



Deep Learning

Machine Learning and Big Data - DATA622

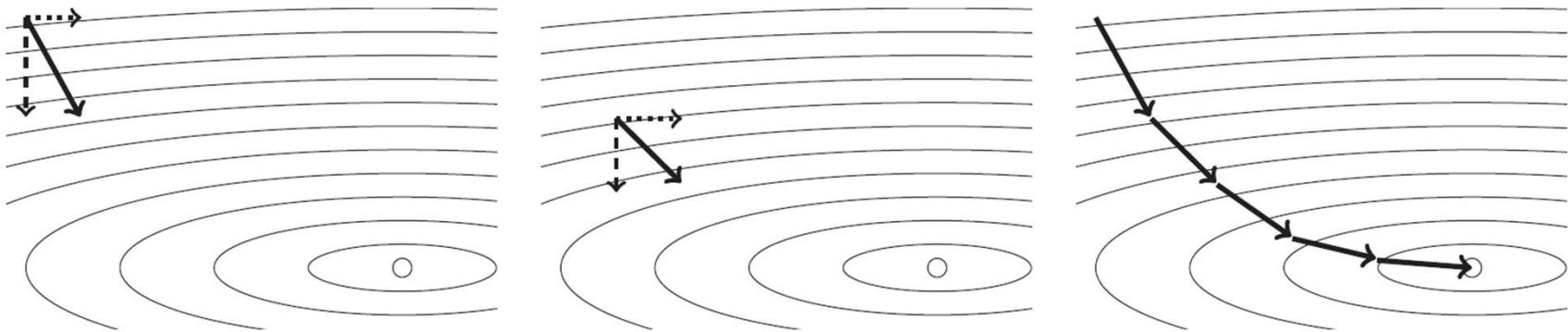
CUNY School of Professional Studies

Learning Rate

You need to adjust the learning rate as you train

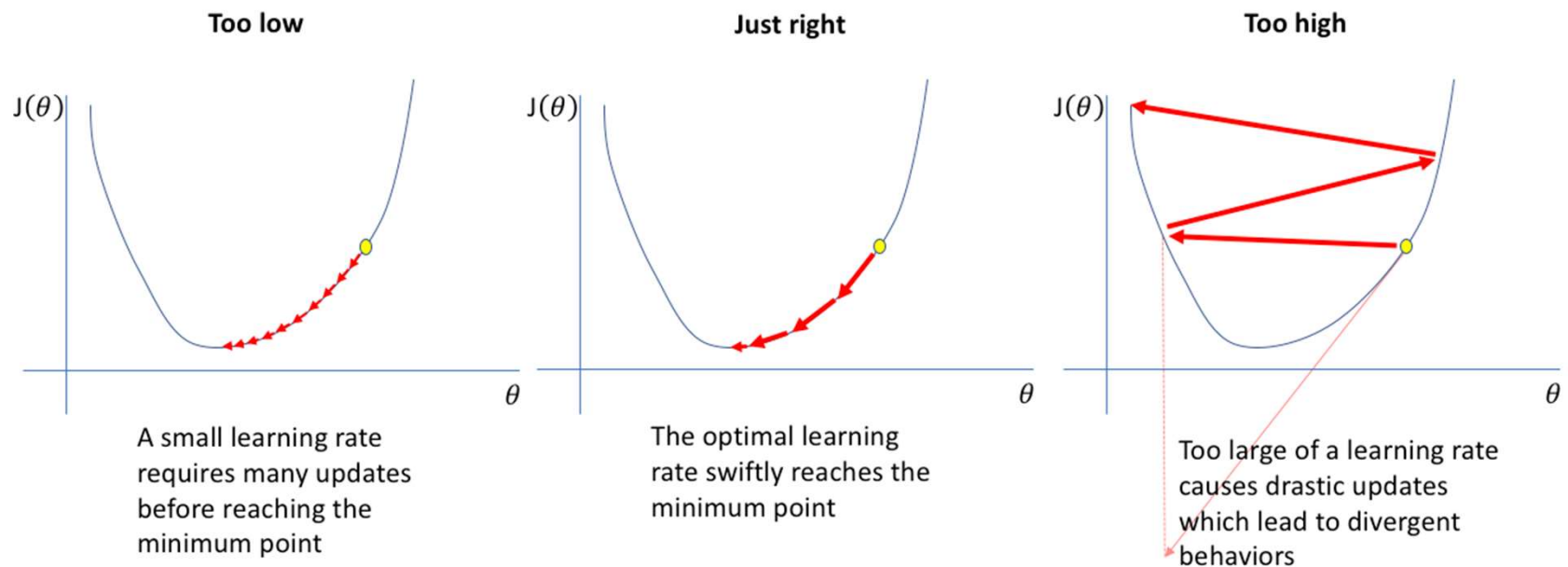
Gradient descent

The gradient tells us the direction in which the loss has the steepest rate of increase, but it does not tell us how far we should step



Learning Rate

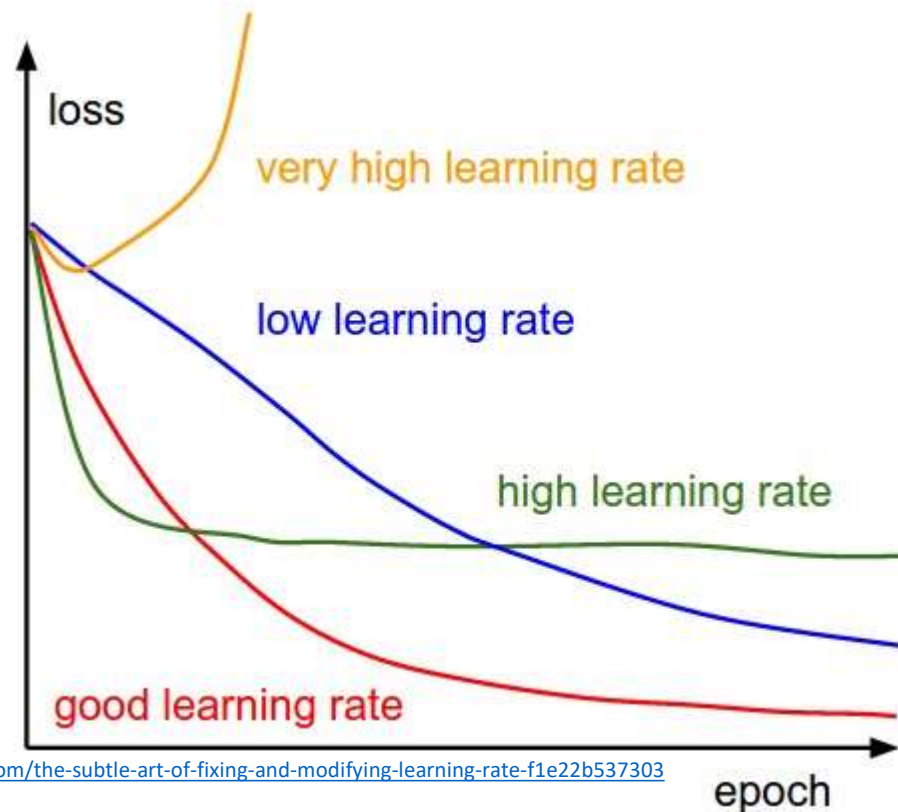
Learning rate (step size) is a hyper-parameter which defines the speed of 'learning'



Source: <https://www.jeremyjordan.me/nn-learning-rate/>

Impact of different learning rates

- High learning rate: loss increases or plateaus too quickly
- Low learning rate: loss decreases too slowly: too many epochs to reach a solution

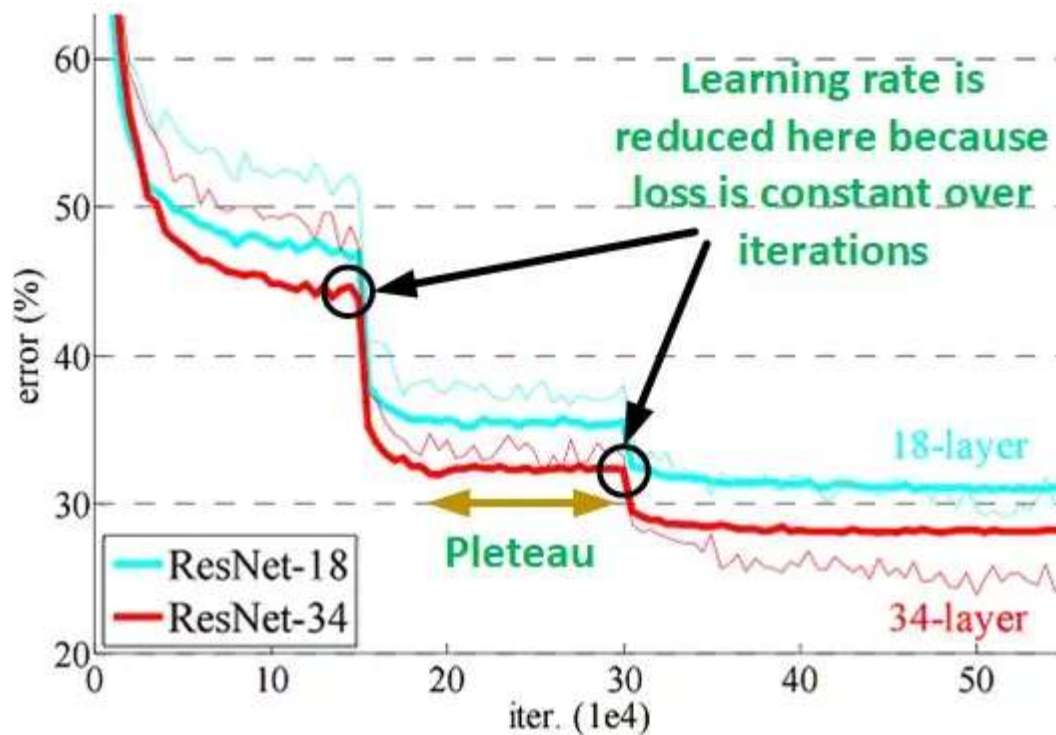


Source: <https://towardsdatascience.com/the-subtle-art-of-fixing-and-modifying-learning-rate-f1e22b537303>

Adjusting learning rates

- Learning rates must be decayed during training
- Approach 1: Automatic Decay
 - Reduce the learning rate by some factor every few epochs
 - Typical values: reduce the learning rate by a half every 5 epochs, or by 10 every 20 epochs
 - Exponential decay reduces the learning rate exponentially over time
 - These numbers depend heavily on the type of problem and the model
- Approach 2: Plateau-driven
 - Reduce the learning rate by a constant (e.g., by half) when the validation loss stops improving
 - In TensorFlow: `tf.keras.callbacks.ReduceLROnPlateau()`
 - Monitor: validation loss
 - Factor: 0.1 (i.e., divide by 10)
 - Patience: 10 (how many epochs to wait before applying it)
 - Minimum learning rate: $1e-6$ (when to stop)

Adjusting learning rates



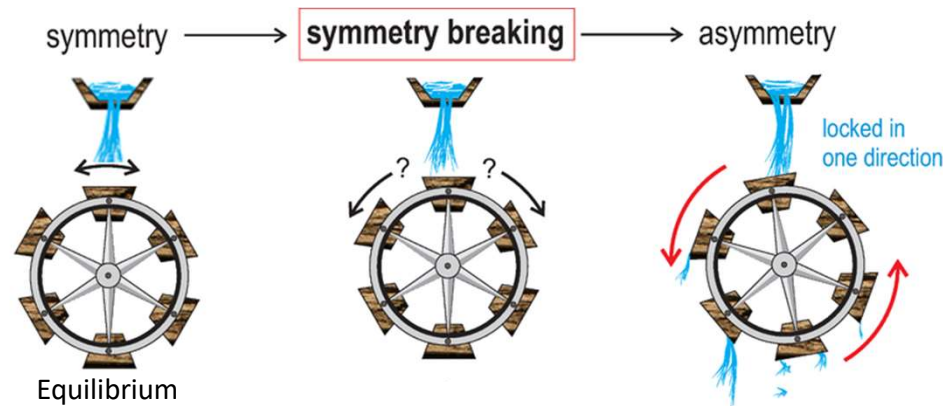
Source: <https://towardsdatascience.com/the-subtle-art-of-fixing-and-modifying-learning-rate-f1e22b537303>

Lottery ticket hypothesis

Breaking symmetry of neural networks with random initialization

Breaking symmetry

- Initializing weights & biases to zeroes (or ones) is problematic

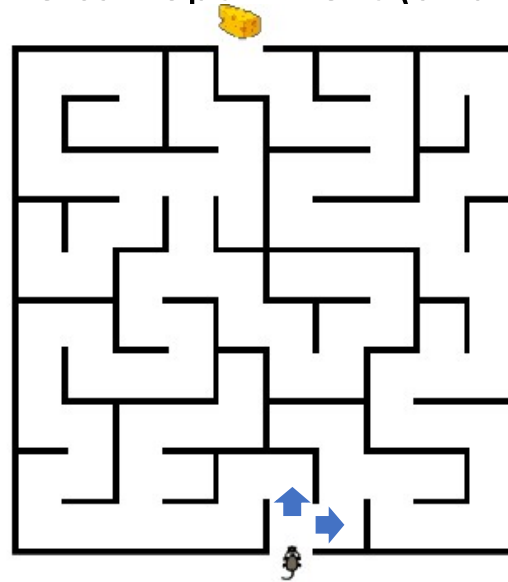


- If all weights are zeros, subsequent hidden layers will get zero signal
- If weights are all the same value (e.g. 1), then subsequent hidden layers will get exactly the same signal i.e. if all weights are initialized to 1, each unit gets signal equal to sum of inputs (and outputs $\text{sigmoid}(\text{sum}(\text{inputs}))$).
- No matter what was the input - if all weights are the same, all units in hidden layer will be the same too. This is why you should initialize weights randomly.

Mouse Maze

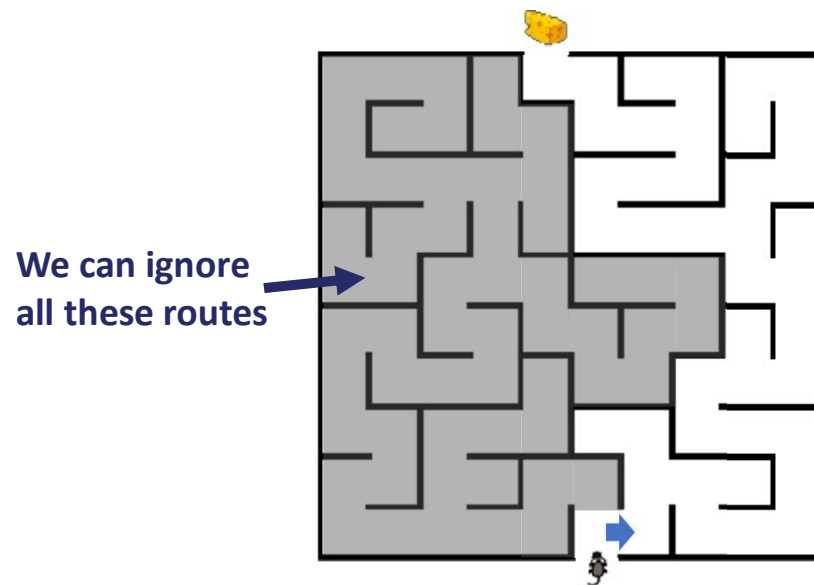
Consider a mouse trying to find a way through a maze:

1. The initial routing will be critical in terms of how quickly a solution is found
2. Once a route is found, we can optimize it (and other routes are ignored)



Mouse Maze

If we have a favorable start and found a route, then the rest of the maze is irrelevant



Lottery ticket hypothesis

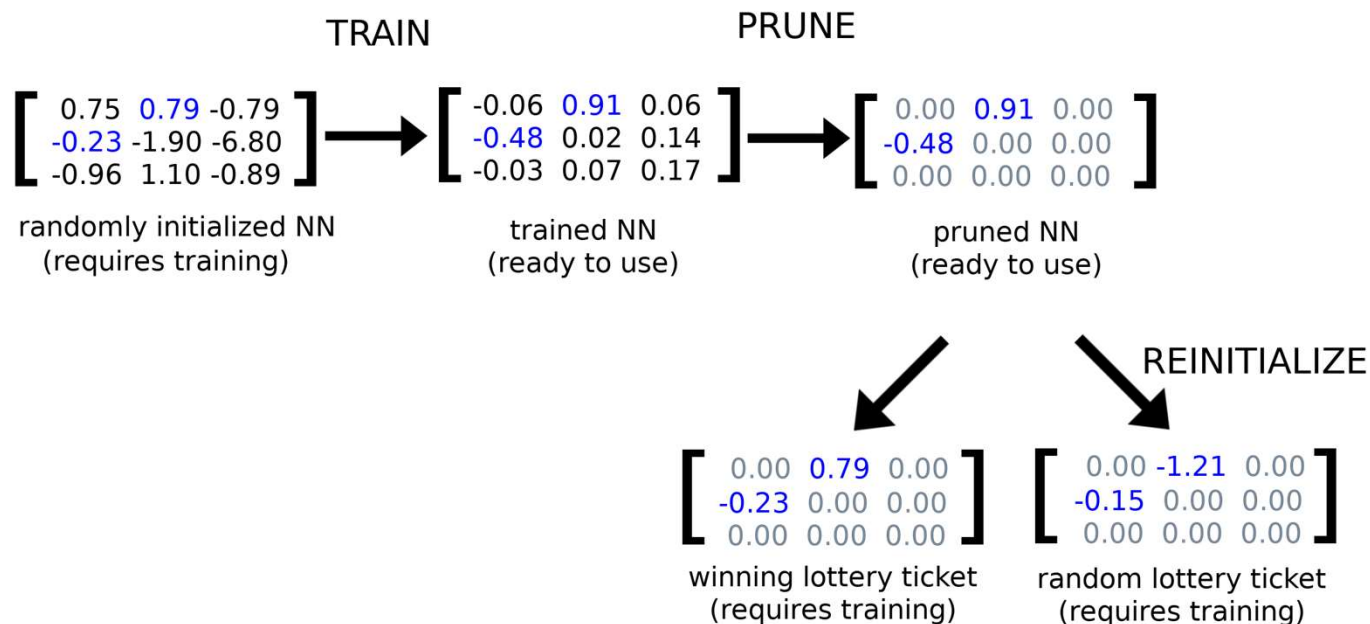
“A randomly-initialized, dense neural network contains a subnetwork that is initialized such that-when trained in isolation-it can match the test accuracy of the original network after training for at most the same number of iterations.”

The subnetwork is referred to as a winning ticket (it has won the initialization lottery!)

Lottery ticket hypothesis: <https://arxiv.org/pdf/1803.03635.pdf>

Lottery ticket hypothesis

- A neural network is typically sparse (most of the weights are zero)
- Pruning the network doesn't change accuracy but makes the model faster
- The lottery ticket (initial optimal weights) will regenerate the optimal model

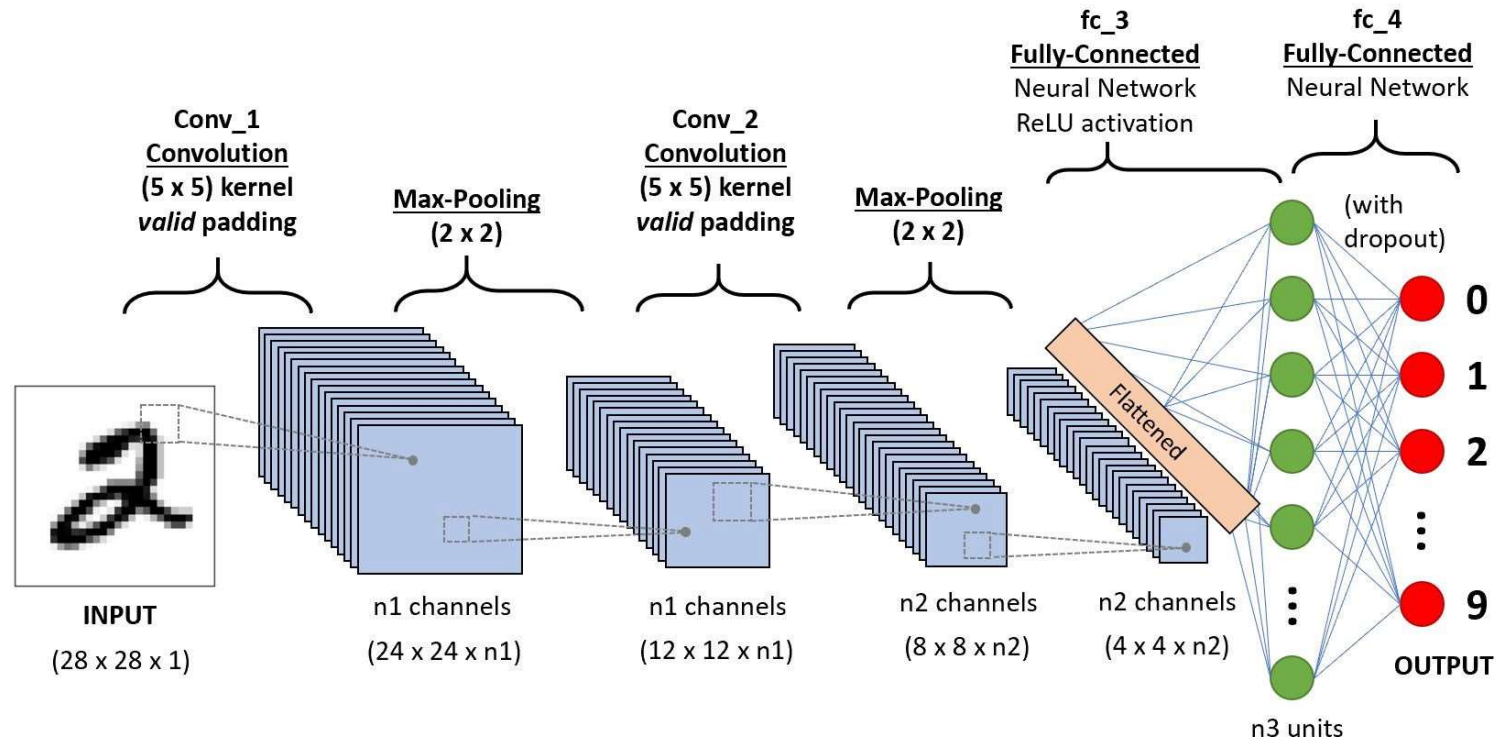


Lottery ticket hypothesis: <https://okl1m3k.github.io/lottery-ticket-hypothesis/>

Convolutional Neural Networks

Image processing (inspired by nature)

Convolutional Neural Networks (CNN)



Source: Jonas Bokstaller

Convolutions

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

+

↓
-498

+

↓
164

+ 1 = -25

↑
Bias = 1

Output

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

At each slide, we perform an element-wise multiplication and sum everything to get a single value, with the kernel is sliding over the whole input.

Pooling

Max Pooling

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

Average pooling

7	3	5	2
8	7	1	6
4	9	3	9
0	8	4	5

$$\frac{7 + 3 + 8 + 7}{4}$$

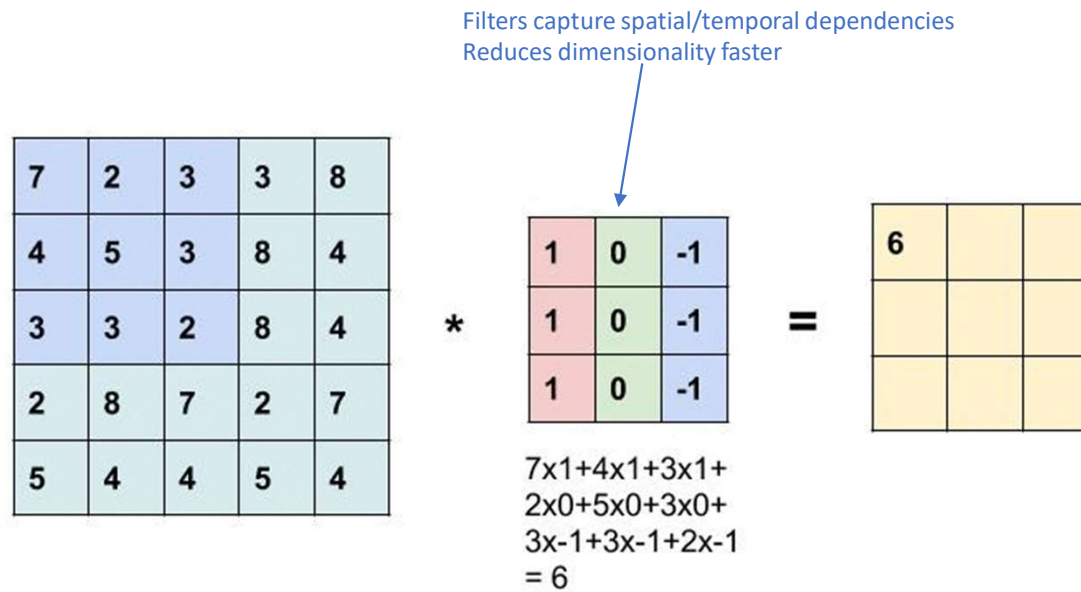
Average pooling

6.25	





The goal of pooling is to reduce the height and width of our image but not the number of channels by using a stride > 1

Source: <https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/>

Convolutions

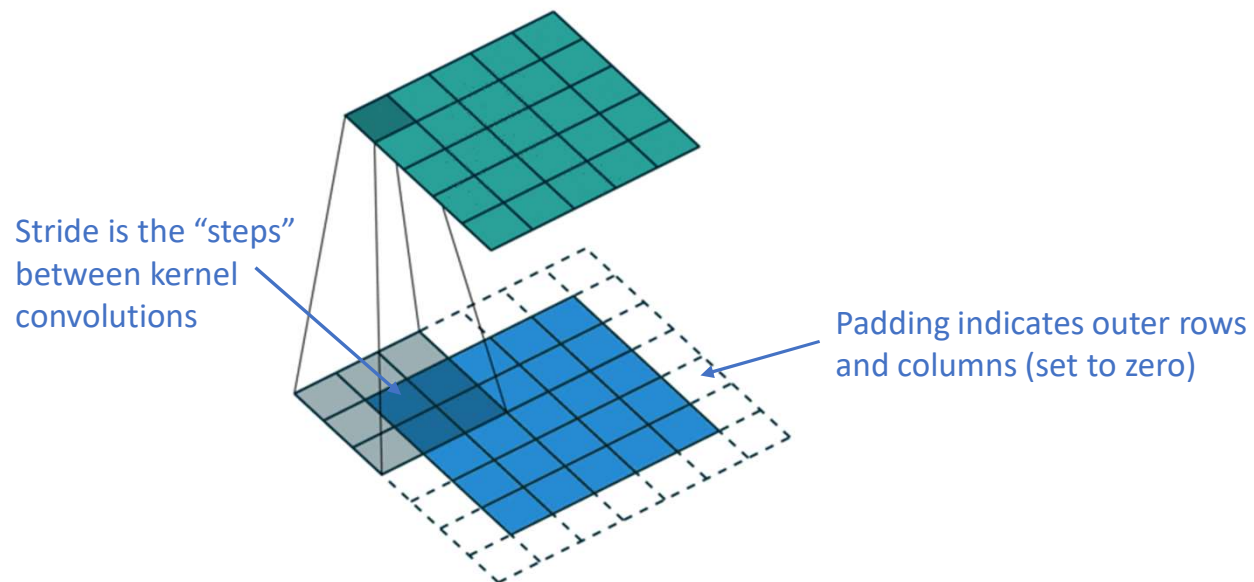


Some examples of kernels

<i>Original</i>	<i>Gaussian Blur</i>	<i>Sharpen</i>	<i>Edge Detection</i>
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$
			

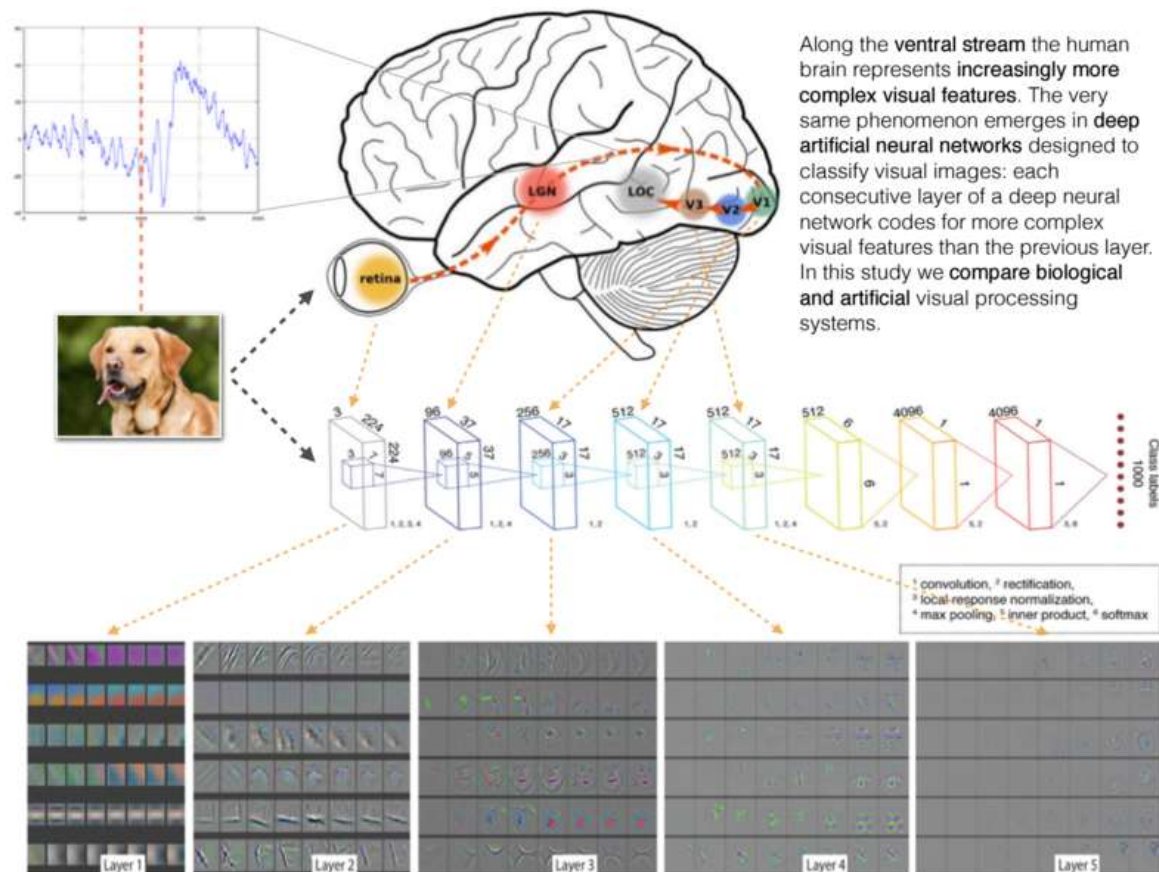
Source: <https://medium.com/analytics-vidhya/understanding-convolution-operations-in-cnn-1914045816d4>

Padding and stride



Source: <https://medium.datadriveninvestor.com/convolutional-neural-networks-3b241a5da51e>

The Brain's visual cortex



CNN example

