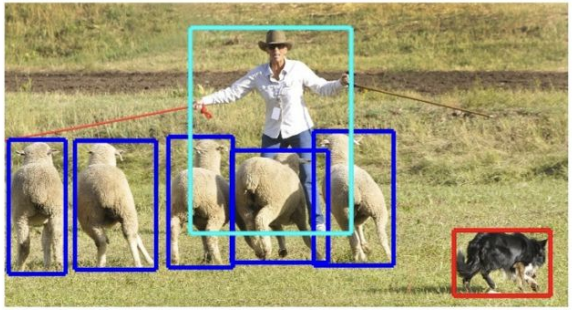
AICP: Image module
Day 4 : Deep Learning Theory

Guillaume Witz
Data Science Lab, University of Bern

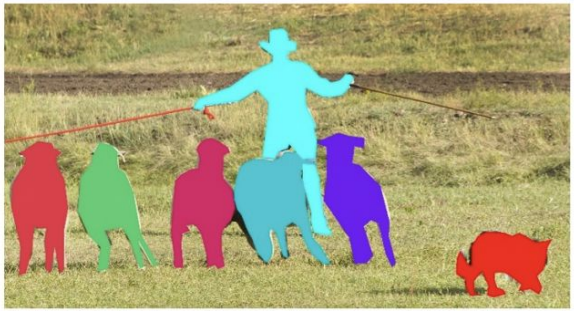
Classical Computer Vision tasks with NN

Classification



Object detection

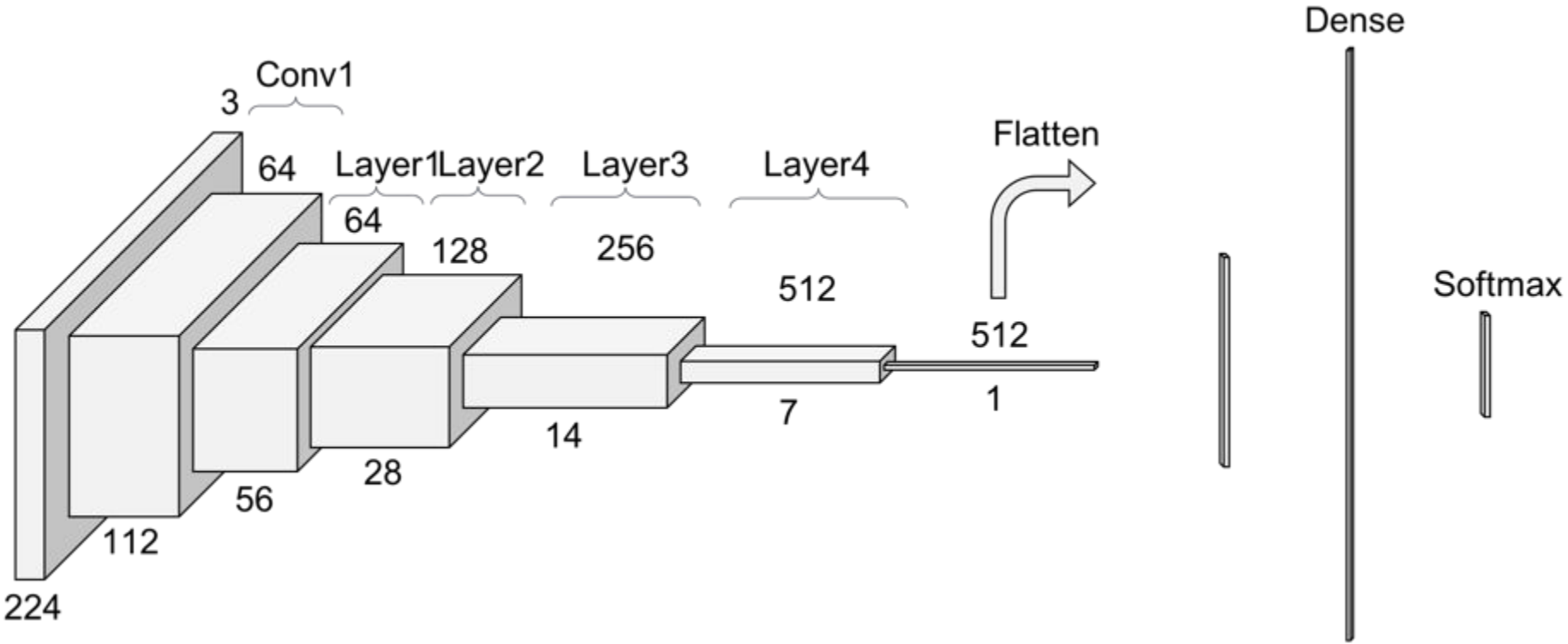
Semantic segmentation



Instance segmentation

Lin T.-Y. et al. arXiv:1405.0312, 2014

You will often see this kind of NN schema



In the following slides we will try to make it clear what this represents !

DSL

Let's start with simple linear regression

Wine dataset

Measure chemical properties

- Wine acidity
- Wine alcohol
- Wine sugar
- ...

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
...	...	...	...	...	...	...	...	...	...	...	...	...

The simplest "AI": linear regression

Can we predict acidity using density?

Model: a simple line

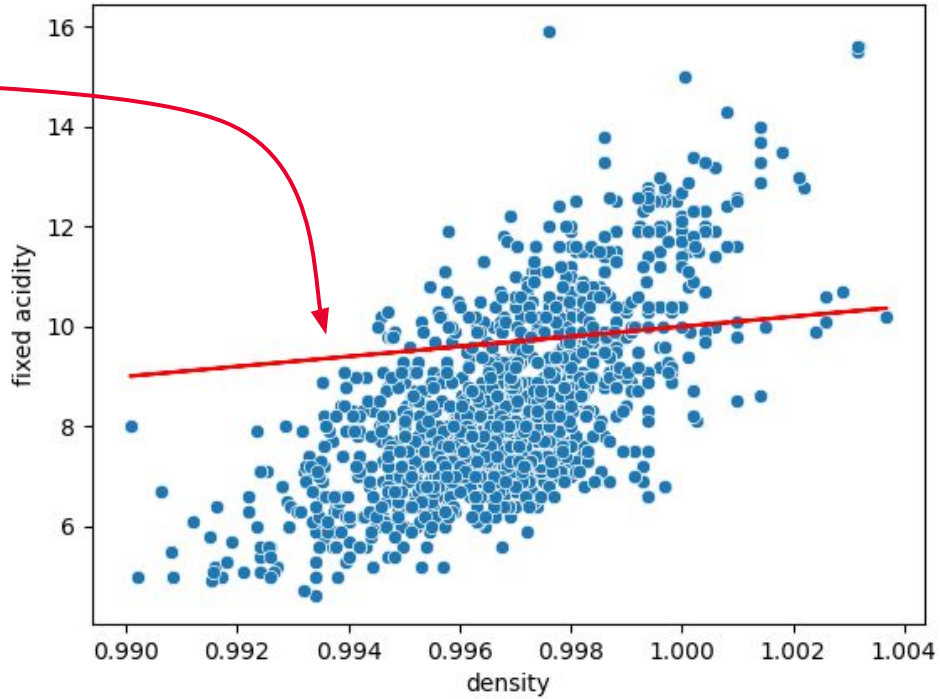
$y = a^*x + b$

"Fixed acidity" = w_1 * "density" + w_0

Target

Feature

Parameters / weights



The simplest "AI": linear regression

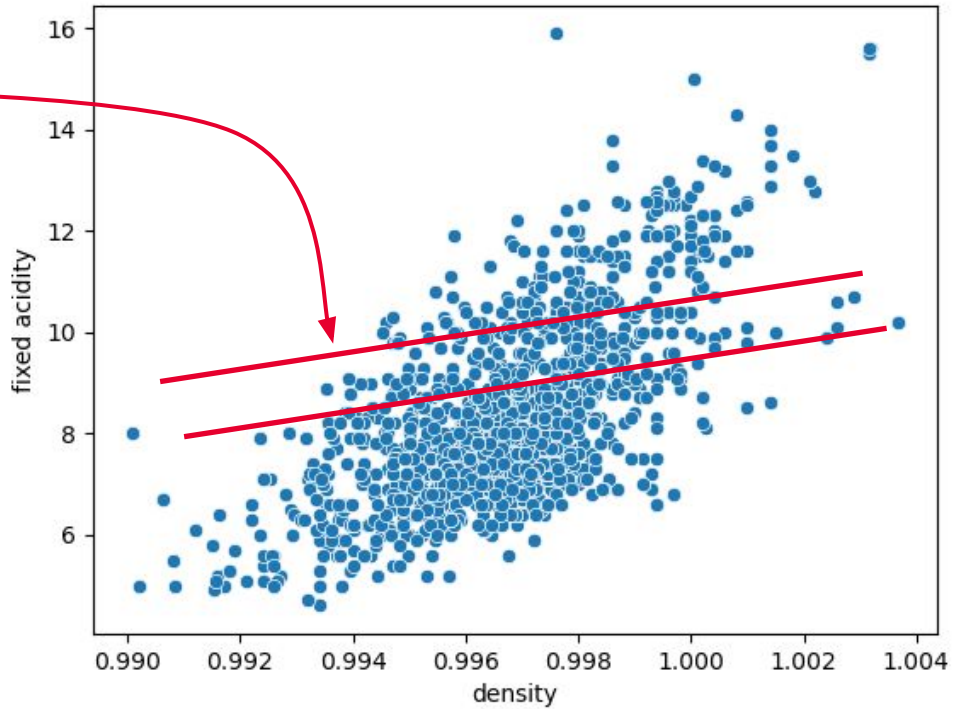
Can we predict acidity using density?

Model: a simple line

$y = a^*x + b$

"Fixed acidity" = w_1 * "density" + w_0

Goal: find the best w_1, w_0 to describe the training data



The simplest "AI": linear regression

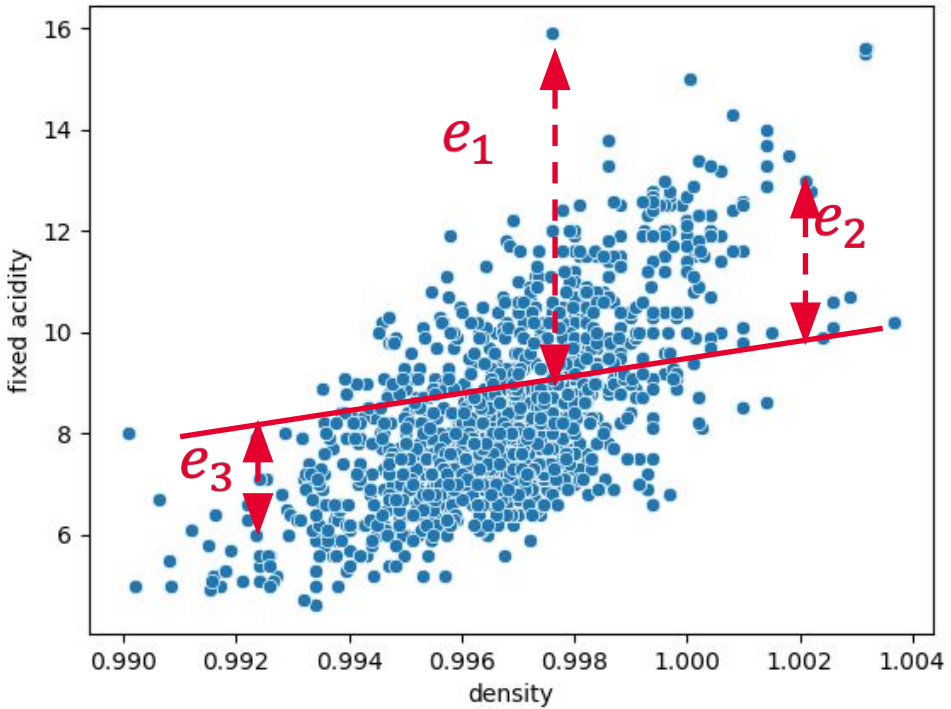
How do we estimate how good a model is ?

For each point in the dataset measure the distance between actual value and predicted value:

$$e_i$$

Compute a total error metric. For example the mean error over all points.

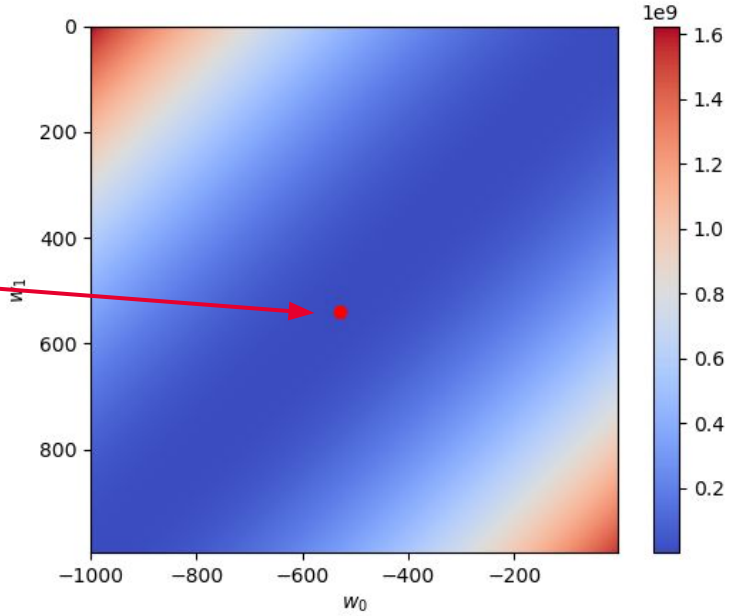
$$\text{loss} = \sum_{i=1}^{N\ points} e_i$$



The simplest "AI": linear regression

- We can try all possible values for our parameters
- For each pair of values we get a loss
- From all the losses, the best model is the one with the smallest value
- If we are close to the optimum, changing a bit a weight is not dramatic -> the loss is "smooth"

"Fixed acidity" = w_1 * "density" + w_0



Bad region

Good region

Number of features and parameters

Generally, models have many more features and parameters to handle

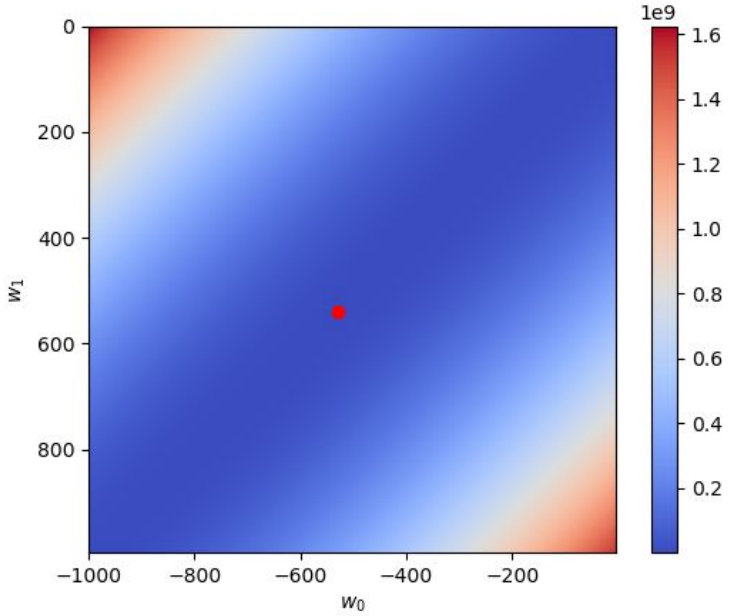
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4
...	...	...	...	...	...	...	...	...	...	...	...

$$y = w_1 * f_1 + w_2 * f_2 + \dots w_n * f_n + w_0$$

To find the best value, we would have to explore a gigantic number of combinations of w_1, w_2, … w_n

In deep learning we would have billions of parameters: we need another solution!

"Fixed acidity" = w_1 * "density" + w_0

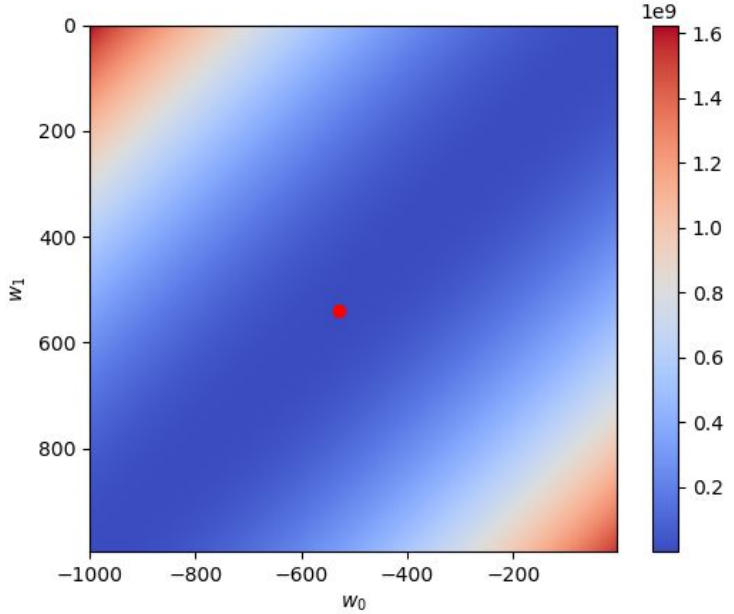


Bad region

Good region

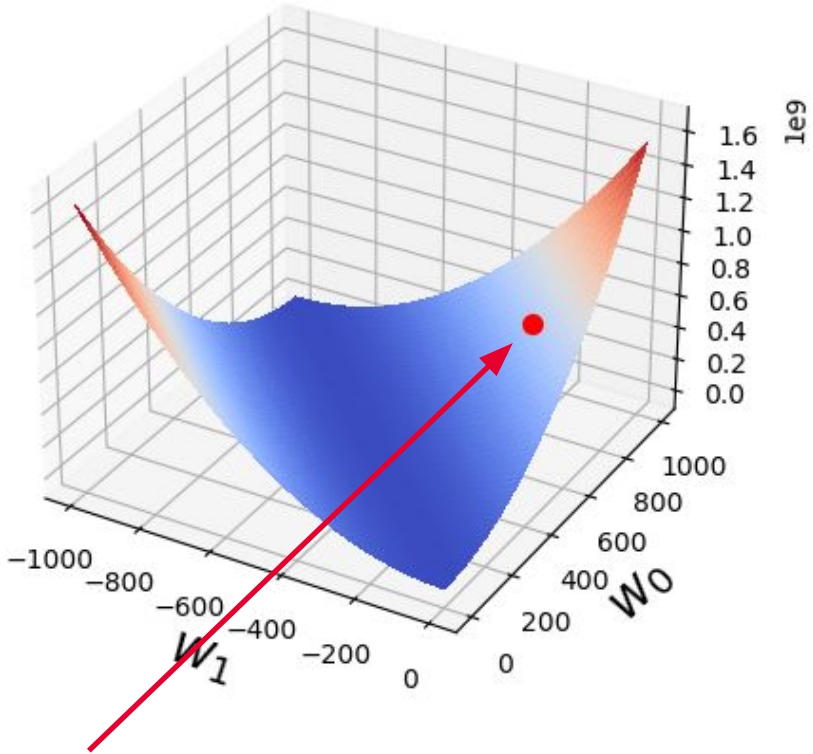
Weights space as mountains and hills

"Fixed acidity" = w_1 * "density" + w_0



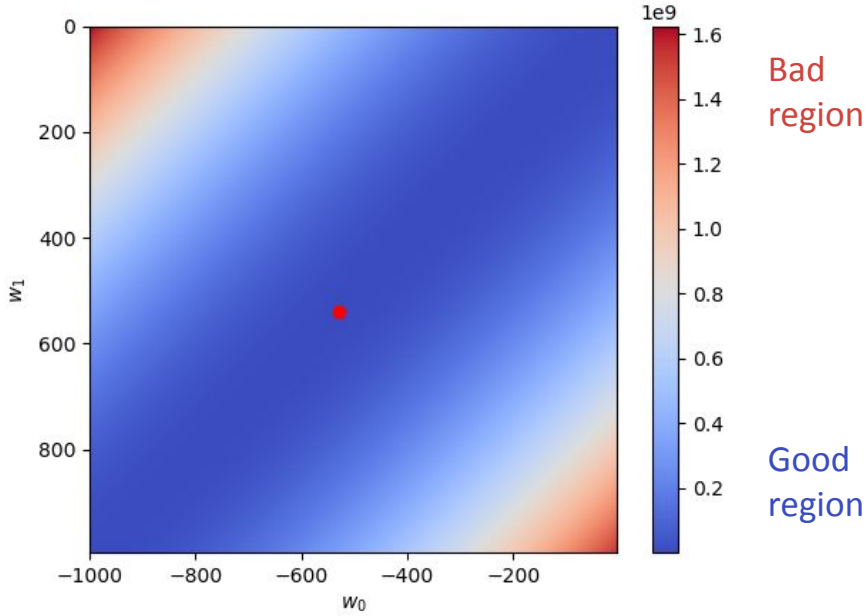
Bad region

Good region



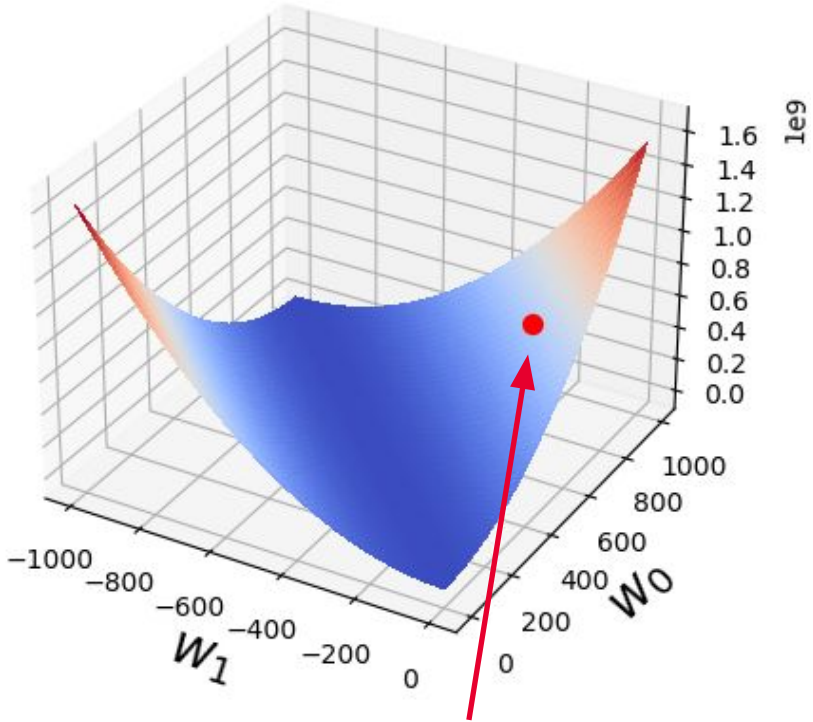
If we start at a random position (random weights), how can we find the deepest valley == the smallest loss == best model ?

Weights space as mountains and hills



Bad region

Good region



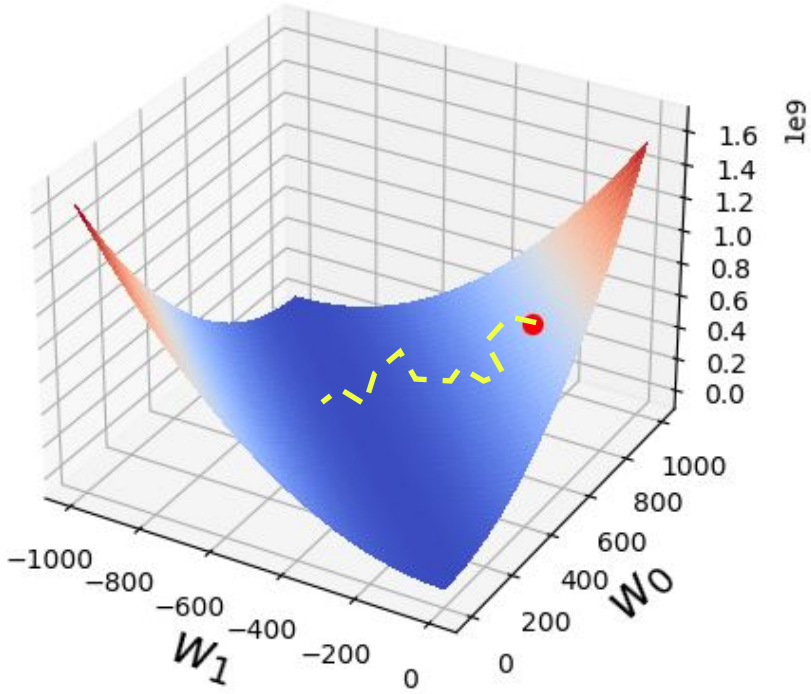
We can view the loss function space as a plane with mountains and hills.

If we start at a random position (random weights), how can we find the deepest valley == the smallest loss == best model ?

Follow the slope!

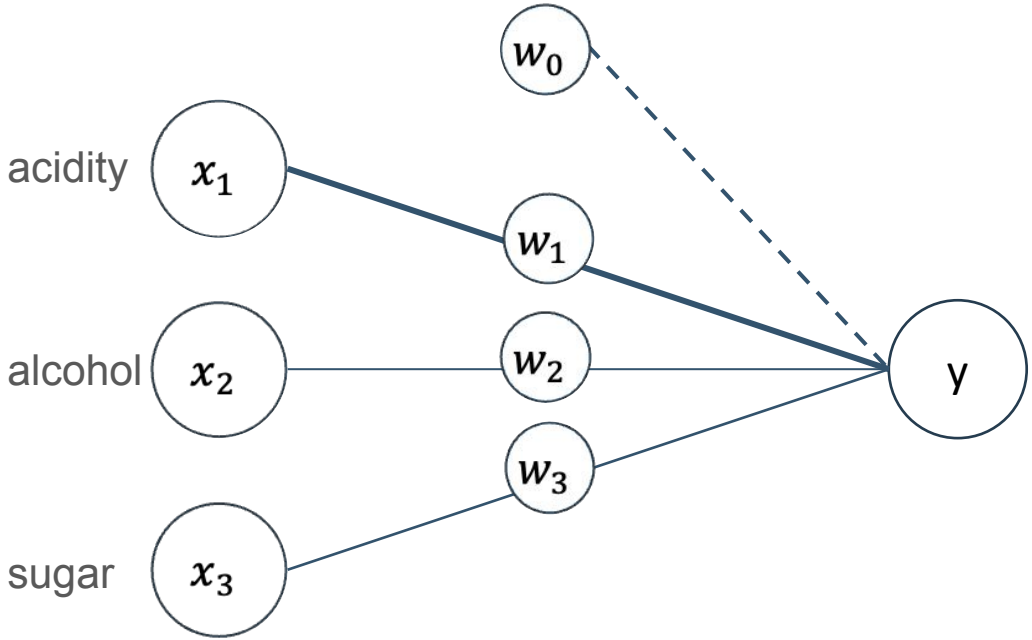
- Check in which direction the slope is steepest
- Take one step in that direction
- Continue until you are in a “flat” region and new steps don’t change the location much
- The same approach is used in hyperspaces with billions of parameters.

This powerful method is called Gradient Descent !



From equation to NN

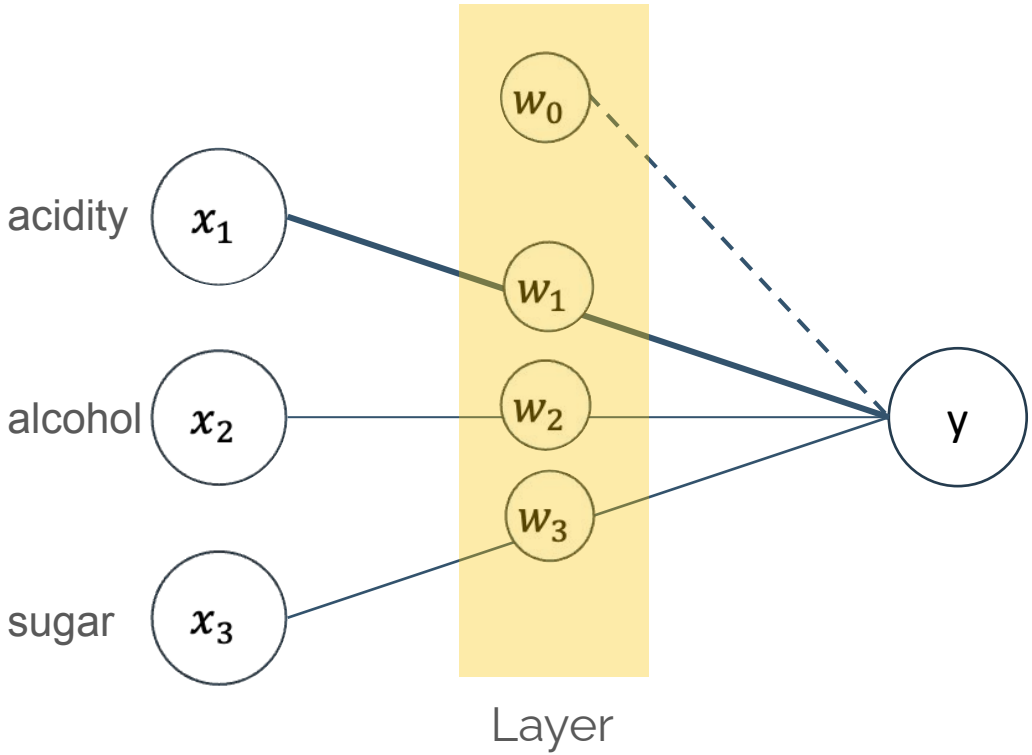
A neural network is just a representation of the type of equations used for linear regression. For example here we use three features to predict the grade.



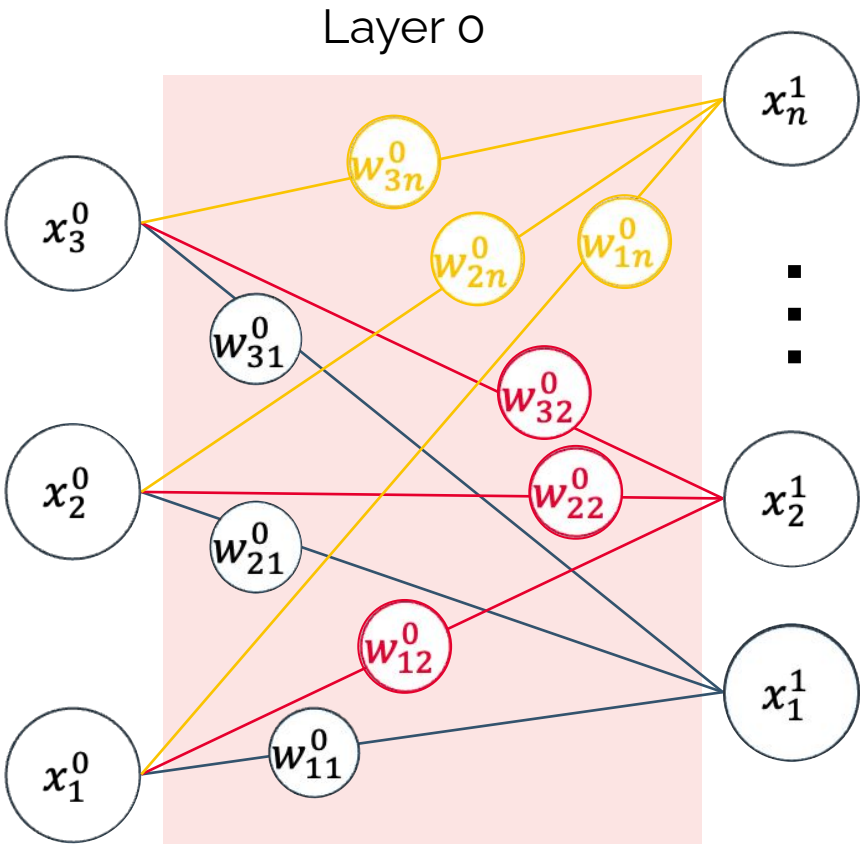
f(acidity, alcohol, sugar) = w_1 *acidity - w_2*alcohol + w_3*sugar + w_0 = y = Prob(grade)

NN layer

f(acidity, alcohol, sugar) = w_1 *acidity - w_2*alcohol + w_3*sugar + w_0 = y = Prob(grade)



Multiple layers

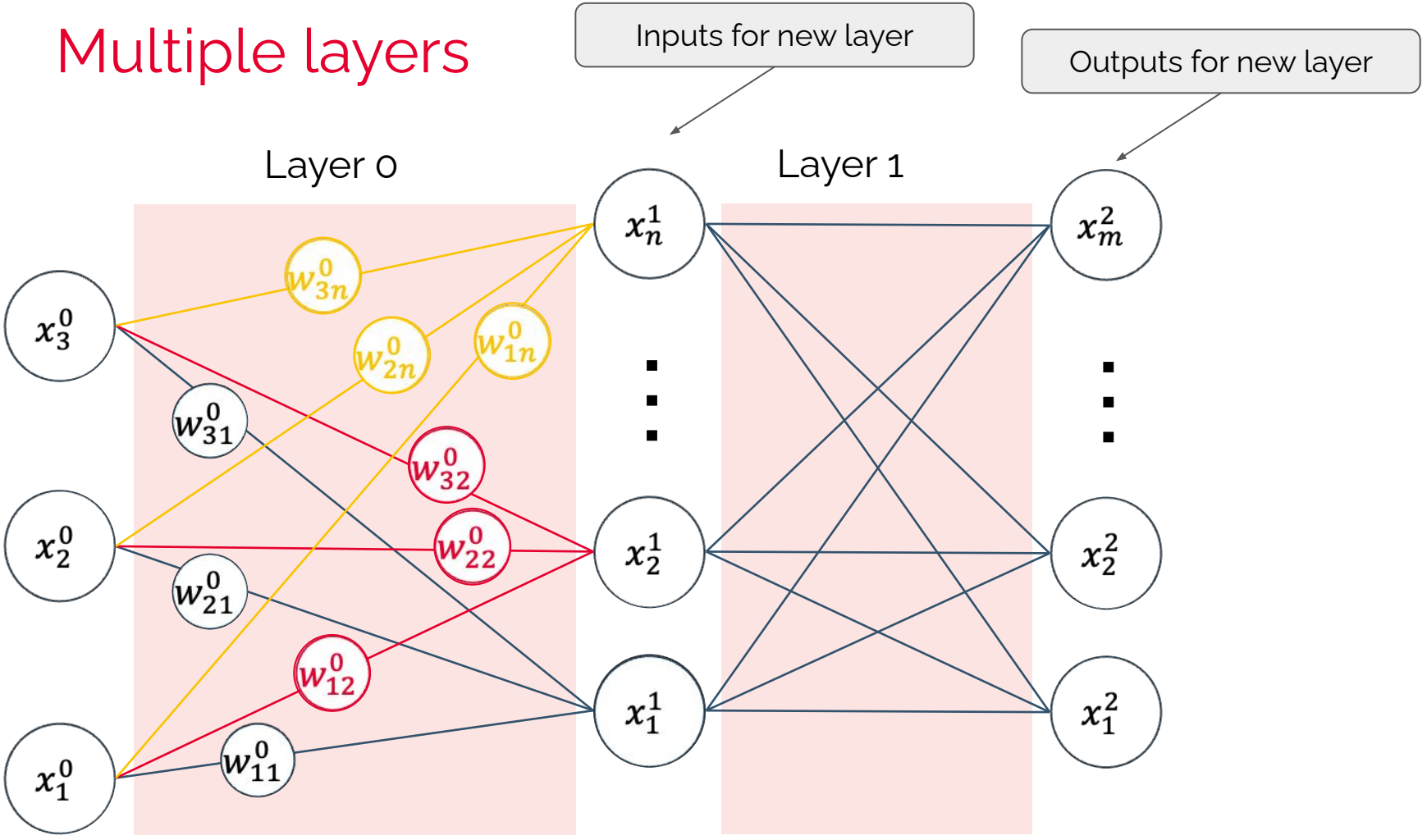


Instead of directly outputting a value, we output a series of them.

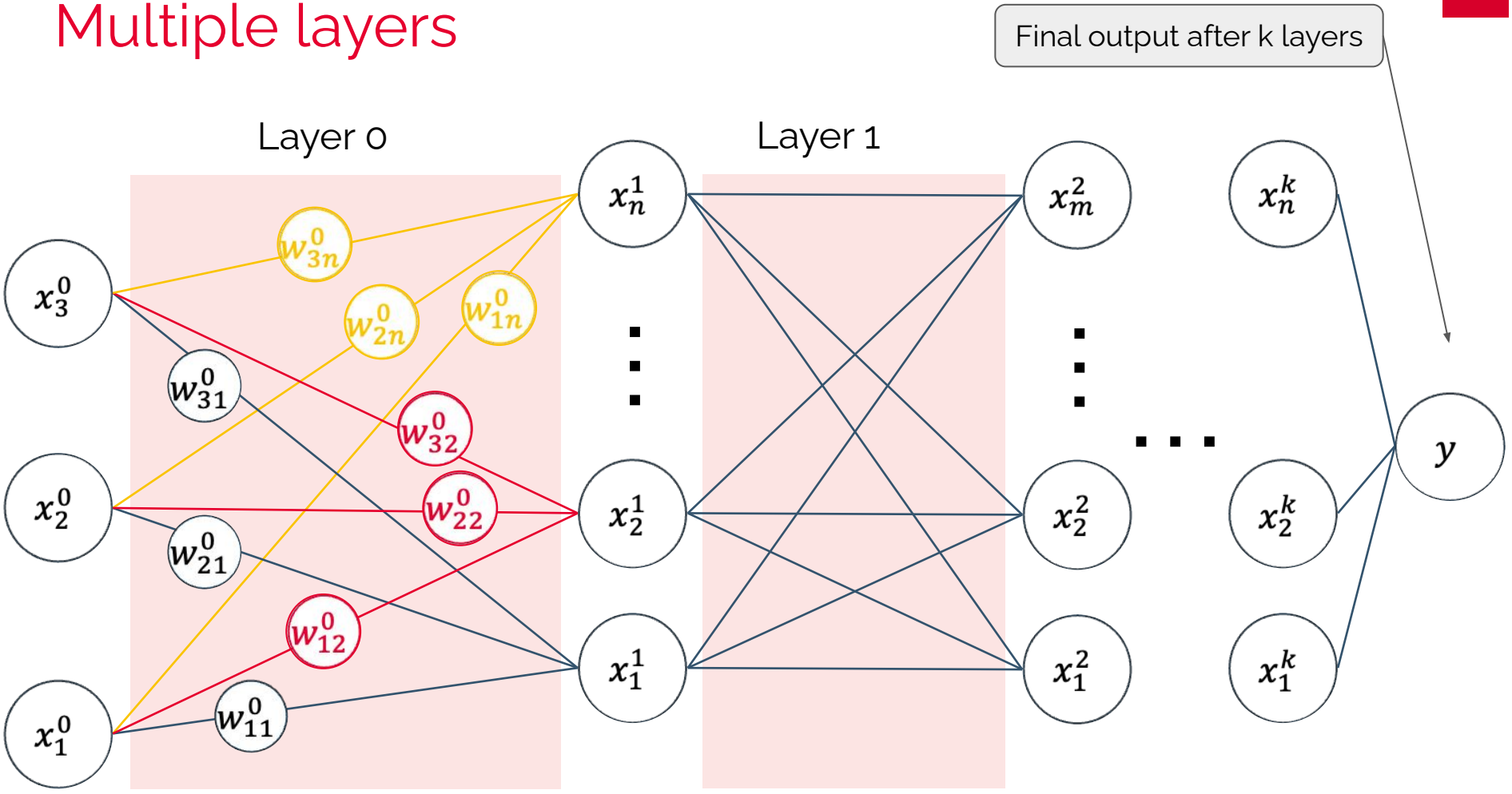
We don't use these outputs directly, they are "hidden"

We'll use them as new inputs for the next layer!

Multiple layers



Multiple layers



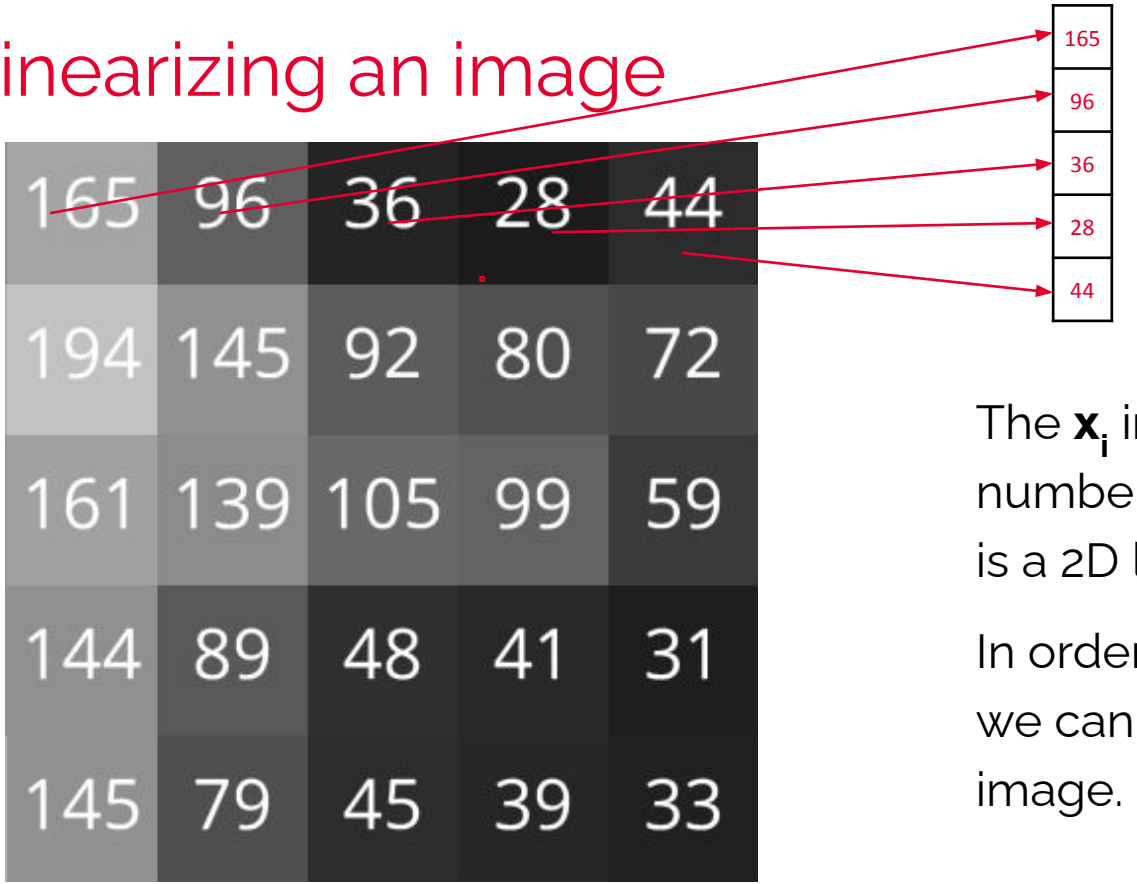
DSL

What are our features in the case of images?

Pixels!

MNIST example: classify 10 numbers

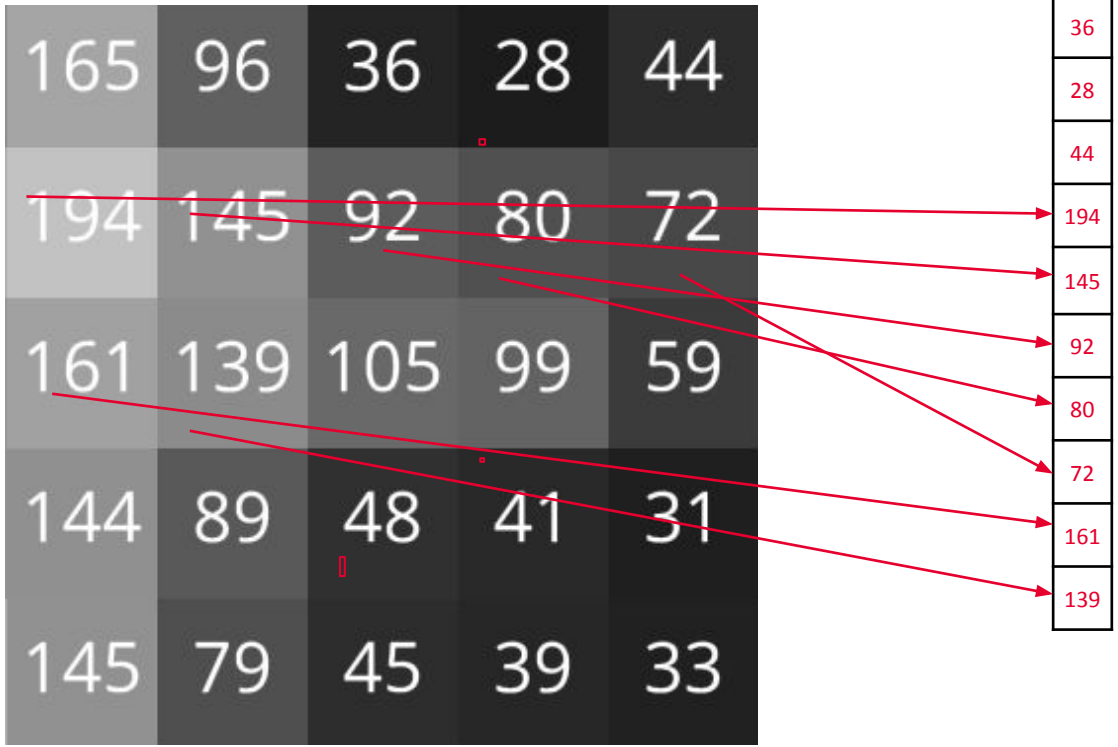
Linearizing an image



The $\mathbf{x}_i$ inputs are just a list of numbers (unlike an image which is a 2D list of numbers).

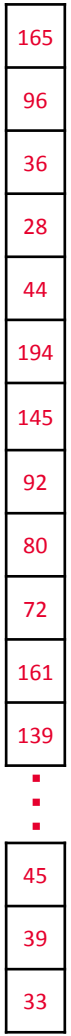
In order to pass an image as input, we can "flatten" or "unwrap" the image.

Linearizing an image



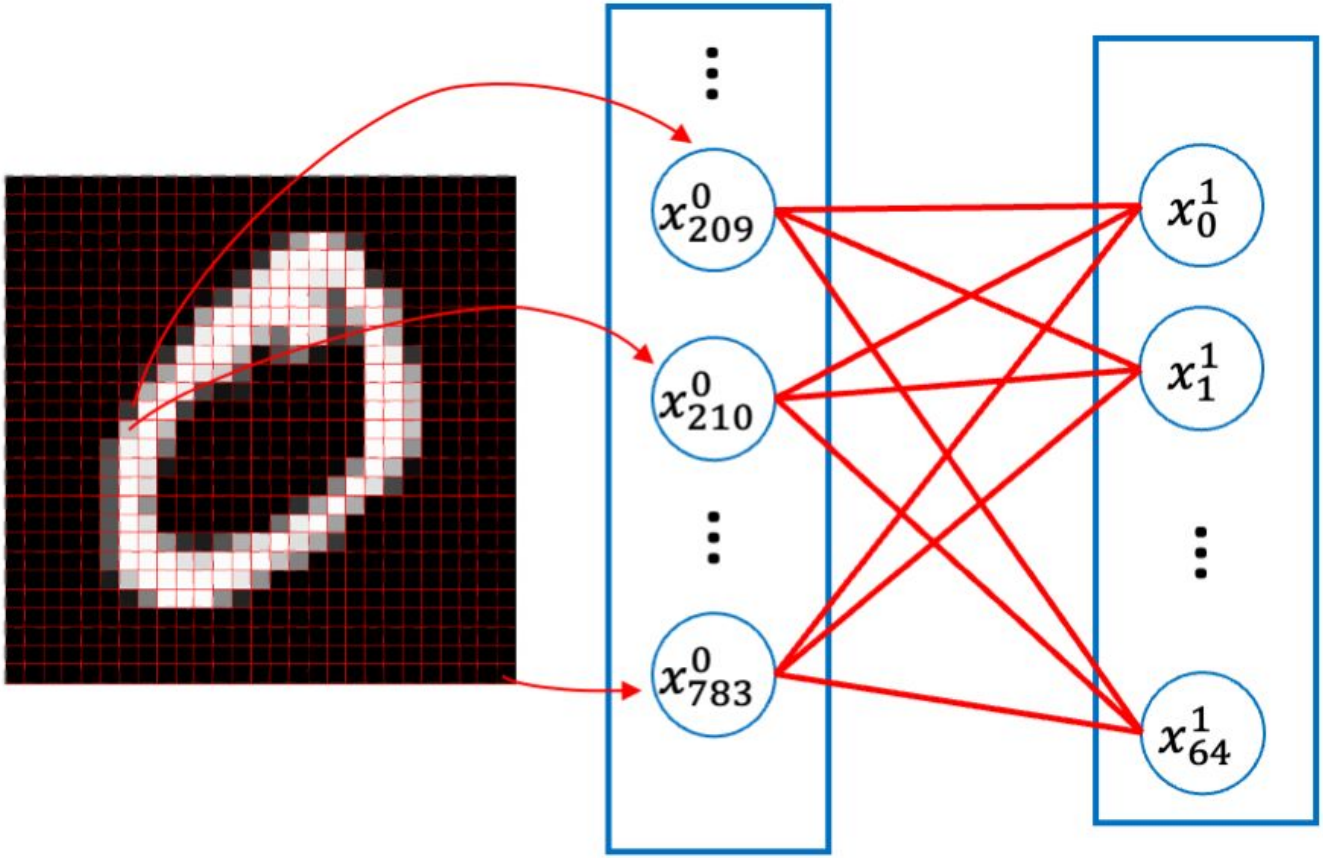
We append all pixel rows to a long 1D list

Linearizing an image

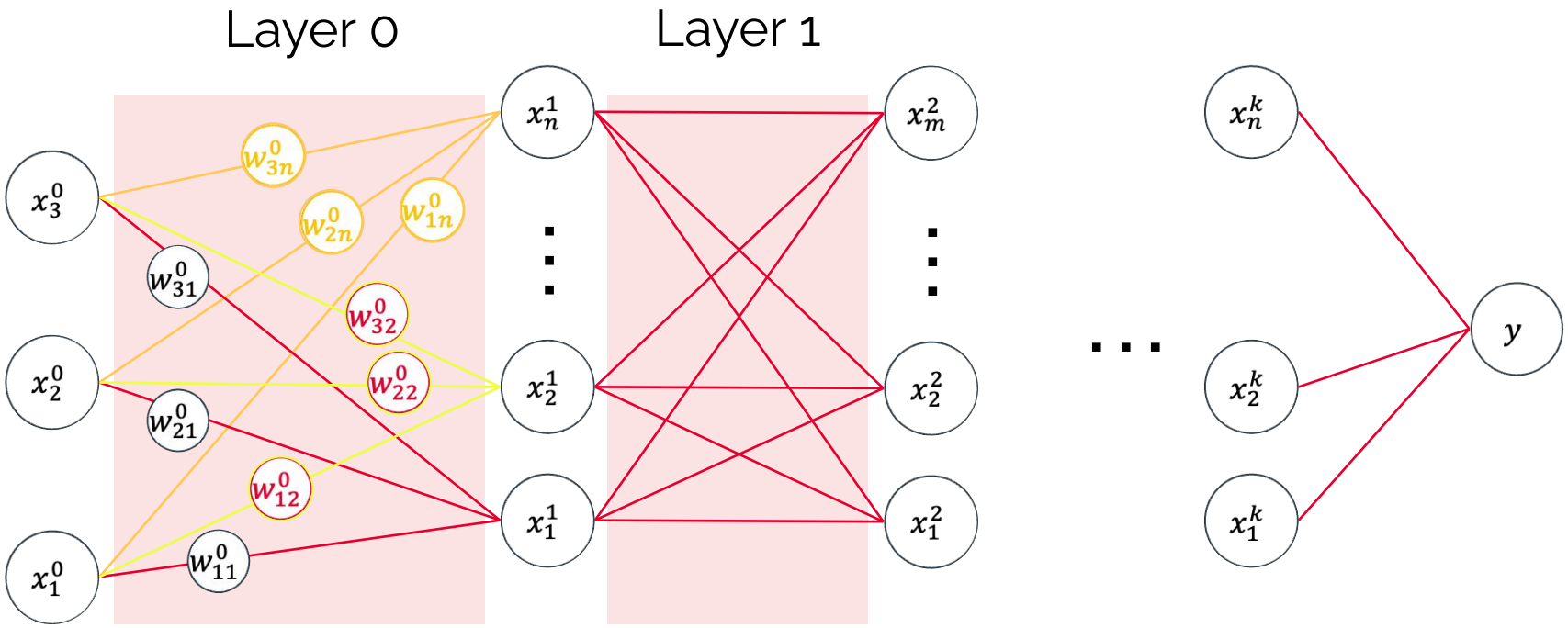


We have now one long list of numbers which are our input features!

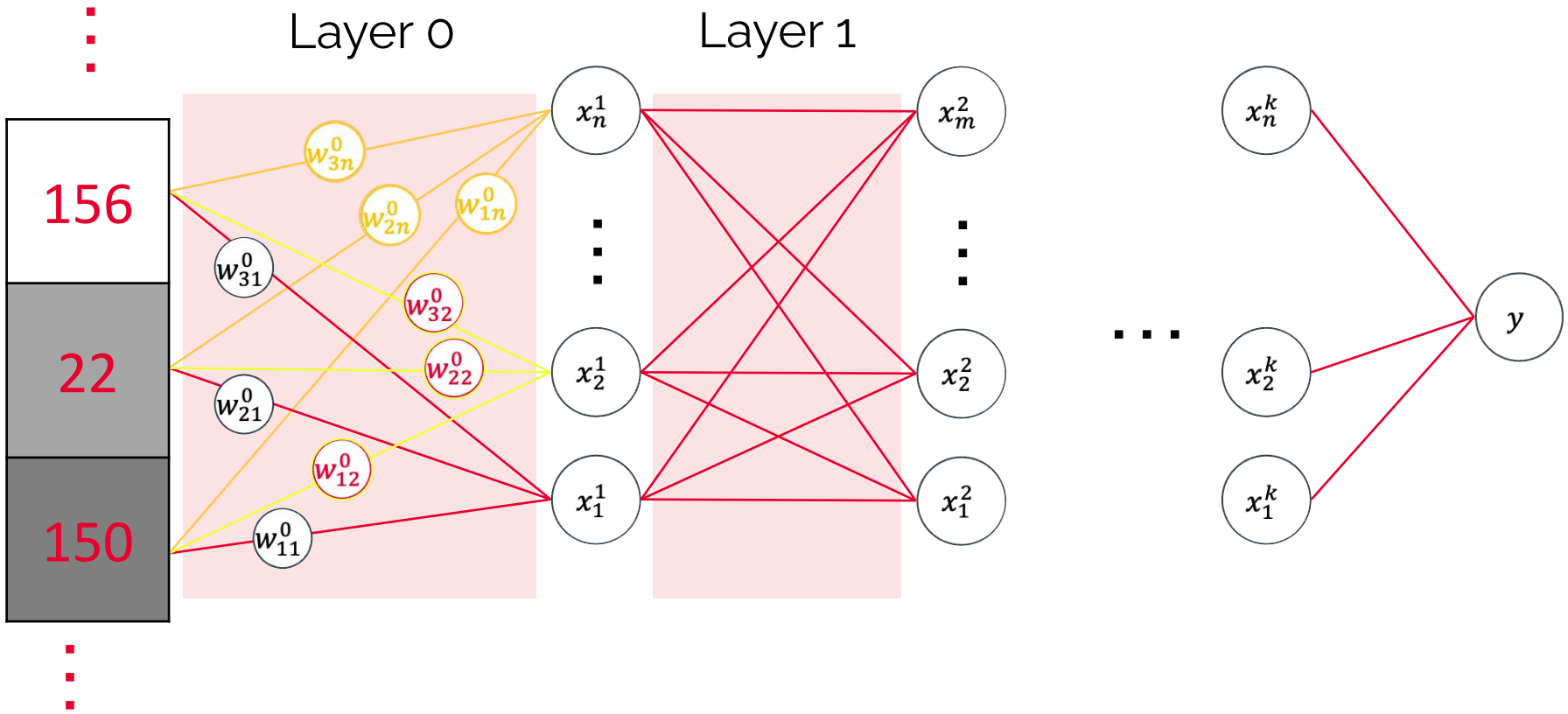
On a real MNIST image



Pixels as input

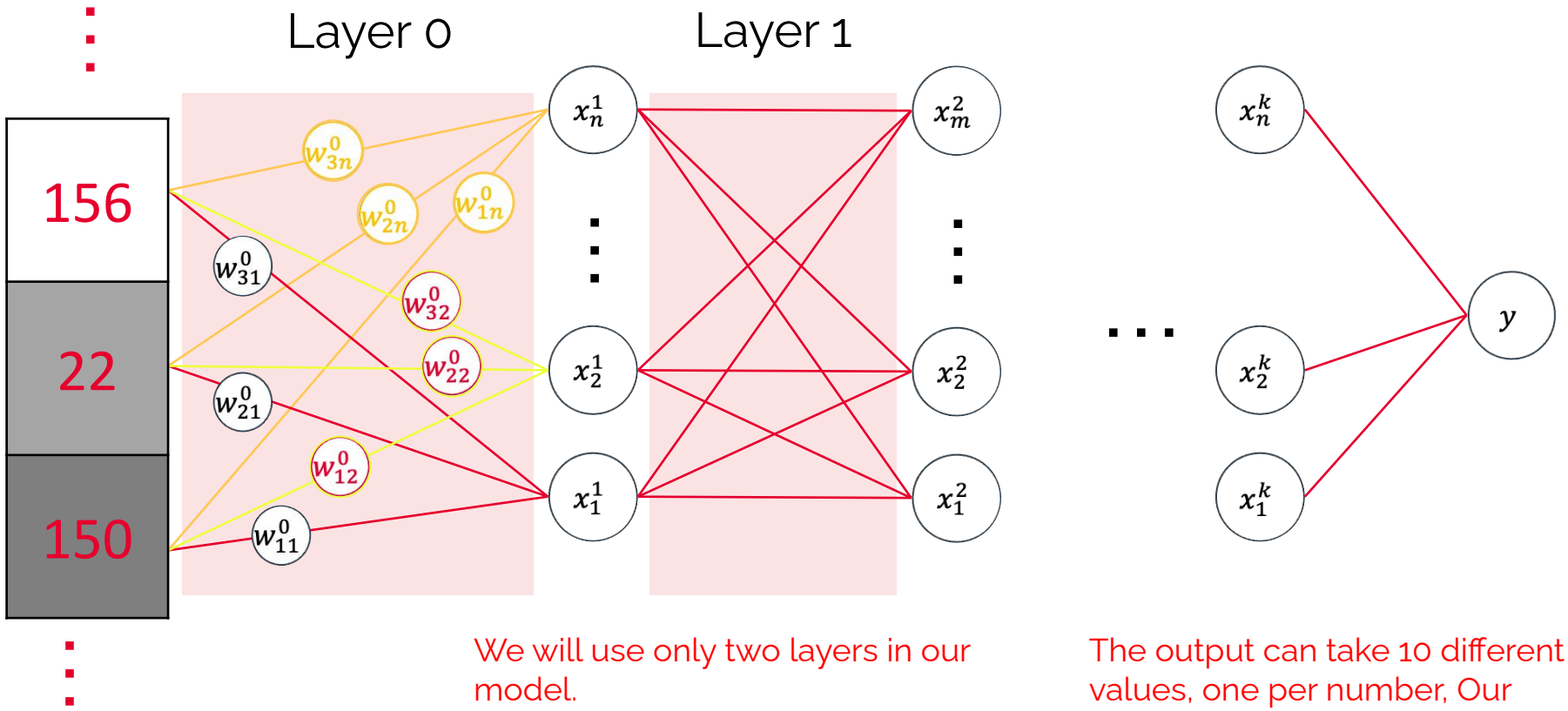


Pixels as input



The list of pixels is now our input to the network.

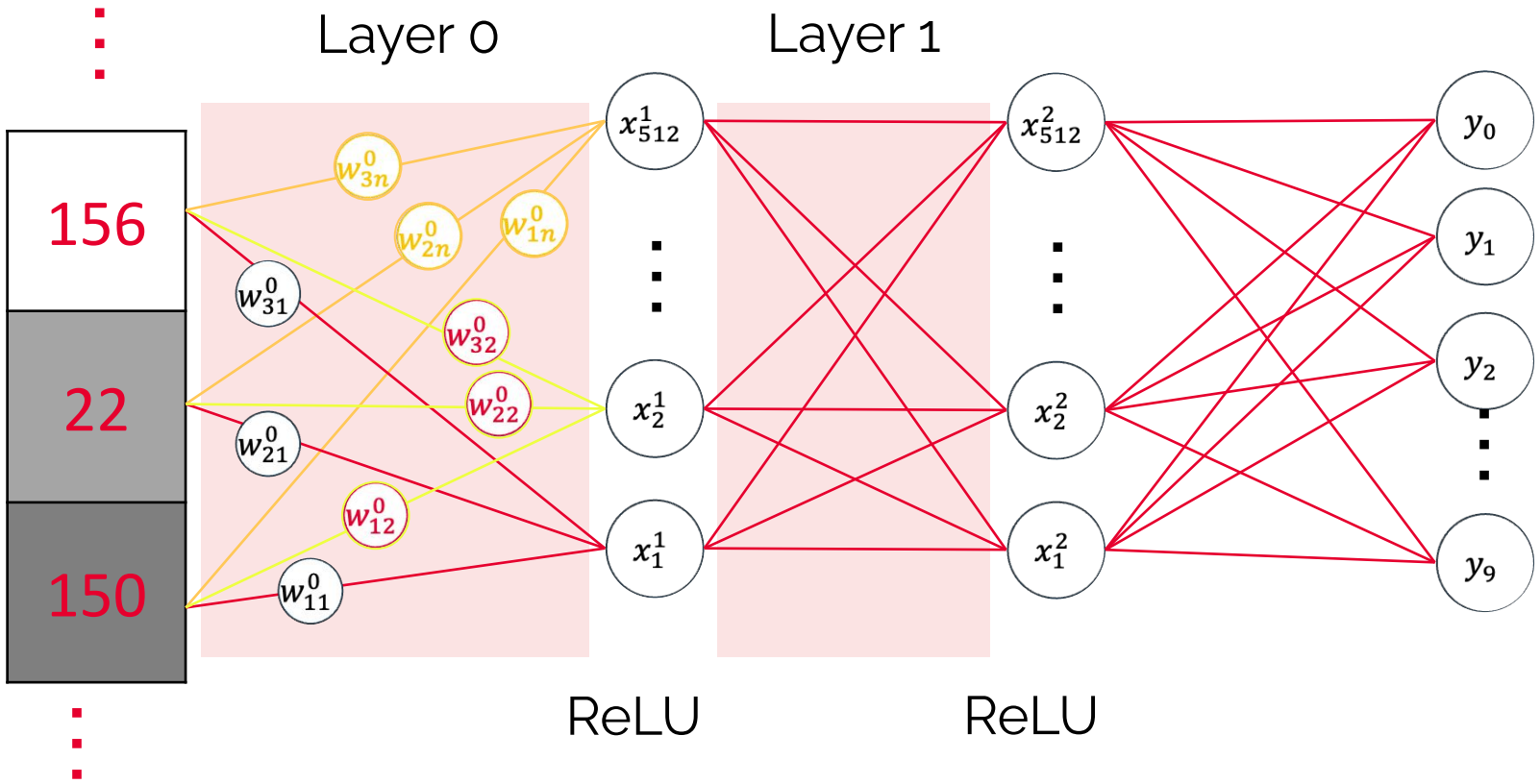
Pixels as input



We will use only two layers in our model.

The output can take 10 different values, one per number, Our output needs 9 units.

Categories as output

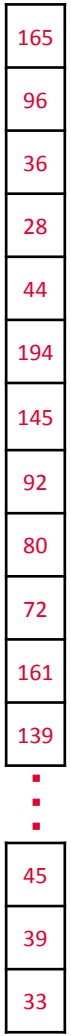


Don't forget that we need non-linearities in our system

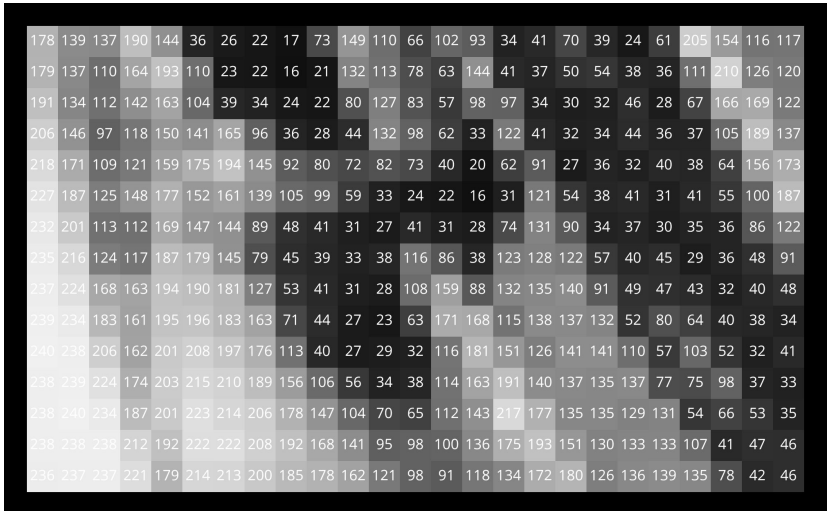
Taking into account local pixel structure: convolutional networks

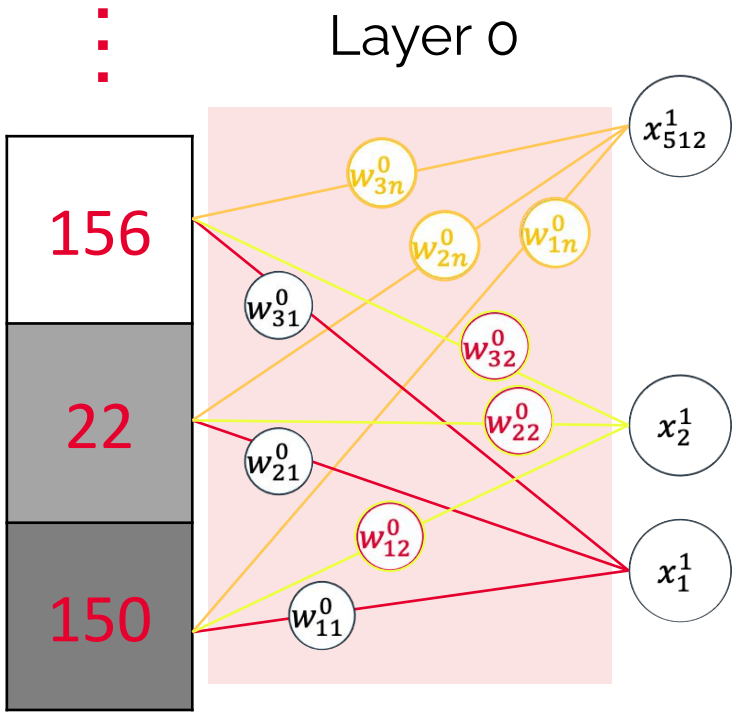
Previous simplistic approach

165	96	36	28	44
194	145	92	80	72
161	139	105	99	59
144	89	48	41	31
145	79	45	39	33



Images have local structure



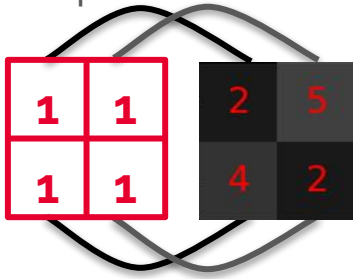


When linearizing an image, we connect **all pixels with all pixels.** But images have a local structure. We need to capture this local information.

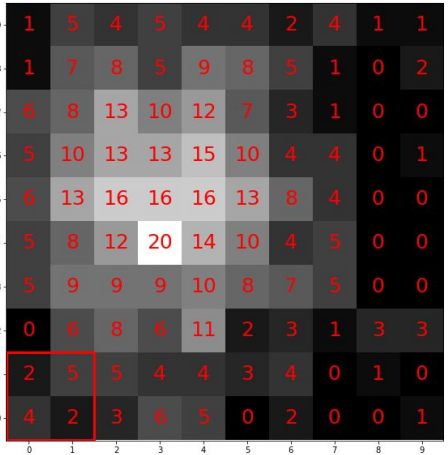
Convolutions

In a convolution, a small image (a filter) travels across an image and performs a local multiplication and sum operation.

Filter

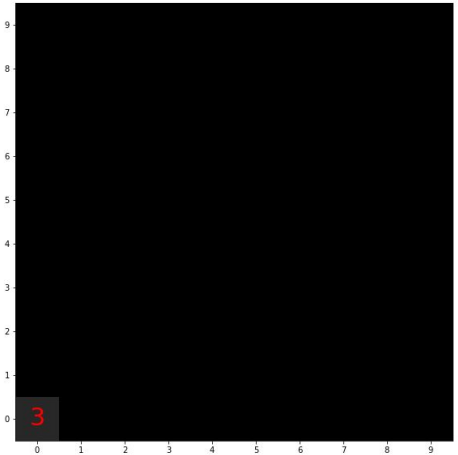


(1*2+1*5+1*4+1*5)/4 = 13/4 = 3.25



Image

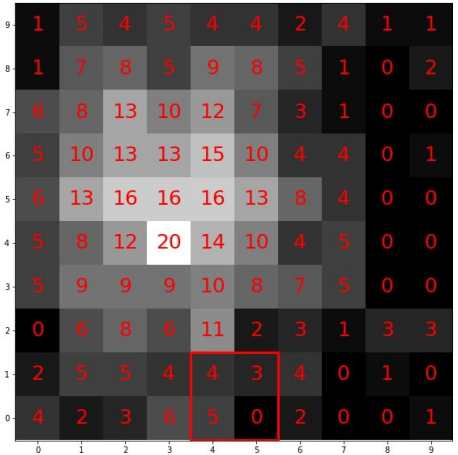
Region on which filter currently operates



Filtered Image

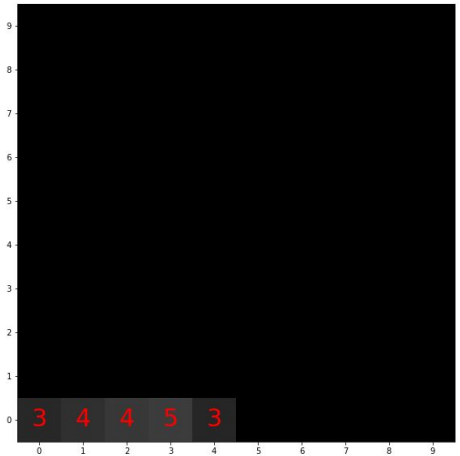
Convolutions

In a convolution, a small image (a filter) travels across an image and performs a local multiplication and sum operation.



Image

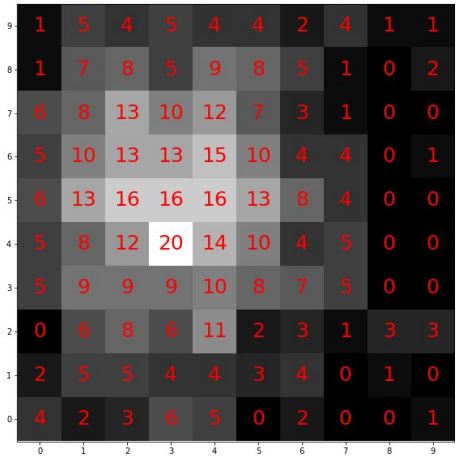
Region on which filter currently operates



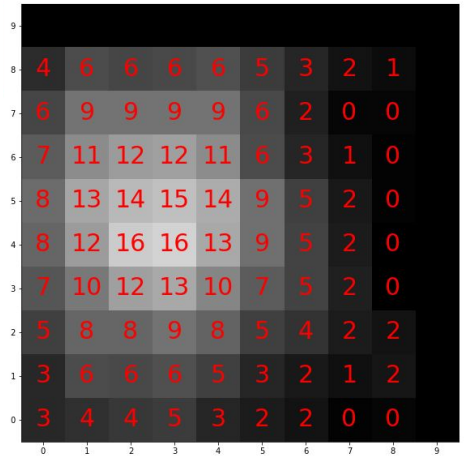
Filtered Image

Convolutions

In a convolution, a small image (a filter) travels across an image and performs a local multiplication and sum operation.



Image



Filtered Image

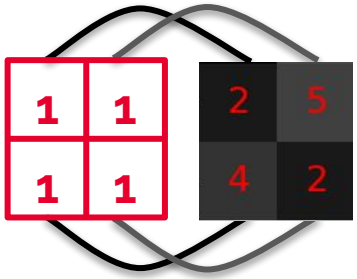
Convolutions

The filter doing the convolution does not have to be 2x2 and does generally not contain 1's. The filter can have:

any size

1	1	1
1	1	1
1	1	1

Filter



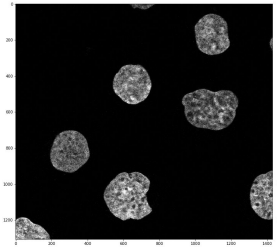
any structure

12	1	7
6	9	1
3	5	8

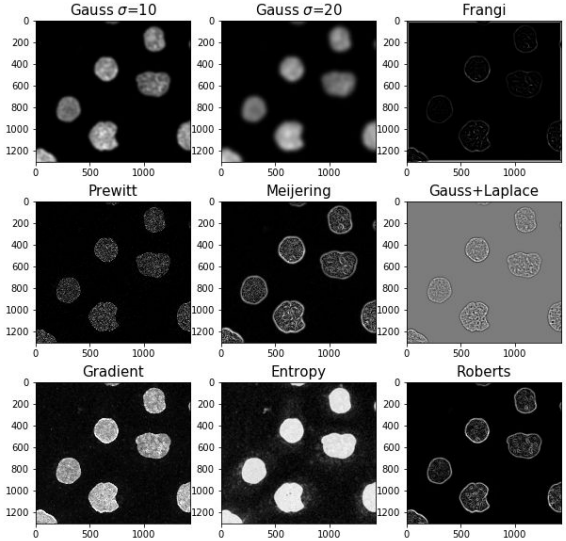
Convolutions / Filters

- Depending on the filter, the filtered image or response will highlight different structures.
- In classical computer vision the filters are designed by a specialist
- In deep learning the filters are **learned**. The weights are the filters!

w_{00}	w_{01}	w_{02}
w_{10}	w_{11}	w_{12}
w_{20}	w_{21}	w_{22}

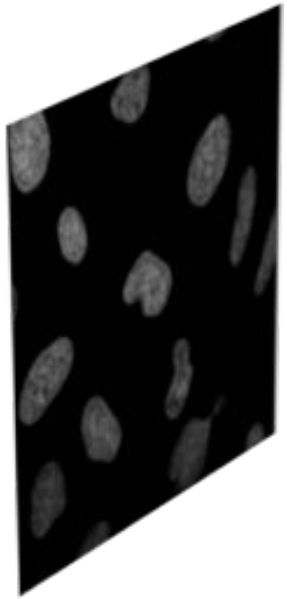


Original image



Various filters applied to the image

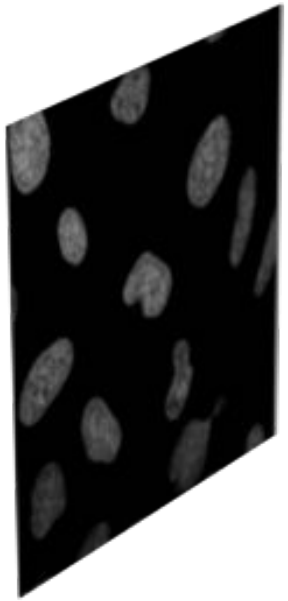
Typical Convolutional network



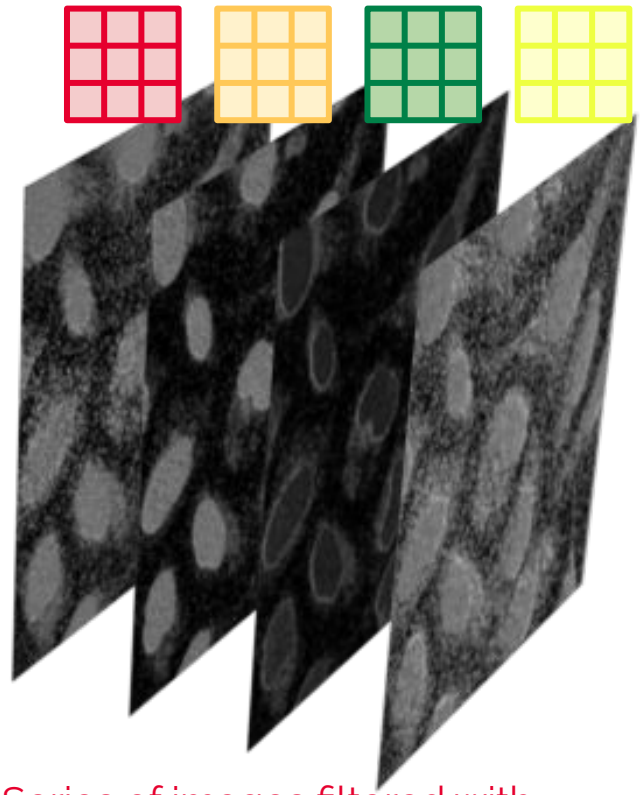
Input image

- The network takes an image as input

Typical Convolutional network



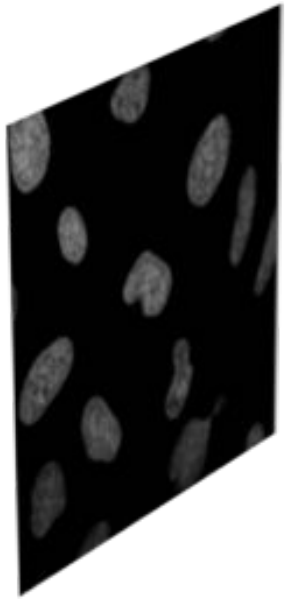
Input image



Series of images filtered with a series of trained filters

- The network takes an image as input
- It gets filtered multiple times, to reveal different structures

Typical Convolutional network

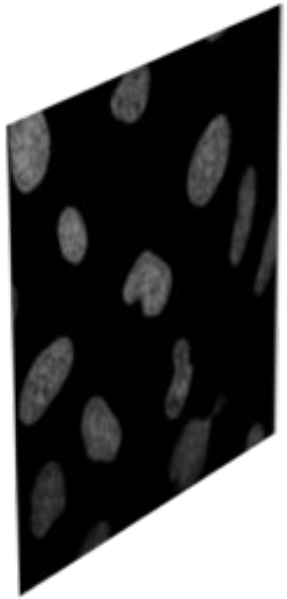


Input image

All images in a filtering round are aggregated

- The network takes an image as input
- It gets filtered multiple times, to reveal different structures
- Images are aggregated into a layer

Typical Convolutional network

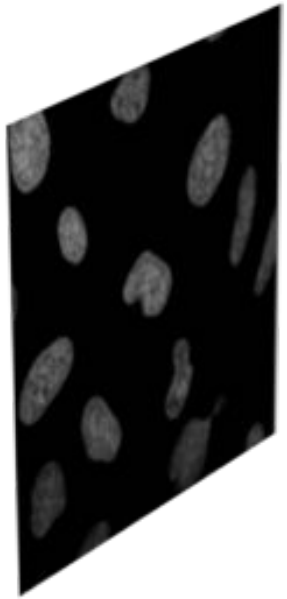


Input image

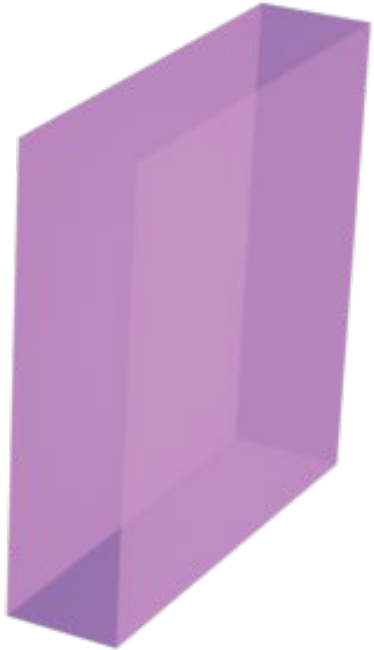
All images in a filtering round are aggregated

- The network takes an image as input
- It gets filtered multiple times, to reveal different structures
- Images are aggregated into a layer

Typical Convolutional network



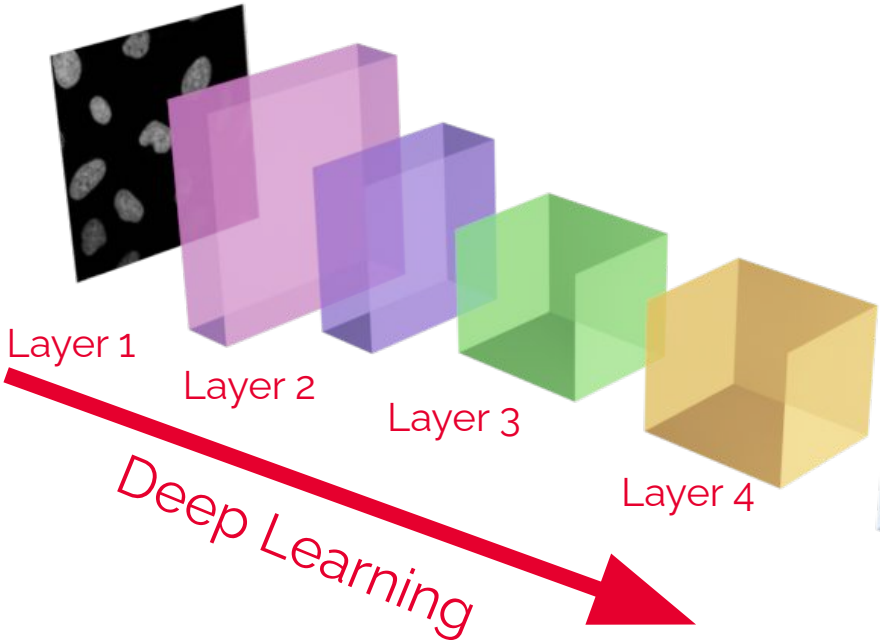
Input image



All images in a filtering round are aggregated

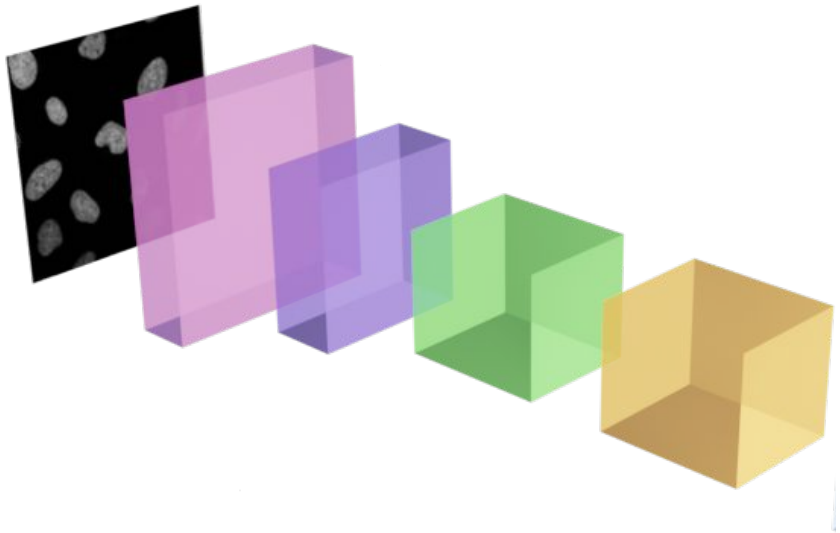
- The network takes an image as input
- It gets filtered multiple times, to reveal different structures
- Images are aggregated into a layer

Typical Convolutional network



- The network takes an image as input
- It gets filtered multiple times, to reveal different structures
- Images are aggregated into a layer
- All filtered images of one layer are the input of the next layer, which produces a new set of filtered images

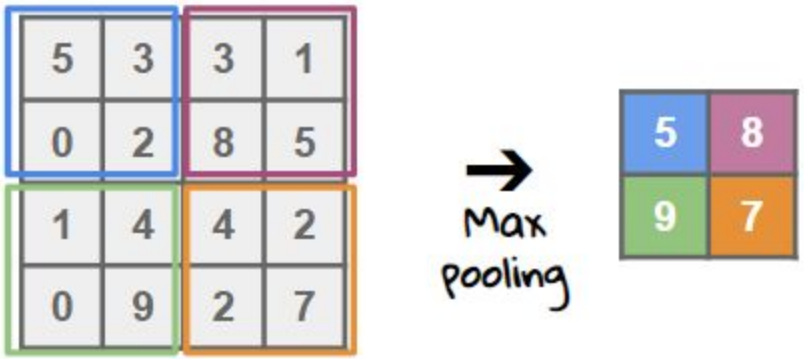
Other layers

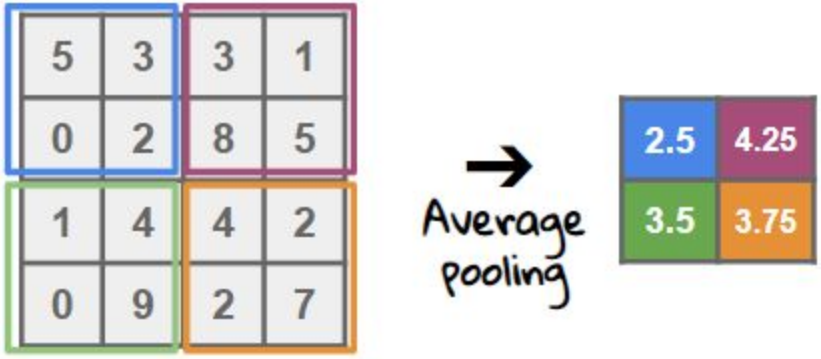


- As seen in this network, the boxes become smaller, i.e. the filtered images become smaller
- Other types of layers are responsible for this effect (and other effects

Downsampling an image

As an example, if we want to reduce the size of the image, we can summarize sets of four pixels into one by finding the maximum or mean over these pixels. This cuts the image size in half.

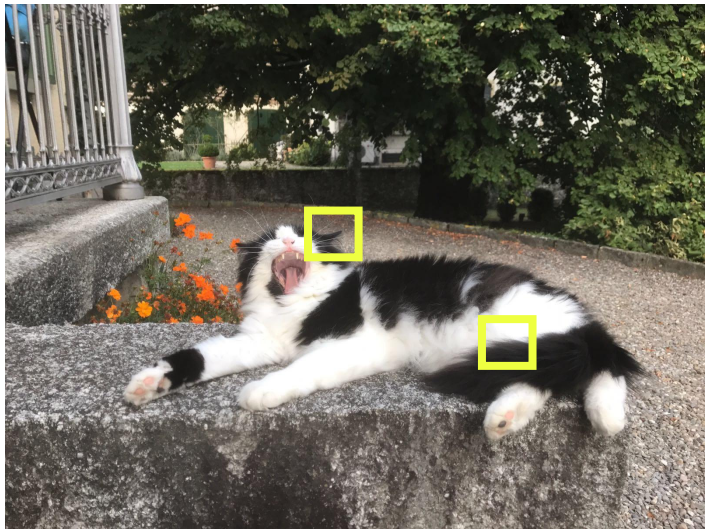




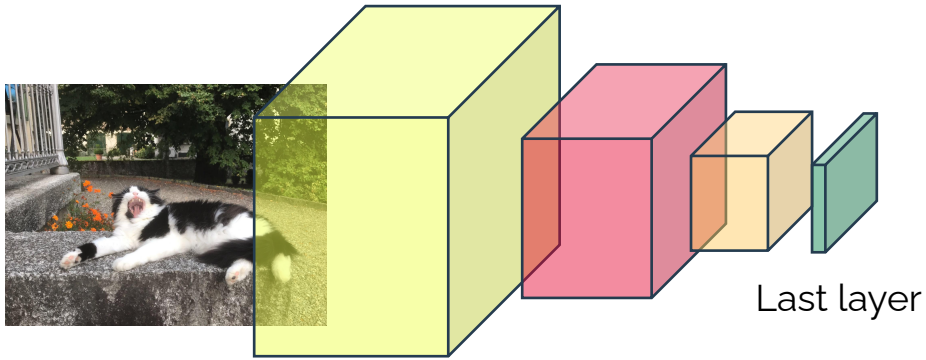
https://towardsdatascience.com/the-most-intuitive-and-easi
est-guide-for-convolutional-neural-network-3607be47480

Why do we need to change the image size?

- By looking at small tiles, we detect local information and structure. E.g. here there is fur or an ear
- This information could give a dog, cat, rabbit etc.
- We need to aggregate the information from each tile
- To perform this aggregation, we analyze each tile and then summarize it by downscaling

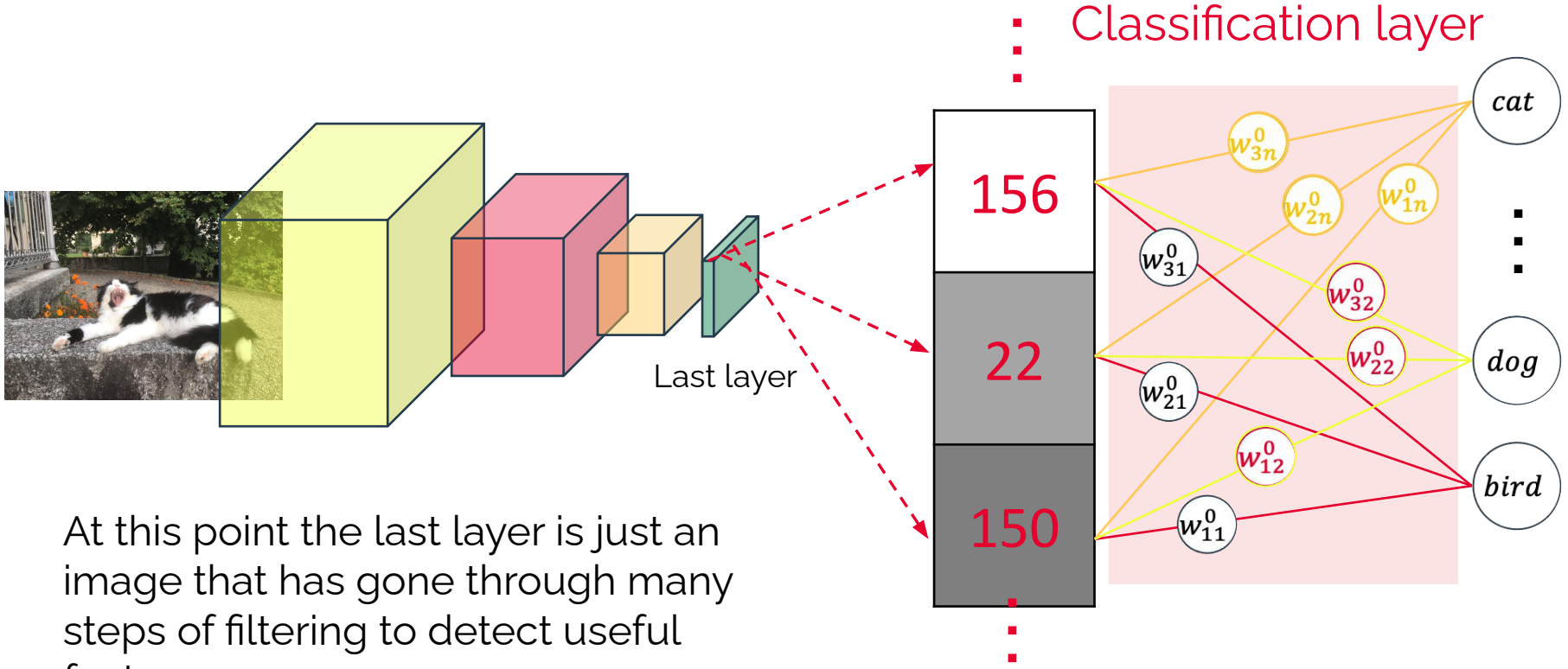


How do we get to classification ?



At this point the last layer is just an image that has gone through many steps of filtering to detect useful features.

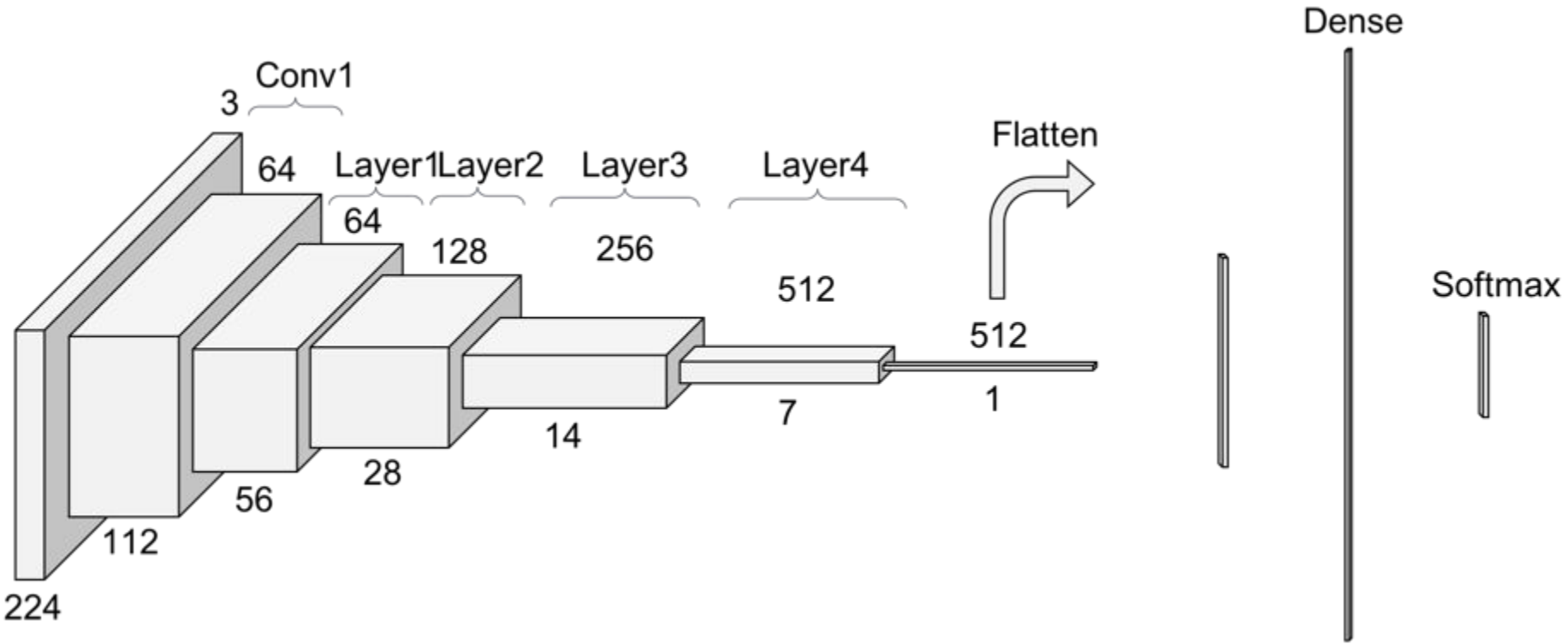
How do we get to classification ?



At this point the last layer is just an image that has gone through many steps of filtering to detect useful features.

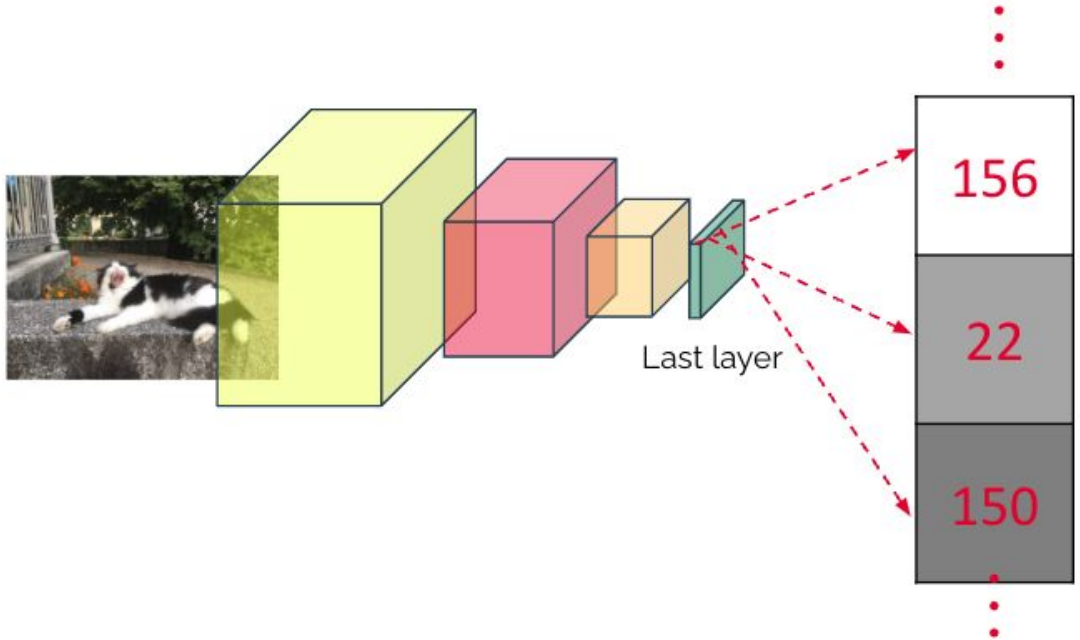
We can unwrap the output image and use it in **a fully connected** or **dense** layer for classification!

Now you should have an idea what this means!



Model outputs

In the last example we wanted a classifier: the output was a number or a list corresponding to probabilities of classification classes. Depending on the taks we may want to have other output shapes.

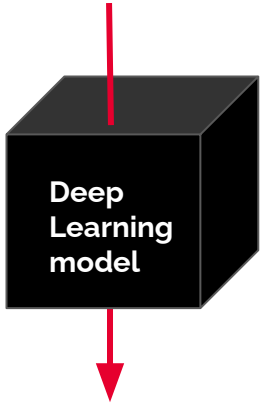


Model outputs

The output of a model is always numbers.

Depending on model architecture you can get:

- a single number (binary classification)



cat / not cat ?

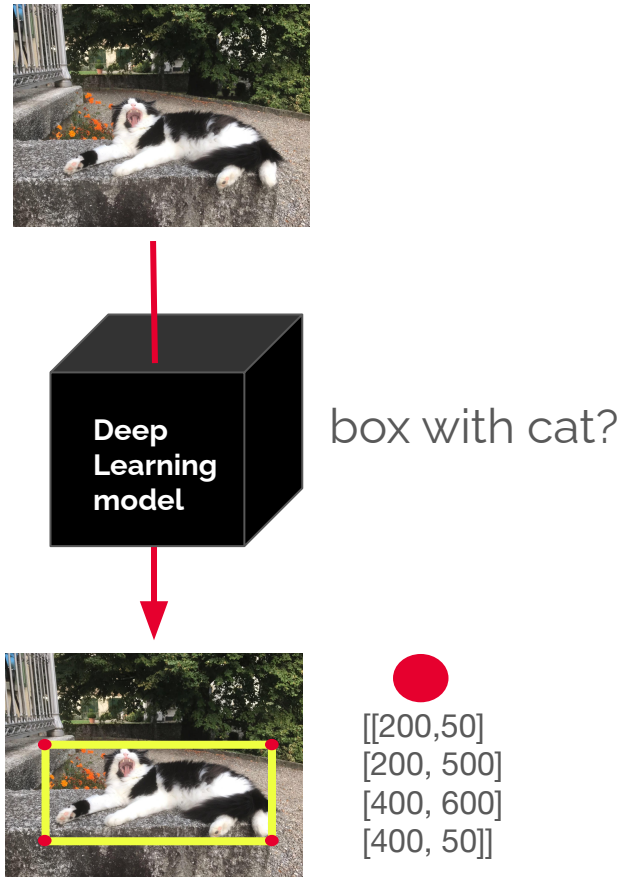
[0.9]

Model outputs

The output of a model is always numbers.

Depending on model architecture you can get:

- a single number (binary classification)
- a series of numbers (coordinates of objects

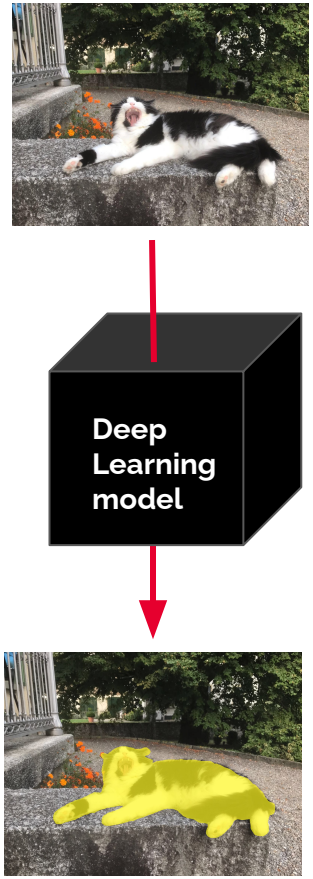


Model outputs

The output of a model is always numbers.

Depending on model architecture you can get:

- a single number (binary classification)
- a series of numbers (coordinates of objects
- a 2D image (a mask with detections)

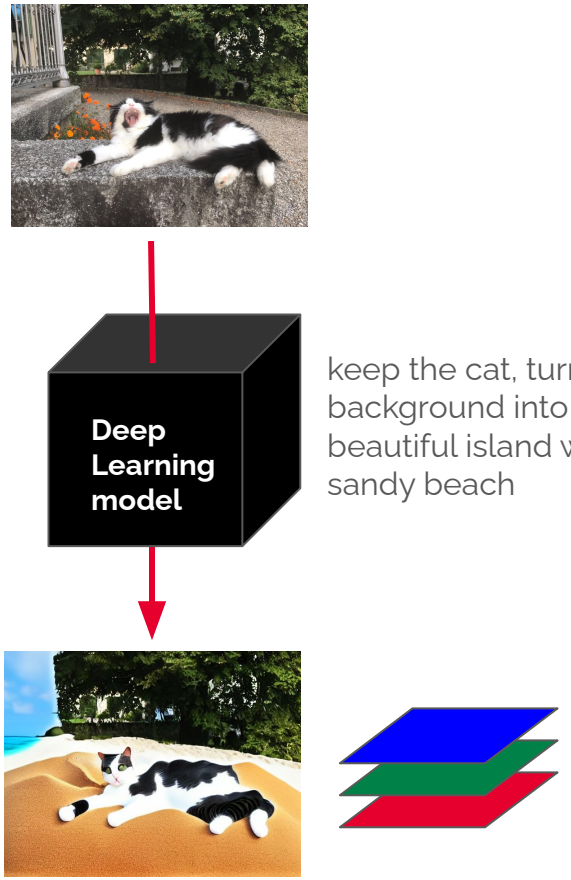


Model outputs

The output of a model is always numbers.

Depending on model architecture you can get:

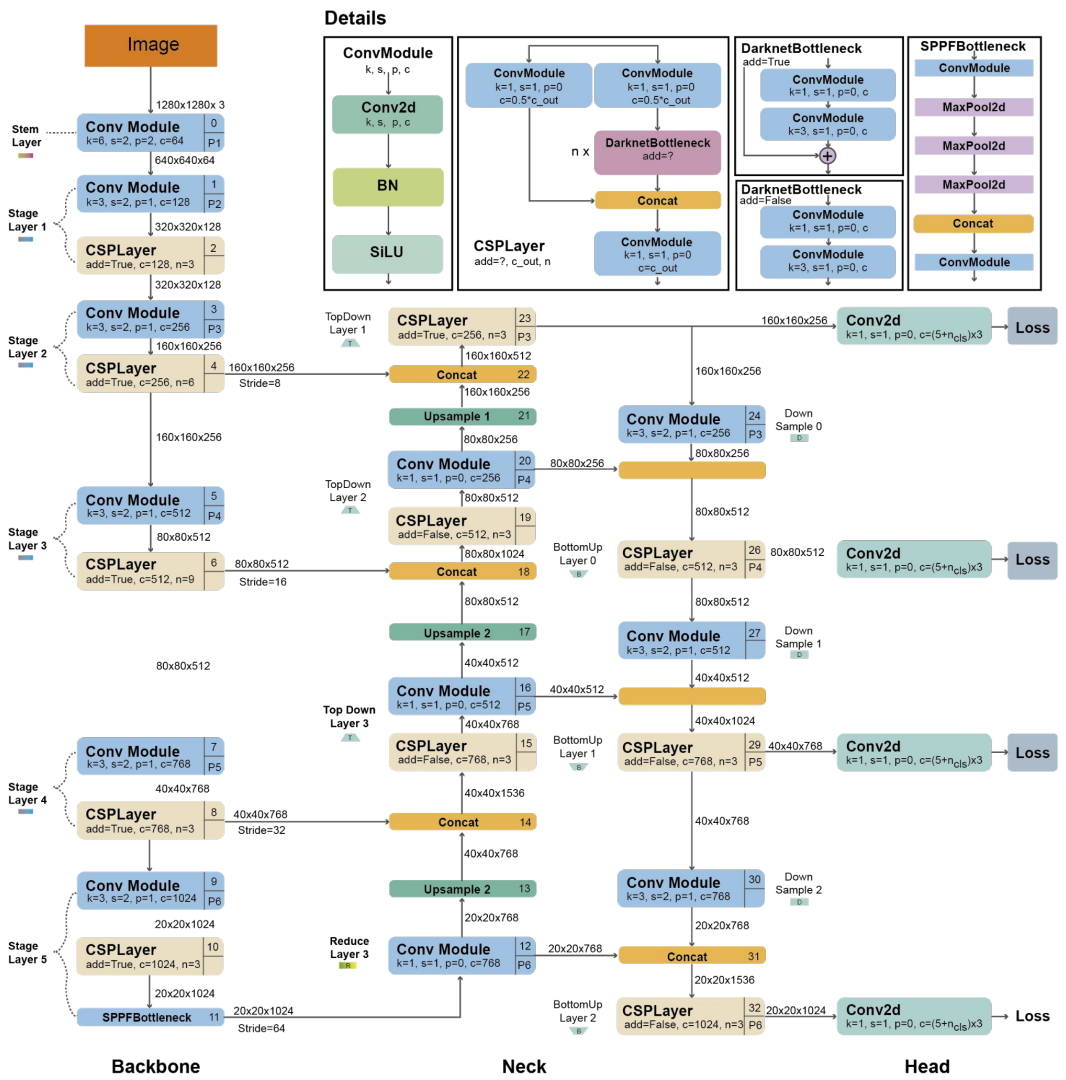
- a single number (binary classification)
- a series of numbers (coordinates of objects
- a 2D image (a mask with detections)
- an RGB image (e.g. generation)



Modern networks

Modern networks are highly complex assemblies of the type of building blocks described earlier.

We won't go through architectures, except for diffusion models



https://arxiv.org/html/2304.00501v6

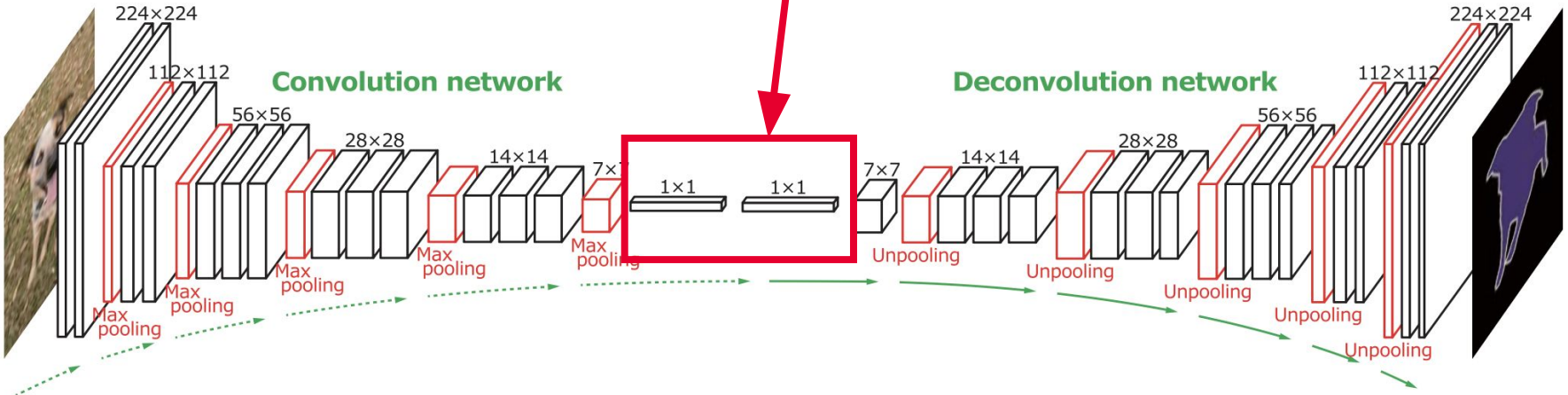
How does it look like in code?

https://github.com/guiwitz/CAS_AICP_M4/blob/main/04-Convolutions.ipynb

U-nets, auto-encoders and latent space

Encoding-decoding

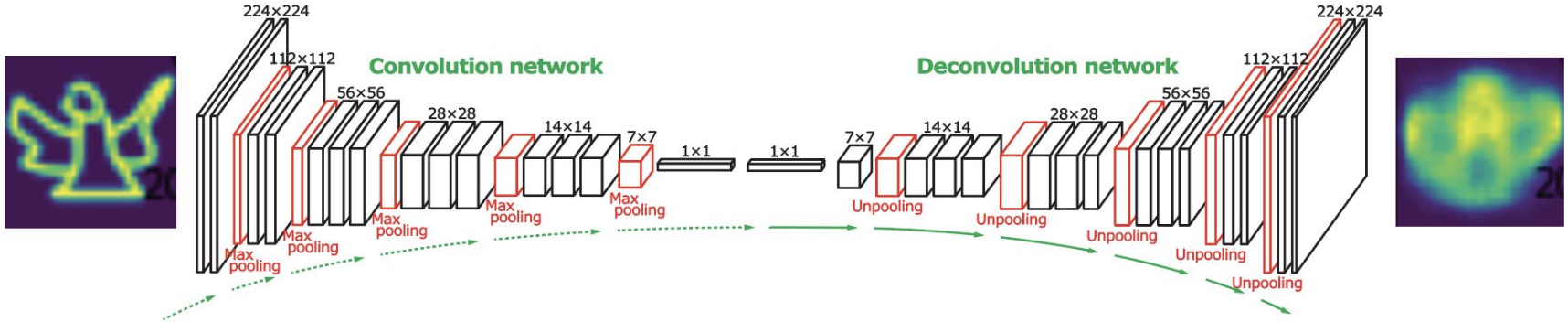
Here the complete image is simply encoded as a long list of numbers! This is the **latent space**



https://arxiv.org/pdf/1505.04366.pdf

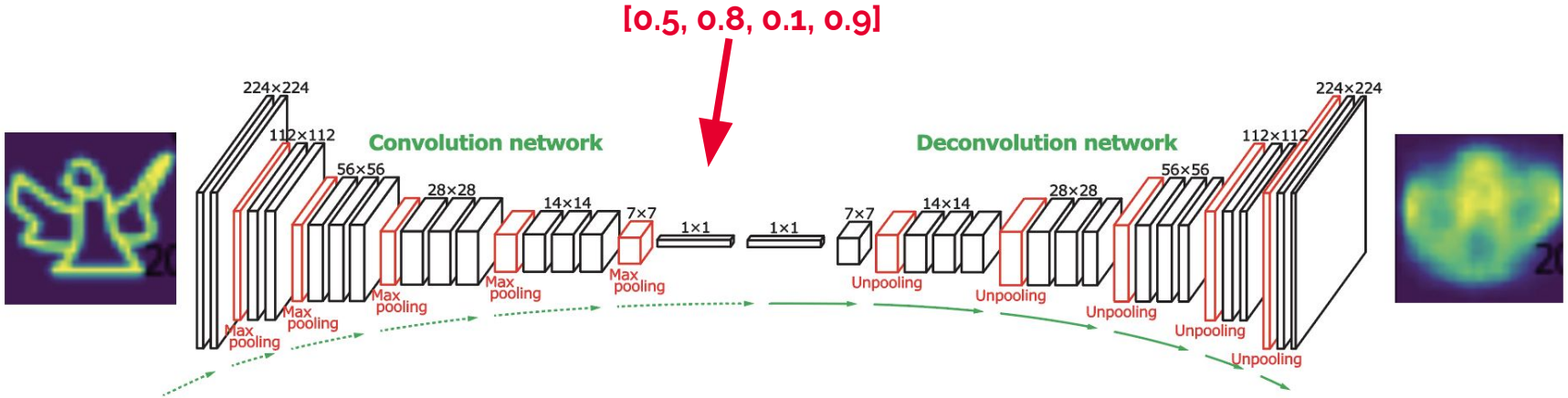
Auto-encoder: self-supervised learning

- Instead of attempting to give a segmentation, we can also teach the network to simply reconstruct the image
- No labels! The input and target are the same image



Auto-encoder: self-supervised learning

- Instead of attempting to give a segmentation, we can also teach the network to simply reconstruct the image
- No labels! The input and target are the same image
- In this example the latent space has a size of 4: we summarize 28x28 pixels with just 4 values!



Auto-encoder: image generation

- In principle we can come up with a set of 4 new numbers and just use the decoding part of the networks to create new images
- If our numbers are “close” to some from the training data, we’ll get a variation of the image. Additional care should be taken to make this work, but this is the basis of image generation!