 3 \\ = 3 \text{ million}$$



<Deep Neural Network, Andrew Ng>

Why not Deep Neural Network?

Why not simply use a regular deep neural network with fully connected layers for image recognition tasks?

Unfortunately, although this works fine for small images (e.g., MNIST), it breaks down for larger images because of the huge number of parameters it requires.

For example, a 100×100 image has 10,000 pixels, and if the first layer has just 1,000 neurons (which already severely restricts the amount of information transmitted to the next layer), this means a total of 10 million connections.

And that's just the first layer.

CNNs solve this problem using [partially connected layers](#).

<Hands-On ML, A. Geron>

Convolutional Layer

The most important building block of a CNN is the *convolutional layer*.

Neurons in the first convolutional layer are not connected to every single pixel in the input image, but only to pixels in their receptive fields.

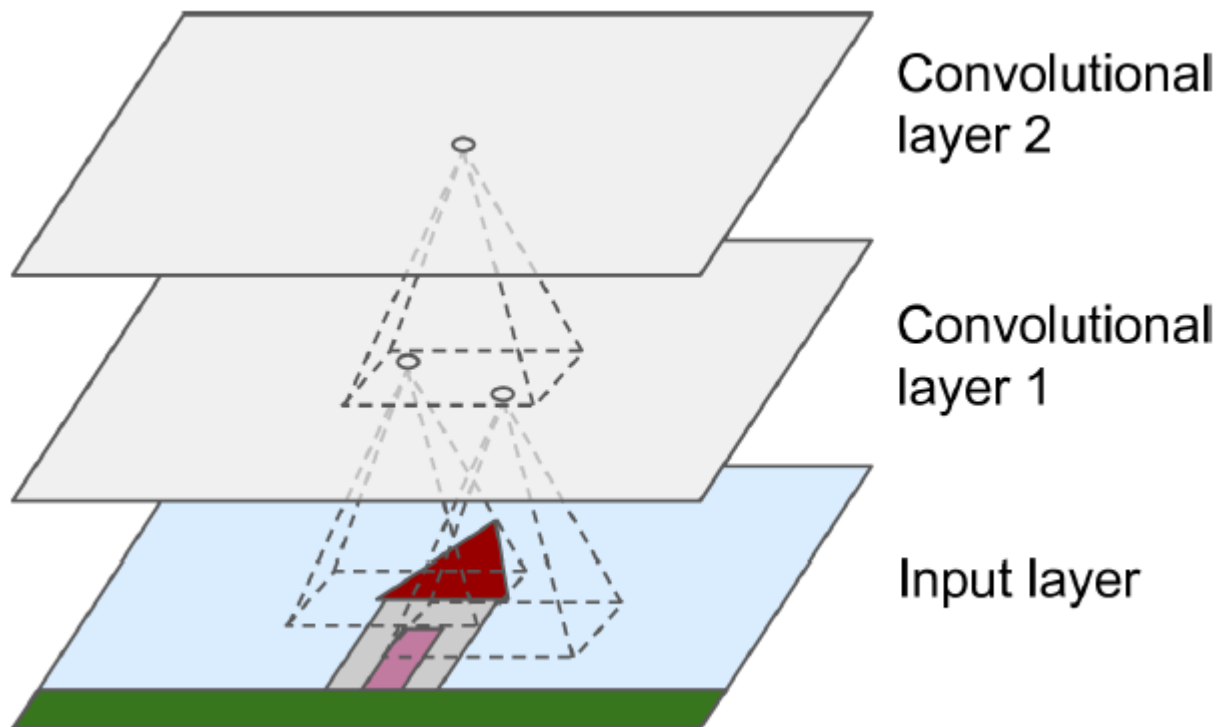
Neuron in the second convolutional layer is connected only to neurons located within a small rectangle in the first layer.

This architecture allows the network to concentrate on low-level features in the first hidden layer, then assemble them into higher-level features in the next hidden layer, and so on.

This hierarchical structure is common in real-world images, which is one of the reasons why CNNs work so well for image recognition.

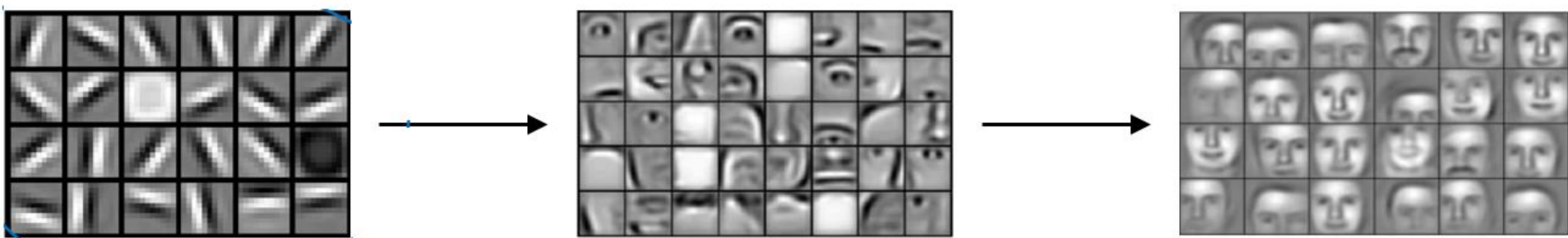
<Hands-On ML, A. Geron>

Convolutional layer



<Hands-On ML, A. Geron>

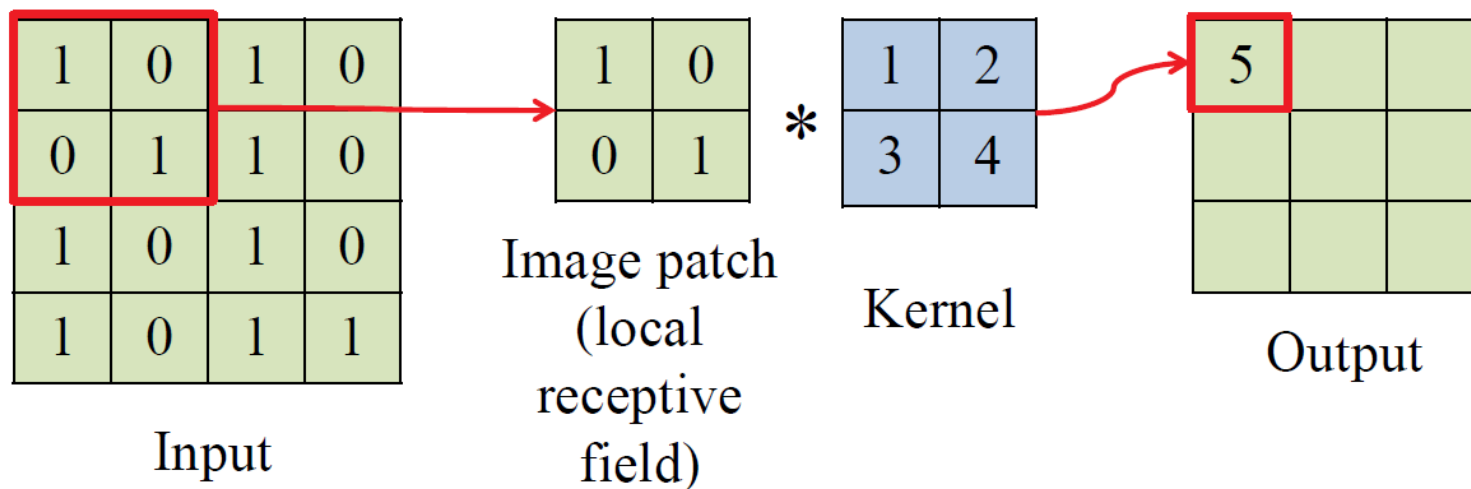
Convolutional layer



<Deep Neural Network, Andrew Ng>

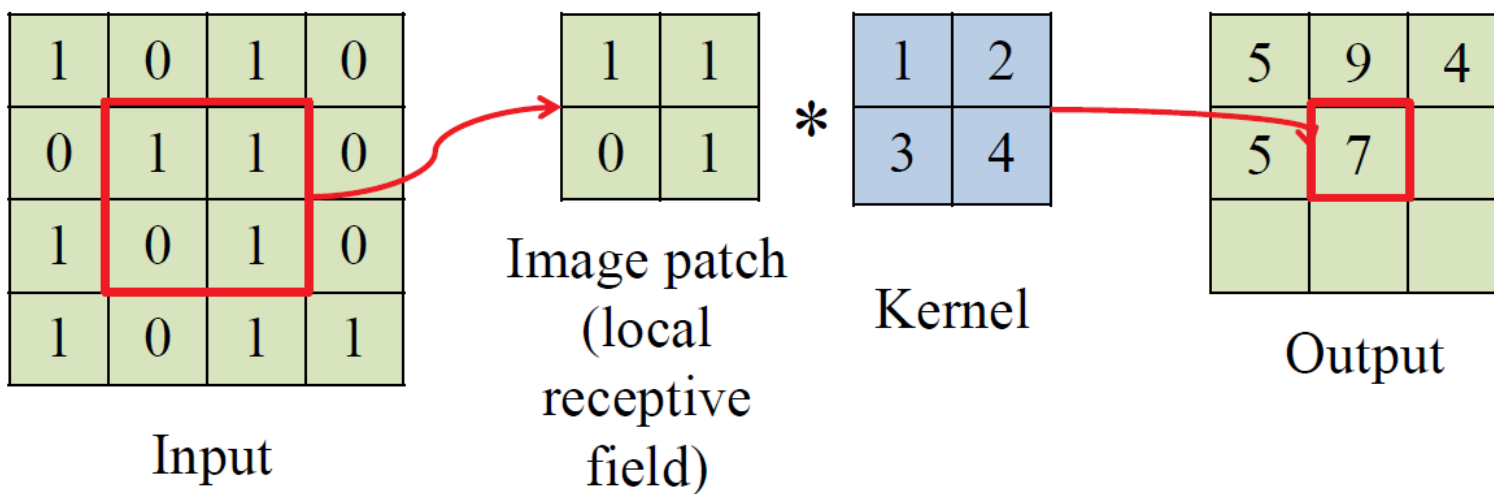
Convolutions

Convolution is a dot product of a **kernel** (or filter) and a patch of an image (**local receptive field**) of the same size



<Intro to deep learning, national research university>

Convolutions



<Intro to deep learning, national research university>

Vertical edge detection

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	<u>10</u>	<u>10</u>	<u>0</u>	0	0
10	<u>10</u>	<u>10</u>	<u>0</u>	0	0
10	<u>10</u>	<u>10</u>	<u>0</u>	0	0

6x6

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

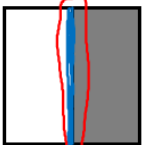
3x3

*

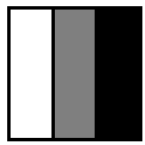
=

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

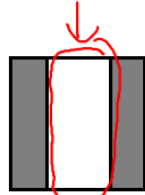
4x4



*




=



<Deep Neural Network, Andrew Ng>


Vertical 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
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
0	30	30	0




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	10	10	10




*

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



=

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



<Deep Neural Network, Andrew Ng>

Horizontal edge detection

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10


6x6

*

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

=

0	0	0	0
30	10	-10	-30
30	10	-10	-30
0	0	0	0



<Deep Neural Network, Andrew Ng>

Edge detection

Kernel

$$\begin{matrix} * & \begin{matrix} \begin{matrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{matrix} \end{matrix} & = & \end{matrix}$$

Edge detection



Original
image

Sums up to 0 (black color)
when the patch is a solid fill



Sharpening

Original image

Kernel

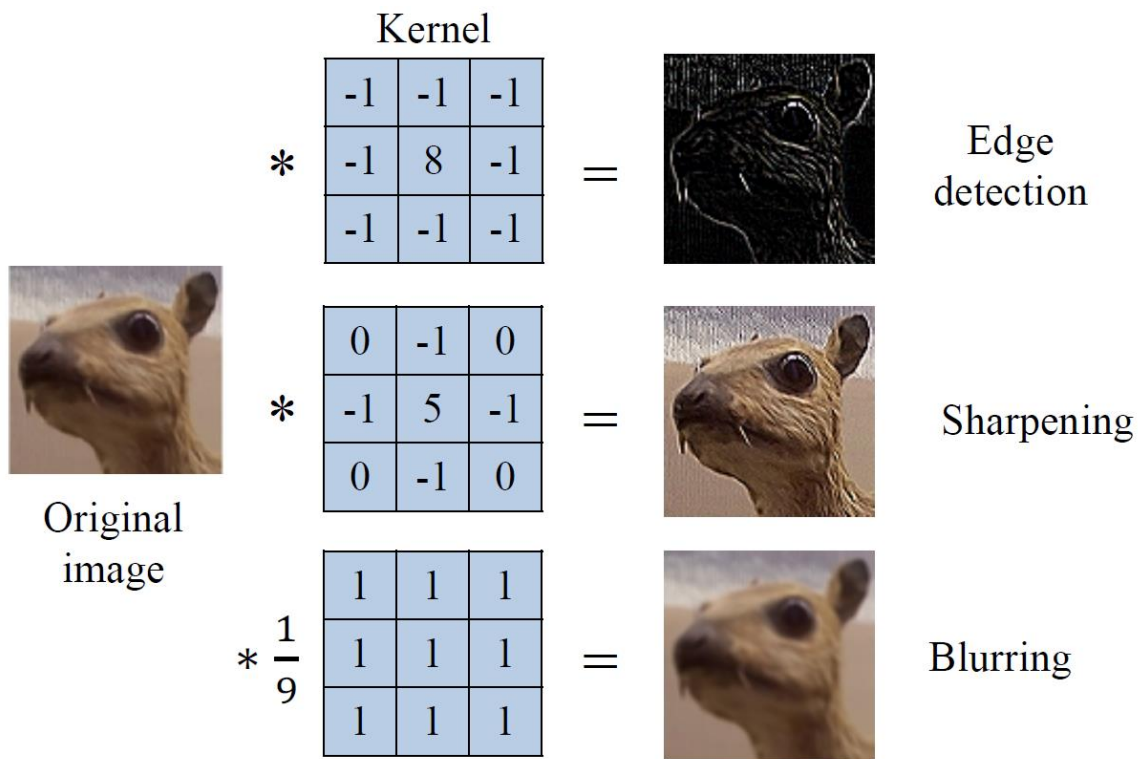
$\begin{matrix} * & \begin{matrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{matrix} & = & \begin{matrix} \text{Edge} \\ \text{detection} \end{matrix} \end{matrix}$

$\begin{matrix} * & \begin{matrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{matrix} & = & \begin{matrix} \text{Sharpening} \end{matrix} \end{matrix}$

Doesn't change an image for solid fills

Adds a little intensity on the edges

Blurring



<Intro to deep learning, national research university>

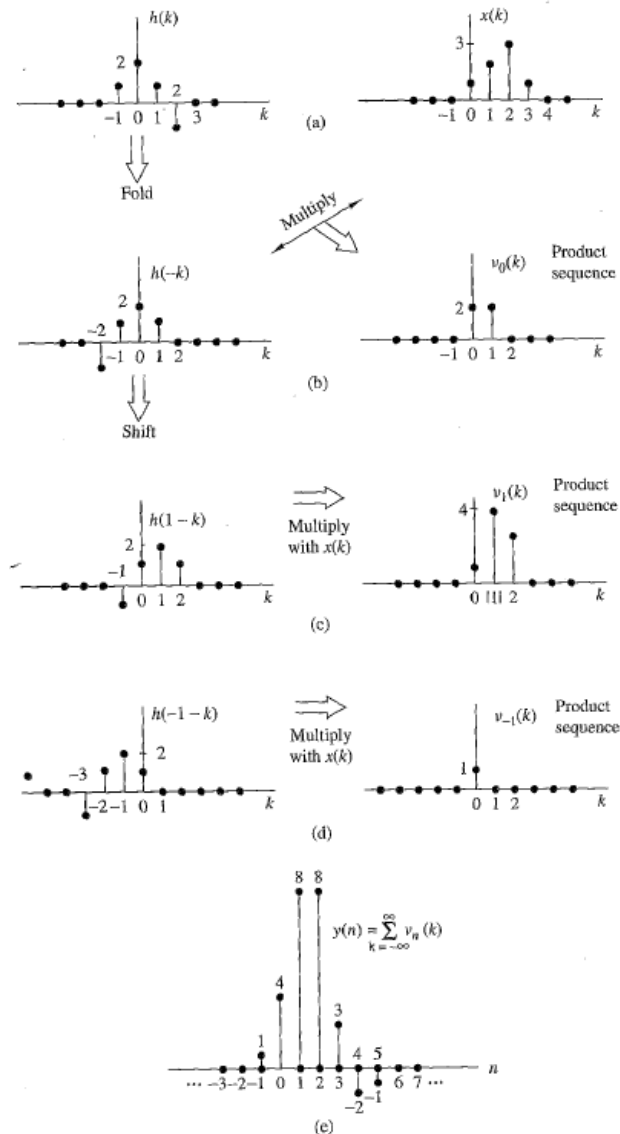
Convolution

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k)$$

<Digital signal processing, J.G., Proakis et al.>

Convolution is a mathematical operation that slides one function over another and measures the integral of their pointwise multiplication. It is heavily used in signal processing.

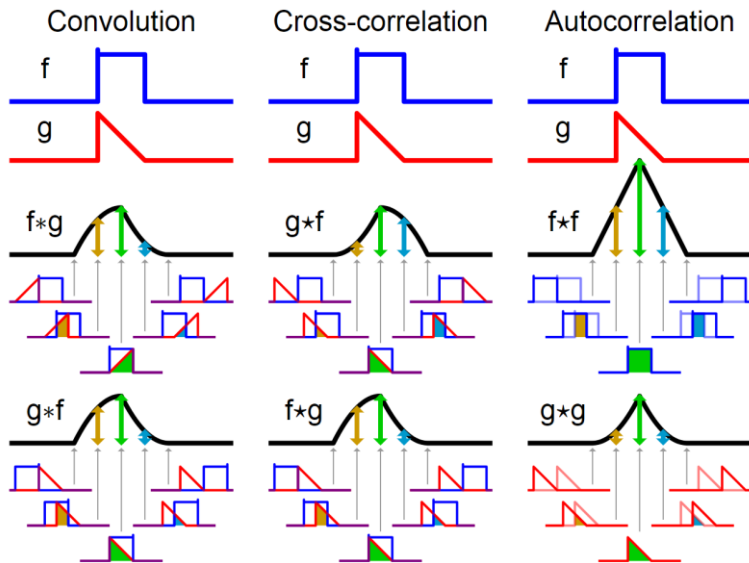
<Hands-On ML, A. Geron>



Convolution

Convolutional layers actually use cross-correlations.

<Hands-On ML, A. Geron>



<Wikipedia>

Convolution as correlation

0	0	0	0
0	0	0	0
0	0	1	0
0	0	0	1

Input

*

1	0
0	1

Kernel

=

0	0	0
0	1	0
0	0	2

Output

0	0	0	0
0	0	0	0
0	0	0	1
0	0	1	0

Input

*

1	0
0	1

Kernel

=

0	0	0
0	0	1
0	1	0

Output

Convolution as translation

0	0	0	0
0	0	0	0
0	0	1	0
0	0	0	1

Input

*

1	0
0	1

Kernel

=

0	0	0
0	1	0
0	0	2

Output

1	0	0	0
0	1	0	0
0	0	0	0
0	0	0	0

Input

*

1	0
0	1

Kernel

=

2	0	0
0	1	0
0	0	0

Output

Advantages of a CNN over a fully connected DNN for image classification

Because consecutive layers are only partially connected and because it heavily reuses its weights, a CNN has many fewer parameters than a fully connected DNN, which makes it much faster to train, reduces the risk of overfitting, and requires much less training data.

When a CNN has learned a kernel that can detect a particular feature, it can detect that feature anywhere on the image. In contrast, when a DNN learns a feature in one location, it can detect it only in that particular location. Since images typically have very repetitive features, CNNs are able to generalize much better than DNNs for image processing tasks such as classification, using fewer training examples.

a DNN has no prior knowledge of how pixels are organized; it does not know that nearby pixels are close. A CNN's architecture embeds this prior knowledge. Lower layers typically identify features in small areas of the images, while higher layers combine the lower-level features into larger features. This works well with most natural images, giving CNNs a decisive head start compared to DNNs.