Introduction to Robotics for cognitive science

Dr. Andrej Lúčny

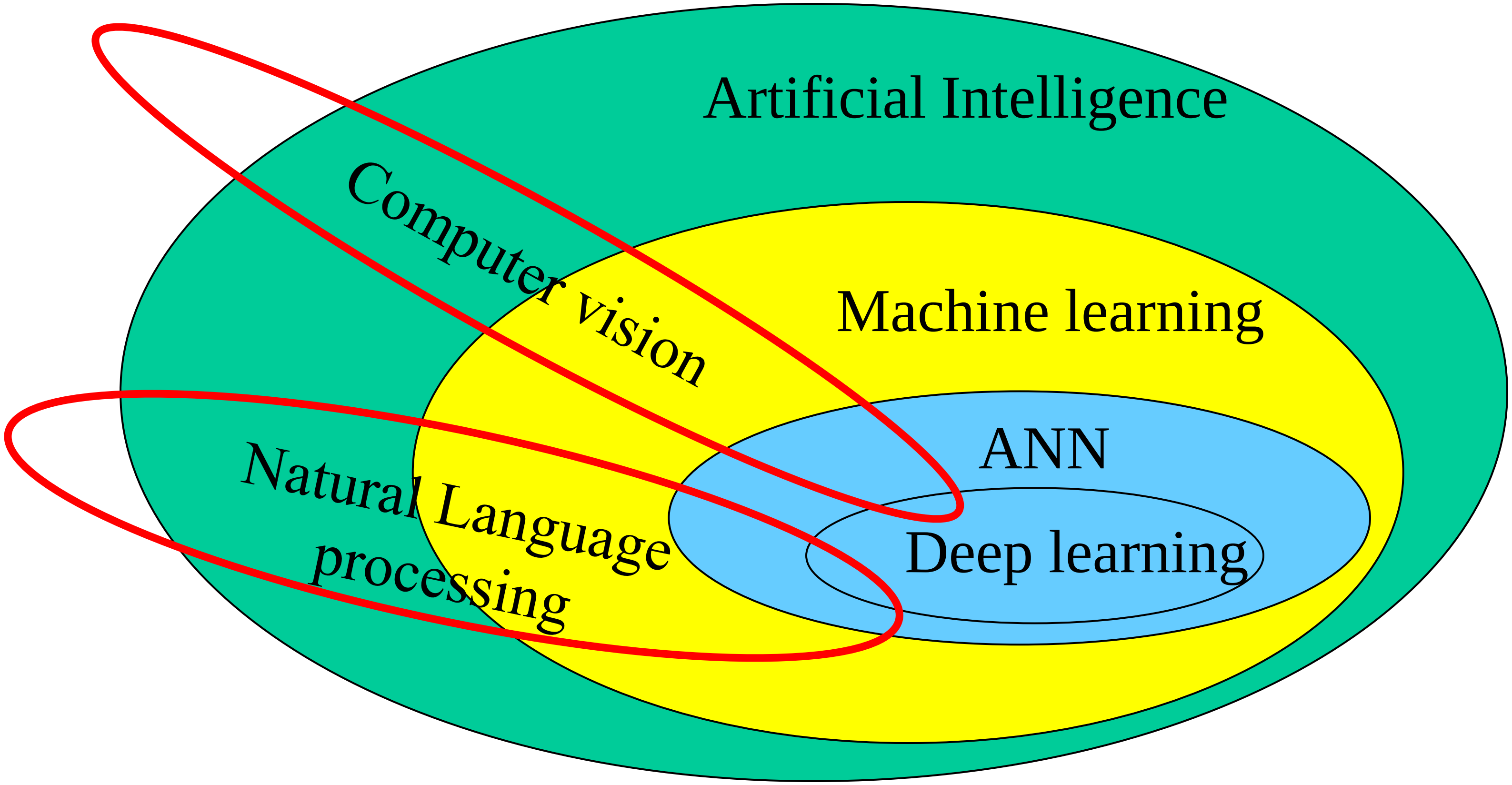
KAI FMFI UK

lucny@fmph.uniba.sk

Web page of the subject

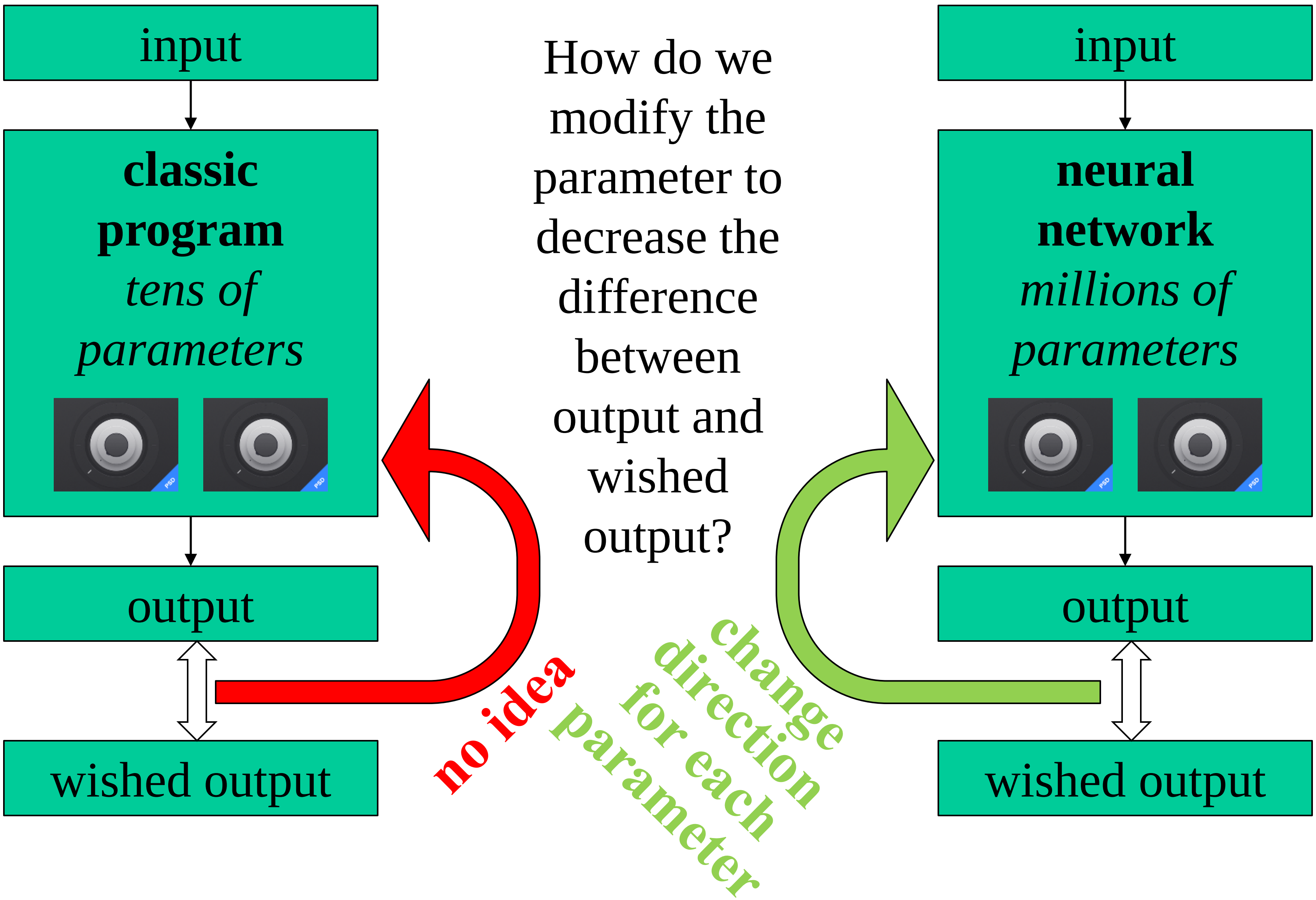
www.agentspace.org/kv

Deep Learning

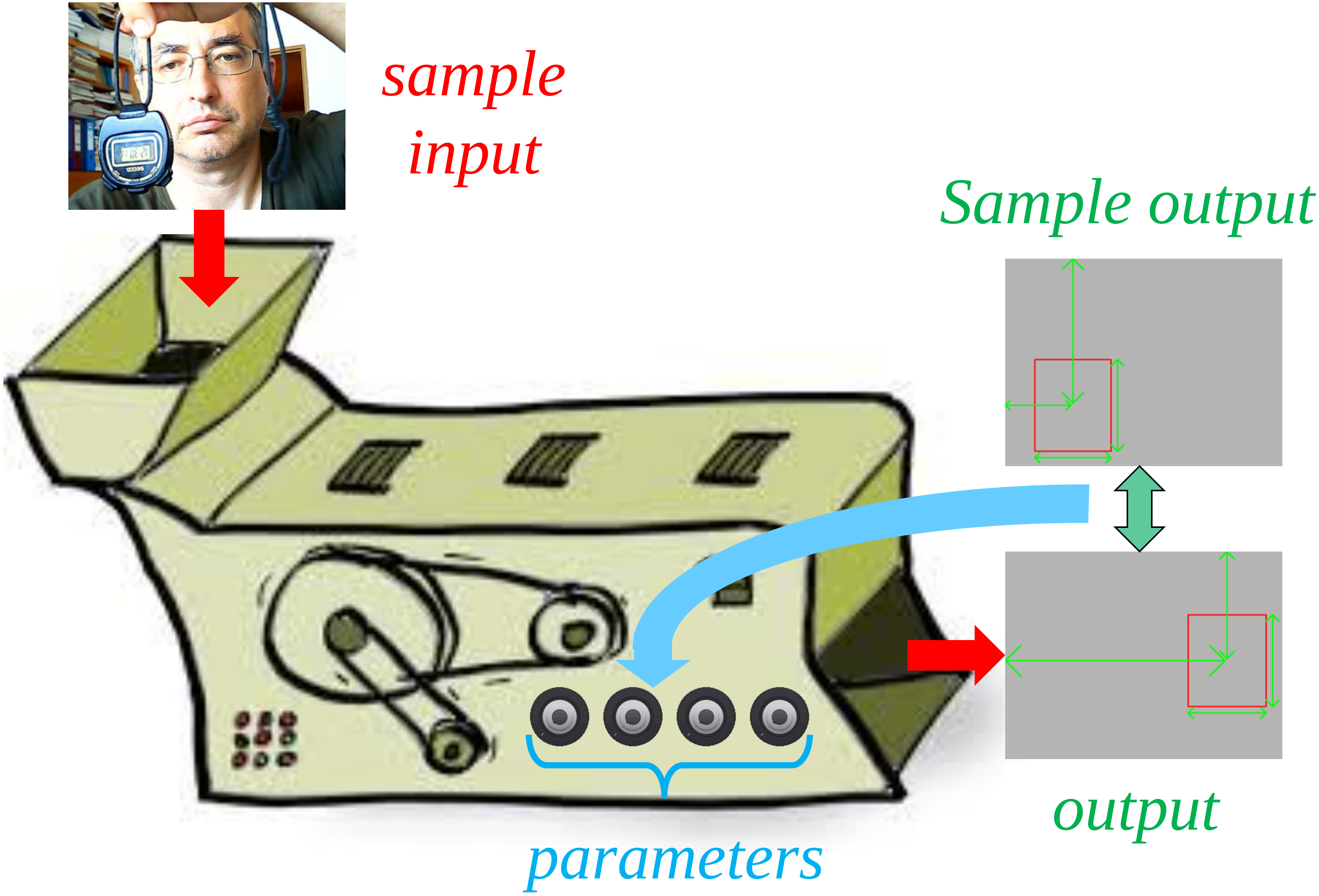


ANN … Artificial Neural Networks

What is neural network



Neural network



Building blocks of neural network

- anything corresponding to a function that we can express and derivate via the symbolic way

Then we can define a loss function, typically the sum of squares of differences between output and wished output for all samples, and calculate its partial derivatives by individual parameters

The value of the partial derivative for the current values of parameters gives the direction in which we need to modify the parameter to decrease the value of the loss function

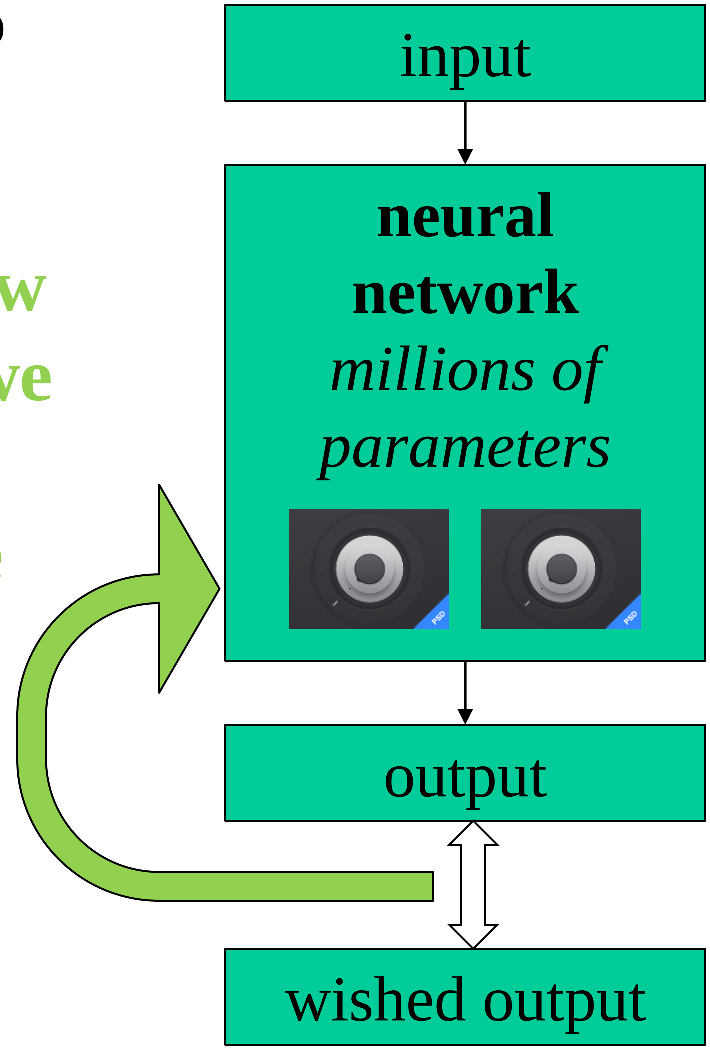
Training (Learning)

How do we modify parameters to decrease the difference between output and wished output?

for each parameter, we know the correct direction, even we know that one parameter needs to be modified more than the other

but we do not know how much

so, we need to guess and try, and return, which is the training or learning process



Training algorithms

According to how many samples we derive the gradient

- Gradient Descent
- Stochastic Gradient Descent
- Batch Gradient Descent
- Minibatch Gradient Descent

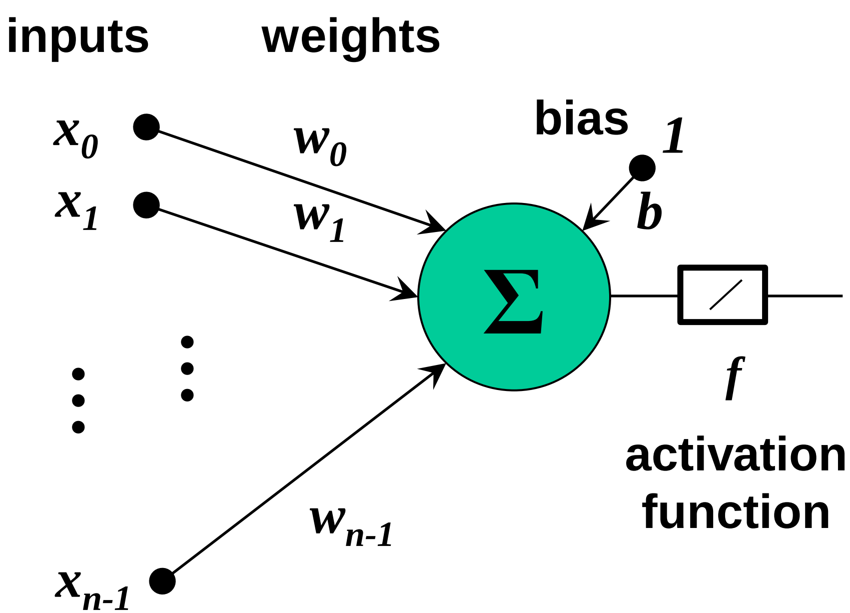
According to how we estimate suitable multiple of the opposite gradient vector:

- rmsprop
- ADAM

Neural networks

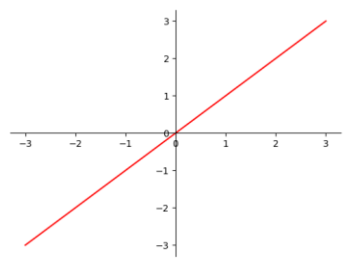
- NN are composed of “neurons”
- The simplest NN has 1 neuron and few parameters
- Typical models for perception have at least tens millions of neurons and parameters
- (Though the brain is a strong motivation for NN, „neuron“ has almost nothing with the neuron cells)

1 neuron with linear activation = linear regressor



output

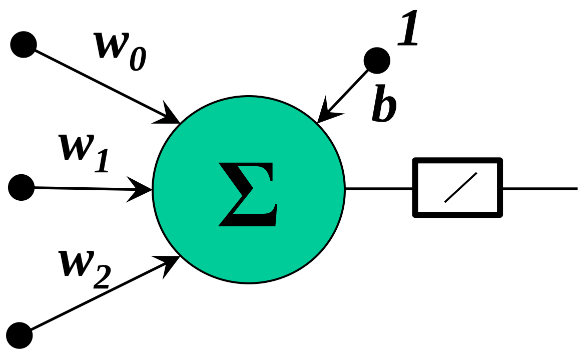
$$f(\sum w_i x_i + b)$$



- Neuron calculates the scalar product of inputs with weights (the weighted average), adds bias, and applies the activation function.

Linear regression

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



109 444 352 727 134

batch

w_0 = ?

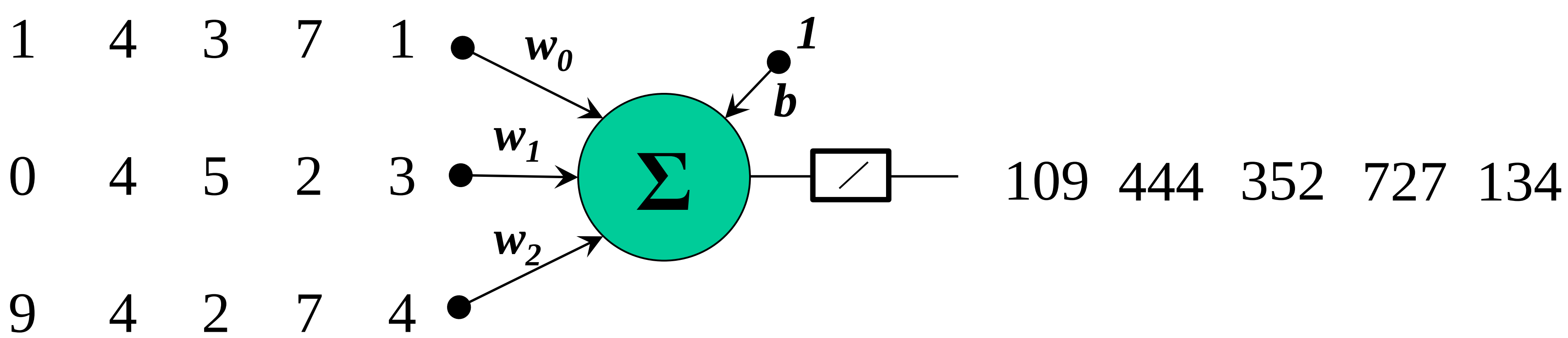
w_1 = ?

w_2 = ?

b = ?

When the activation is linear, finding suitable weights corresponds to linear regression

Linear regression



$w_0 =$ 100

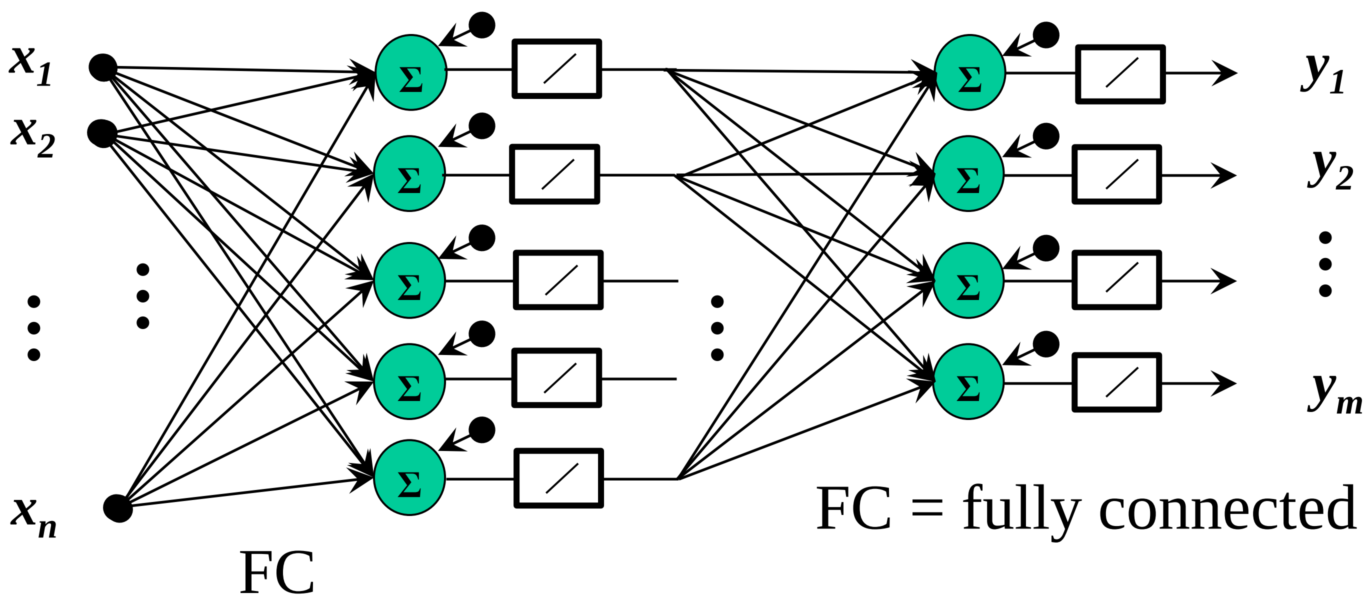
$w_1 =$ 10

$w_2 =$ 1

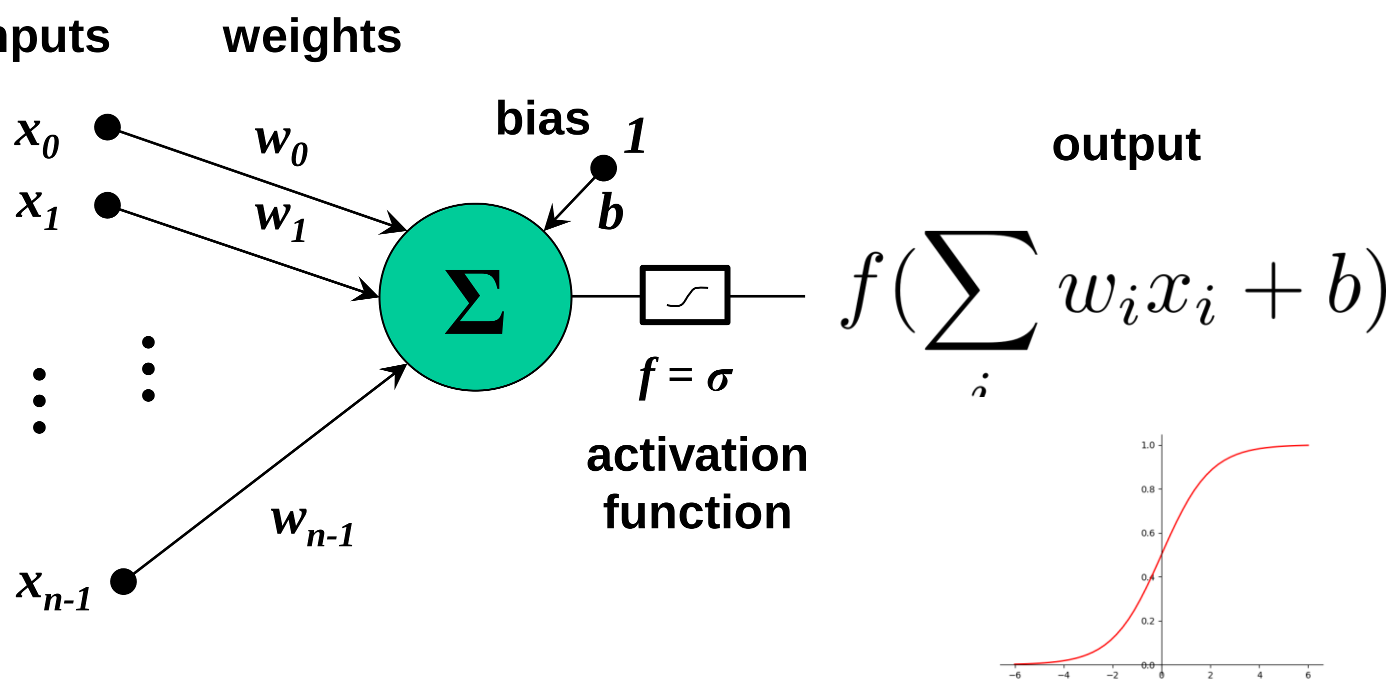
$b =$ 0

Perceptron

- Neural network from at least two fully connected layers
- With linear activations it is an interesting but not very useful machine



1 neuron with sigmoid activation = logistic regressor

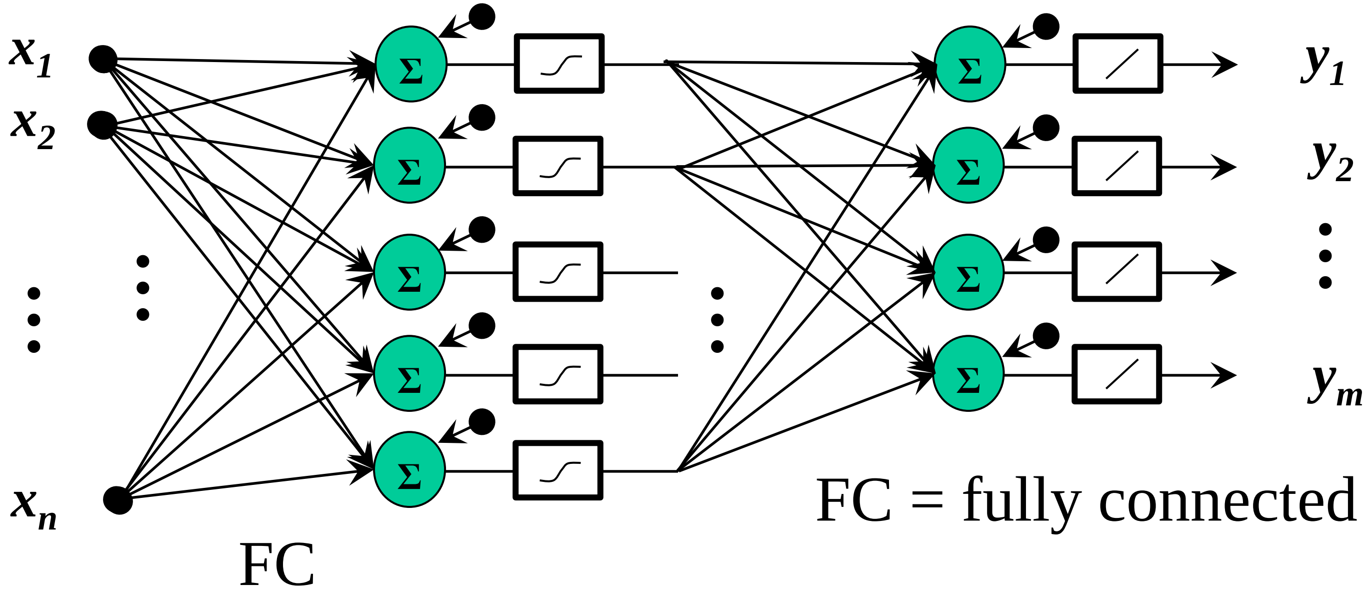


$$f(\sum_i w_i x_i + b)$$

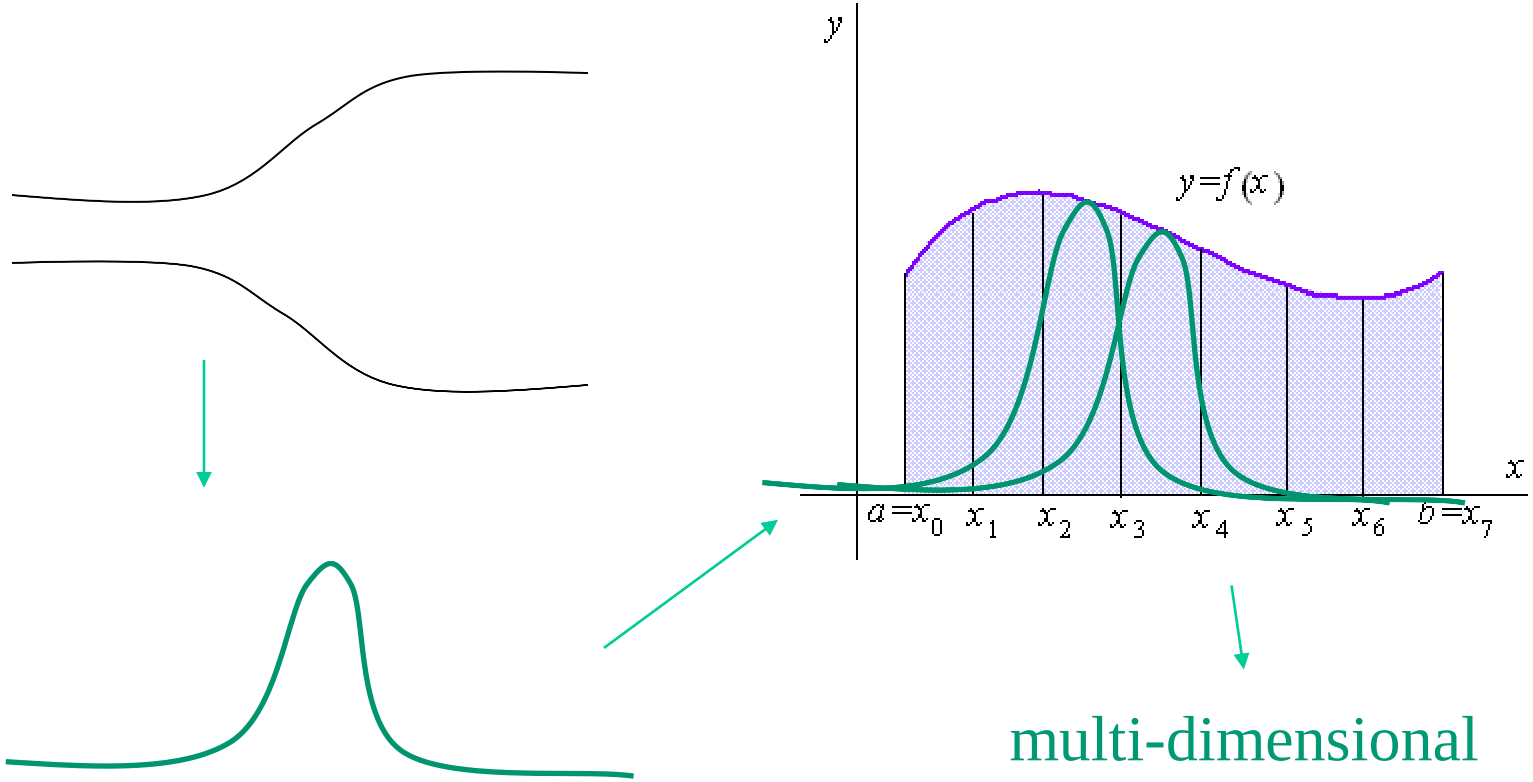
- Neuron calculates the scalar product of inputs with weights, adds bias, and applies the activation function.
- Sigmoidal functions: sigmoid, hyperbolic tangent

Perceptron

- With non-linear activation in the hidden layer, it is an **universal approximator**
- It is still less useful in practice when we process multi-dimensional data like images



Universal approximation



multi-dimensional
functions

Convolutional neural networks

- Perceptron works for low dimensional data
- Images are high dimensional data
- Solution? We will code images to features
- How? By classic CV – by kernels
- What kernels? We will find by training
- Pixel where kernel is applied = neuron
- Kernel coefficients = shared weights of neurons

Kernel

240x320x1

1x1x1

1.5

weight

0.15

bias

240x320x1

Kernel

240x320x1

1x1x1

1.5

weight

0

bias

240x320x1

weight changes contrast

Kernel

240x320x1

1x1x1

$\boxed{1.0}$

weight

$\boxed{0.15}$

bias

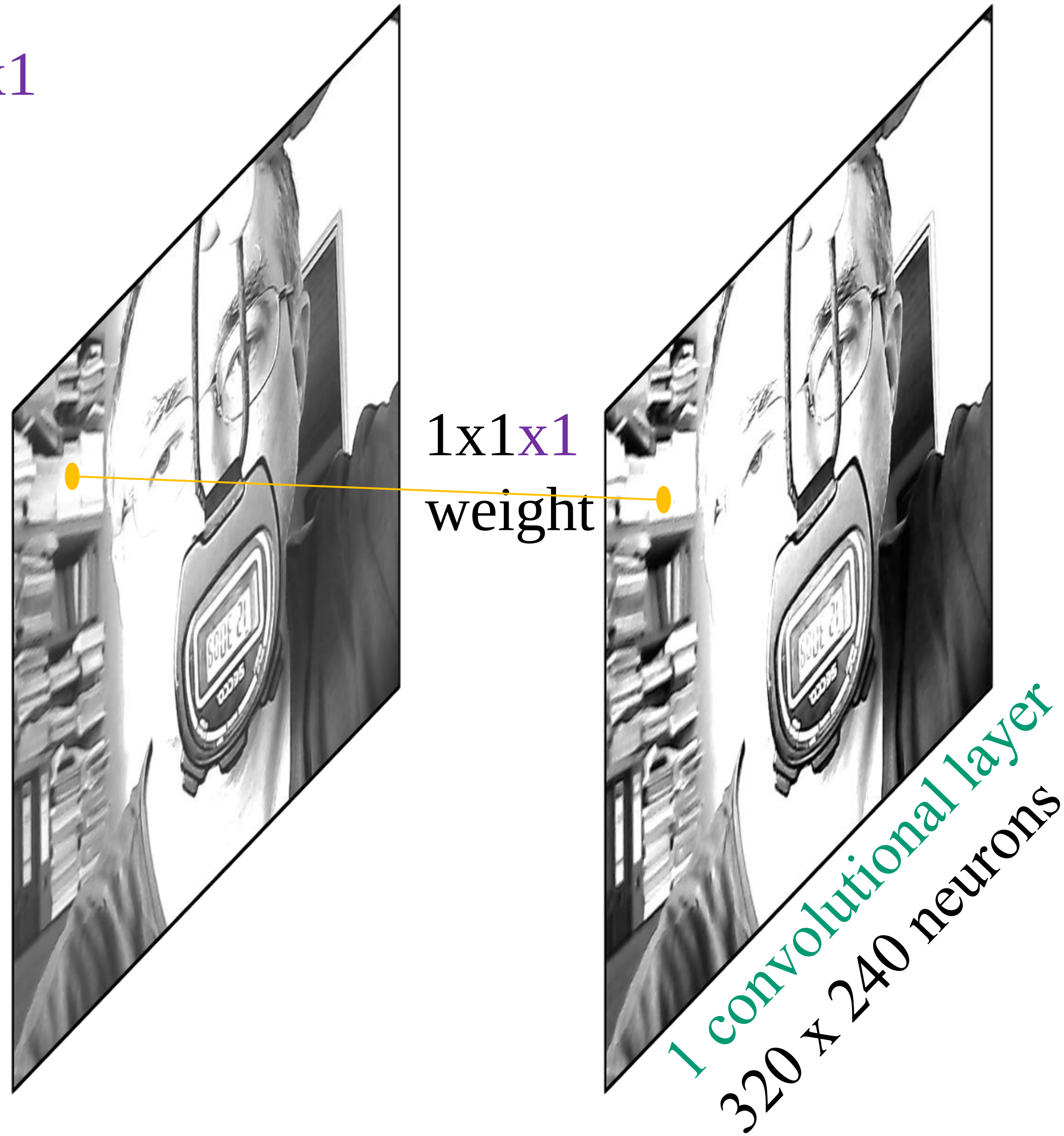
240x320x1

bias changes brightness

The Block of Convolutional Layers

320x240x1
input
values

input
dimension
76800



neurons share
1 weight and
1 bias

the layer contains
76800 neurons
but has 2
parameters only

320x240x1
output
values

activation: linear

Training

sample input

sample output

1.5	contrast
weight	
0.15	brightness
bias	

Kernel

240x320x3

3x3x3

-1/3	0	1/3
-2/3	0	2/3
-1/3	0	1/3

weights

+0

bias

240x320x1

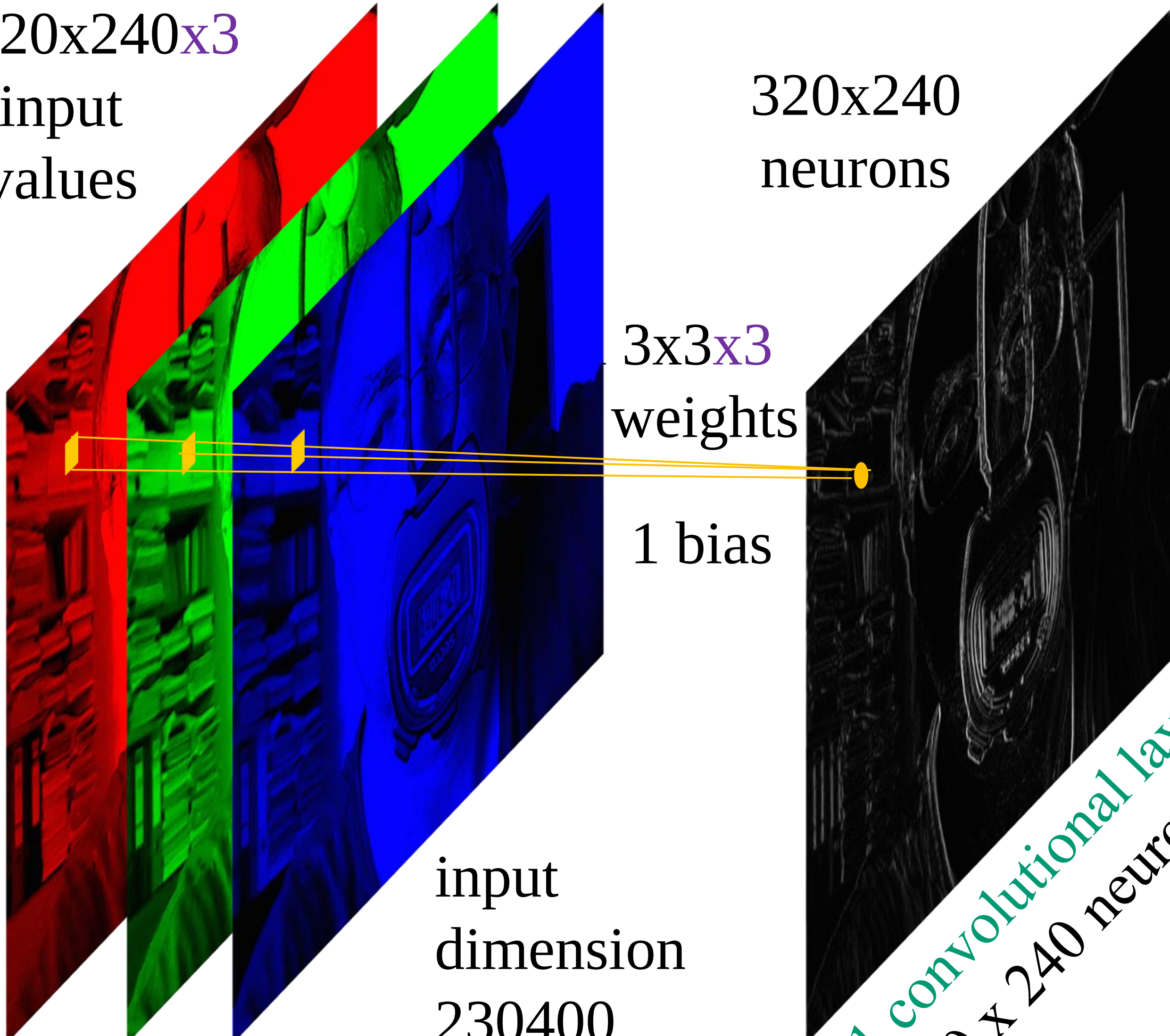
Sobel (vertical) kernel provides vertical edges

The Block of Convolutional Layers

320x240x3
input
values

320x240
neurons

neurons share
27 weights and
1 bias



3x3x3
weights

1 bias

the layer contains
76800 neurons
and 28 parameters

input
dimension
230400

1 convolutional layer
320 x 240 neurons

320x240x1
output
values

activation: linear

Training

sample input

sample output

weights

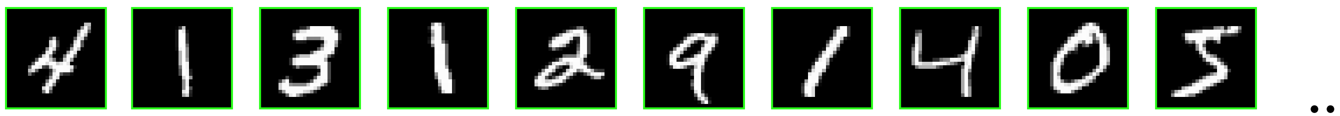
Sobel kernel

-1/3	0	1/3
-2/3	0	2/3
-1/3	0	1/3

+0 bias

Deap Learning

DL works with a data set

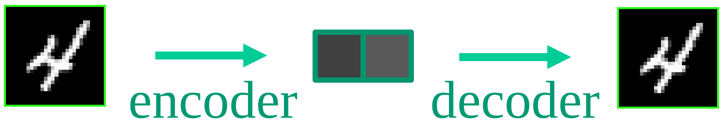


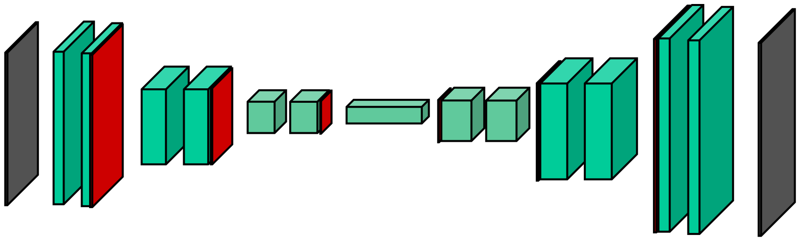
60000 images 28×28

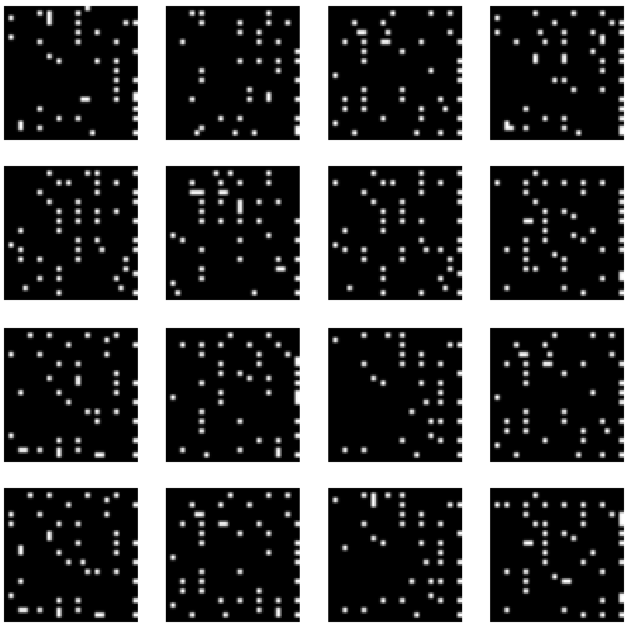
HU creates models that transform one data to other data

The key idea behind DL is creation of encoder (feature extractors) and decodes (feature generators)

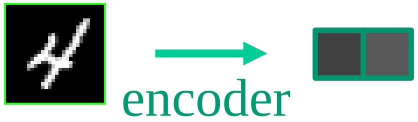
Autoencoder

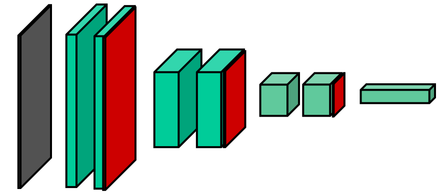


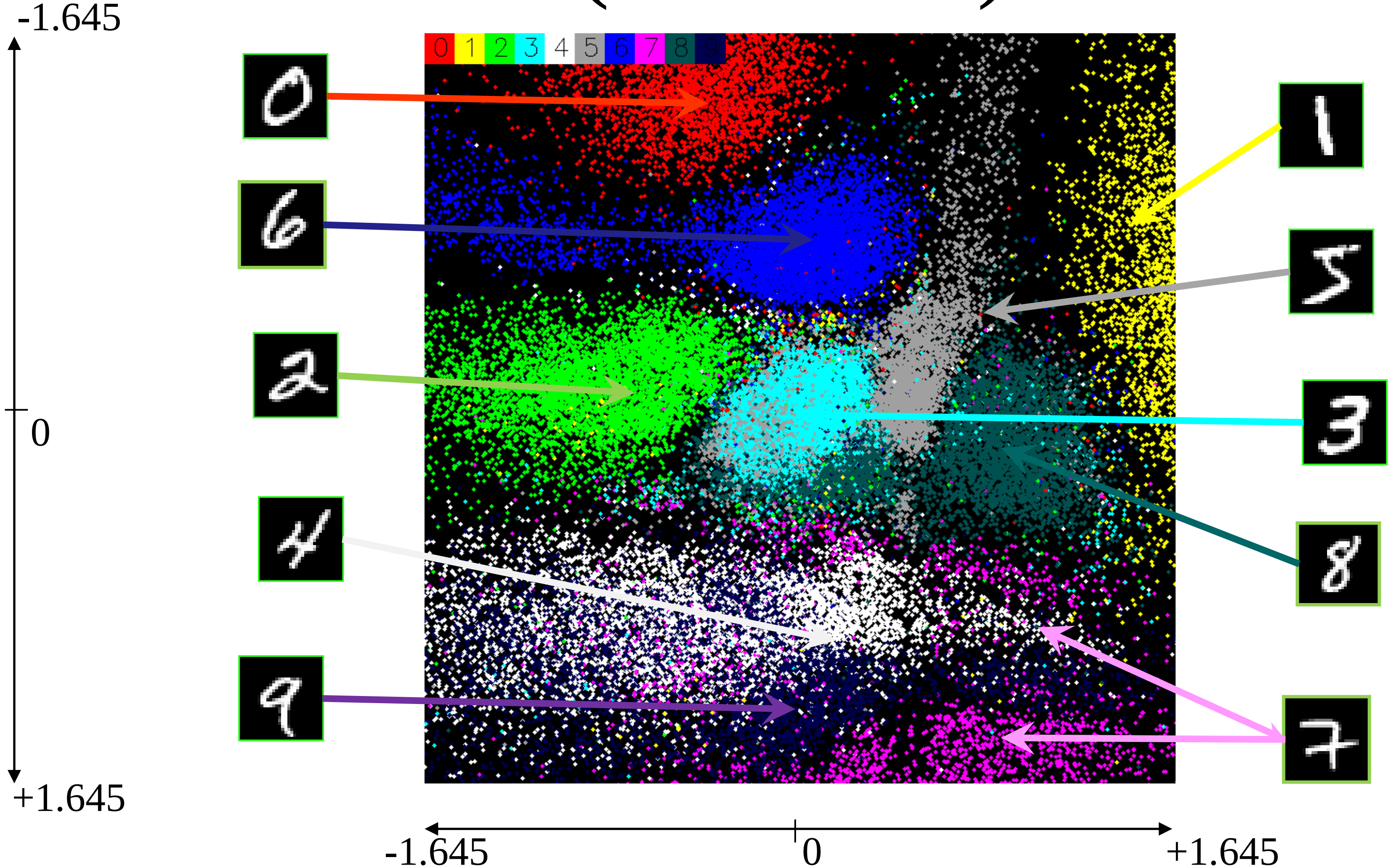




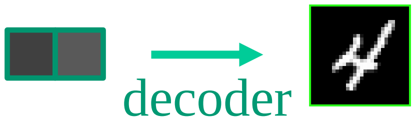
Encoder (Extractor)

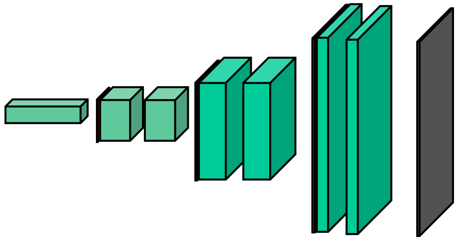


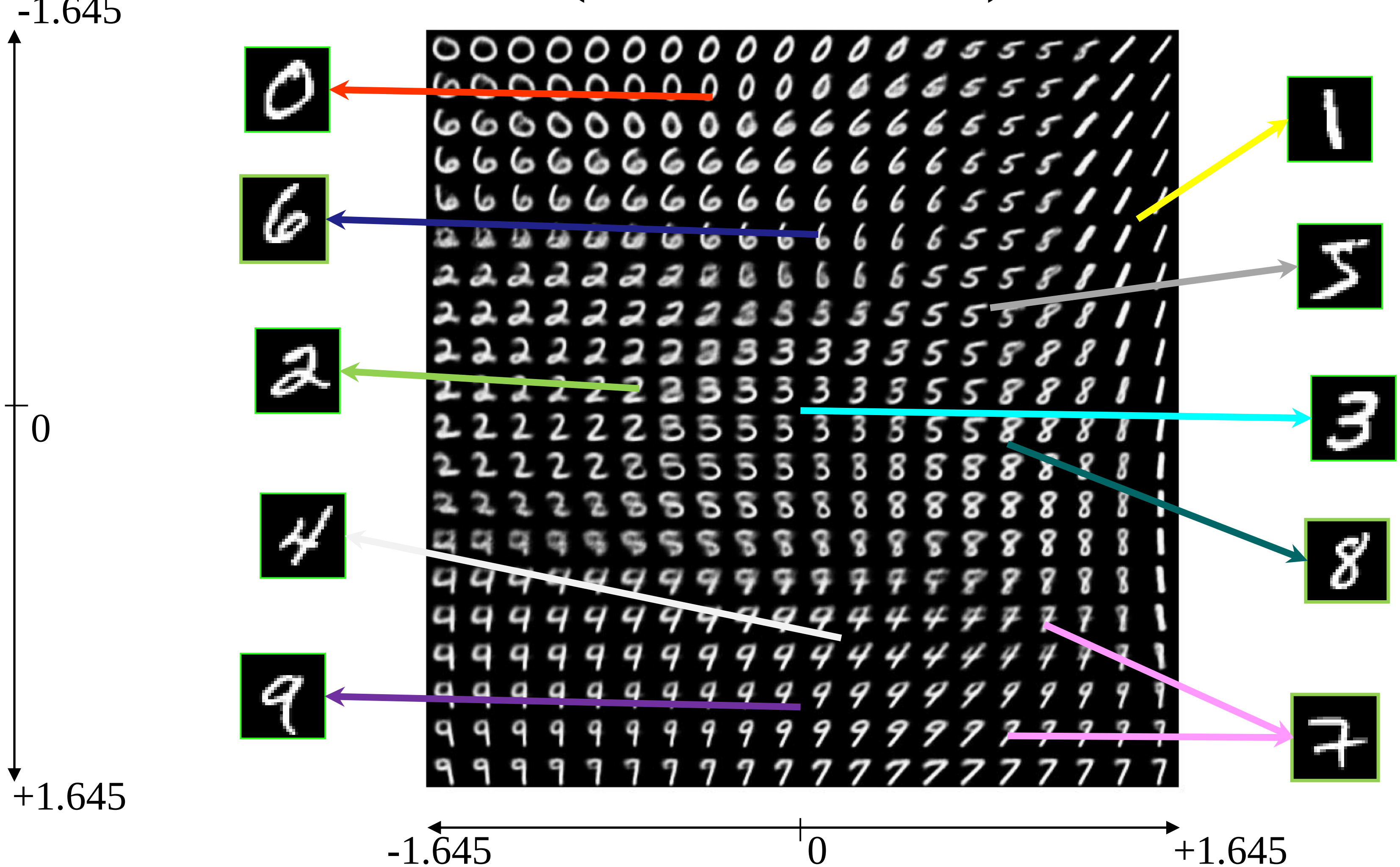




Decoder (Generator)







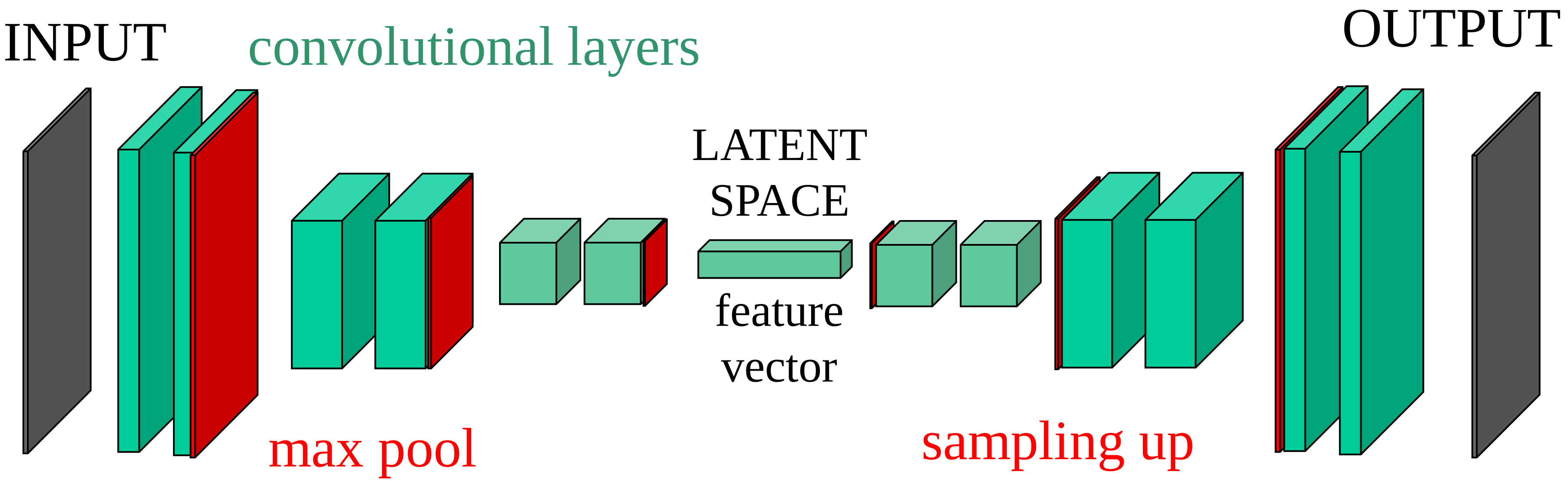
Fundamental features of feature vectors

- Feature vector are points in a latent space

- Though only finite number of points correspond to the samples from the dataset, all points represent a reasonable instance of data

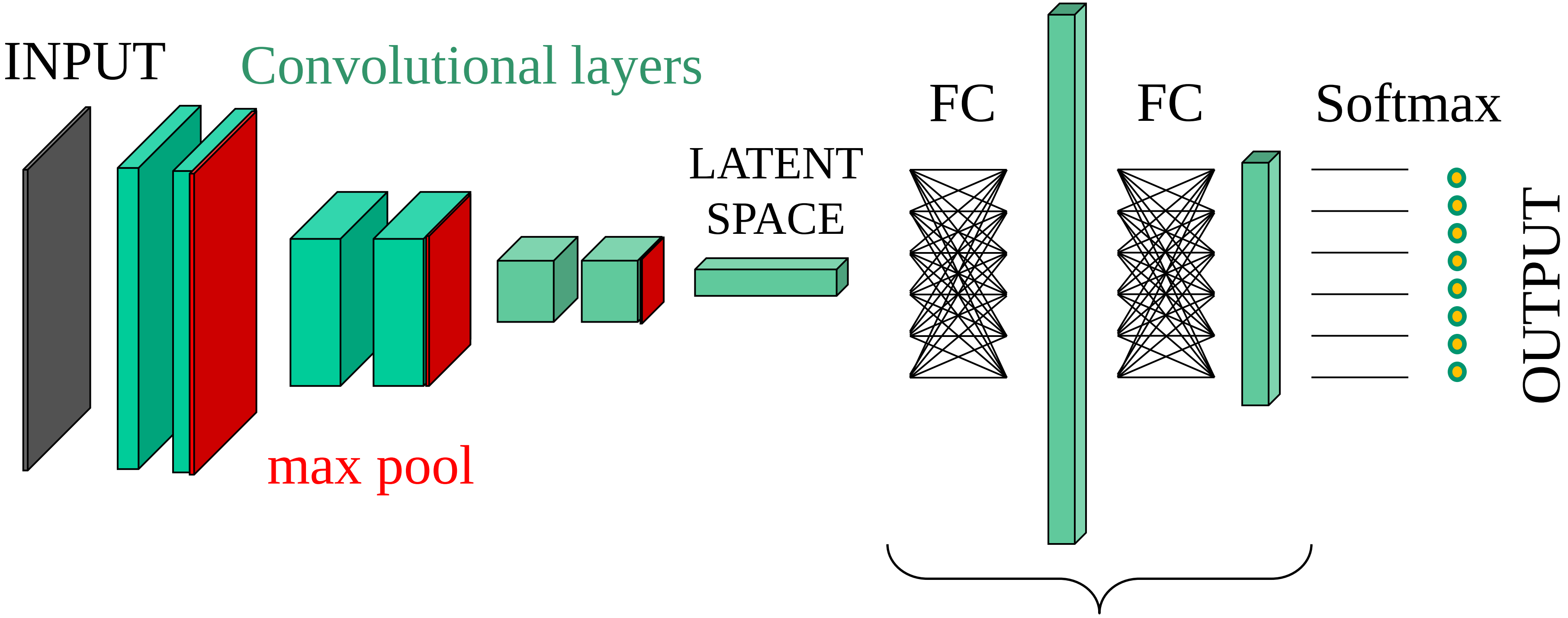
- The latent space has no holes (it is not sparse) and it is fluent (uniformly continuous)

(Convolutional) autoencoder



Encoder (the first half of autoencoder) transforms image to features (point in the latent space) ...

Classifier



... and for features perceptron works also in practice

Example

Emotions

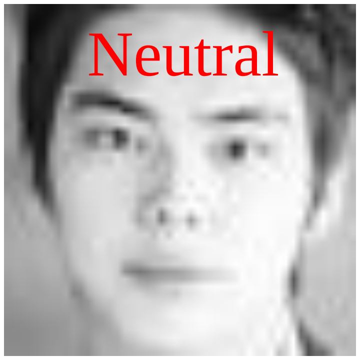
- various lists of some global states of mind (7 – 40)

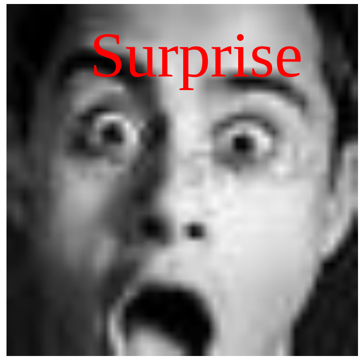
Plutchik's theory

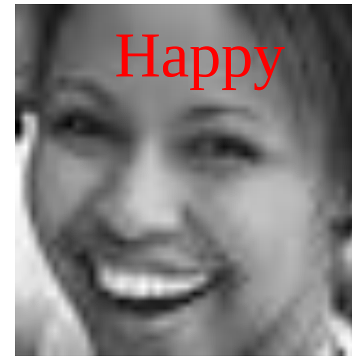
Fear → feeling of being afraid, frightened, scared.
Anger → feeling angry. A stronger word for anger is rage
Sadness → feeling sad. Other words are sorrow, grief
Joy → feeling happy. Other words are happiness, gladness
Disgust → feeling something is wrong or nasty. Strong disapproval.
Surprise → being unprepared for something.
Trust → a positive emotion; admiration is stronger; acceptance is weaker.
Anticipation → in the sense of looking forward positively to something which is going to happen. Expectation is more neutral

Emotion recognition

- Having dataset,

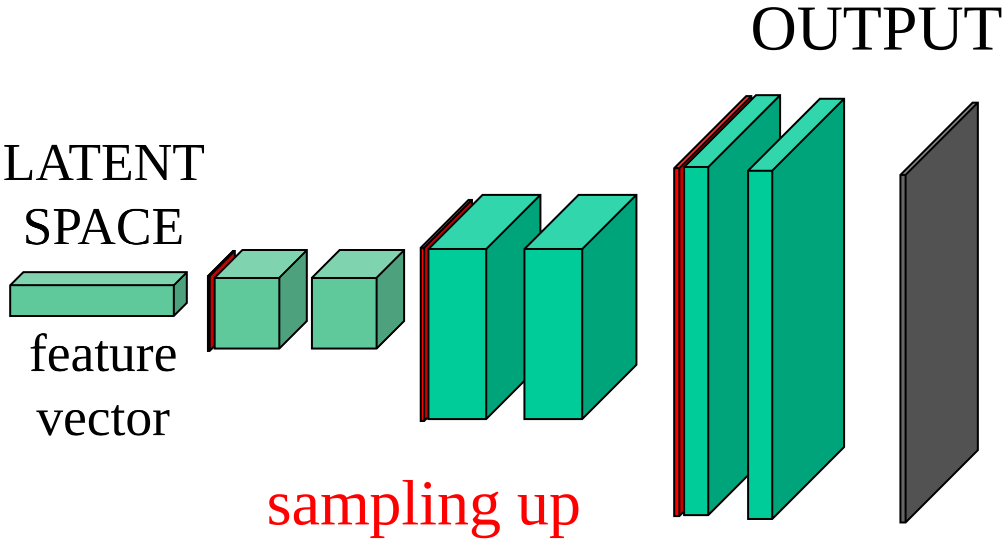






- we can train a deep model which output probabilities of individual emotions (classifier)
- Of course, before emotion recognition, we need to localize the face on images. This task can be provided by another deep model (detector)

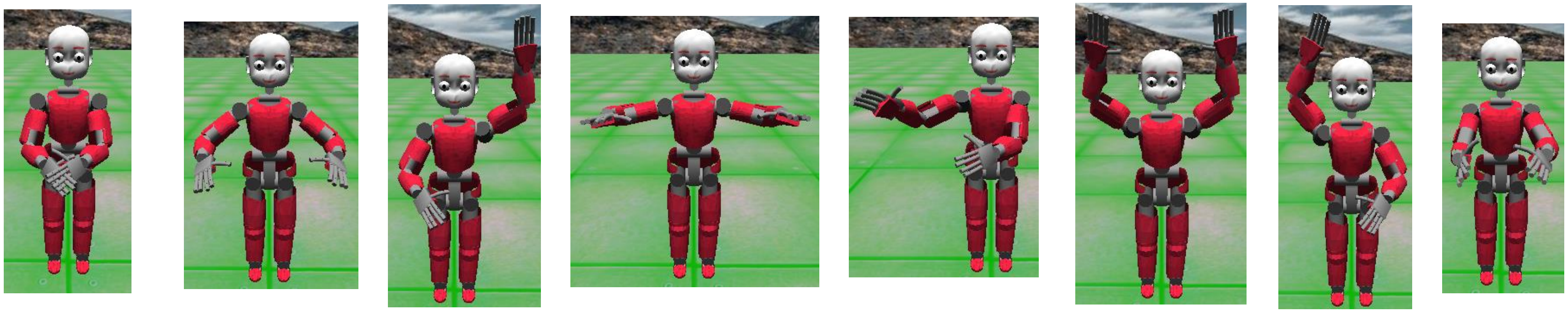
Generator



Example

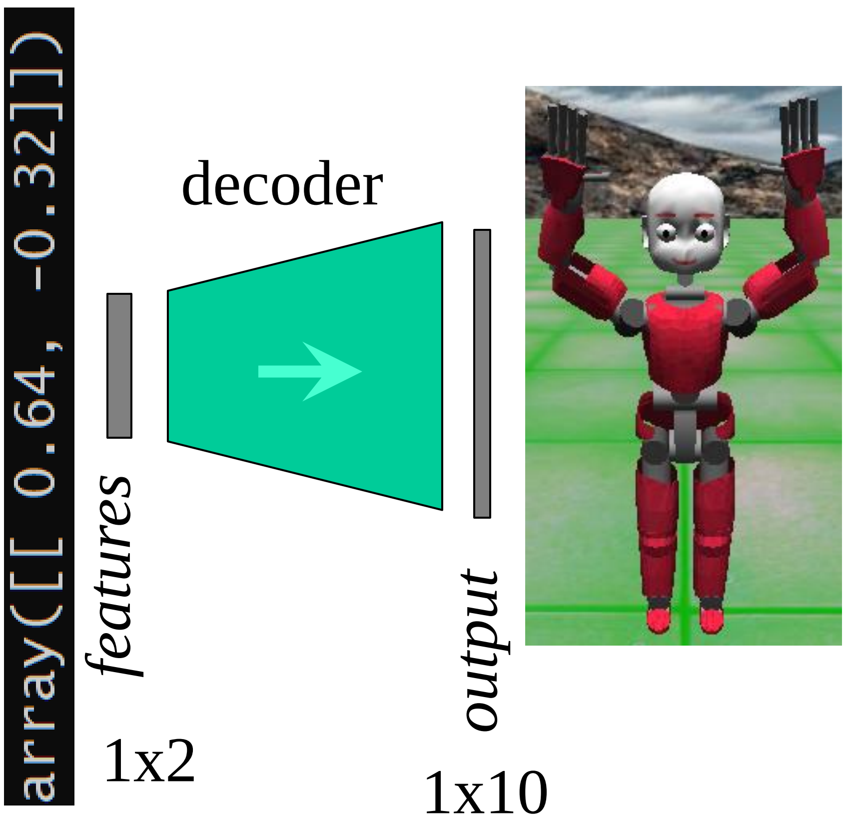
Dataset of robot's postures

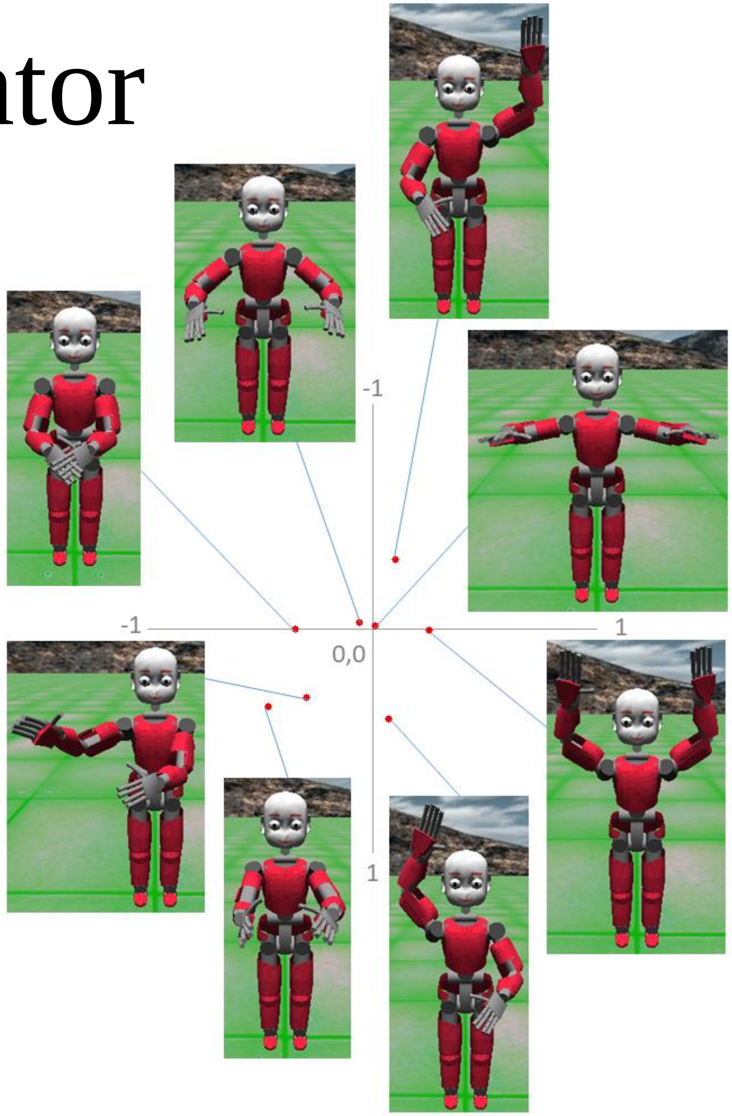
- 10 DOF, 60000 postures



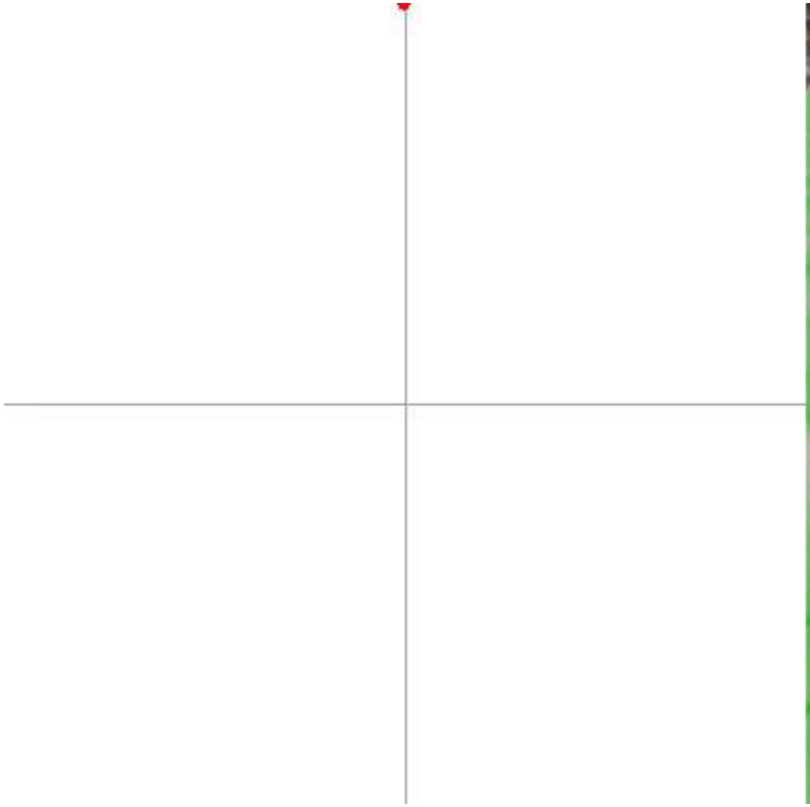
- We can train autoencoder and dissect it into the encoder and decoder

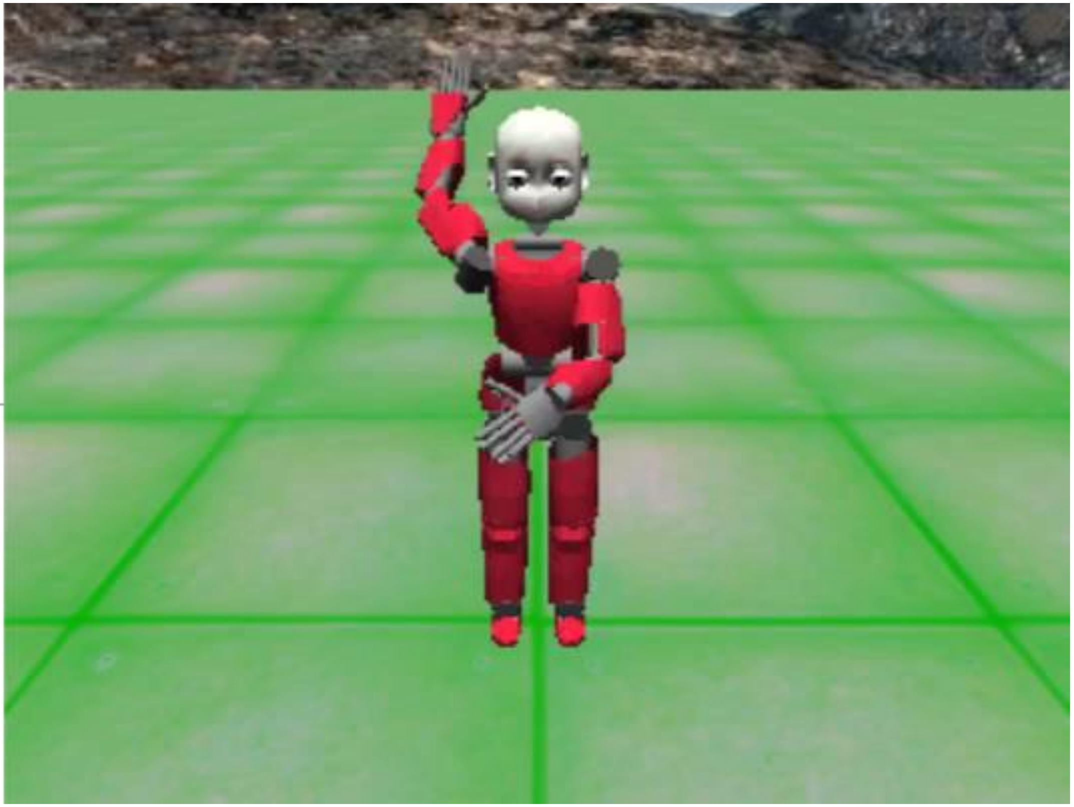
The posture generator





Generator





Detector

Classifier that is run in parallel over many regions on the image

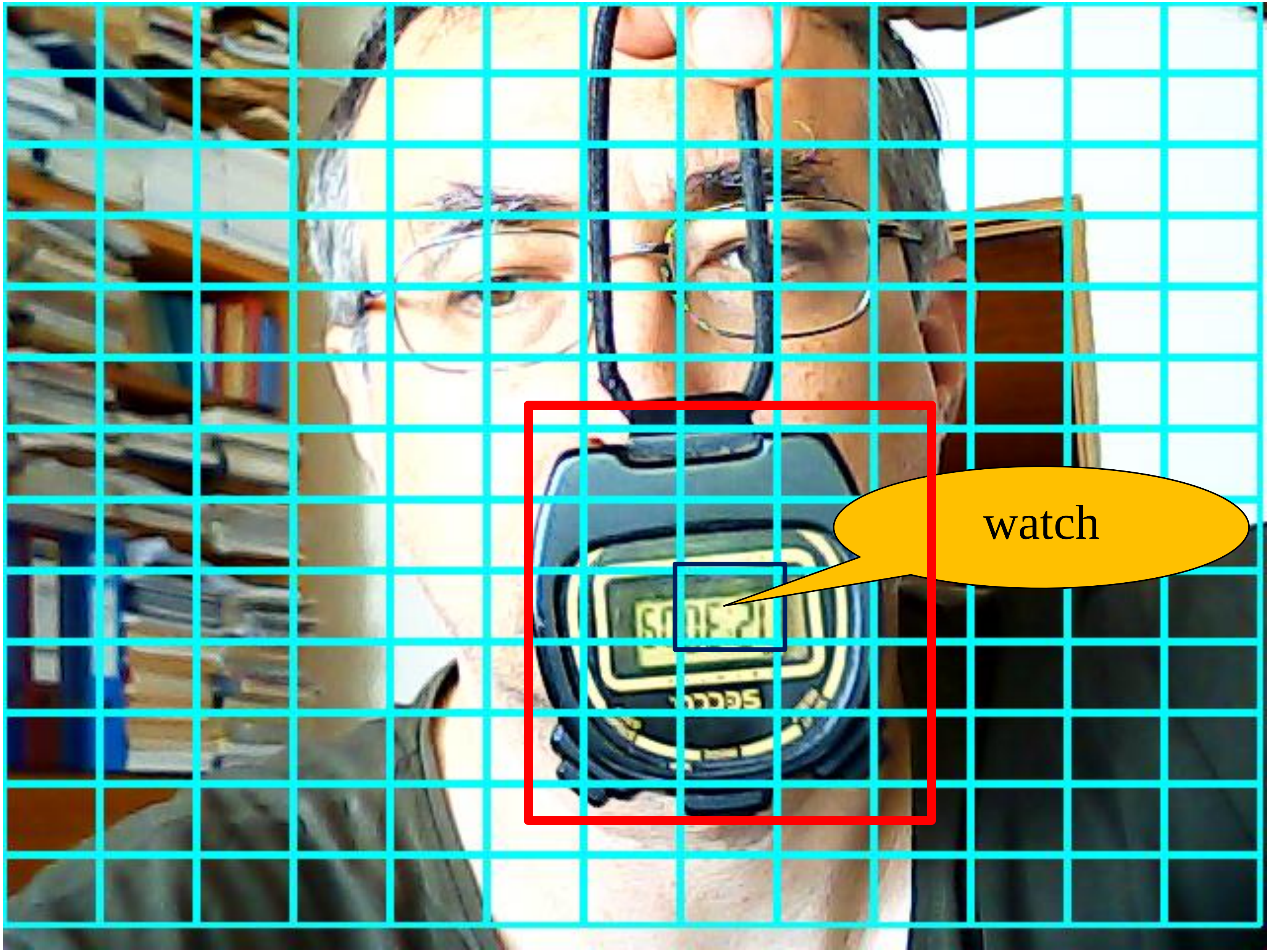
Classifier is materialized by perceptron and all these perceptrons share weights

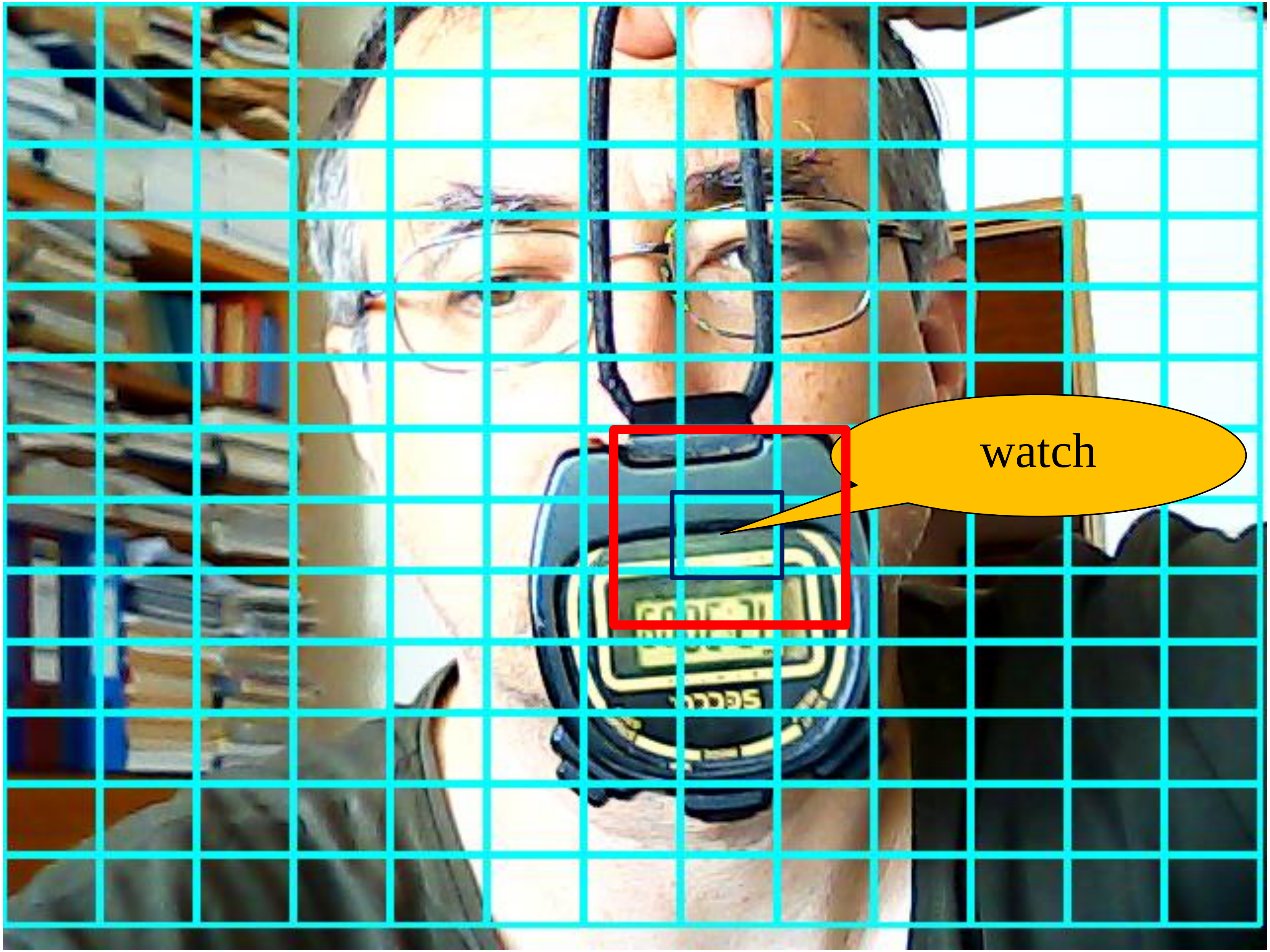
How do we build parallely operated perceptron? Unbelievable: by two or three blocks of the convolutional layers

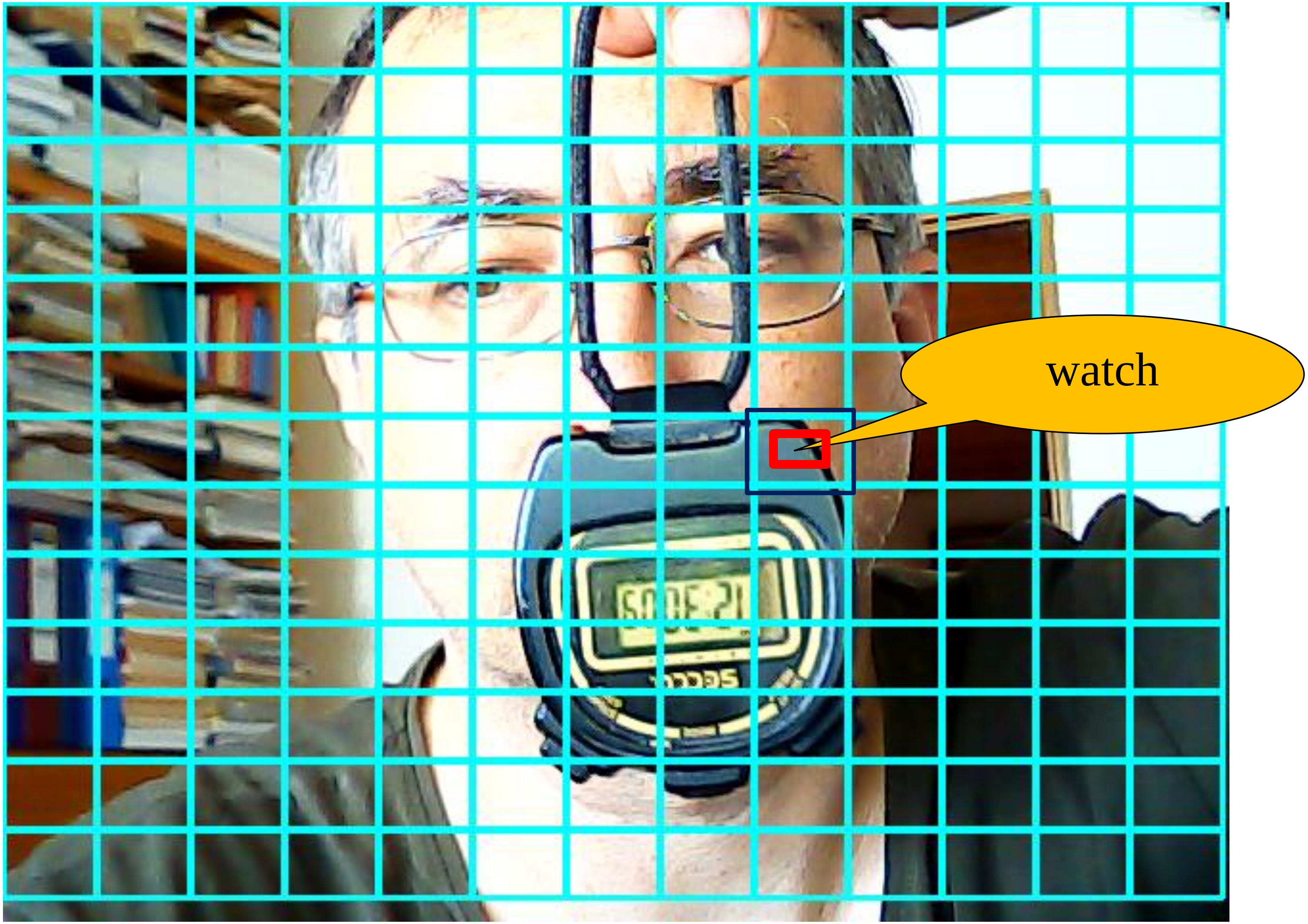
YOLO Detector (You Look Only Once)

- It covers image by non-overlapping regions
- it combines classification and regression tasks for each region and summarize them

- The classifier predicts the probability that the region belongs to the object.
- The regressor predicts the bounding box of the object lying in the region.


watch

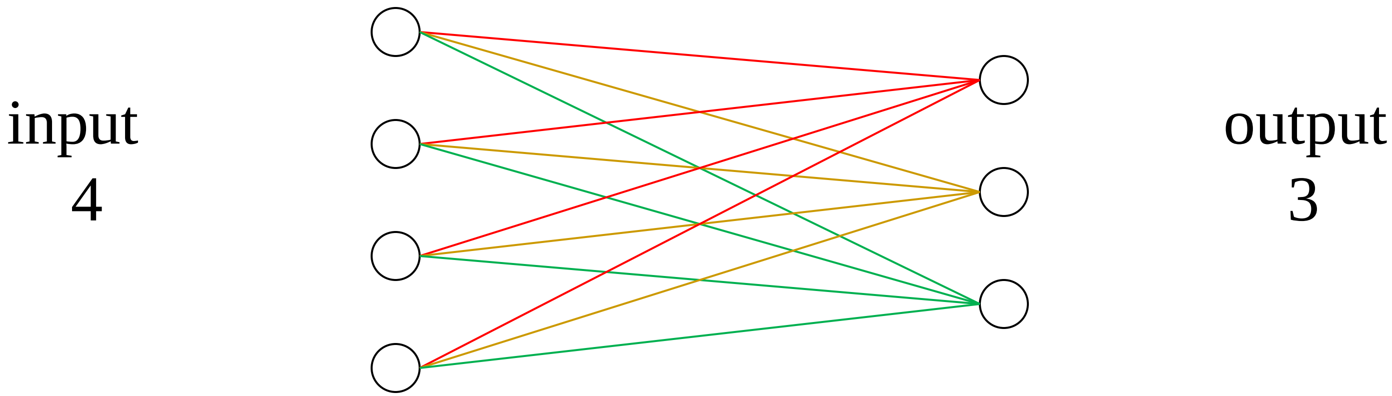

watch


watch

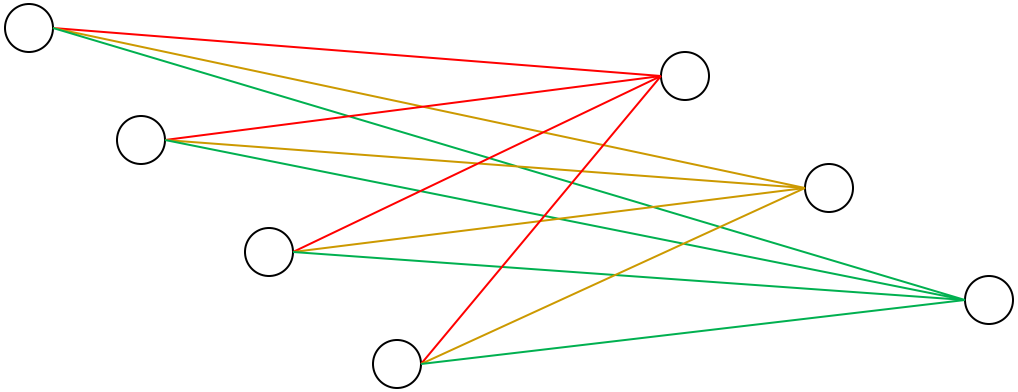

face


face

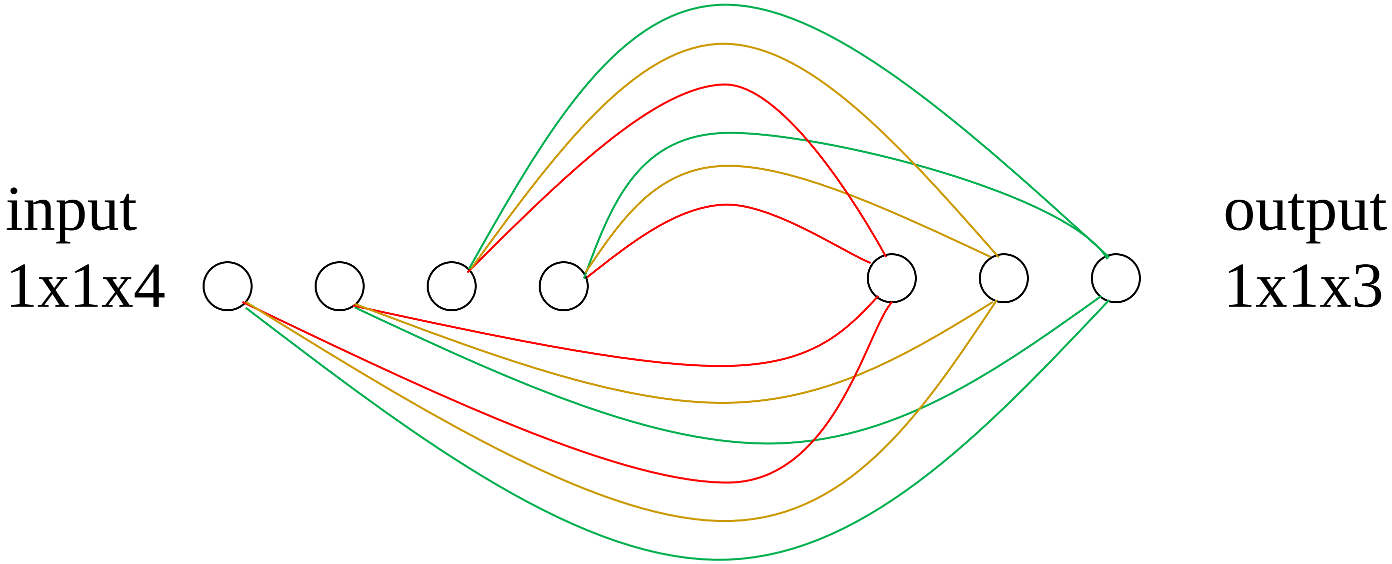
How could we perform the classification and regression tasks for each region? Well, we need to run the same perceptron for each spot. In other words: we need a building block of neural networks for running perceptrons sharing weights in parallel. Surprisingly, the block is already available to us: the block of convolutional layers with kernel 1x1.



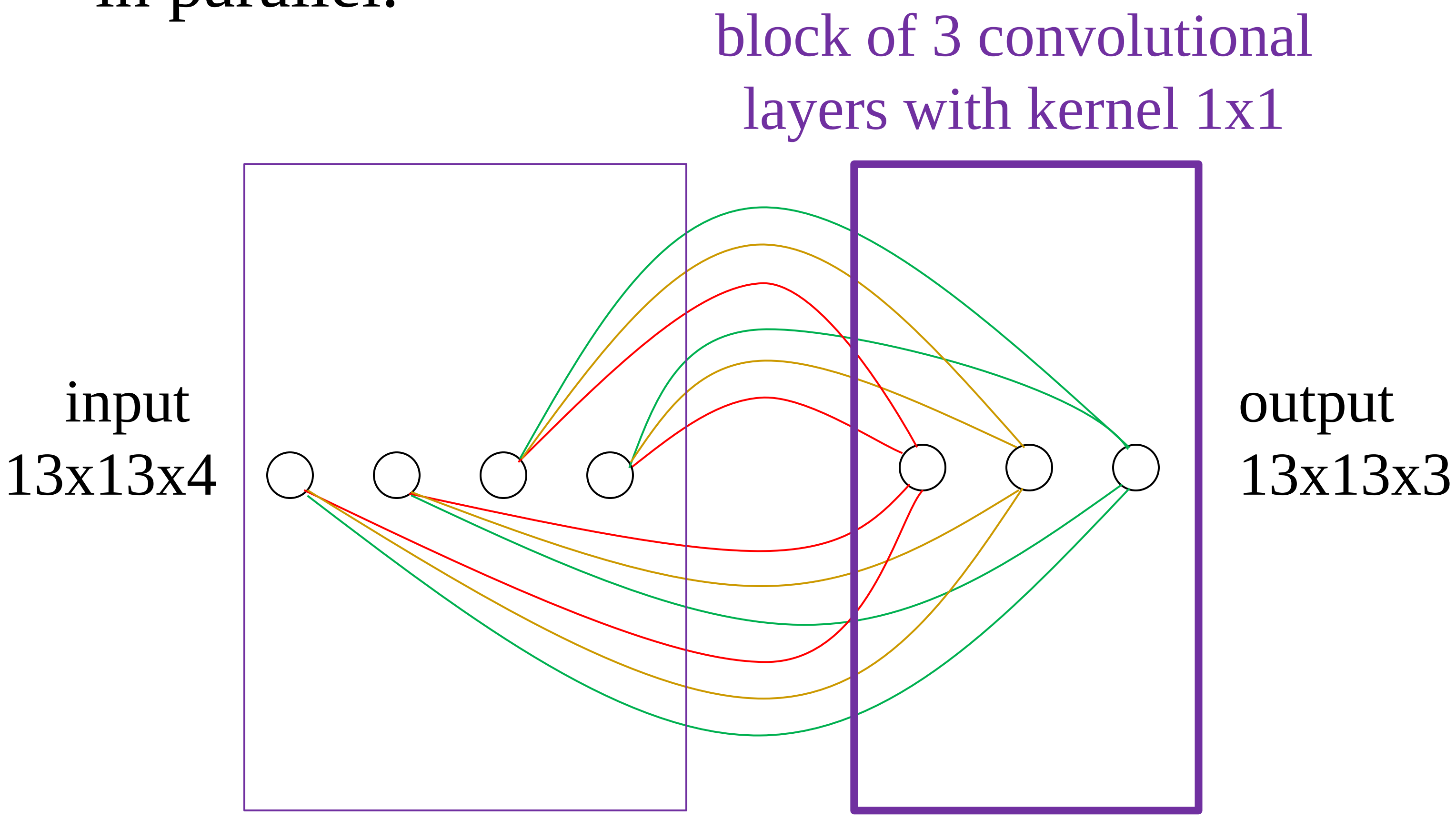
How could we perform the classification and regression tasks for each region? Well, we need to run the same perceptron for each spot. In other words: we need a building block of neural networks for running perceptrons sharing weights in parallel. Surprisingly, the block is already available to us: the block of convolutional layers with kernel 1x1.



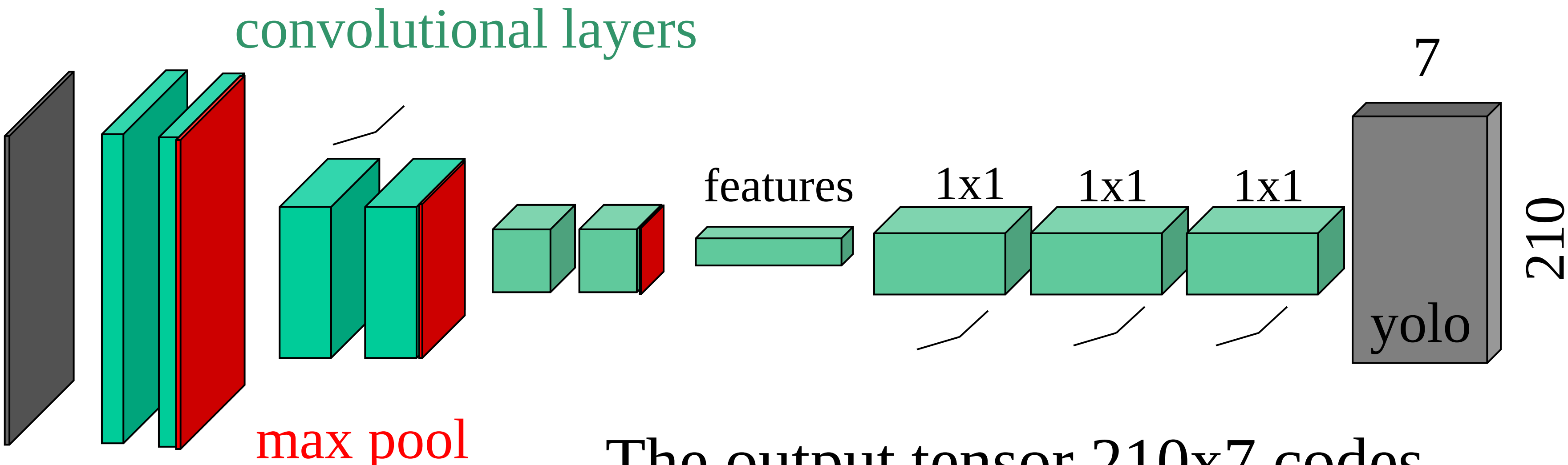
How could we perform the classification and regression tasks for each region? Well, we need to run the same perceptron for each spot. In other words: we need a building block of neural networks for running perceptrons (sharing weights and biases) in parallel. Surprisingly, the block is already available to us: the block of convolutional layers with kernel 1x1.



When we feed input 13x13x4 we get output 13x13x3 corresponding to the production of 169 perceptrons (that shares weights and biases) running in parallel.



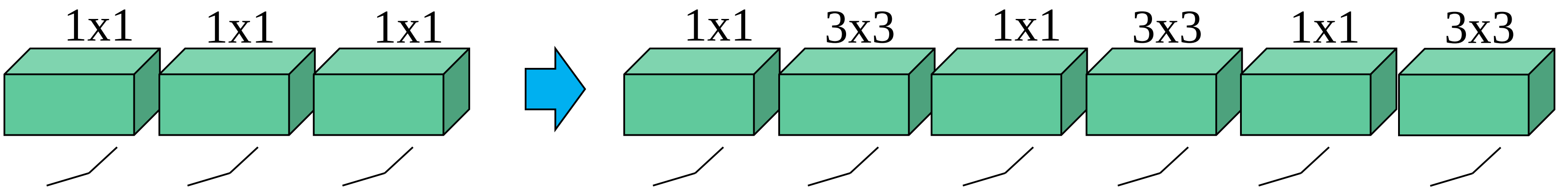
YOLO v1



The output tensor 210x7 codes maximally 210 detections, each containing: relevance, category, confidence, x, y, w, and h

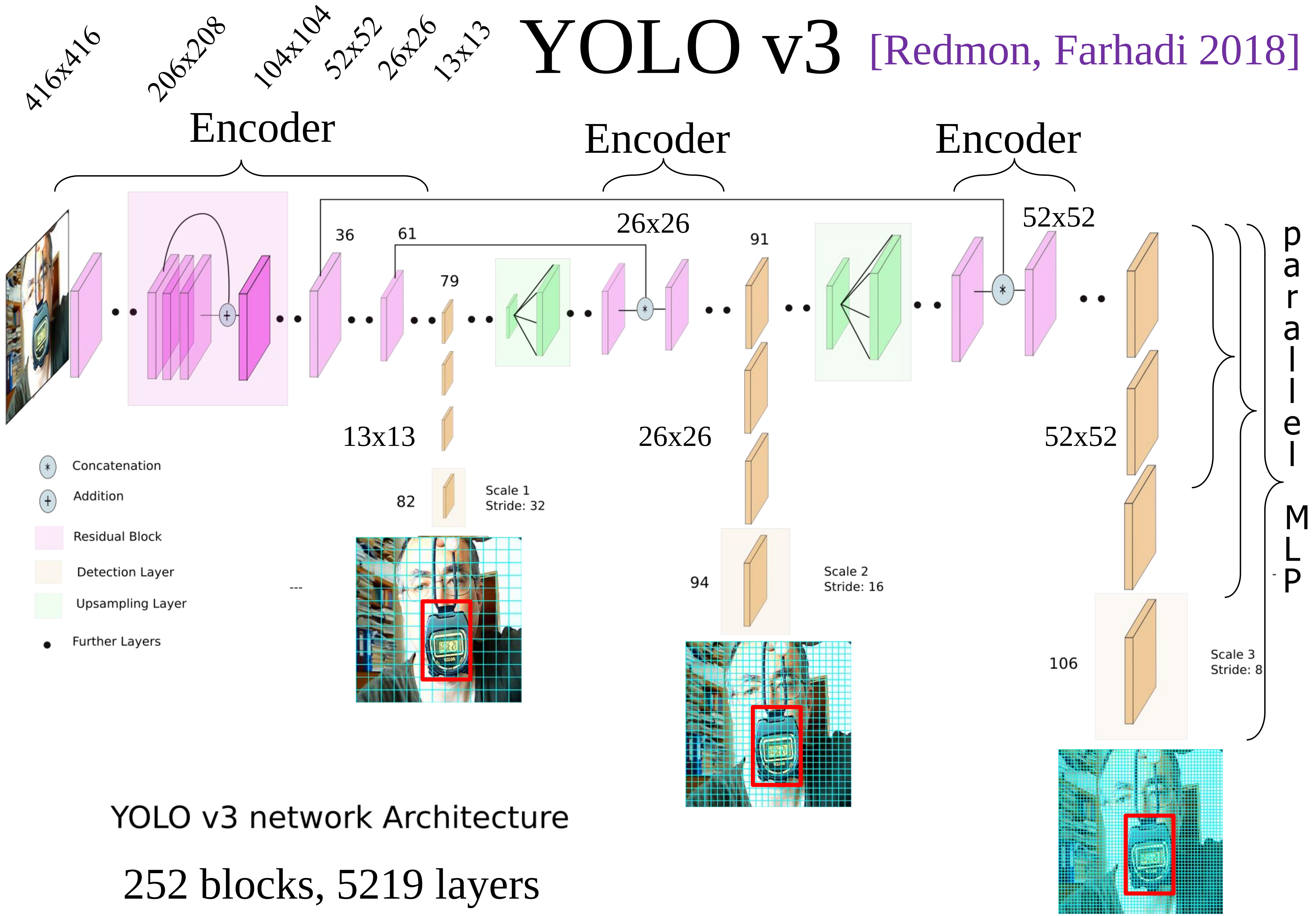
(summarized by the special building block called yolo)

- If we have a piece of a watch in one region, it increases the probability of having it in neighboring regions
- How could the parallel perceptrons cooperate?
- We use kernel 3x3 for that



- How could we treat different sizes of objects? We can run more processing pipelines in parallel for 13x13, 26x26, and 52x52 regions
- Then, the less-detailed pipeline can advise the more-detailed one

YOLO v3 [Redmon, Farhadi 2018]



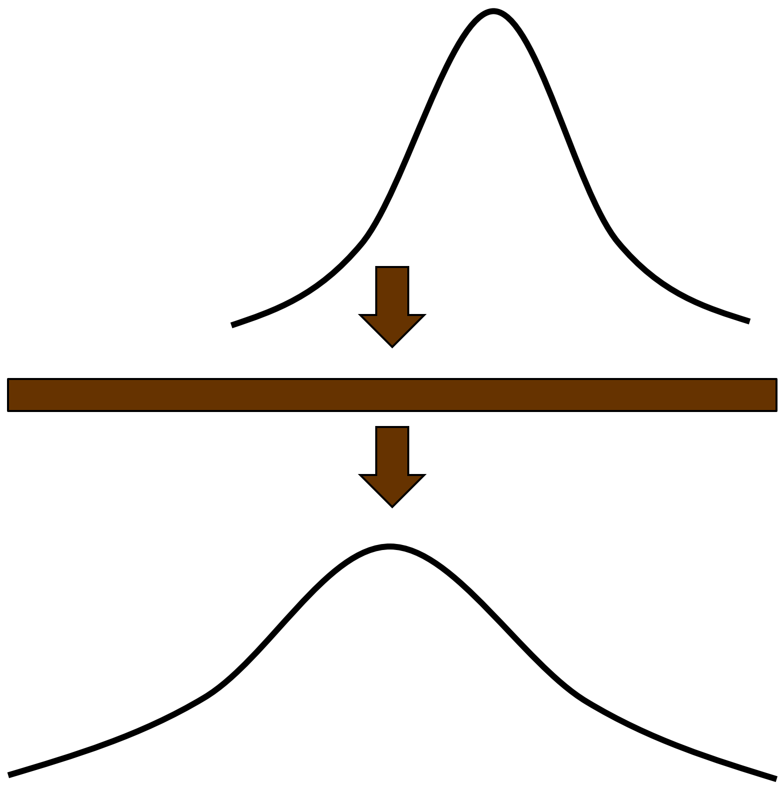
YOLO v3 network Architecture

252 blocks, 5219 layers

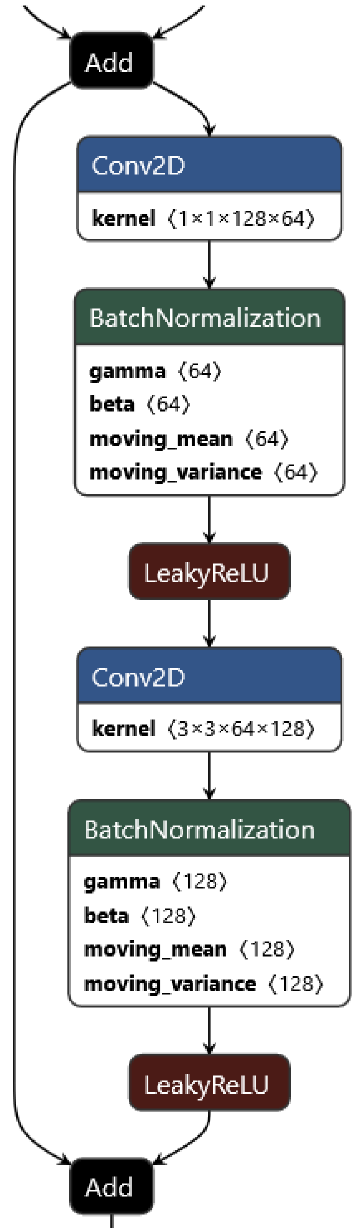
Training Deep Neural Networks

- Depth (about 200 neurons) enables us to design a sophisticated architecture. However, it makes training much harder (the problem o vanishing gradients).
- Therefore, we have to add building blocks that give us a chance to handle the training. YOLO v3 employs batch normalization and residual connections for this purpose.

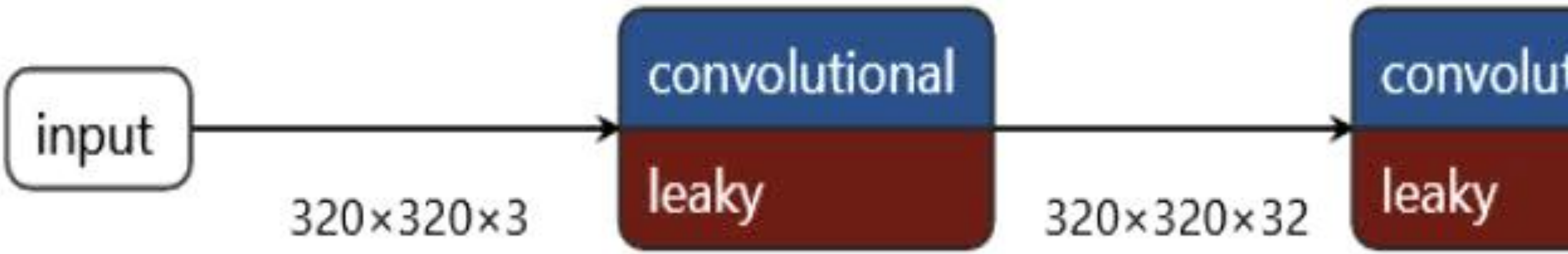
Batch Normalization



Residuals



YOLO v3 schema



YOLO v3 output

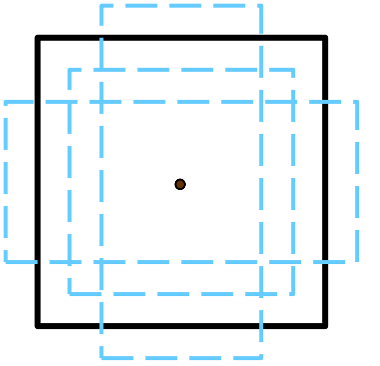
- 13x13x255, 26x26x255, 52x52x255
- Each perceptron approximates three detections, 255 = 3 x 85
- Each detection 85 = 4+1+80 contains:

  4 … x, y, w, h

  1 … confidence

  80 … probabilities for individual categories

Anchors



- Each detection expresses x,y,w, and h relative to a given anchor. YOLO employ three anchors for each object size (nine altogether) with different aspect ratio.
- Anchors in YOLO v3:
  [116x90, 156x198, 373x326] (output 52x52)
  [30x61, 62x45, 59x119] (output 26x26)
  [10x13, 16x30, 33x23] (output 13x13)

Transfer learning

YOLO v3 provides the pre-trained model, trained on the COCO dataset (80 categories):

person, bicycle, car, motorbike, aeroplane, bus, train, truck, boat, traffic light, fire hydrant, stop sign, parking meter, bench, bird, cat, dog, horse, sheep, cow, elephant, bear, zebra, giraffe, backpack, umbrella, handbag, tie, suitcase, frisbee, skis, snowboard, sports, ball, kite, baseball bat, baseball glove, skateboard, surfboard, tennis racket, bottle, wine glass, cup, fork, knife, spoon, bowl, banana, apple, sandwich, orange, broccoli, carrot, hot dog, pizza, donut, cake, chair, sofa, pottedplant, bed, diningtable, toilet, tvmonitor, laptop, mouse, remote, keyboard, cell phone, microwave, oven, toaster, sink, refrigerator, book, clock, vase, scissors, teddy bear, hair drier, toothbrush

Starting training from the pre-trained model gives us a better chance to train our custom detector.

Training YOLO v3

Processing the video