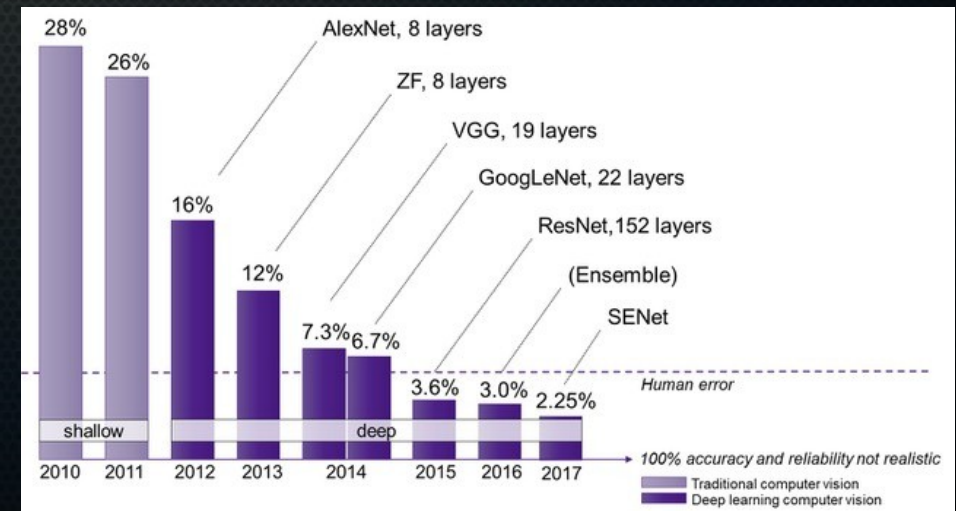
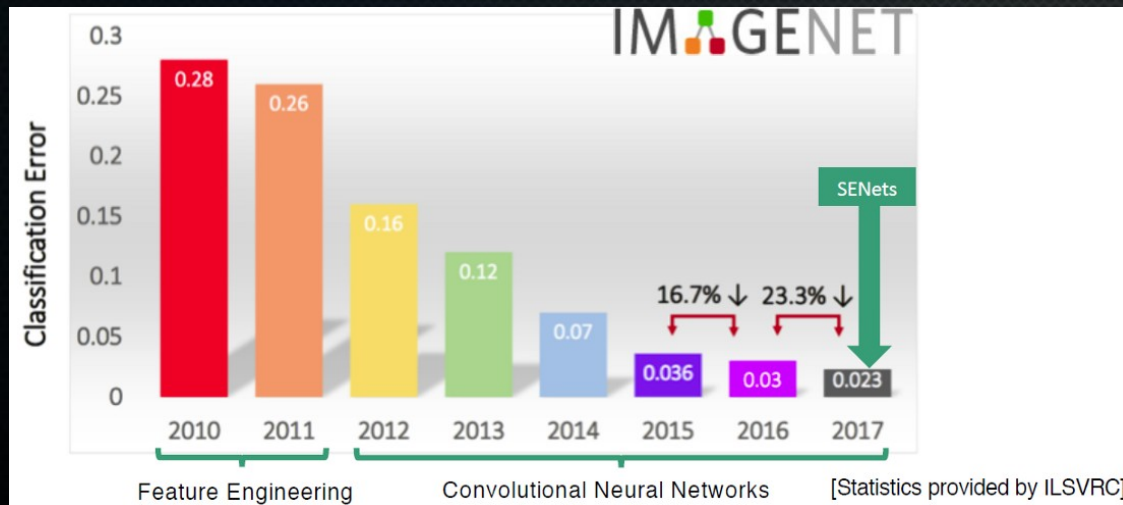


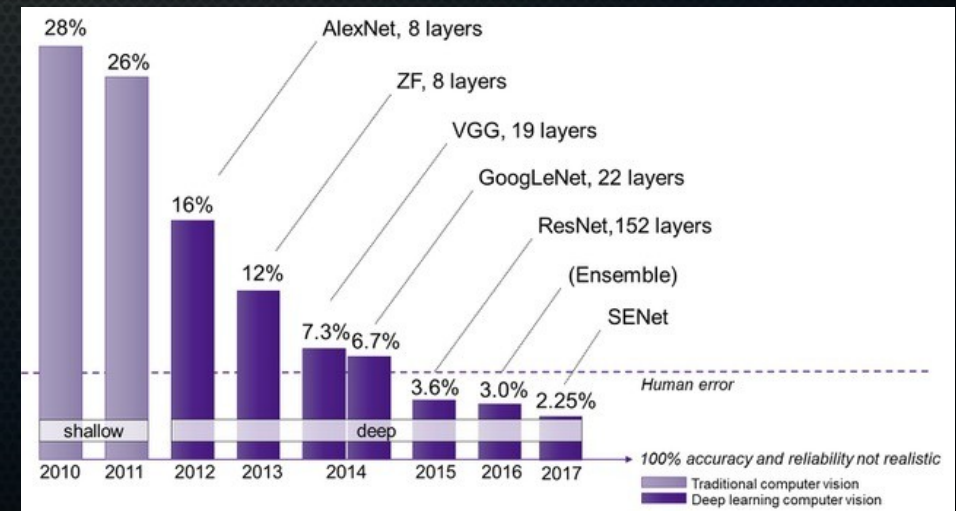
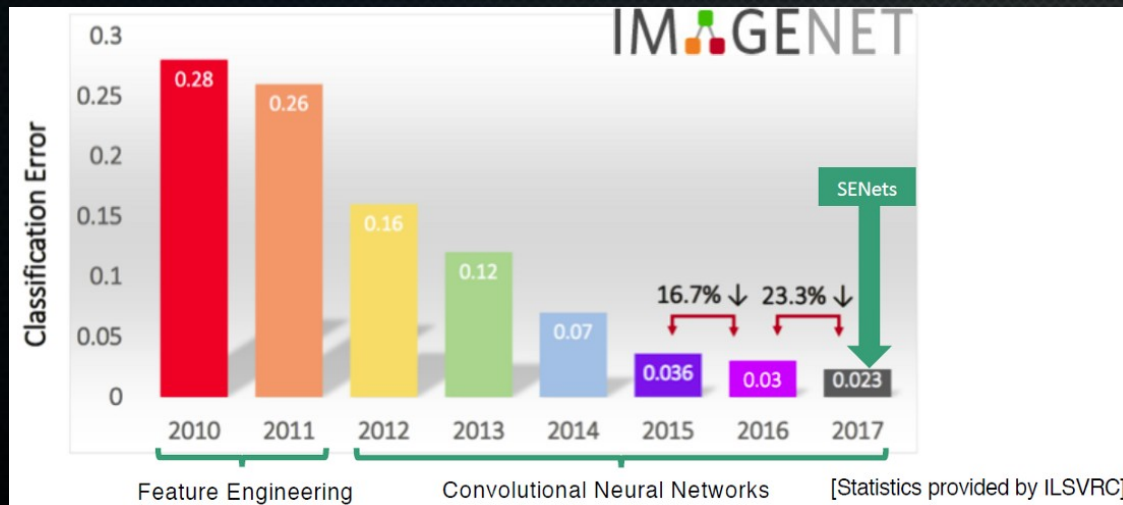
Convolutional neural networks

- These dense neural networks are not what made the huge strides in deep learning over the last few years.



Convolutional neural networks

- These dense neural networks are not what made the huge strides in deep learning over the last few years.
- Instead, those are deep convolutional neural networks



Convo-what now?

- Let's look at an image



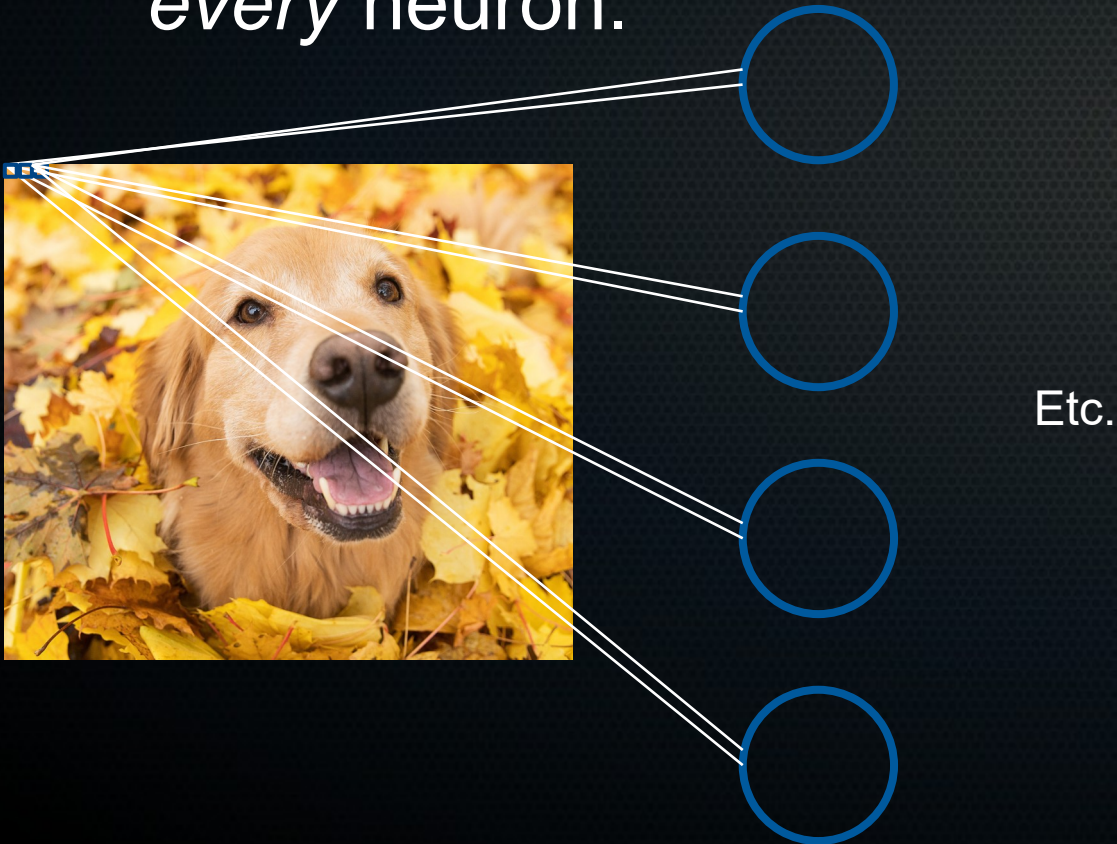
Convo-what now?

- Let's look at an image
- In a dense architecture, every pixel value is connected to every neuron.



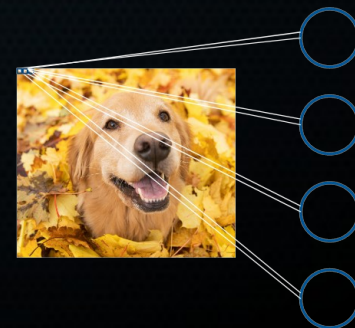
Convo-what now?

- Let's look at an image
- In a dense architecture, every pixel value is connected to every neuron.



Convo-what now?

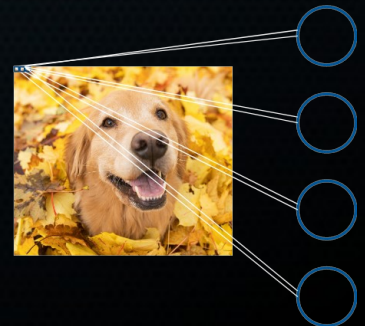
- Let's look at an image
- In a dense architecture, every pixel value is connected to *every* neuron.
- This gives problems:
 - You get an *insane* amount of parameters to optimise. 250×250 pixels * 20 input neurons = 1,250,020 weights and biases. You can forget about any sort of findable or achievable (global) optimum.
 - There is no locality: if you want your network to know whether or not there is a dog in an image, all these parameters must be optimised so that you can recognise the dog anywhere.



Convo-what now?



This is madness!



Convo-what now?

- The answer: convolution. Let's look at a 1D example!



- When is the heart beating?

Convo-what now?

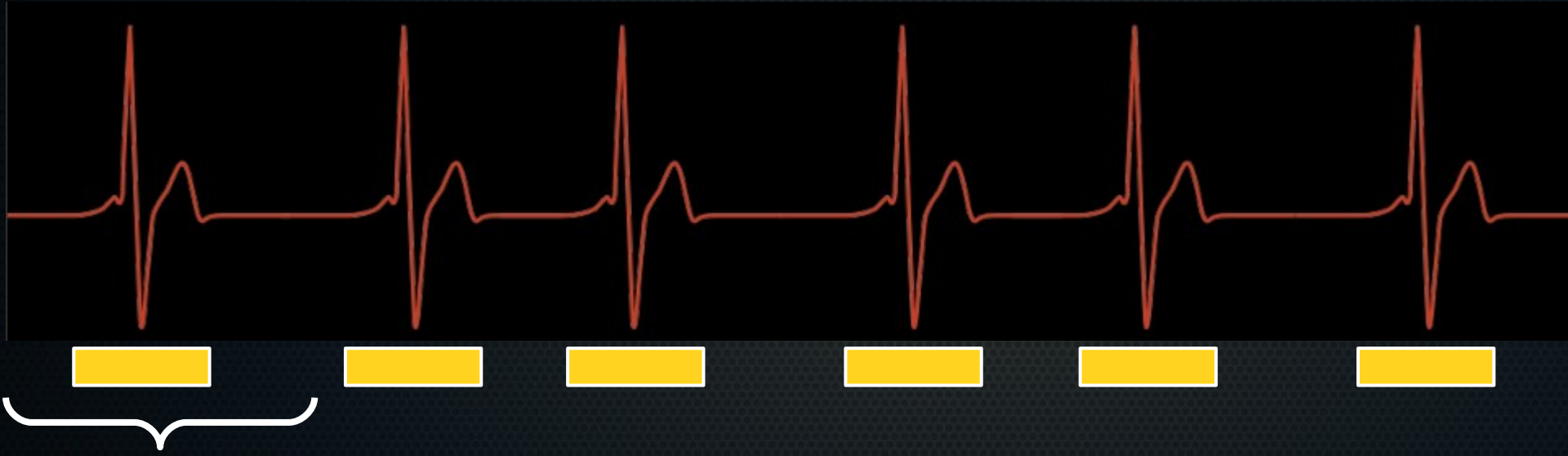
- The answer: convolution. Let's look at a 1D example!



- When is the heart beating?

Convo-what now?

- The answer: convolution. Let's look at a 1D example!



signal = $x = [0 \quad 0 \quad 0 \quad 1 \quad -1 \quad 0.5 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$

label = $y = [0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$

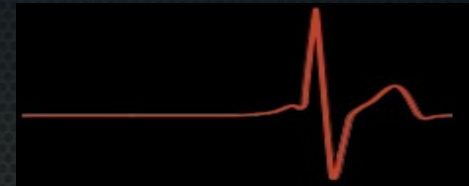
Convo-what now?

- Signal can be at different positions in the sequence:

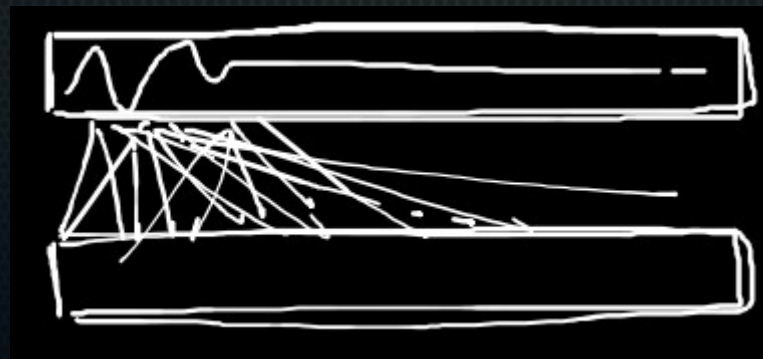


Convo-what now?

- Signal can be at different positions in the sequence:



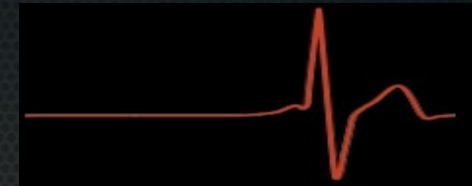
- Dense network needs to optimise such that different weights somehow cause the network to output 1 for different positions of the signal:



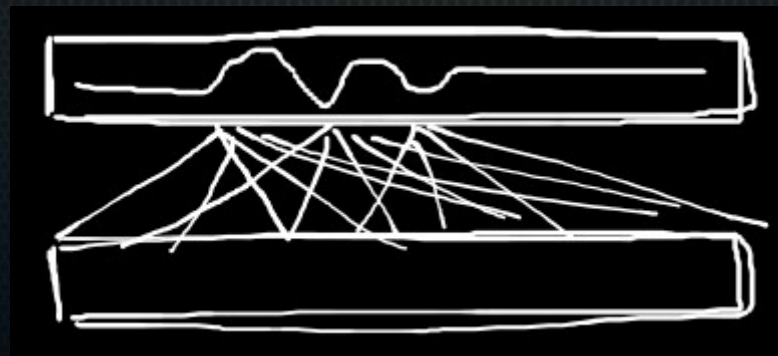
$$\text{label} = y = [1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]$$

Convo-what now?

- Signal can be at different positions in the sequence:



- Dense network needs to optimise such that different weights somehow cause the network to output 1 for different positions of the signal:



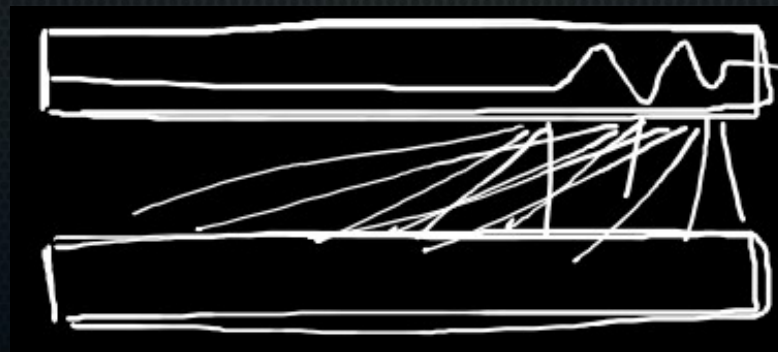
label = $y = [0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0]$

Convo-what now?

- Signal can be at different positions in the sequence:



- Dense network needs to optimise such that different weights somehow cause the network to output 1 for different positions of the signal:



$$\text{label} = y = [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1]$$

Convo-what now?

- Convolution:



Input

0	0	0	1	-1	0.5	0	0	0	0
---	---	---	---	----	-----	---	---	---	---

Kernel

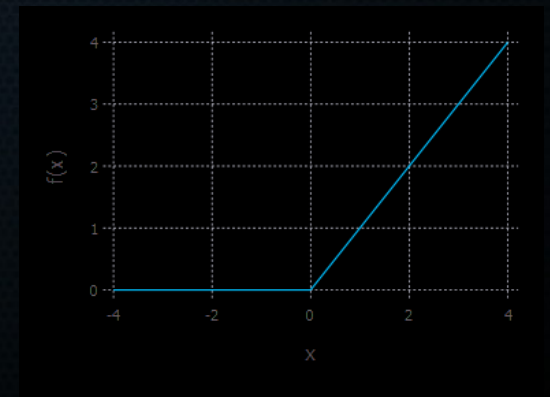
2

Kernel output

0									
---	--	--	--	--	--	--	--	--	--

ReLU output

0	0	0	1	-1	0.5	0	0	0	0
---	---	---	---	----	-----	---	---	---	---



Convo-what now?

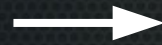
- Convolution:



Input

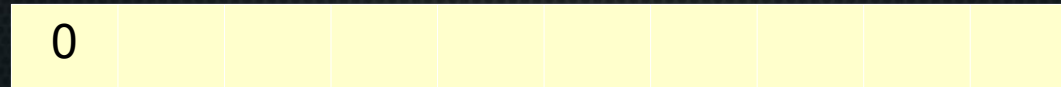


Kernel

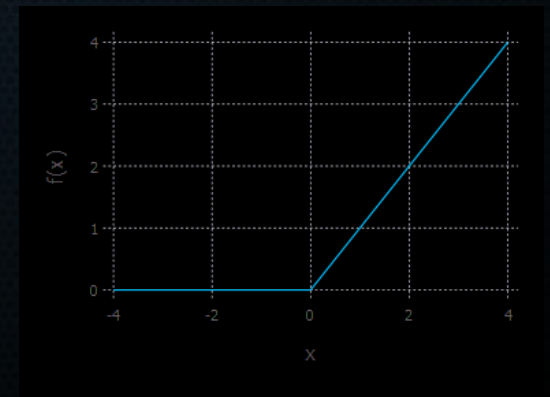


Convolve (move) the
kernel over the sequence

Kernel output



ReLU output



Convo-what now?

- Convolution:



Input

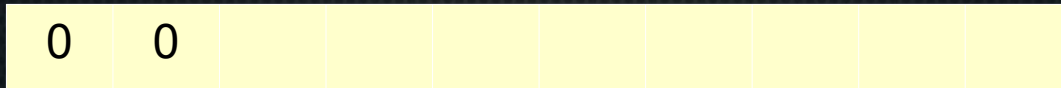


*

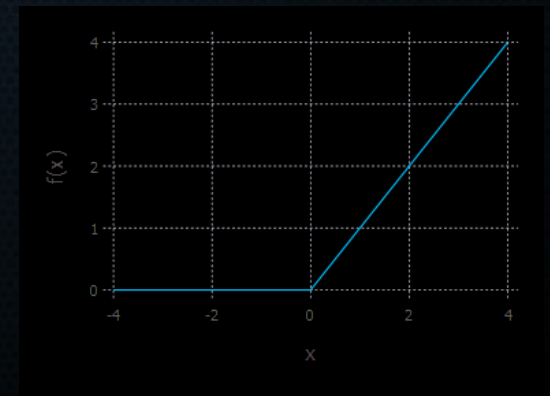
Kernel



Kernel output



ReLu output



Convo-what now?

- Convolution:



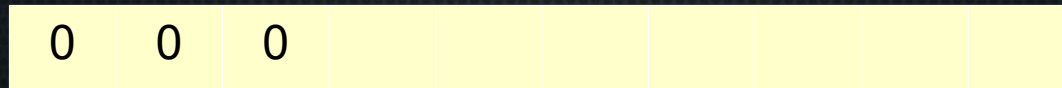
Input



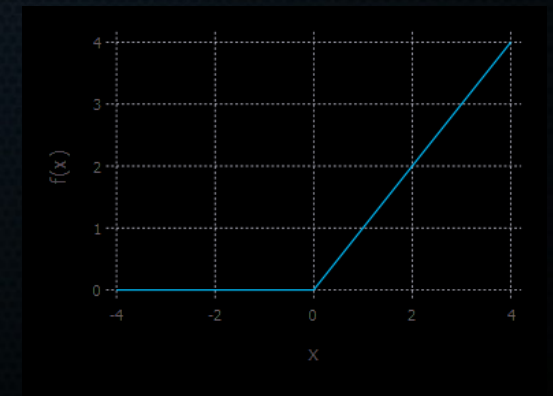
Kernel



Kernel output



ReLU output



Convo-what now?

- Convolution:

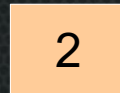


Input

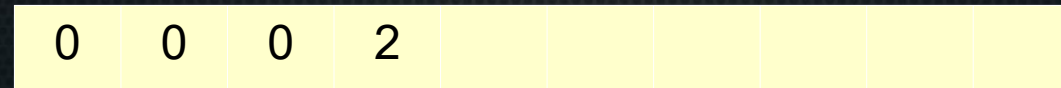


*

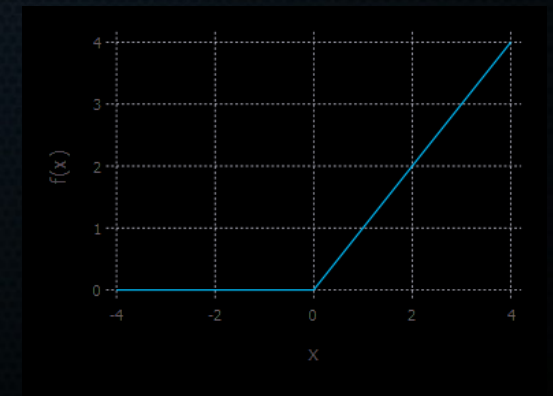
Kernel



Kernel output



ReLU output



Convo-what now?

- Convolution:



Input

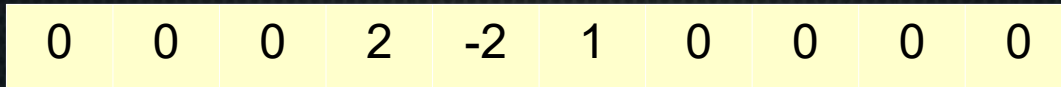


*

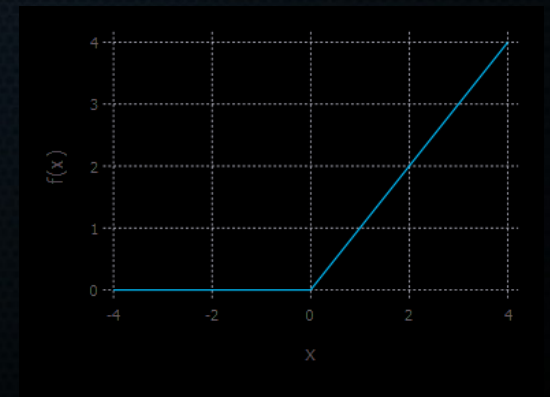
Kernel

2

Kernel output



ReLU output



Convo-what now?

- Convolution:



Input

0	0	0	1	-1	0.5	0	0	0	0
---	---	---	---	----	-----	---	---	---	---

*

Kernel

2

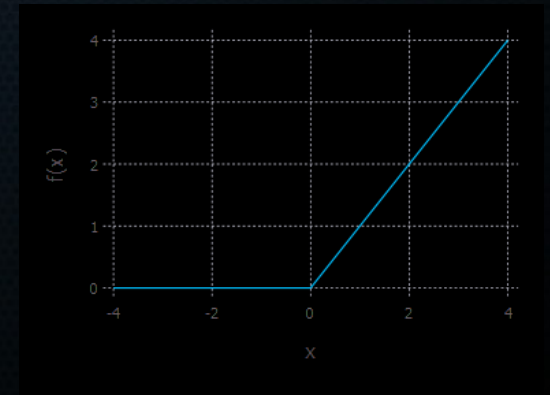
Kernel output

0	0	0	2	-2	1	0	0	0	0
---	---	---	---	----	---	---	---	---	---

ReLU output

0	0	0	2	0	1	0	0	0	0
---	---	---	---	---	---	---	---	---	---

This is a kernel that detects
positive numbers



Convo-what now?

- Convolution:



Input



Kernel

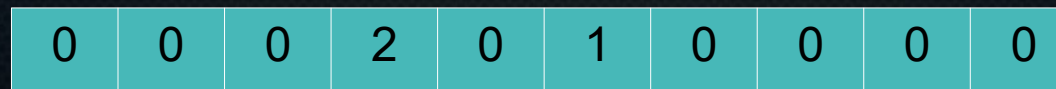


Just like a neuron has weights, this kernel has a trainable weight (2)

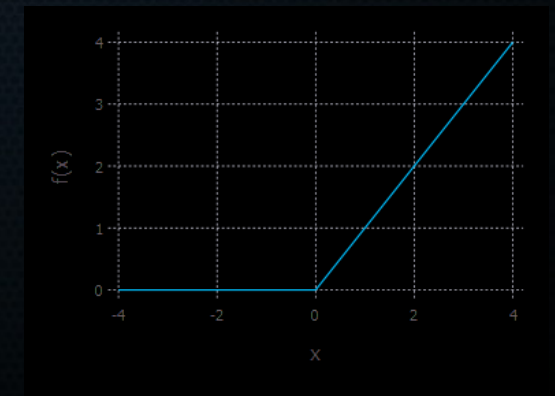
Kernel output



ReLU output

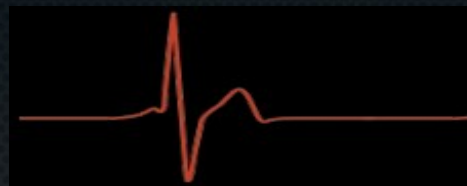


This is a kernel that detects positive numbers

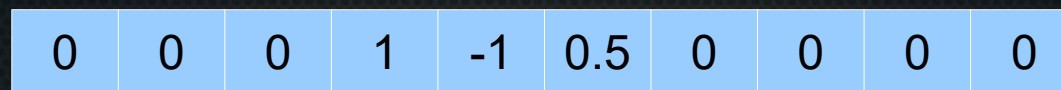


Convo-what now?

- Convolution:



Input

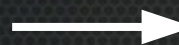
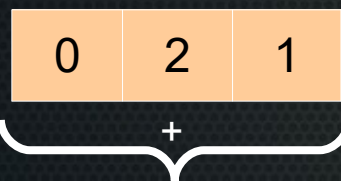


*

*

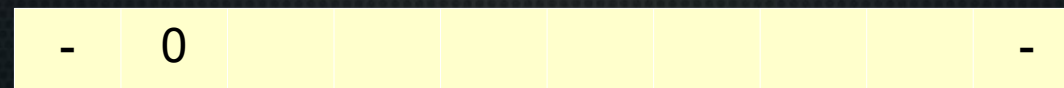
*

Kernel

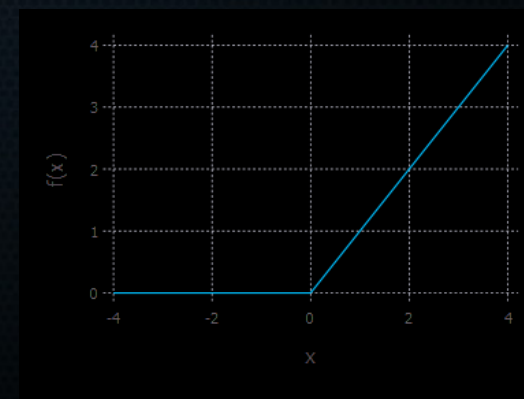


A kernel can have a size >1

Kernel output



ReLU output



Convo-what now?

- Convolution:



Input

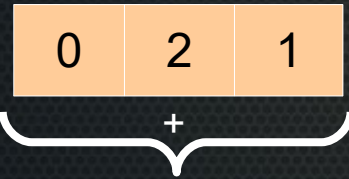


*

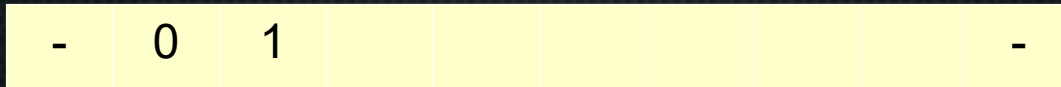
*

*

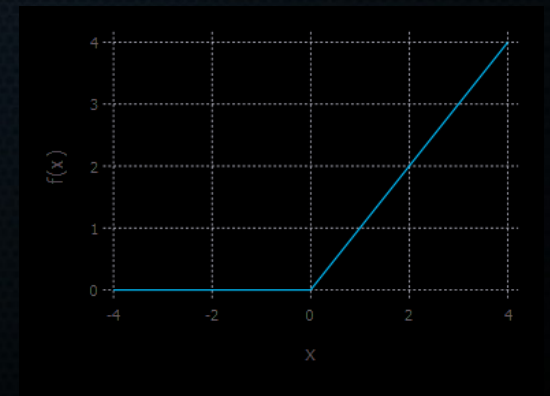
Kernel



Kernel output



ReLU output



Convo-what now?

- Convolution:



Input

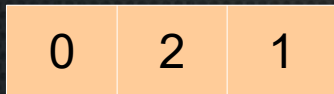


*

*

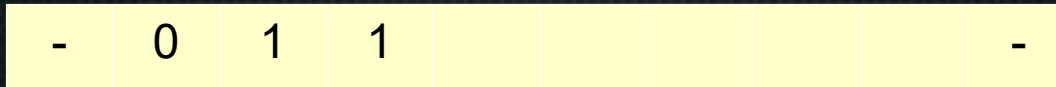
*

Kernel

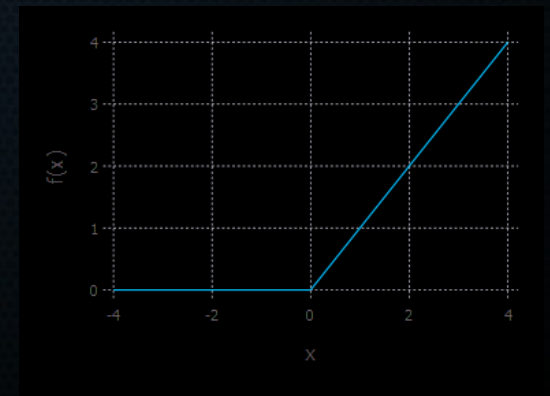


+

Kernel output



ReLU output



Convo-what now?

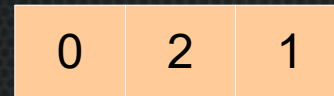
- Convolution:



Input

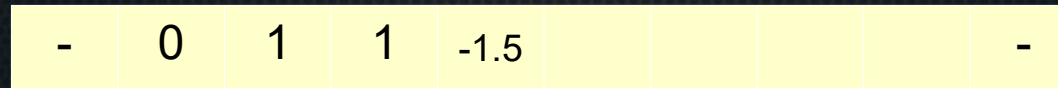


Kernel

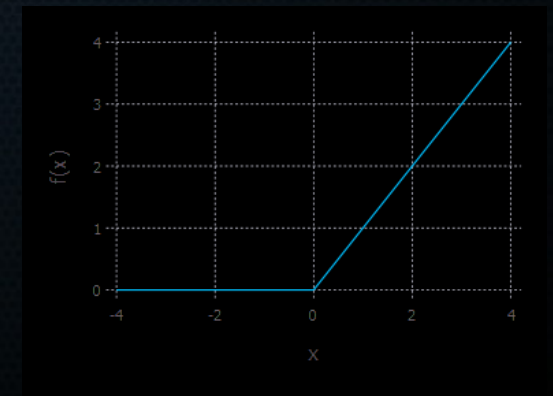


+

Kernel output



ReLU output



Convo-what now?

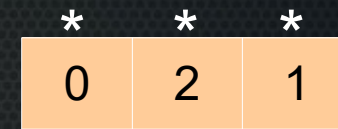
- Convolution:



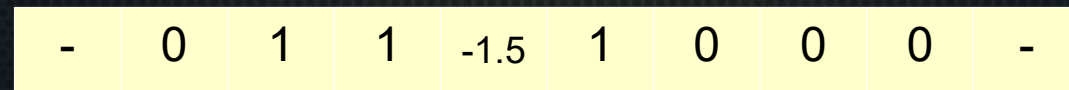
Input



Kernel

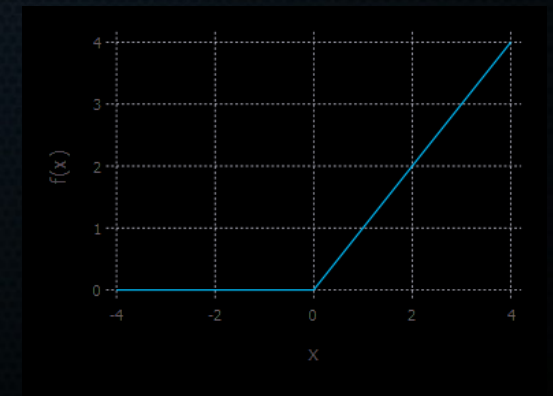
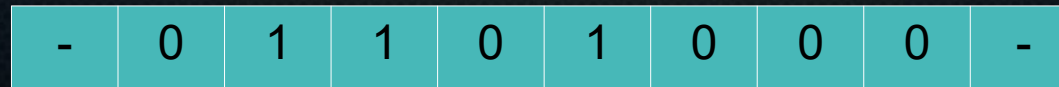


Kernel output



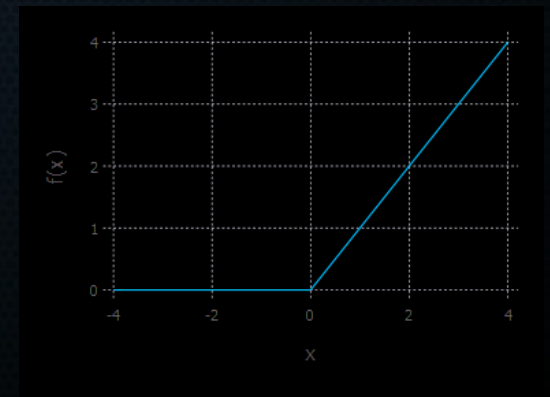
Note shrinkage due to edge effects

ReLU output



Convo-what now?

- Convolution:



Convo-what now?

- Convolution:



Input

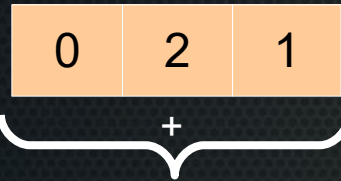


*

*

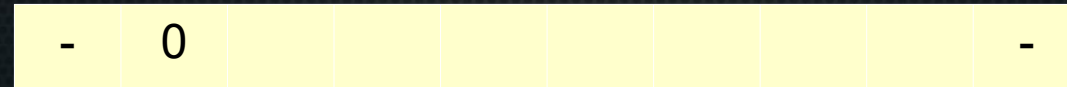
*

Kernel

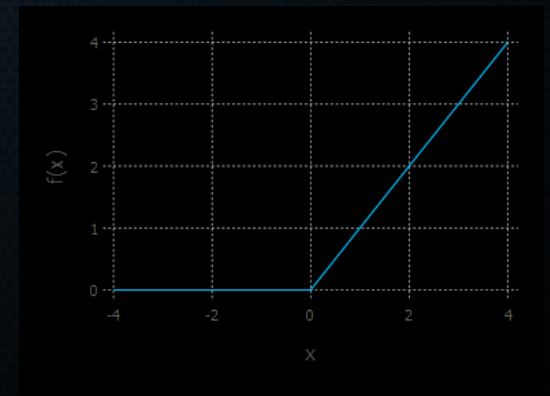
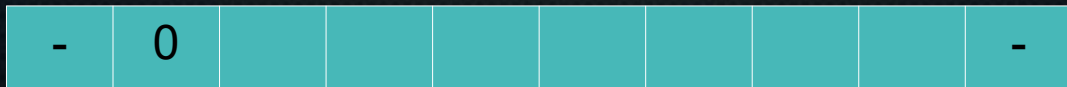


Or could compress output size further by changing *stride* (step size). Stride = 3 now.

Kernel output



ReLU output



Convo-what now?

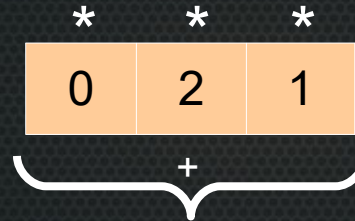
- Convolution:



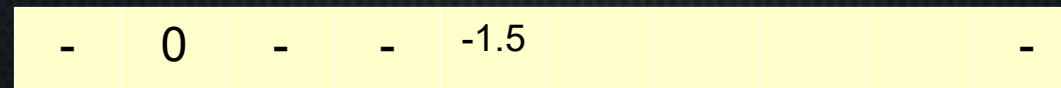
Input



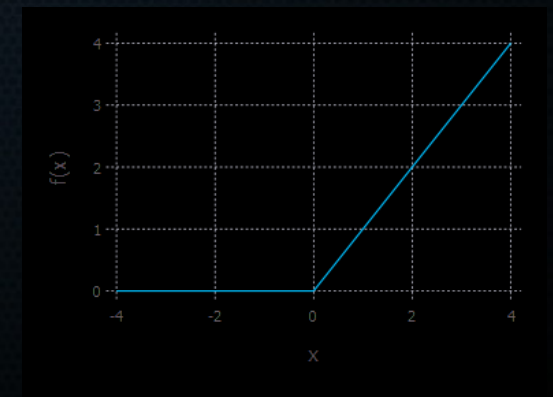
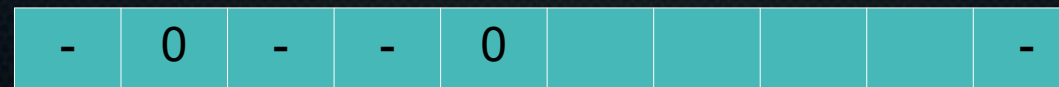
Kernel



Kernel output



ReLU output



Convo-what now?

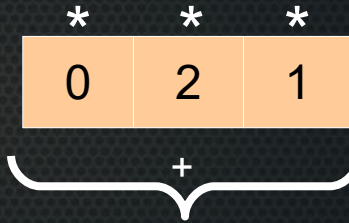
- Convolution:



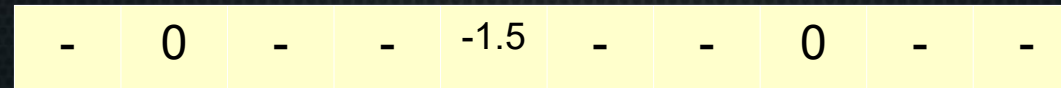
Input



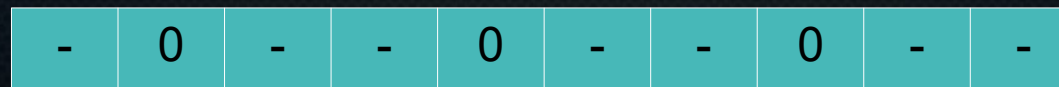
Kernel



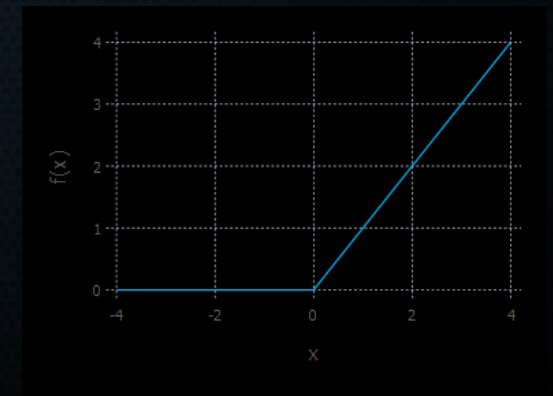
Kernel output



ReLU output



Size = 3
Stride = 3



Convo-what now?

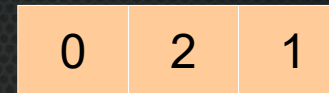
- Convolution:



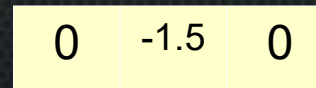
Input



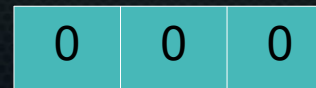
Kernel



Kernel output

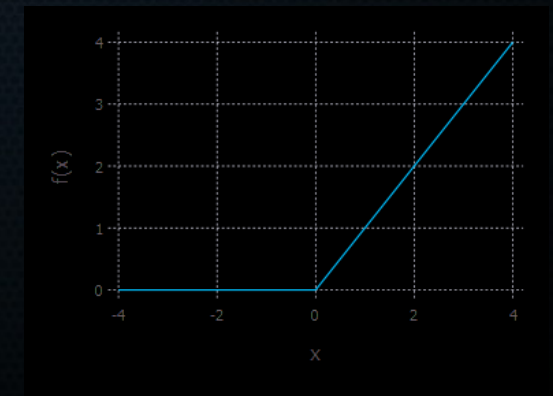


ReLU output



Size = 3
Stride = 3

This example is not useful because I randomly picked some weights for the kernel. But normally you can train these weights by backpropagation such that the network works well!



Convo-what now?

- Convolution:



Input

0	0	0	1	-1	0.5	0	0	0	0
---	---	---	---	----	-----	---	---	---	---

Kernel

0	2	1
---	---	---

+

Kernel output

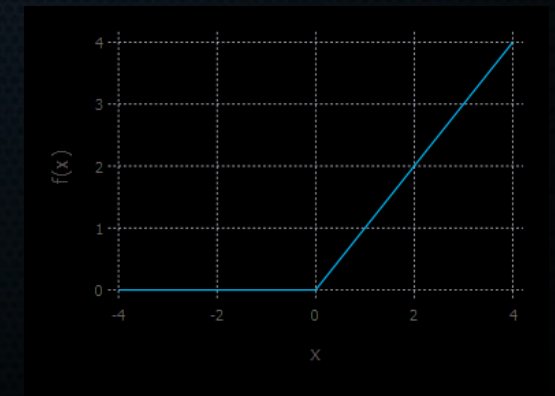
-	0	1	1	-1.5	1	0	0	0	-
---	---	---	---	------	---	---	---	---	---

ReLU output

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

Size = 3
Stride = 1

Note shrinkage due
to edge effects

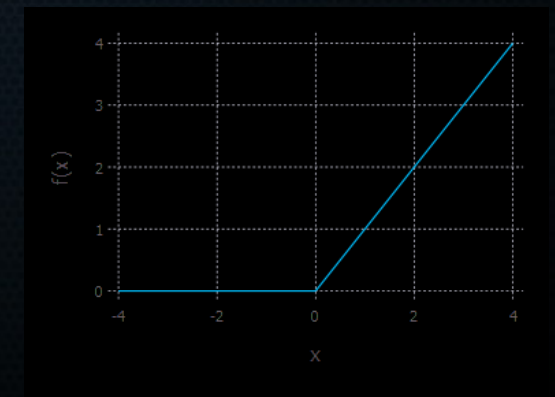
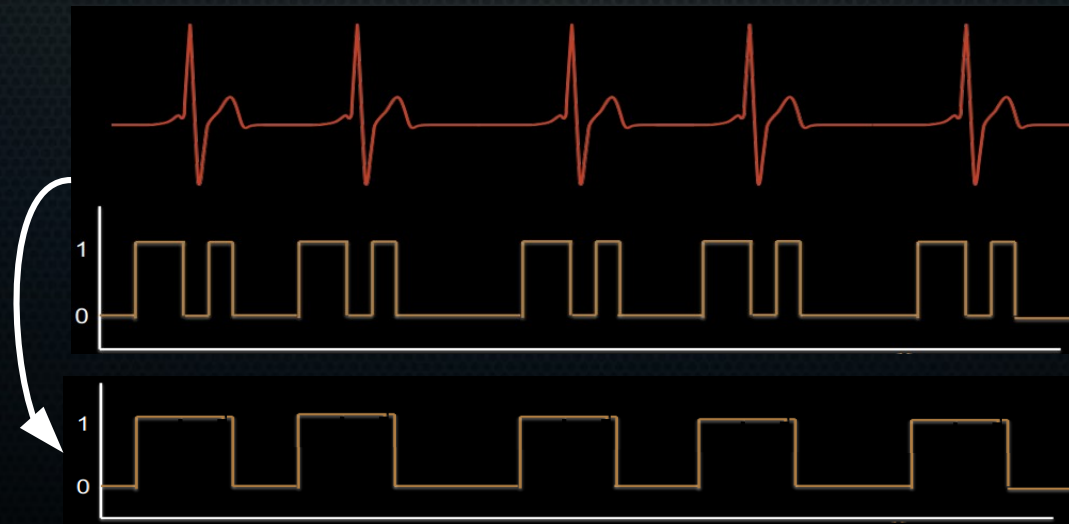


Convo-what now?

Input	0	0	0	1	-1	0.5	0	0	0	0
Kernel							*	*	*	
							0	2	1	
							+			
Kernel output	-	0	1	1	-1.5	1	0	0	0	-
ReLu output	-	0	1	1	0	1	0	0	0	-

Size = 3
Stride = 1

Not optimal, but by adding another convolution layer, you might get to something like:



2D convolution

- Size = 2×2 ; stride = 1

0	22	0	1
12	2	3	23
3	34	26	2
0	22	86	3
4	3	1	4

0	2
2	0

68		

2D convolution

- Size = 2×2 ; stride = 1

0	22	0	1
12	2	3	23
3	34	26	2
0	22	86	3
4	3	1	4

0	2
2	0

68	4	

2D convolution

- Size = 2×2 ; stride = 1

0	22	0	1
12	2	3	23
3	34	26	2
0	22	86	3
4	3	1	4

0	2
2	0

68	4	8

2D convolution

- Size = 2×2 ; stride = 1

0	22	0	1
12	2	3	23
3	34	26	2
0	22	86	3
4	3	1	4

0	2
2	0

68	4	8
10		

Etc.

Another type of convolution: max pooling

- Size = 2×2 ; stride = 1; just take the maximum value in the kernel area

0	22	0	1
12	2	3	23
3	34	26	2
0	22	86	3
4	3	1	4

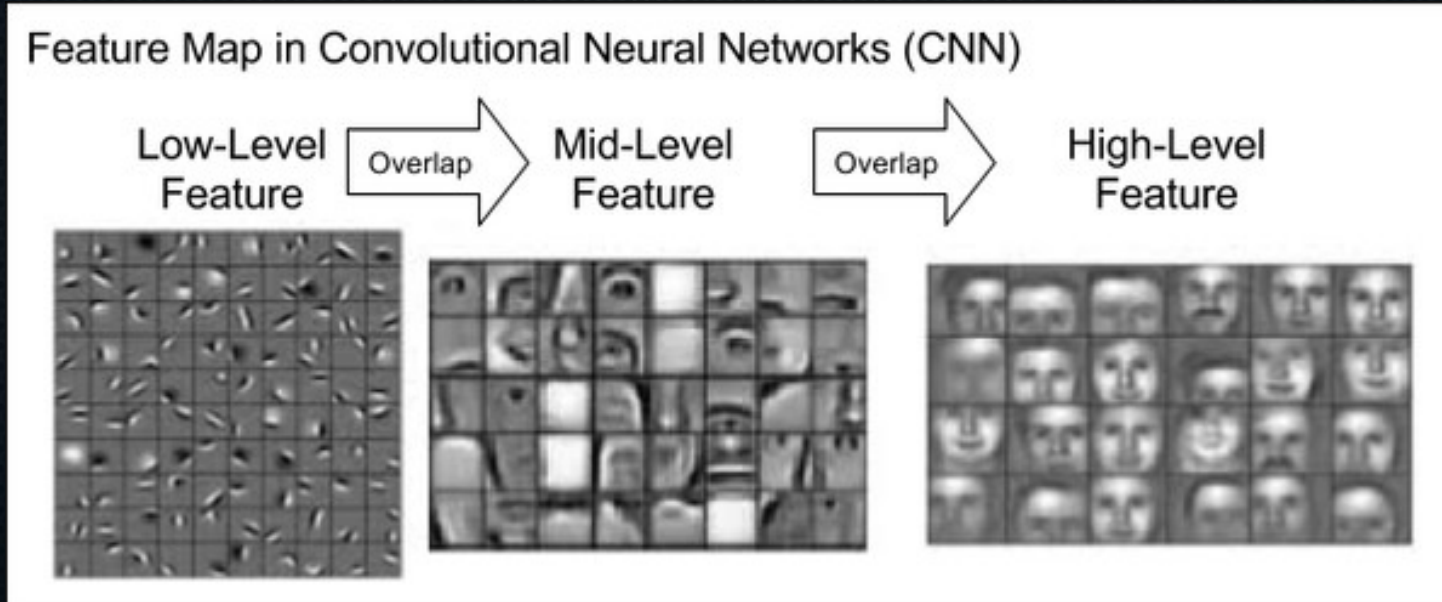
22	22	23
34	26	26
34	86	86
22	86	86

Another type of convolution: pooling/averaging

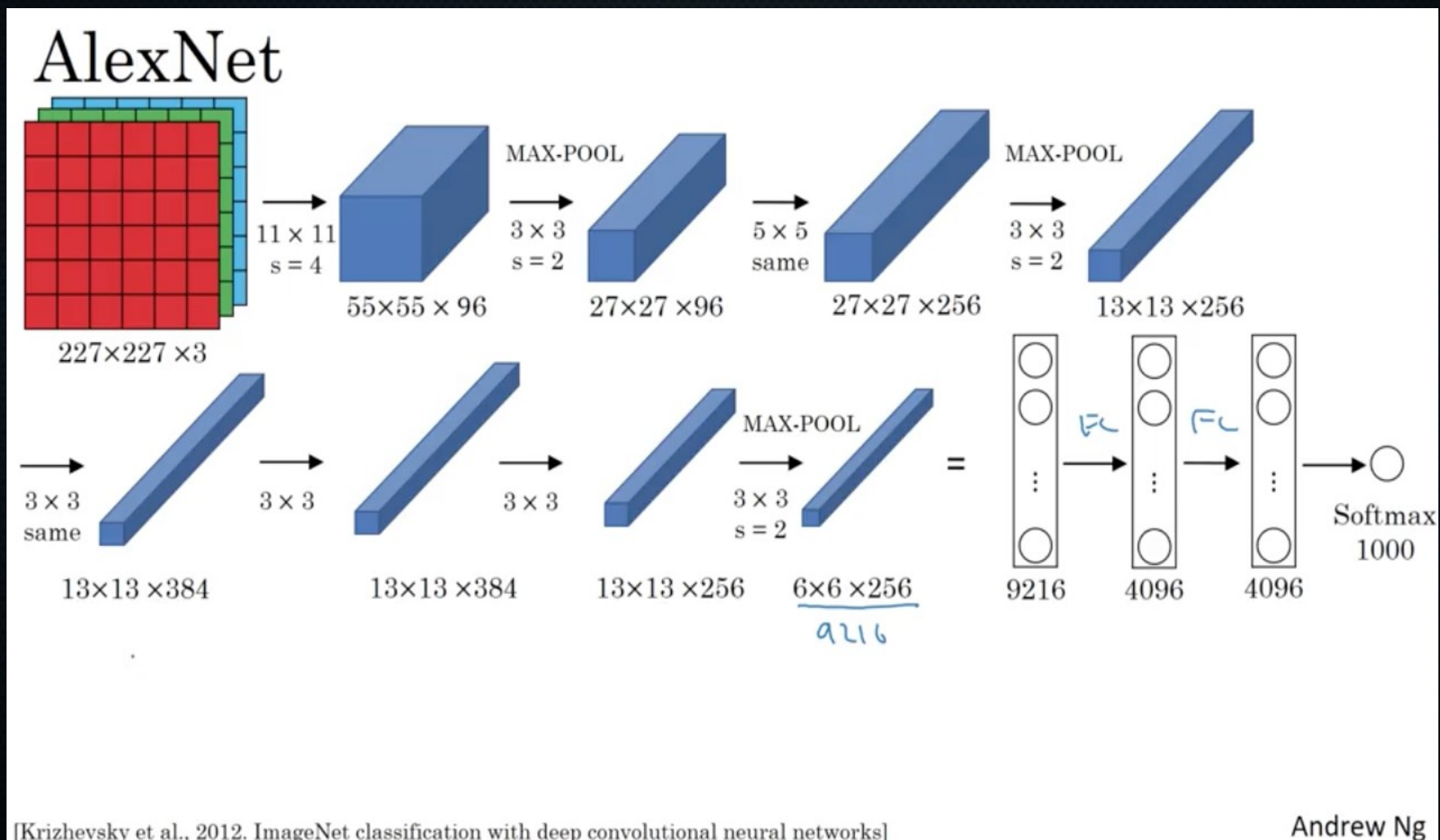


Use in face detection

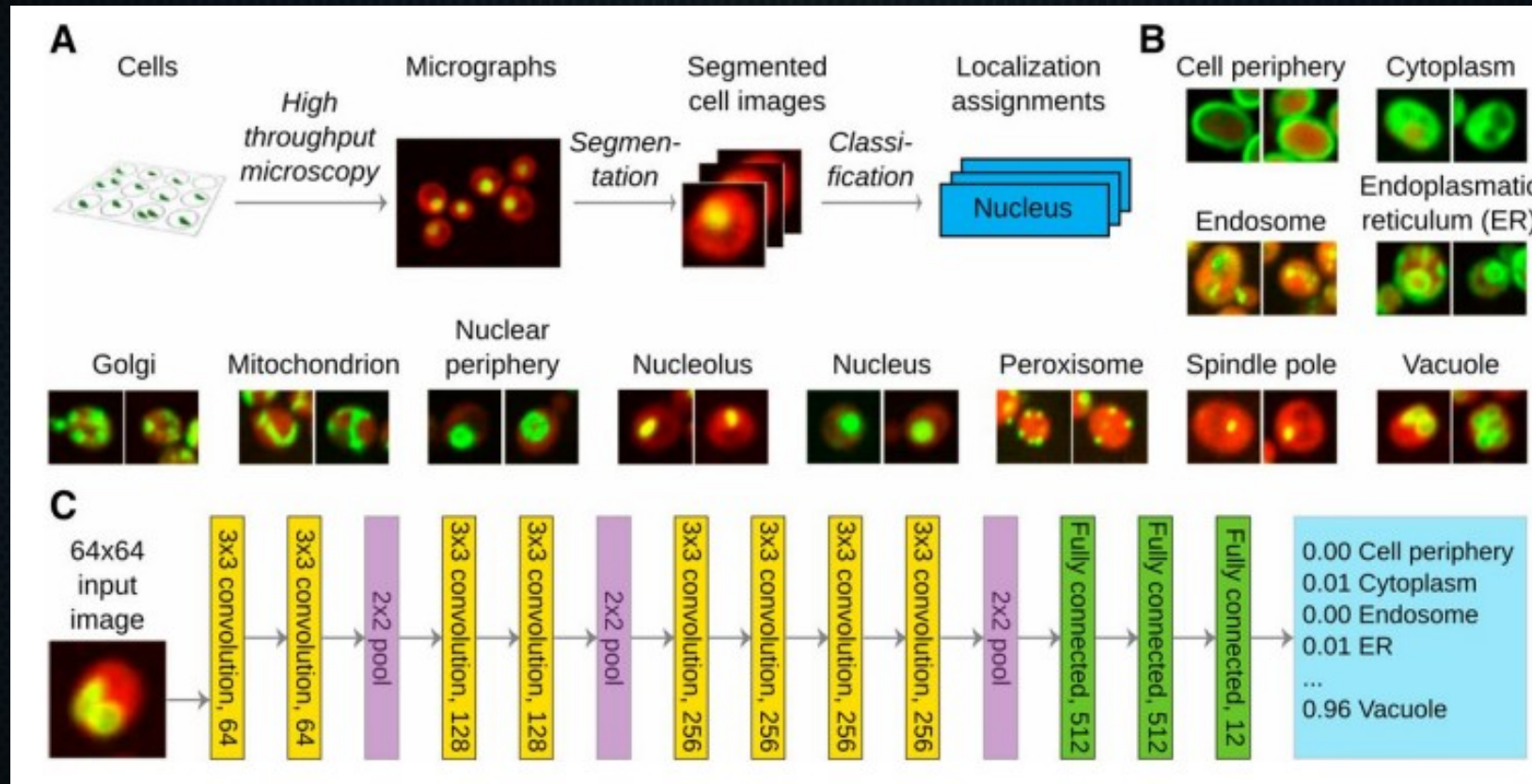
- Since kernels so few parameters: can use *many* of them per layer → each becomes sensitive to different image features



Example AlexNet (2012)



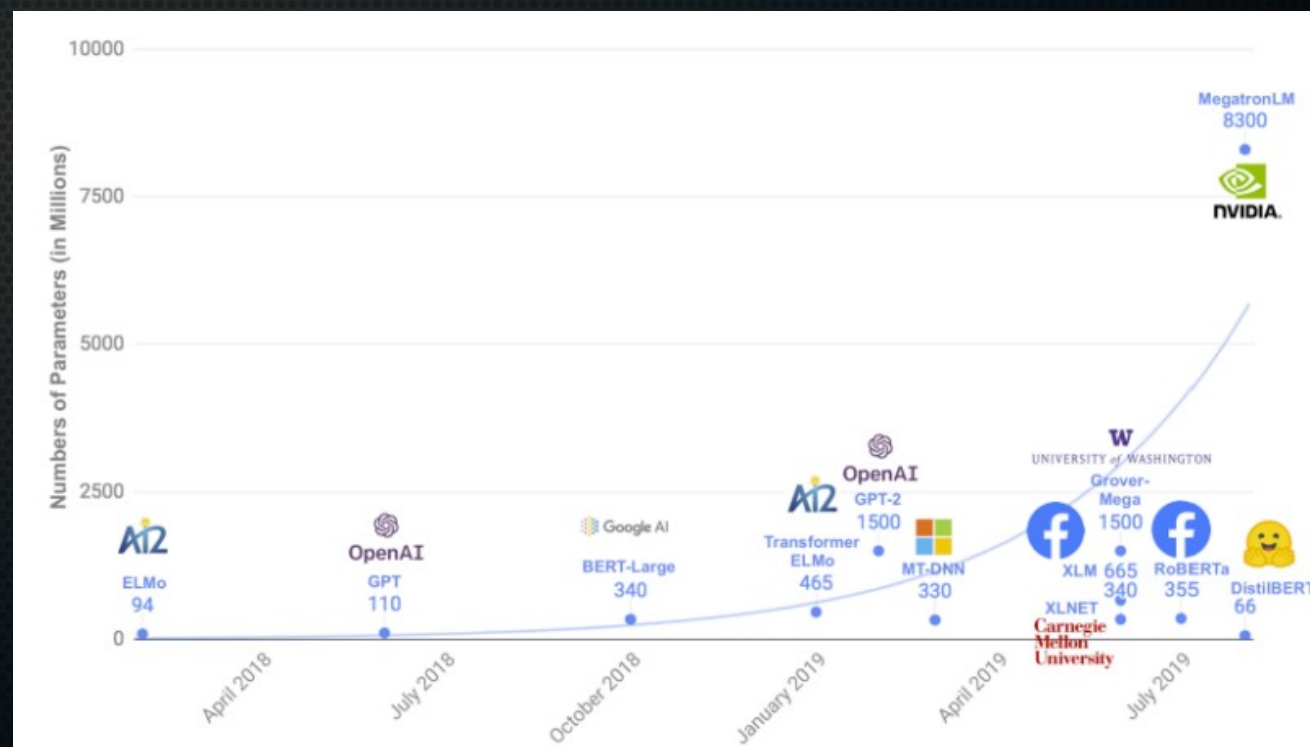
Biological use



Pärnamaa, T., & Parts, L. (2017). Accurate classification of protein subcellular localization from high-throughput microscopy images using deep learning. *G3: Genes, Genomes, Genetics*, 7(5), 1385-1392.

There's a lot more

- Batch normalisation
- Vanishing gradient problem
- Dropout
- Recurrent neural nets



Implementation

- We are not going to implement convolutional neural networks ourselves: implementing backpropagation properly on a simple dense network is already taxing enough.
- Still, doing that should give you a solid basis for understanding convolutional neural networks, and we'll introduce the Keras library for building (convolutional) neural networks next Monday.

Talk 13:15

- Marc Pages Gallego
- Jeroen de Ridder group
- Working with Oxford NanoPore sequencing data
- Talk about ML/deep learning in interpreting sequencing data from nanopores

Afternoon practical

- Implement backpropagation yourself
- Train a dense neural network on the MNIST dataset