ML for natural and physical scientists 2023 9

Deep Learning 2 - Convolutional NNs

dr.federica bianco | *fbb.space* | *fedhere* | *fedhere*

this slide deck:

https://slides.com/federicabianco/mlpns23_9

- **Machine Learning basic concepts**
  - interpretability
  - parameters vs hyperparameters
  - supervised/unsupervised
- **CART methods**
- **Clustering methods**
- **Neural Networks**

- **Neural Networks**
  - the brain connection
  - perceptron
  - learning
  - activation functions
  - shallow nets
  - deep nets architecture
  - back-propagation
  - preprocessing and whitening (minibatch)

neural networks

perceptrons

Perceptrons are ***linear classifiers:*** makes its predictions based on a linear predictor function

combining a set of weights (=parameters) with the feature vector.

recap

$$y = wx + b$$

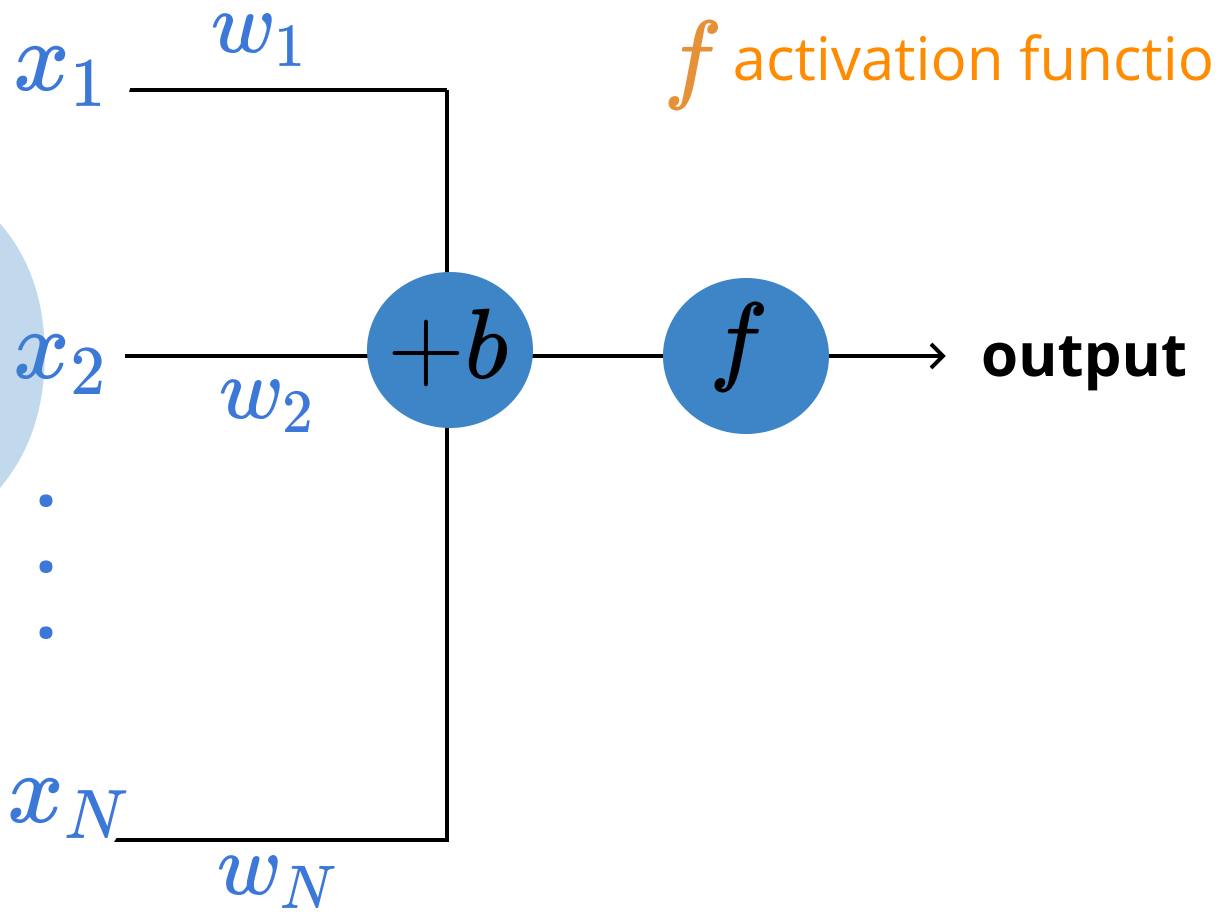
$$y = \sum_i w_i x_i + b$$

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

w_i weights

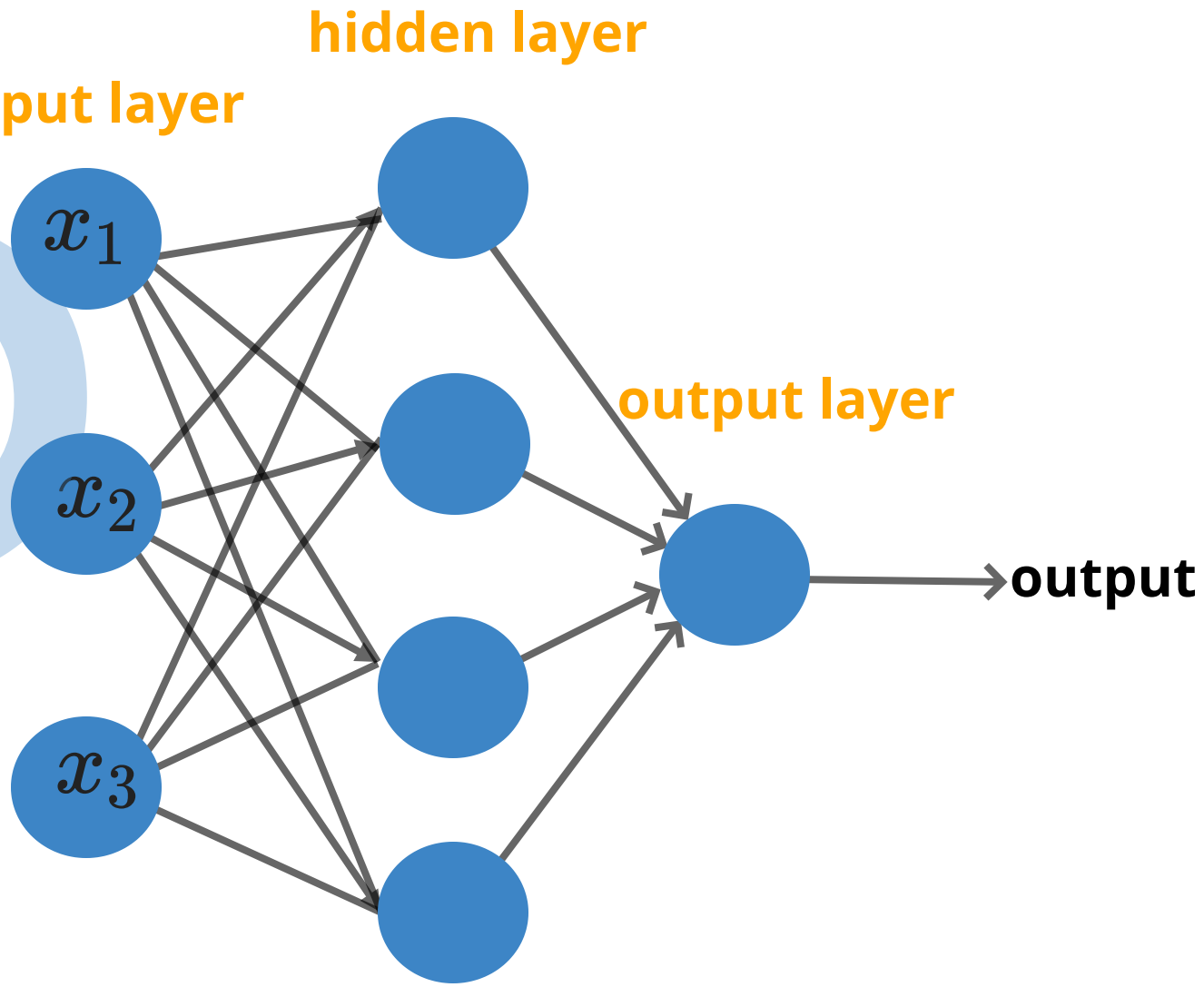
b bias

f activation function



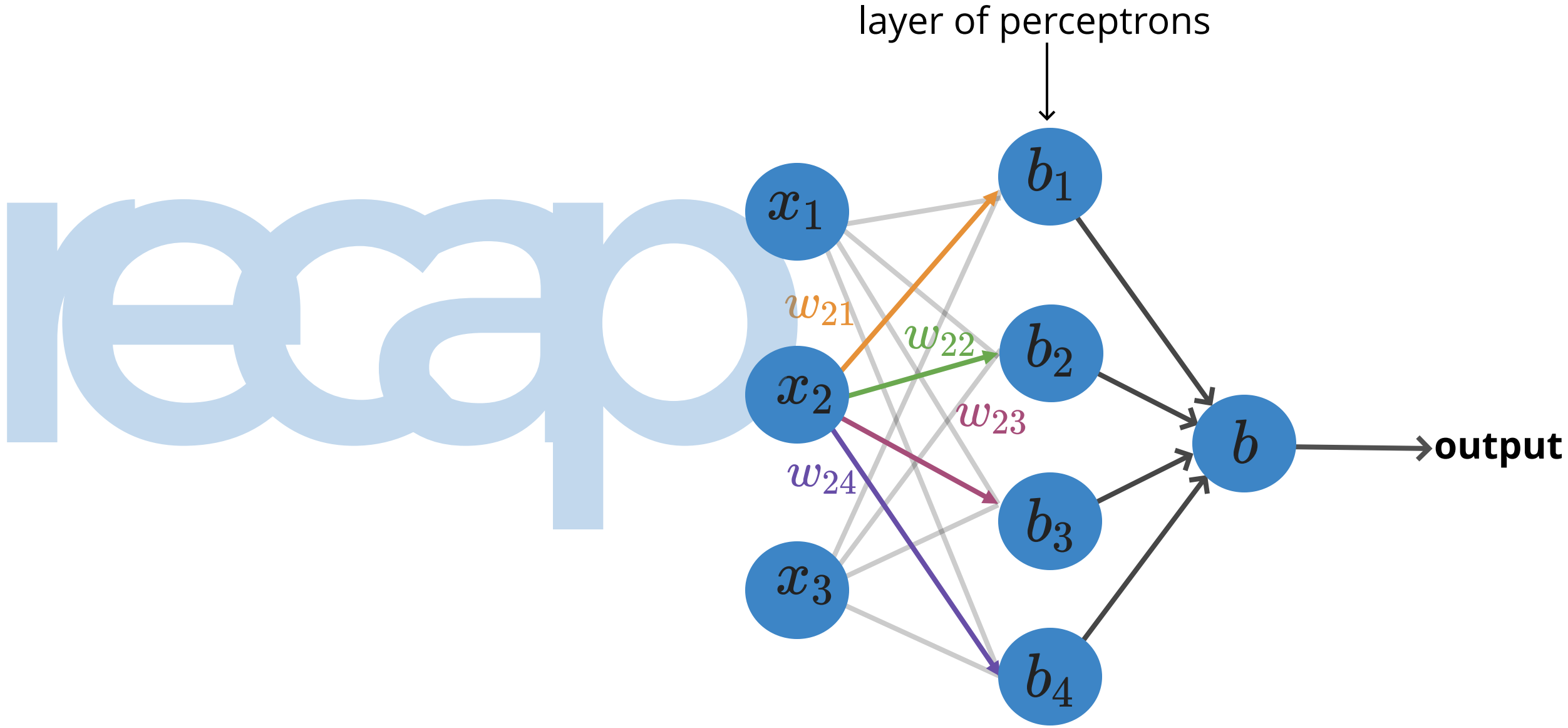
multilayer perceptron

1970: multilayer perceptron architecture



Fully connected: all nodes go to all nodes of the next layer.

multilayer perceptron



multilayer perceptron

Fully connected: all nodes go to all nodes of the next layer.

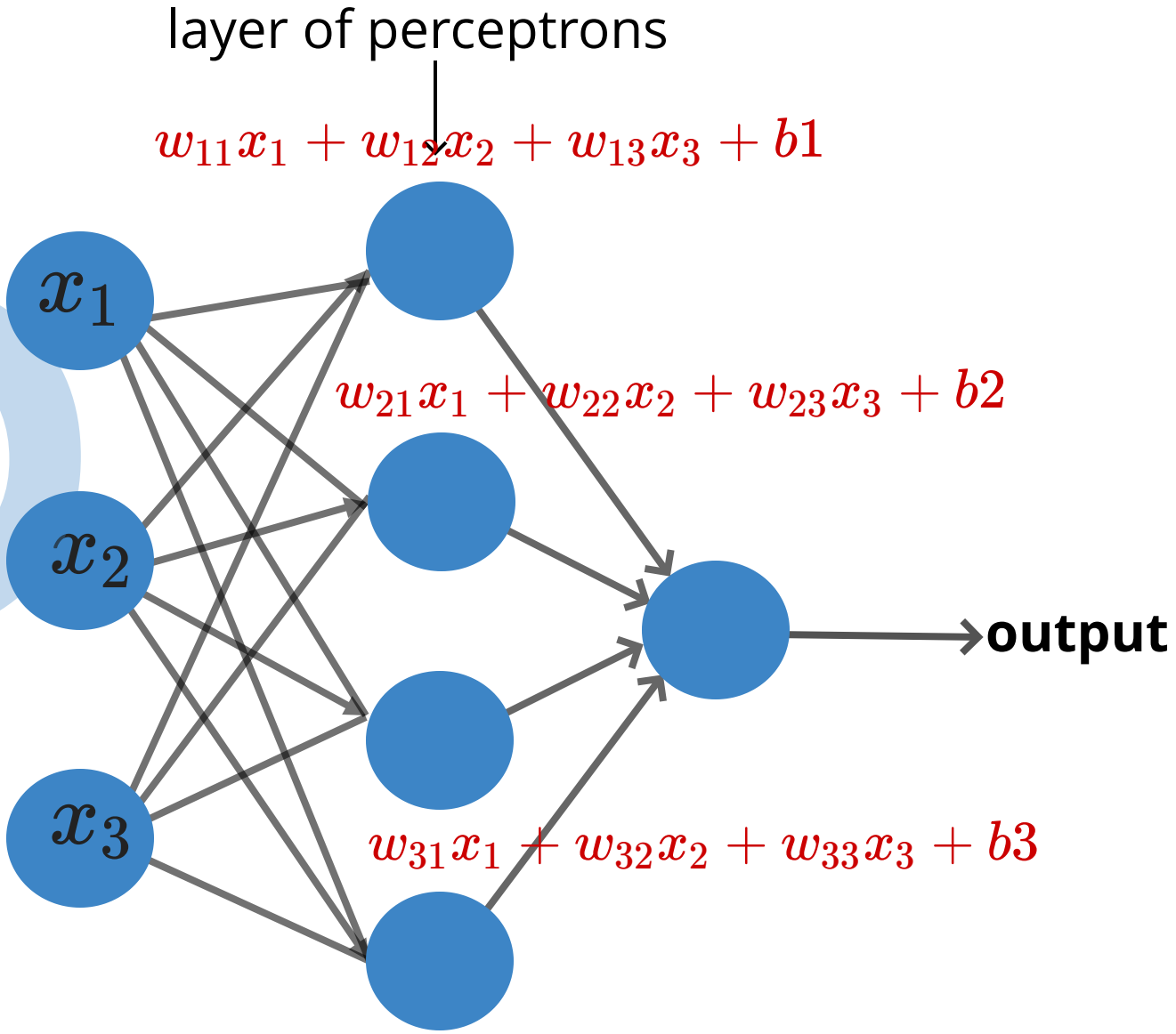
learned parameters

w: weight

sets the sensitivity of a neuron

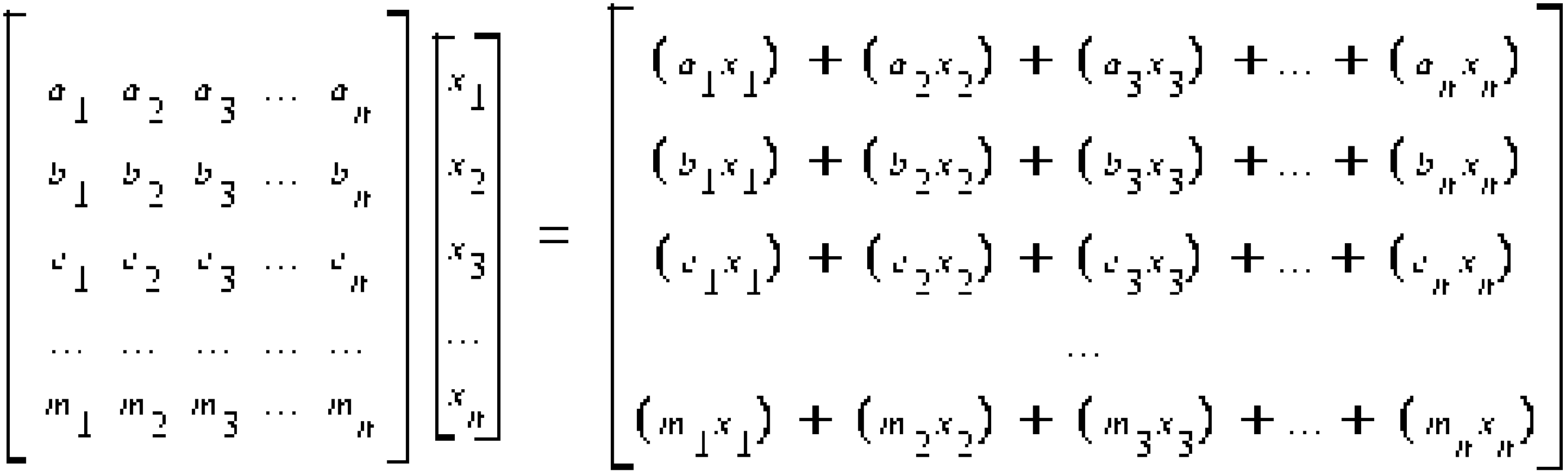
b: bias:

up-down weights a neuron



multilayer perceptron

what we are doing is exactly a series of matrix multiplictions.



multilayer perceptron

Fully connected: all nodes go to all nodes of the next layer.

f: activation function:

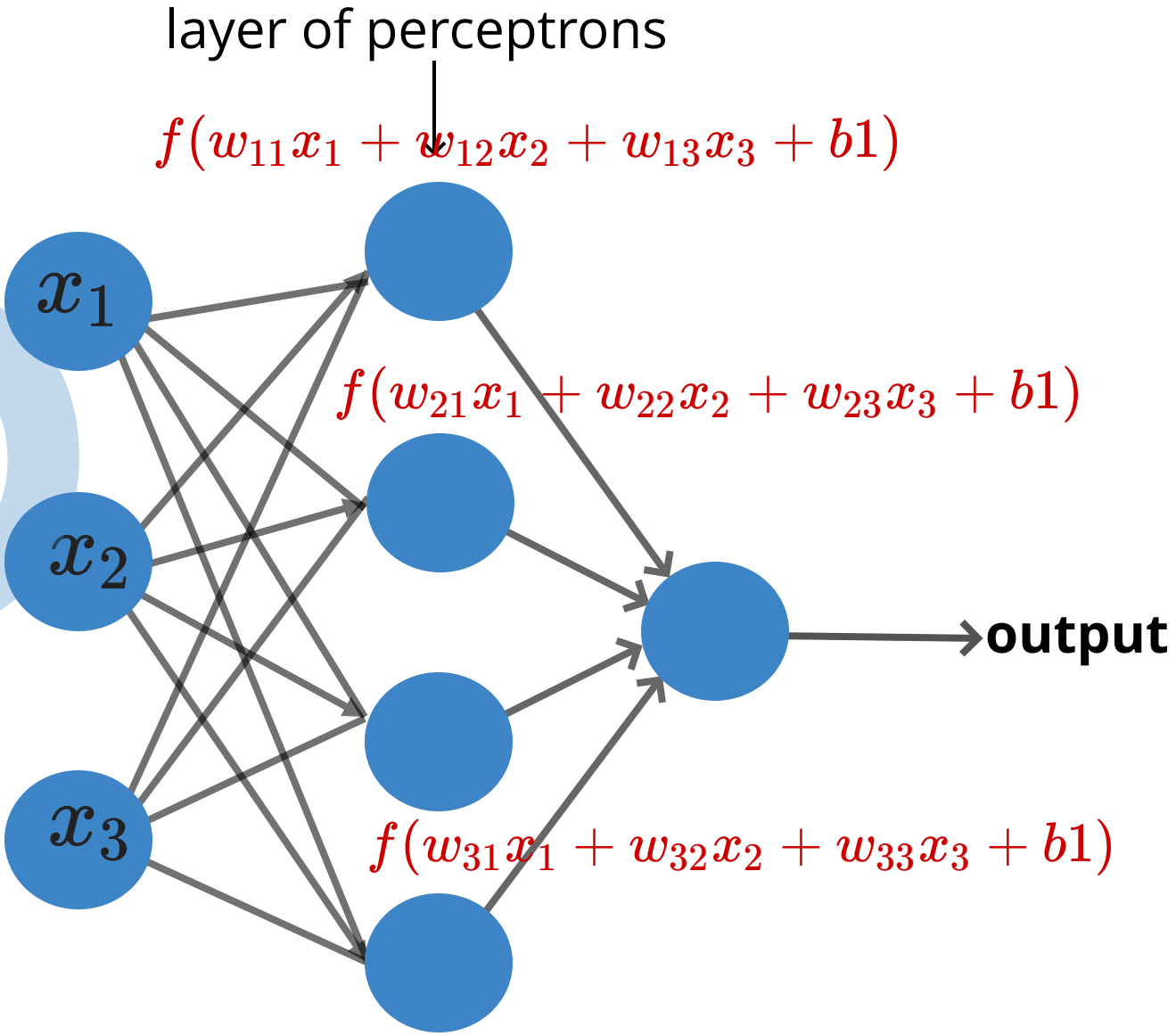
turns neurons on-off

w: weight

sets the sensitivity of a neuron

b: bias:

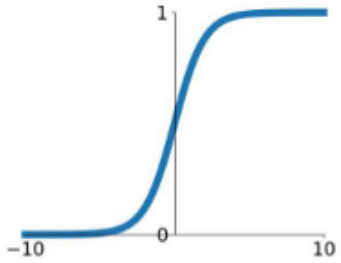
up-down weights a neuron



activation functions

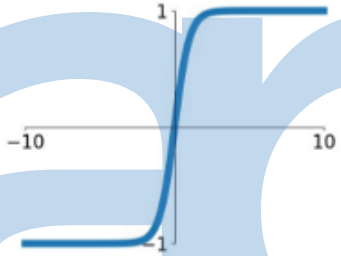
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



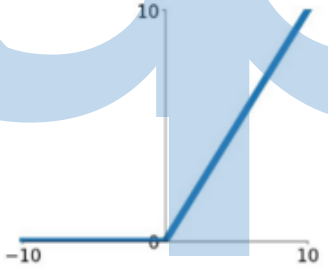
tanh

$$\tanh(x)$$



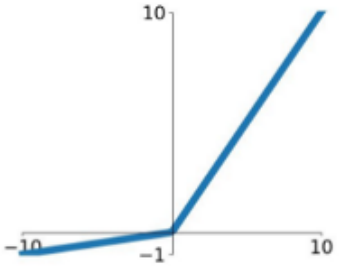
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

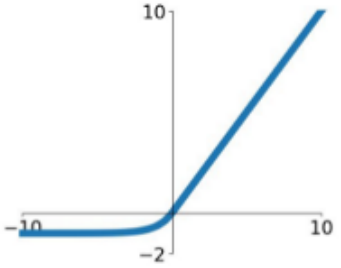


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

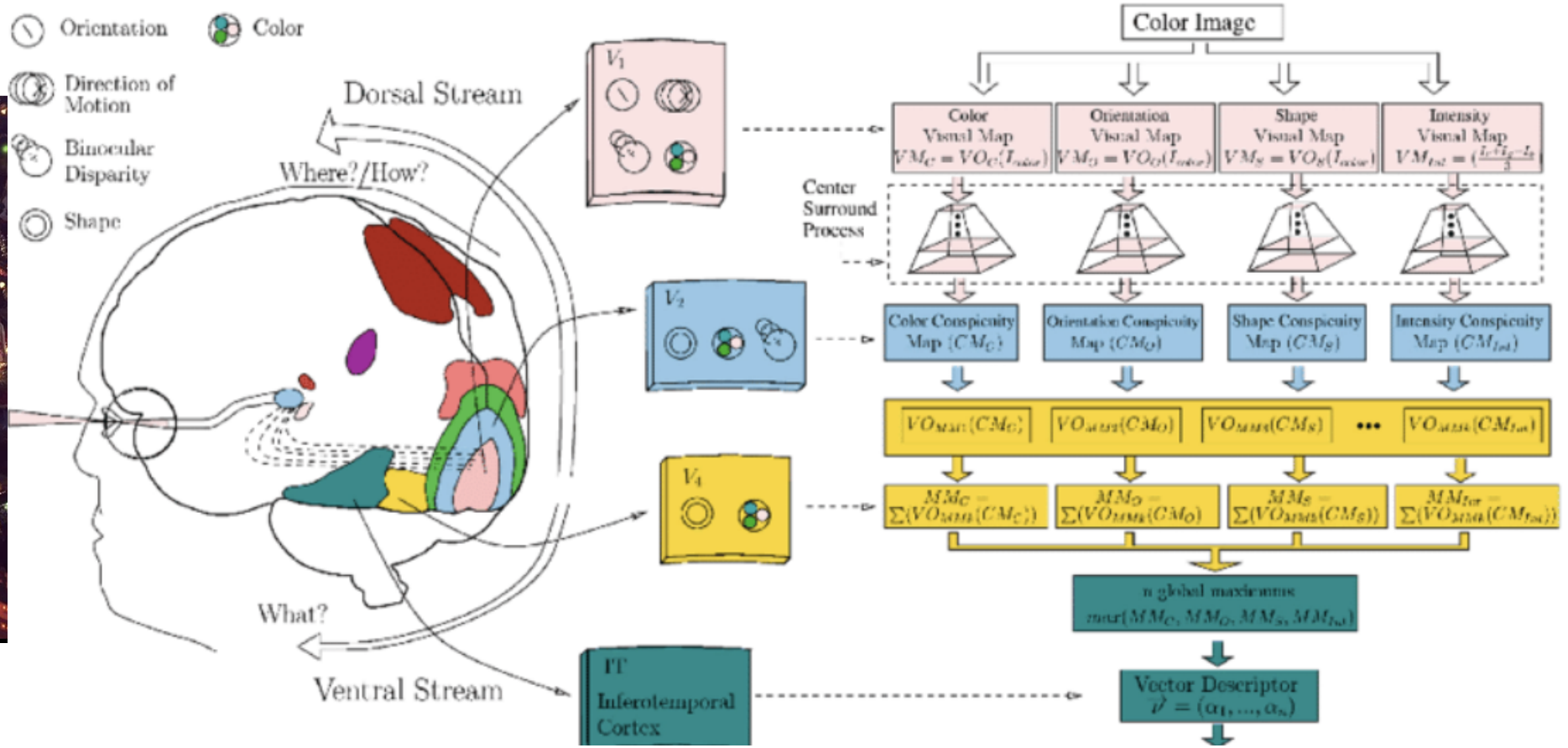


CNN

1

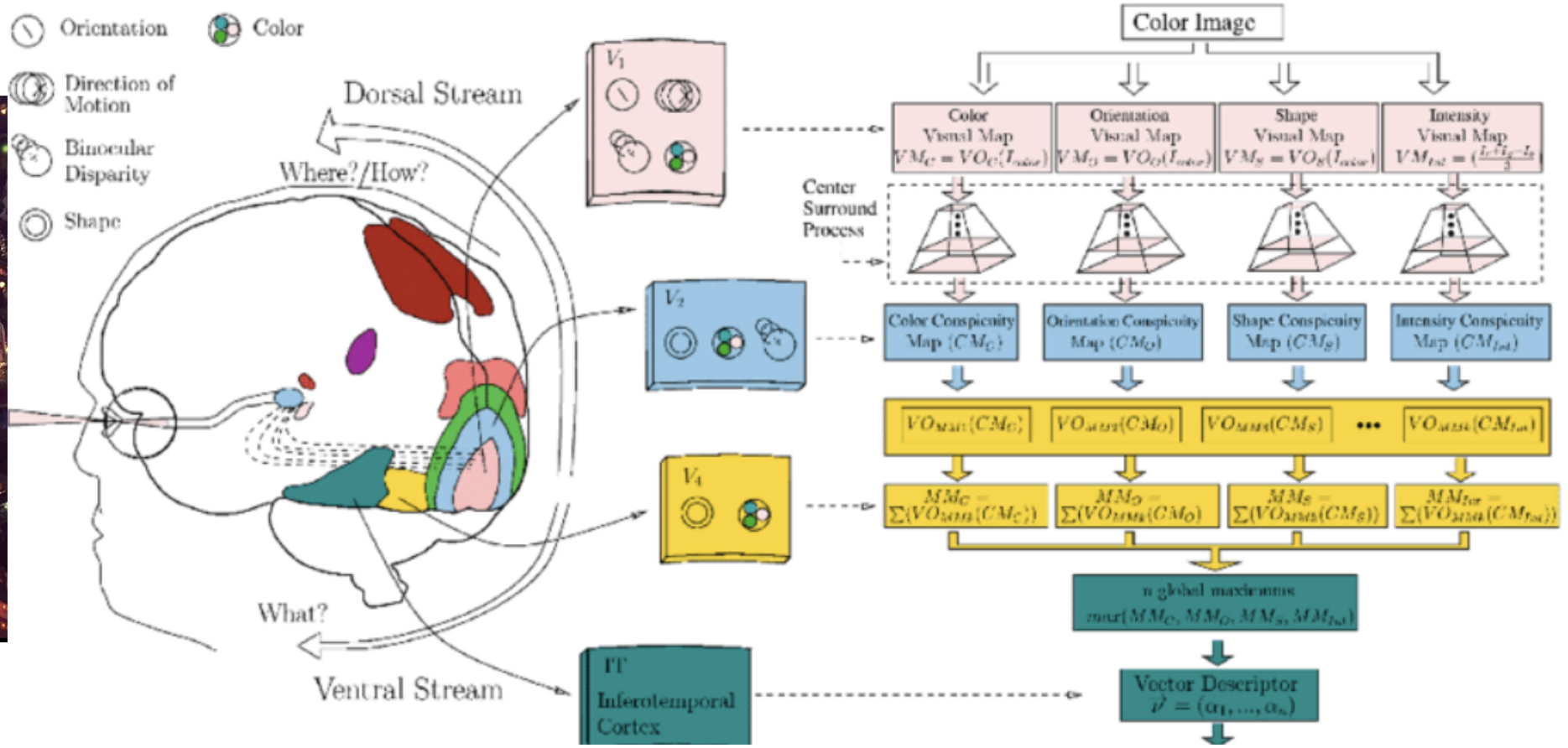
Convolutional Neural Nets

@akumadog



Brain Programming and the Random Search in Object Categorization Olague et al 2017

The visual cortex learns hierarchically: first detects simple features, then more complex features and ensembles of features



Convolution

Convolution

convolution is a mathematical operator on two functions

f and g

that produces a third function

$f \otimes g$

expressing *how the shape of one is modified by the other.*

Convolution Theorem

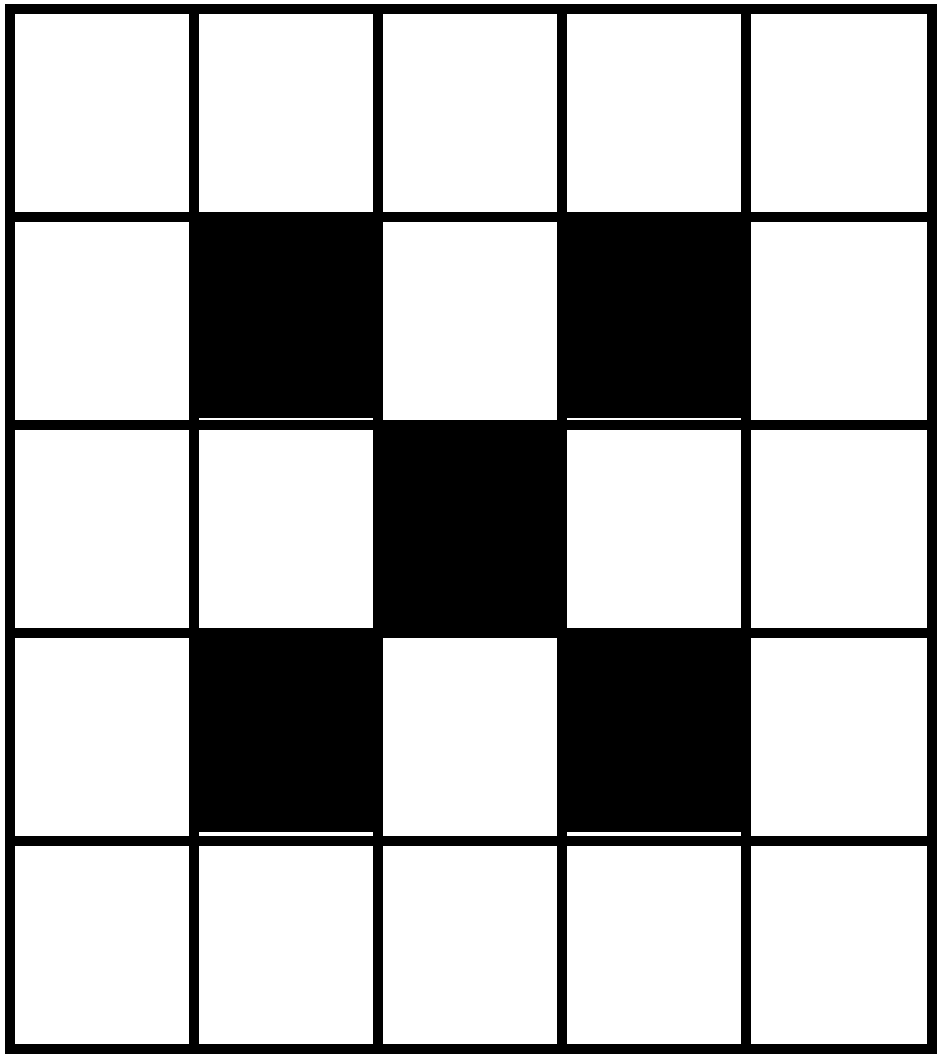
$$F(\nu) = \int_{\mathbb{R}^n} f(x) e^{-2\pi i x \cdot \nu}\, dx,$$

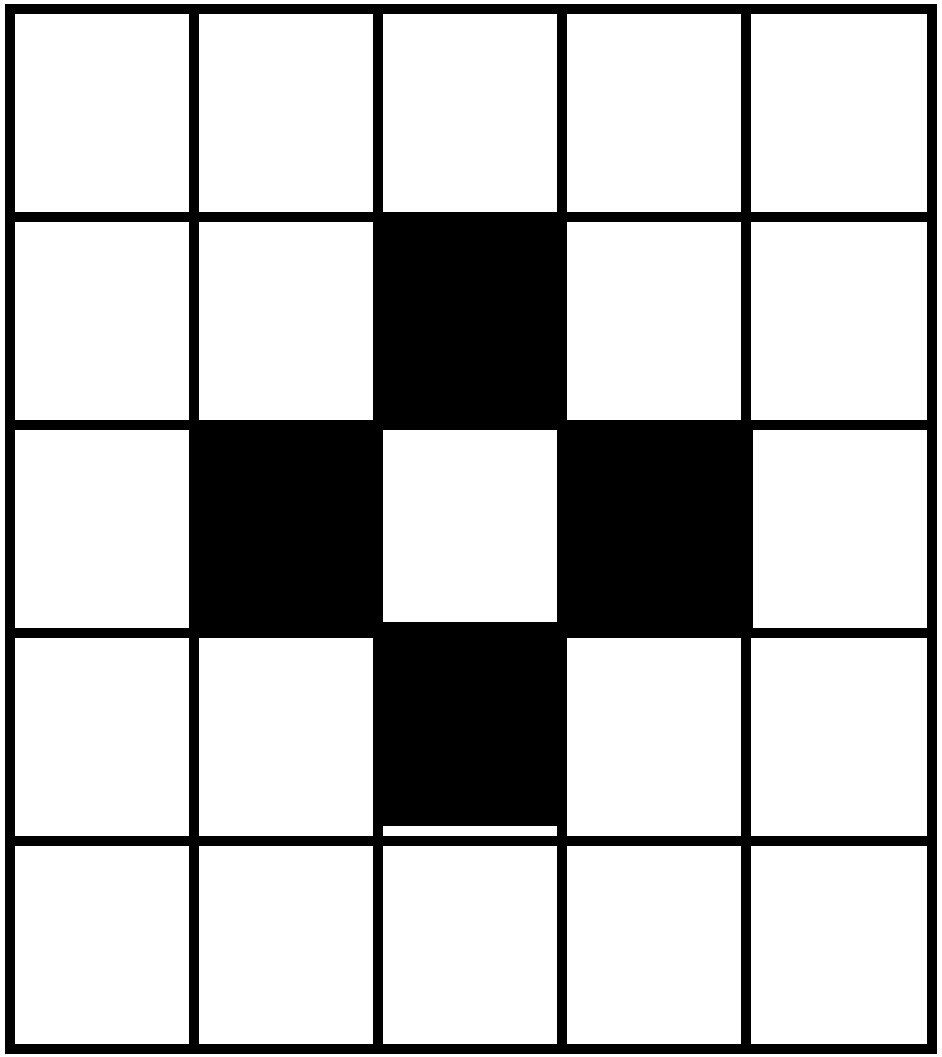
$$G(\nu) = \int_{\mathbb{R}^n} g(x) e^{-2\pi i x \cdot \nu}\, dx,$$

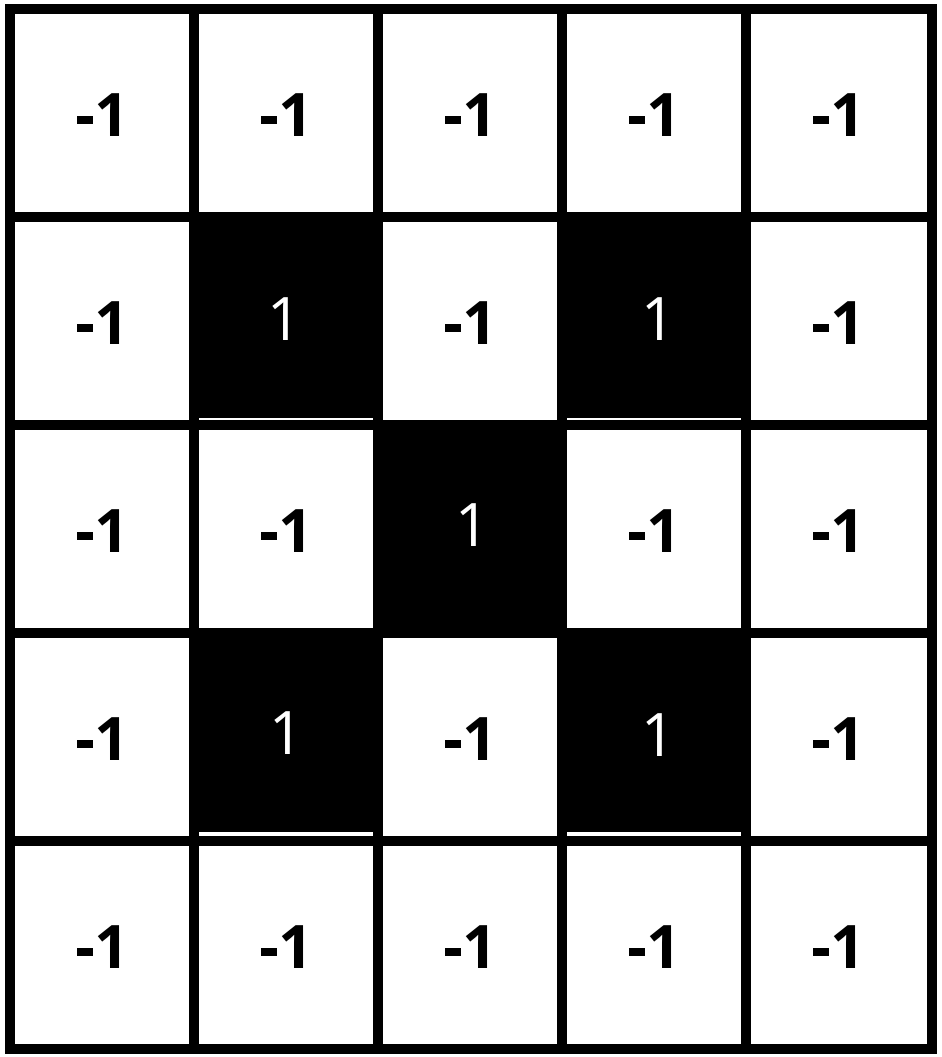
$$f * g = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\}$$

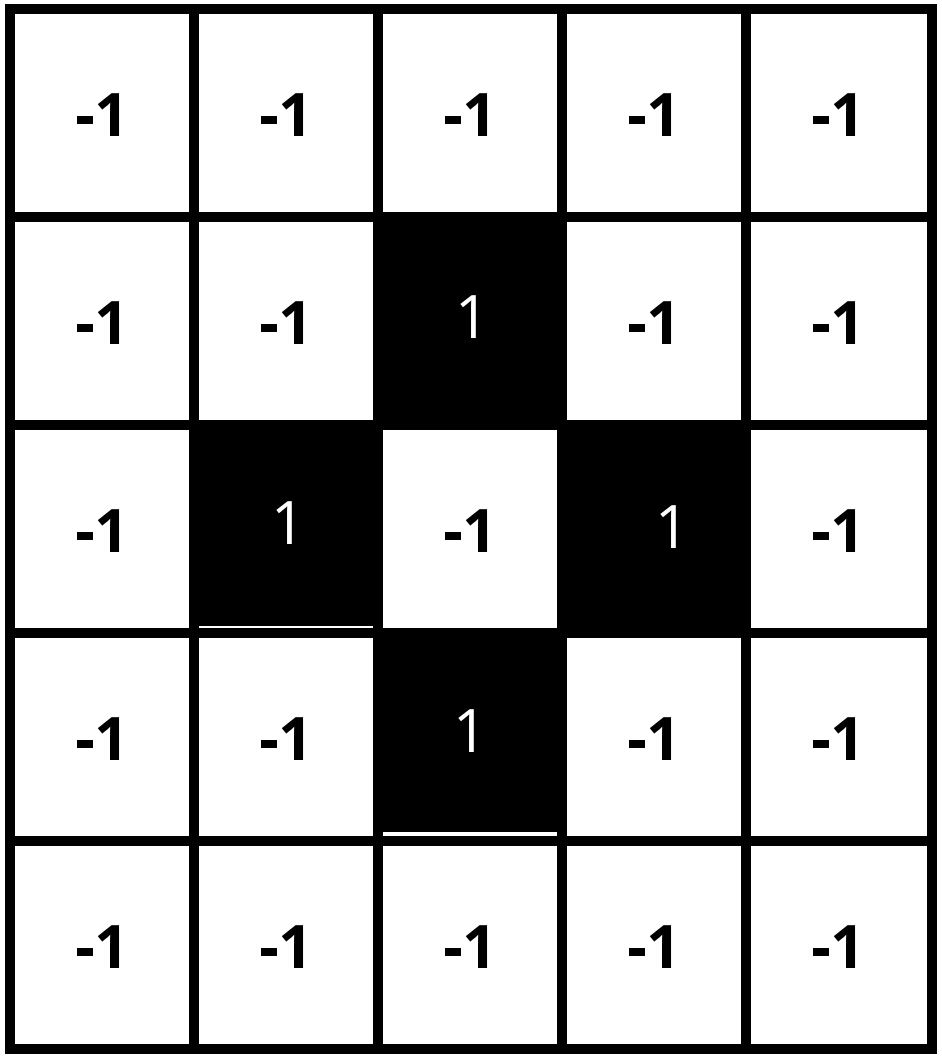
$\mathcal{F}$ fourier transform

two images.






-1 -1 -1 -1 -1
-1 1 -1 1 -1
-1 -1 1 -1 -1
-1 1 -1 1 -1
-1 -1 -1 -1 -1

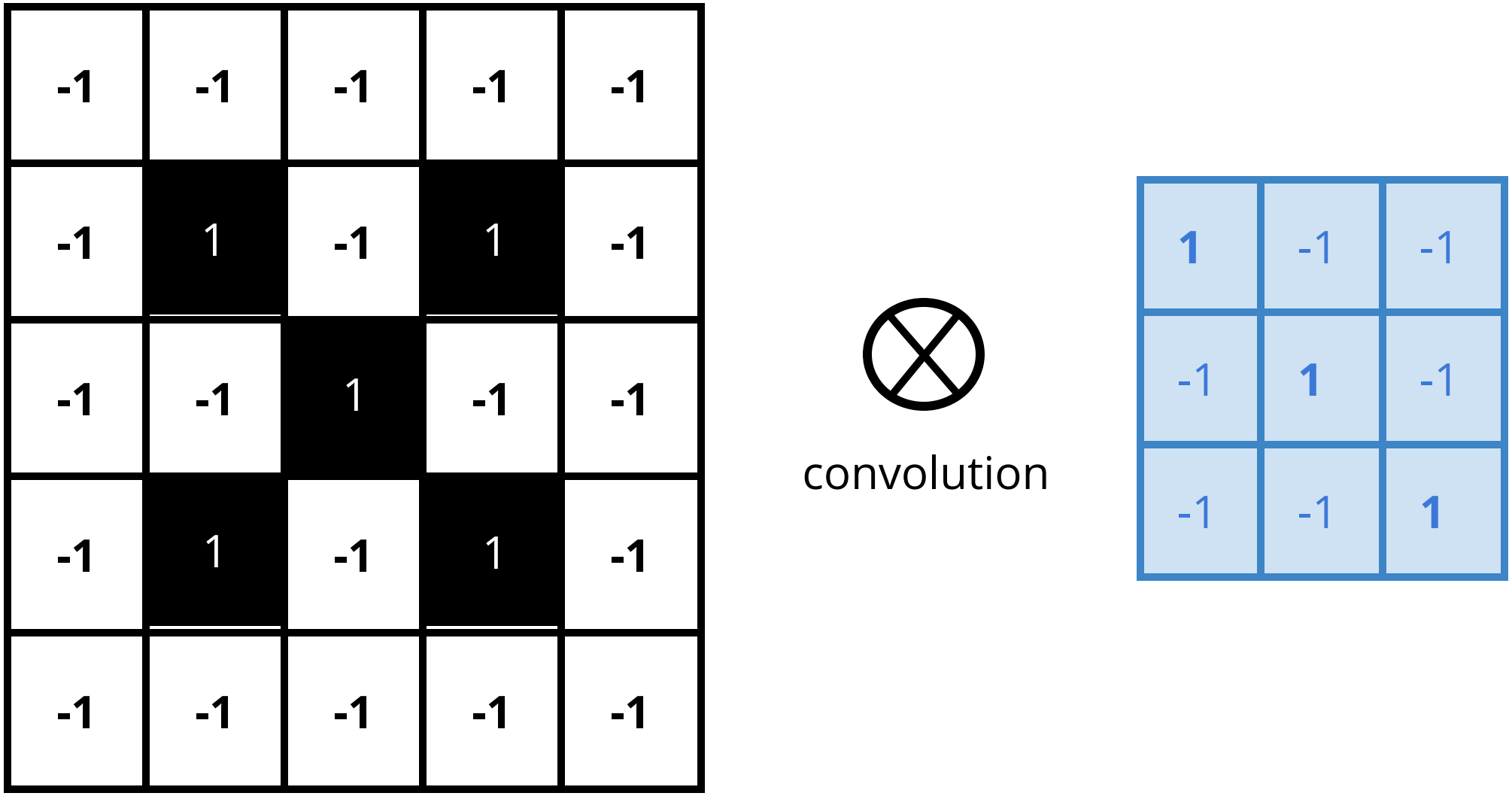

-1 -1 -1 -1 -1
-1 -1 1 -1 -1
-1 1 -1 1 -1
-1 -1 1 -1 -1
-1 -1 -1 -1 -1

-1	-1	-1	-1	-1
-1	1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	1	-1
-1	-1	-1	-1	-1

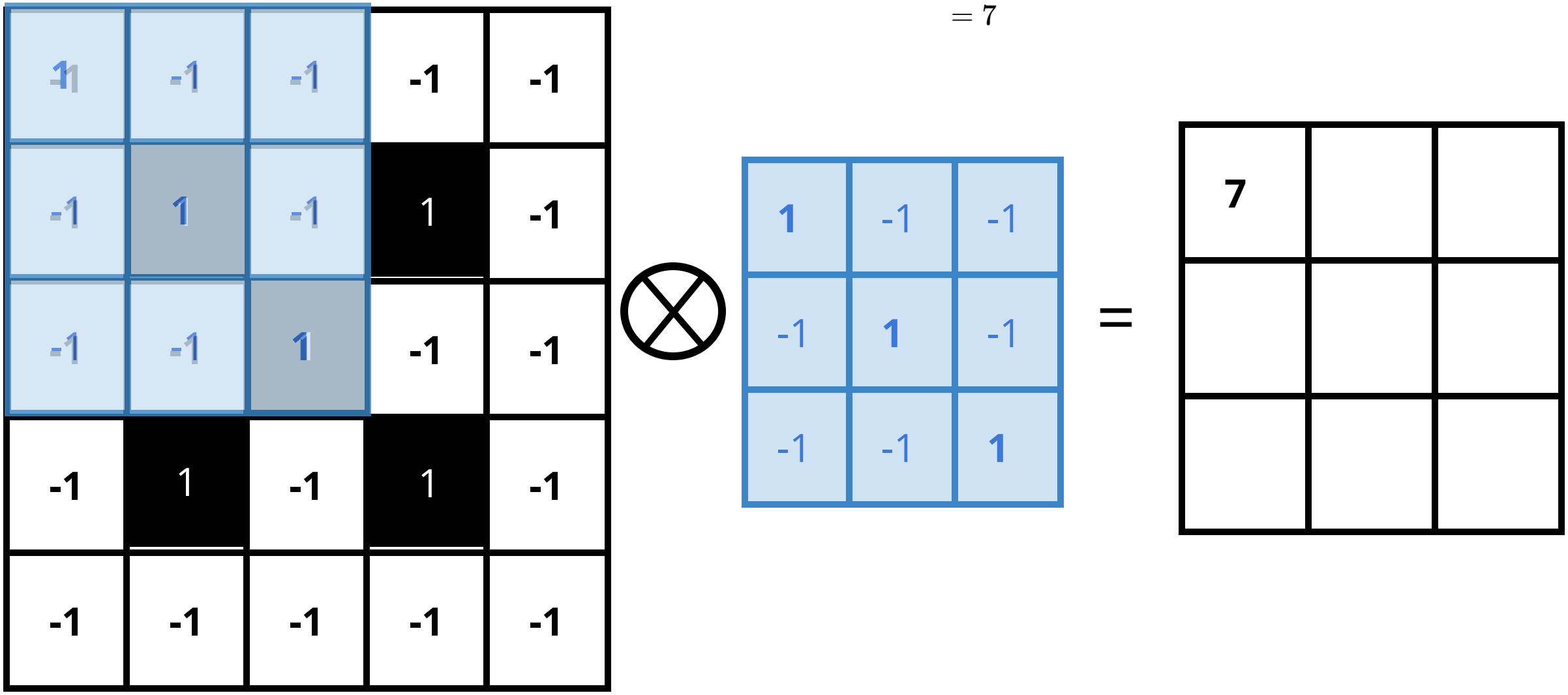
feature maps

1	-1	-1
-1	1	-1
-1	-1	1

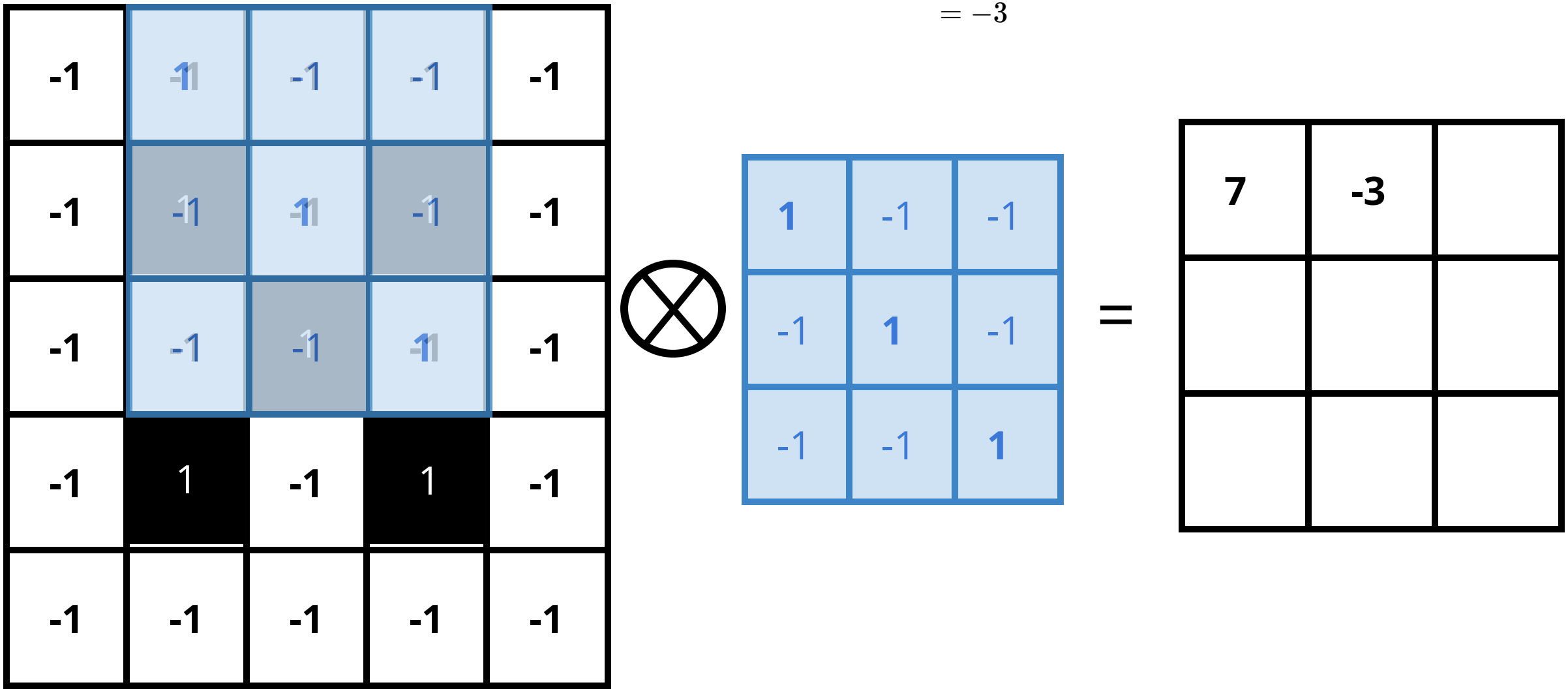
-1	-1	1
-1	1	-1
1	-1	-1

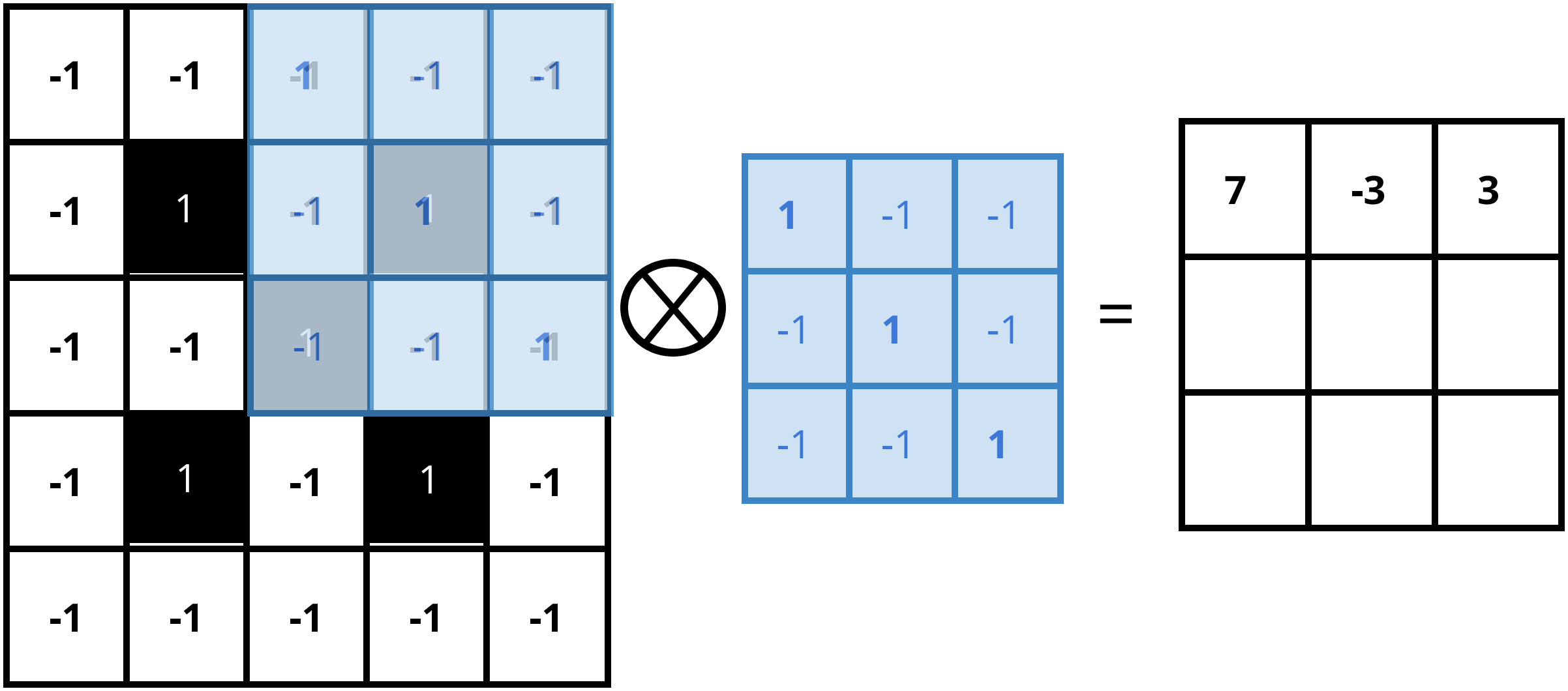

-1 -1 -1 -1 -1
-1 1 -1 1 -1
-1 -1 1 -1 -1
-1 1 -1 1 -1
-1 -1 -1 -1 -1
convolution
1 -1 -1
-1 1 -1
-1 -1 1

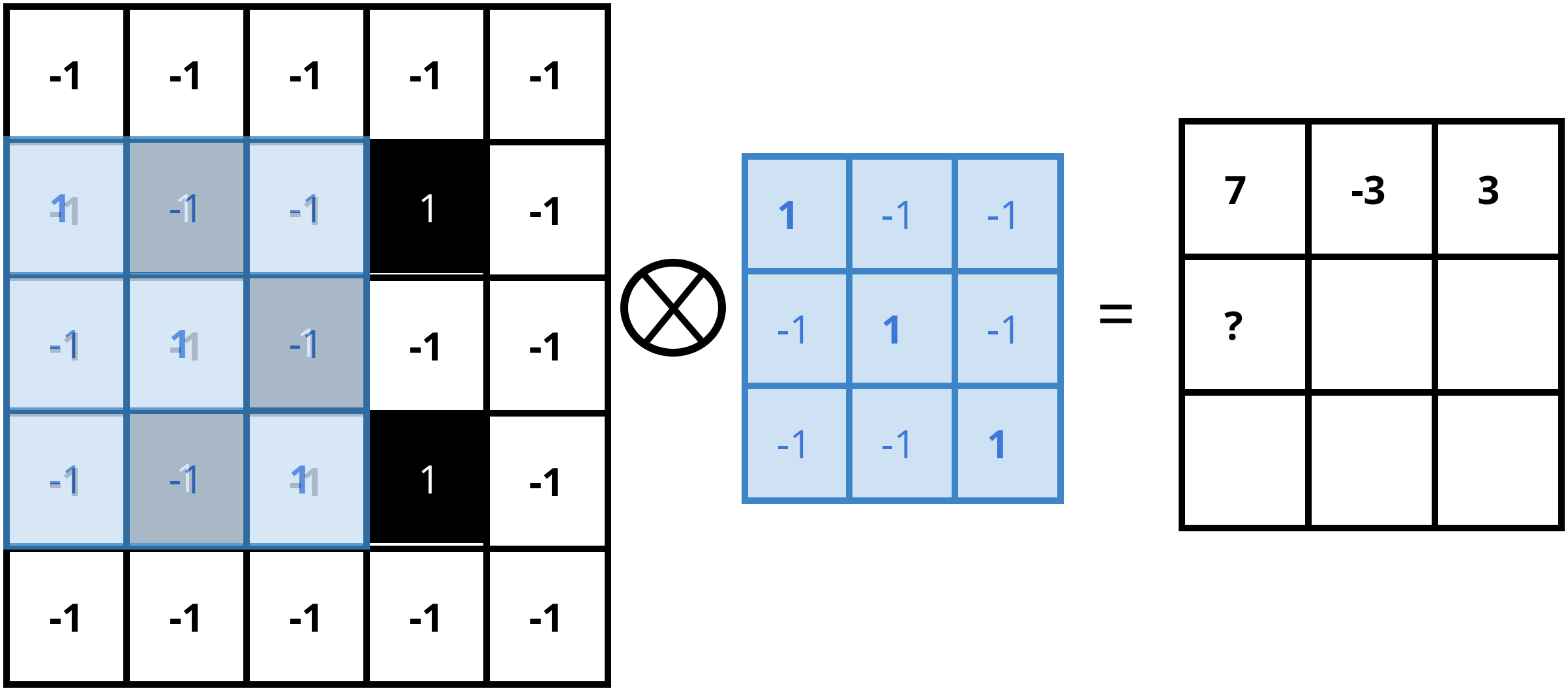
$$(-1 * 1) + (-1 * -1) + (-1 * -1) +$$
$$(-1 * -1) + (1 * 1) + (-1 * -1)$$
$$(-1 * -1) + (-1 * -1) + (1 * 1)$$
$$= 7$$

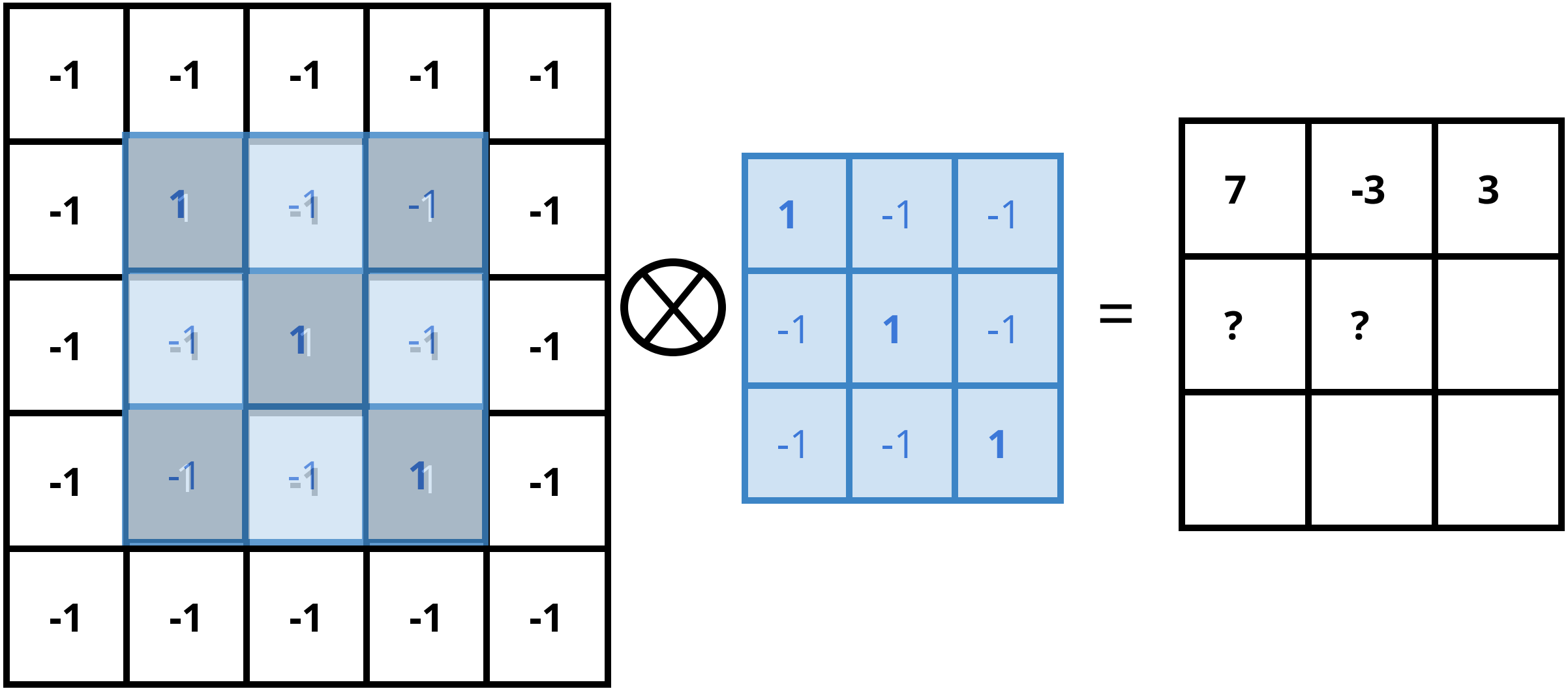


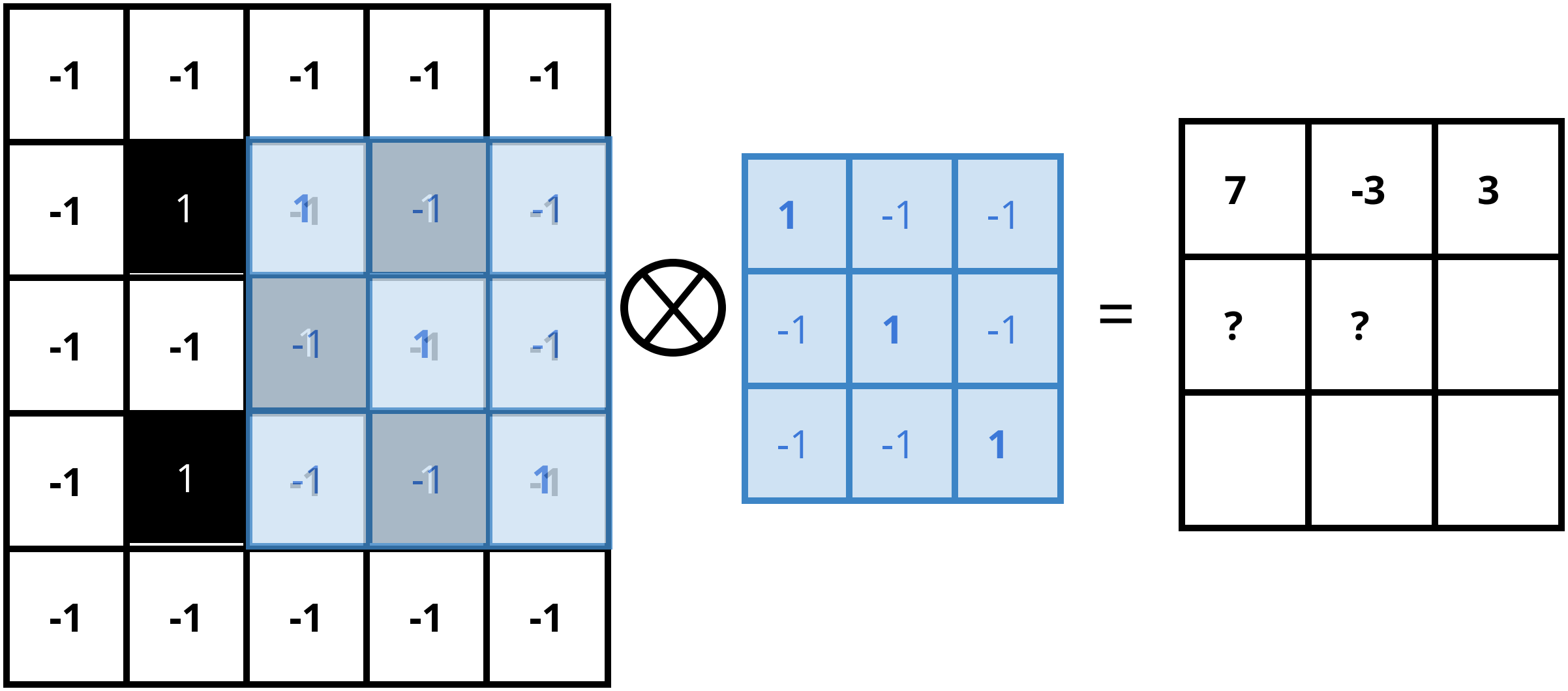
$$(-1 * 1) + (-1 * -1) + (-1 * -1) +$$
$$(-1 * 1) + (-1 * 1) + (-1 * 1)$$
$$(-1 * -1) + (-1 * 1) + (-1 * 1)$$
$$= -3$$

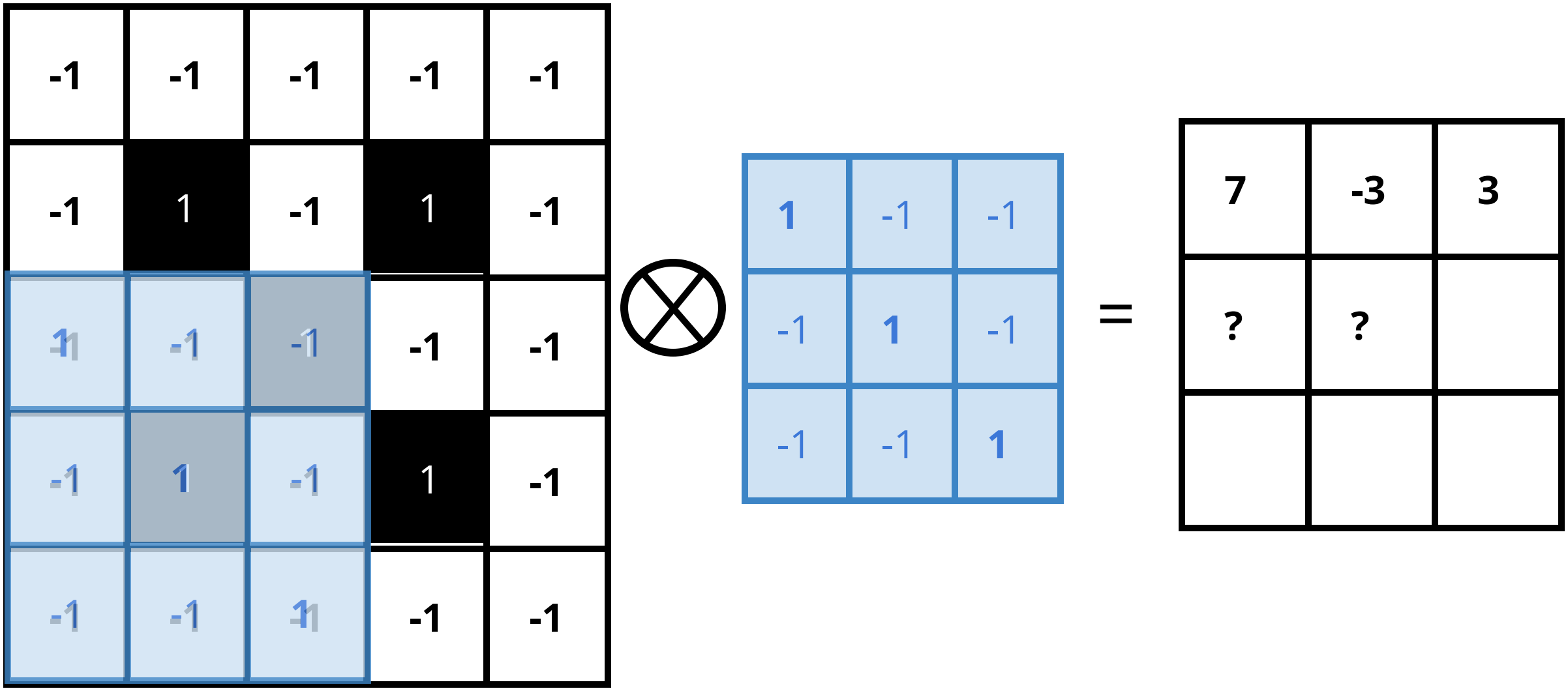


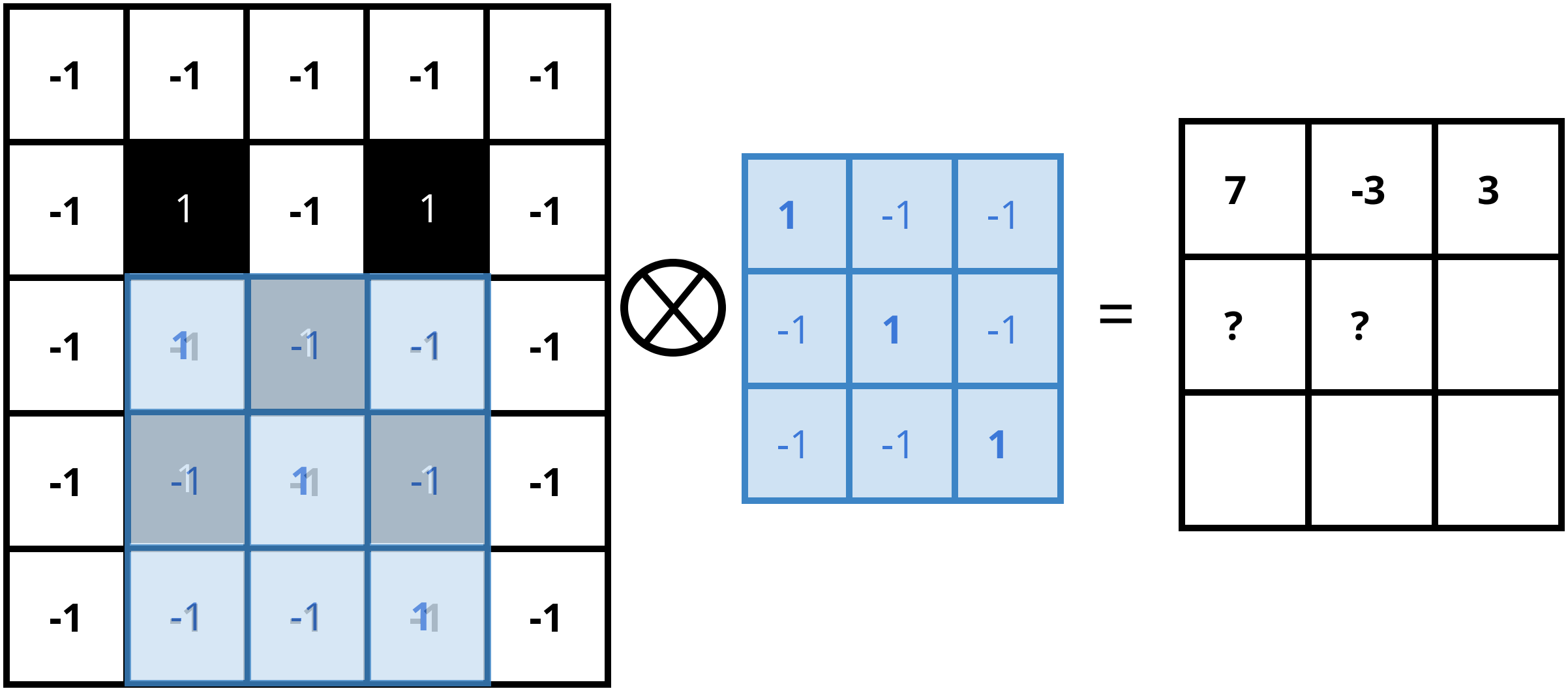

-1 -1 -1 -1 -1
-1 1 -1 1 -1
-1 -1 -1 -1 -1
-1 1 -1 1 -1
-1 -1 -1 -1 -1
1 -1 -1
-1 1 -1
-1 -1 1
7 -3 3

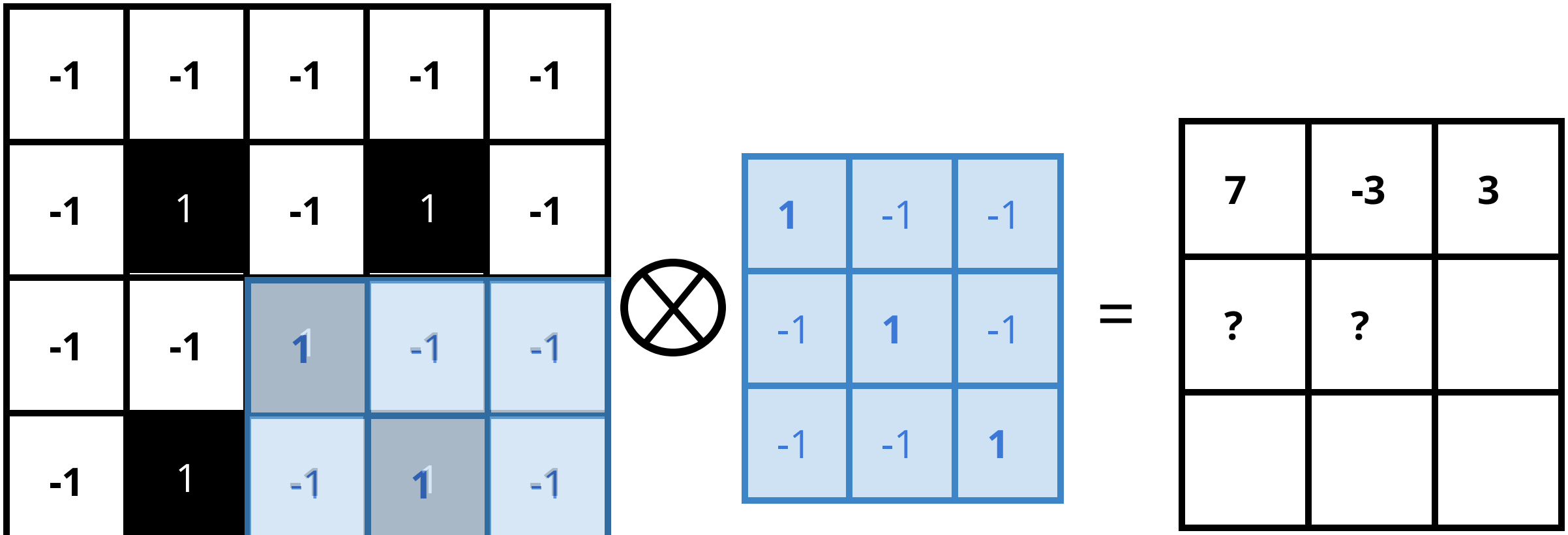

-1 -1 -1 -1 -1
-1 -1 -1 1 -1
-1 -1 -1 -1 -1
-1 -1 -1 1 -1
-1 -1 -1 -1 -1
1 -1 -1
-1 1 -1
-1 -1 1
7 -3 3
?

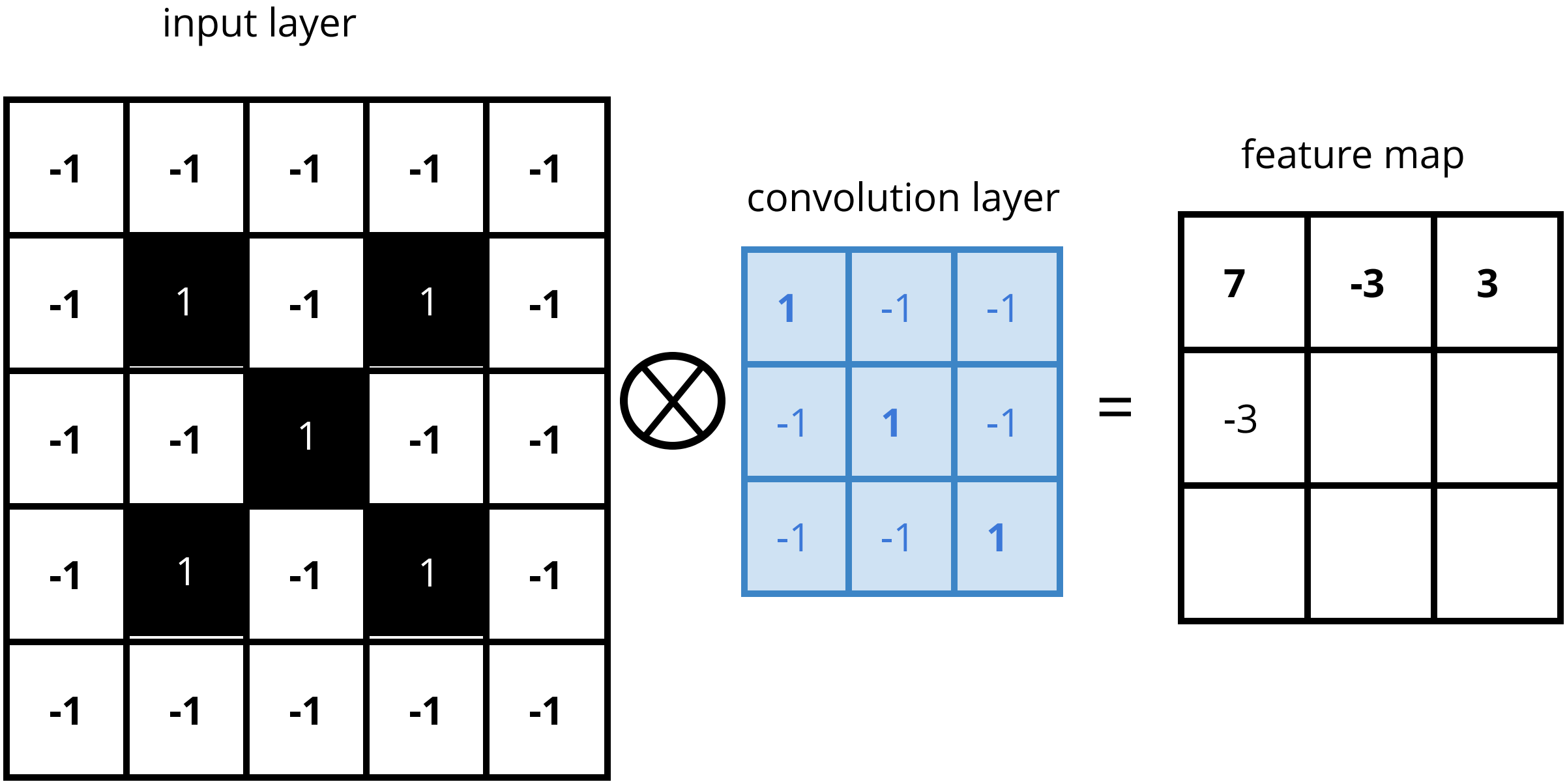

-1 -1 -1 -1 -1
-1 1 -1 -1 -1
-1 -1 1 -1 -1
-1 -1 -1 1 -1
-1 -1 -1 -1 -1
1 -1 -1
-1 1 -1
-1 -1 1
7 -3 3
? ?


-1 -1 -1 -1 -1
-1 1 1 -1 -1
-1 -1 -1 1 -1
-1 1 -1 -1 1
-1 -1 -1 -1 -1
1 -1 -1
-1 1 -1
-1 -1 1
7 -3 3
? ?


-1 -1 -1 -1 -1
-1 1 -1 1 -1
-1 -1 -1 -1 -1
-1 1 -1 1 -1
-1 -1 -1 -1 -1
1 -1 -1
-1 1 -1
-1 -1 1
7 -3 3
? ?


-1 -1 -1 -1 -1
-1 1 -1 1 -1
-1 1 1 -1 -1 -1
-1 -1 1 1 -1 -1
-1 -1 -1 -1 -1 -1
-1 -1 -1 1 -1
1 -1 -1
-1 1 -1
-1 -1 1
=
7 -3 3
? ?


-1 -1 -1 -1 -1
-1 1 -1 1 -1
-1 -1 1 -1 -1
-1 1 -1 1 -1
-1 -1 -1 -1 1
1 -1 -1
-1 1 -1
-1 -1 1
7 -3 3
? ?


input layer
-1 -1 -1 -1 -1
-1 1 -1 1 -1
-1 -1 1 -1 -1
-1 1 -1 1 -1
-1 -1 -1 -1 -1
convolution layer
1 -1 -1
-1 1 -1
-1 -1 1
=
feature map
7 -3 3
-3

input layer

-1	-1	-1	-1	-1
-1	1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	1	-1
-1	-1	-1	-1	-1

⊗

convolution layer

1	-1	-1
-1	**1**	-1
-1	-1	**1**

=

feature map

7	-3	3
-3	5	-3
3	-3	7

the feature map is "richer": we went from binary to R

input layer

-1	-1	-1	-1	-1
-1	1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	1	-1
-1	-1	-1	-1	-1

⊗

convolution layer

1	-1	-1
-1	**1**	-1
-1	-1	**1**

=

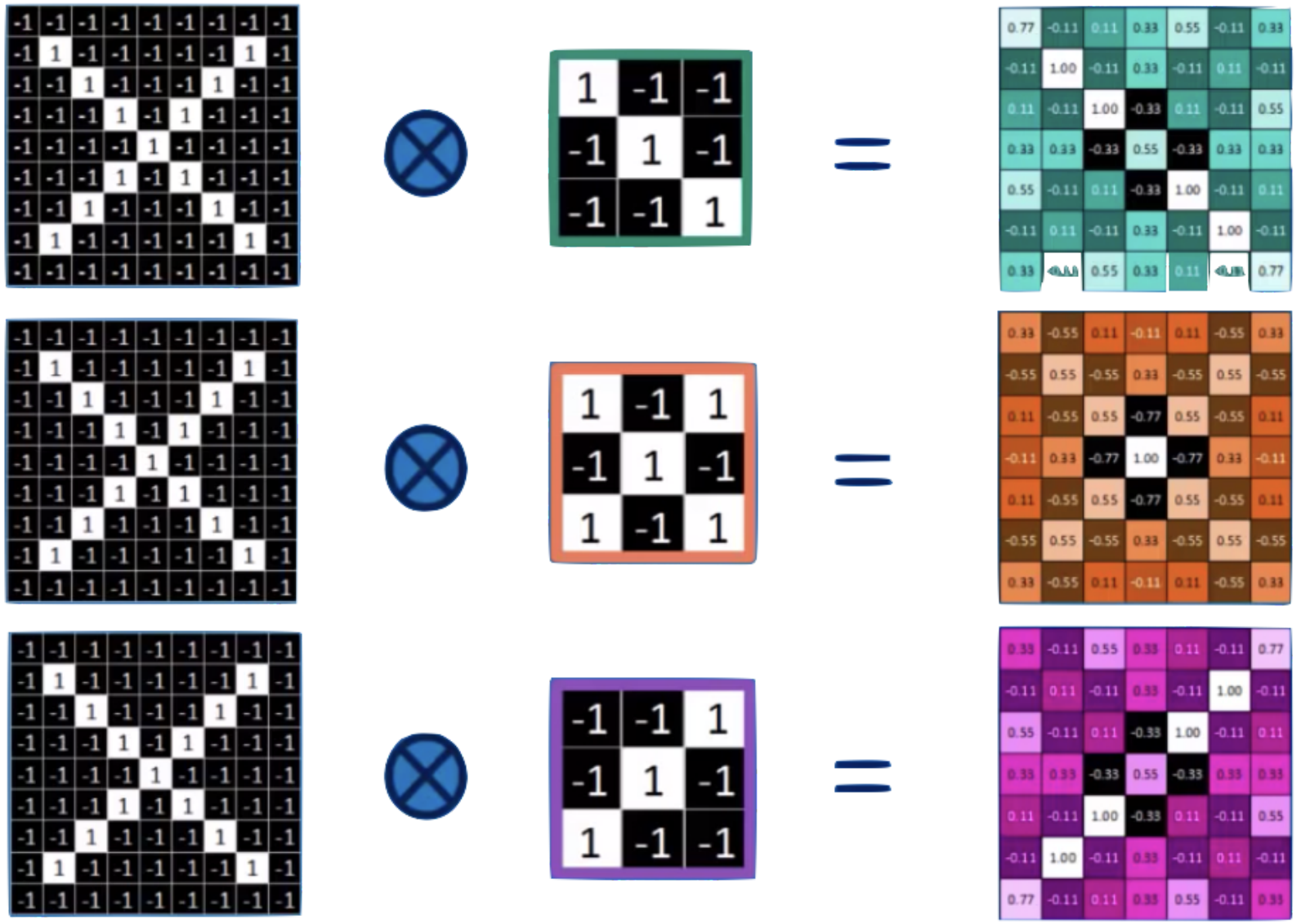
feature map

7	-3	3
-3	5	-3
3	-3	7

the feature map is "richer": we went from binary to R

and it is reminiscent of the original layer

Convolve with different feature: each neuron is 1 feature



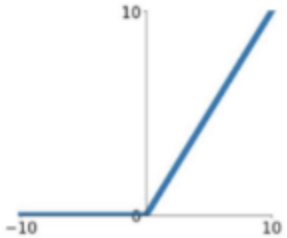
ReLu

ReLu: normalization that replaces negative values with 0's

7	-3	3
-5	5	-3
-6	-1	7

ReLU

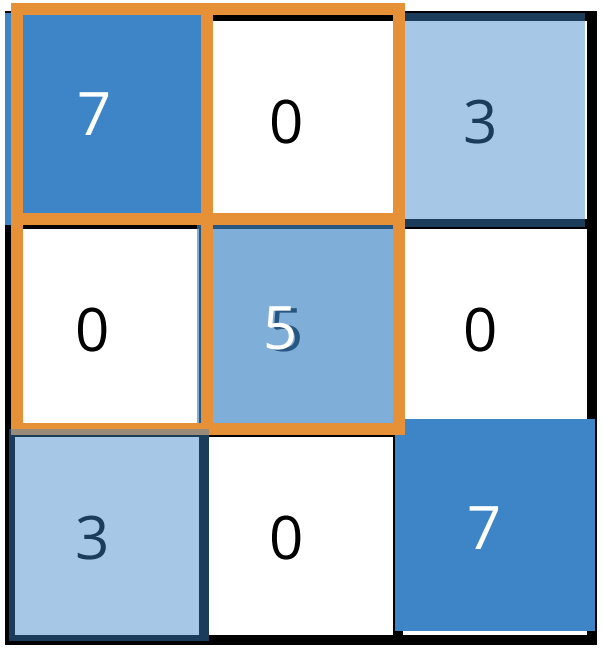
$$\max(0, x)$$

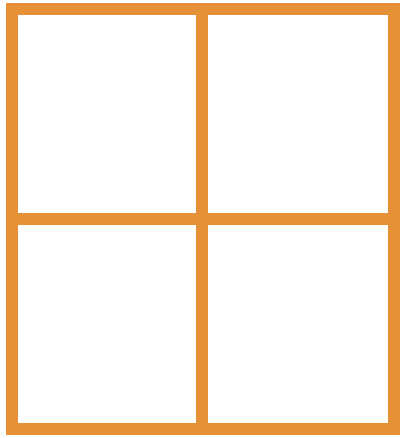


7	0	3
0	5	0
3	0	7

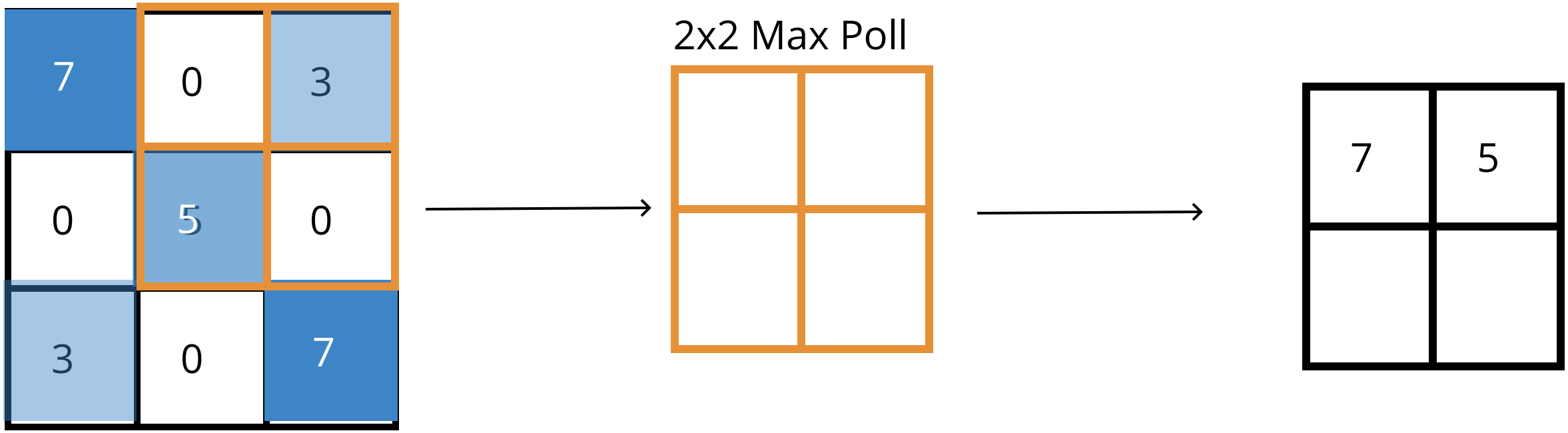
Max-Pool

MaxPooling: reduce image size, generalizes result

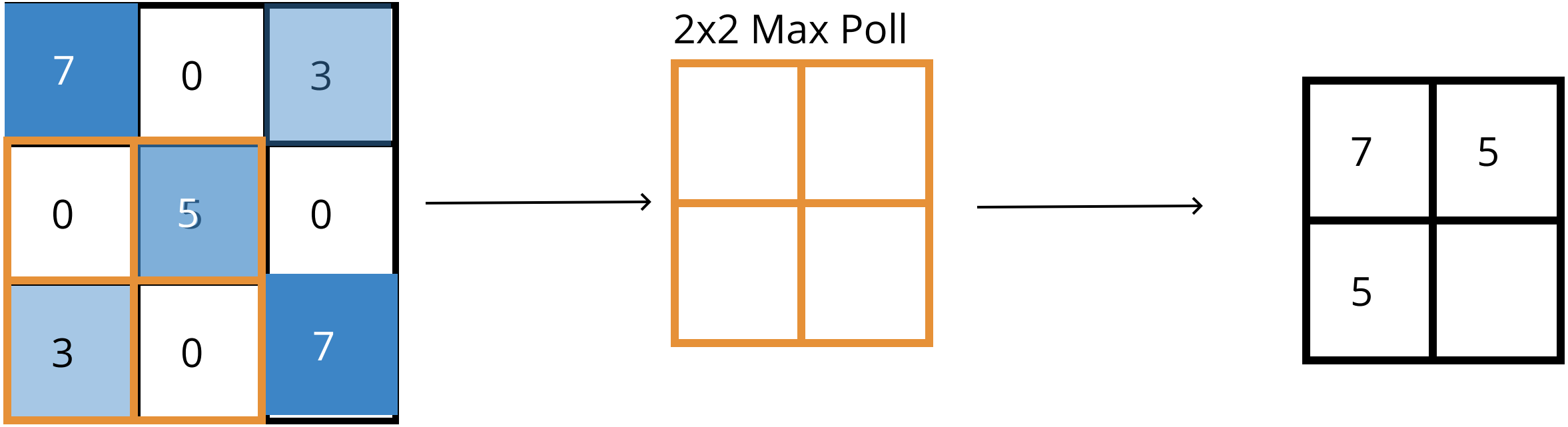




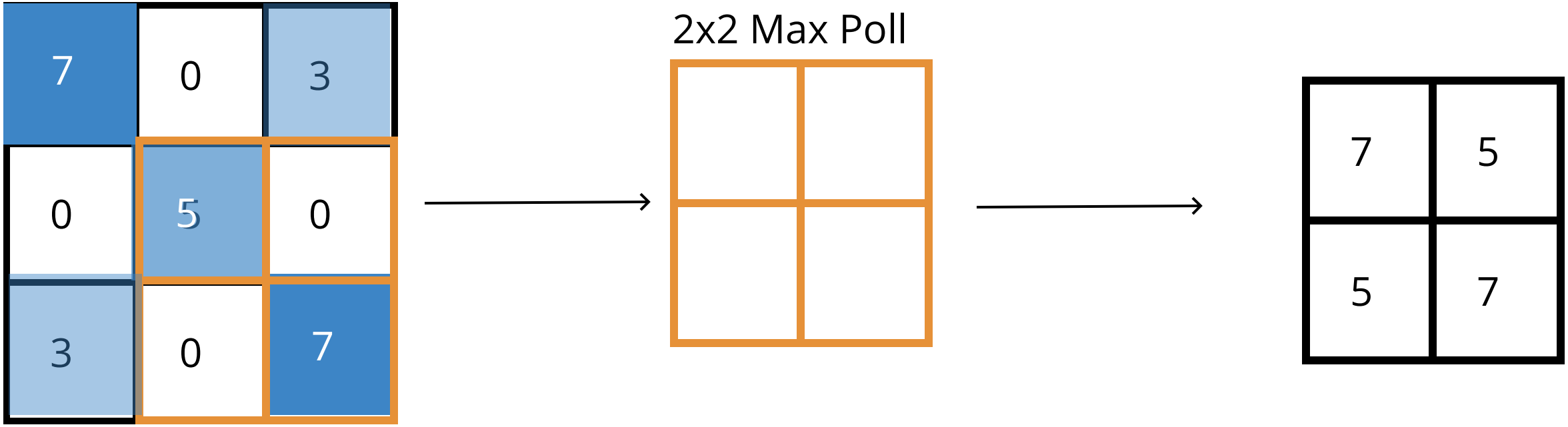
MaxPooling: reduce image size, generalizes result

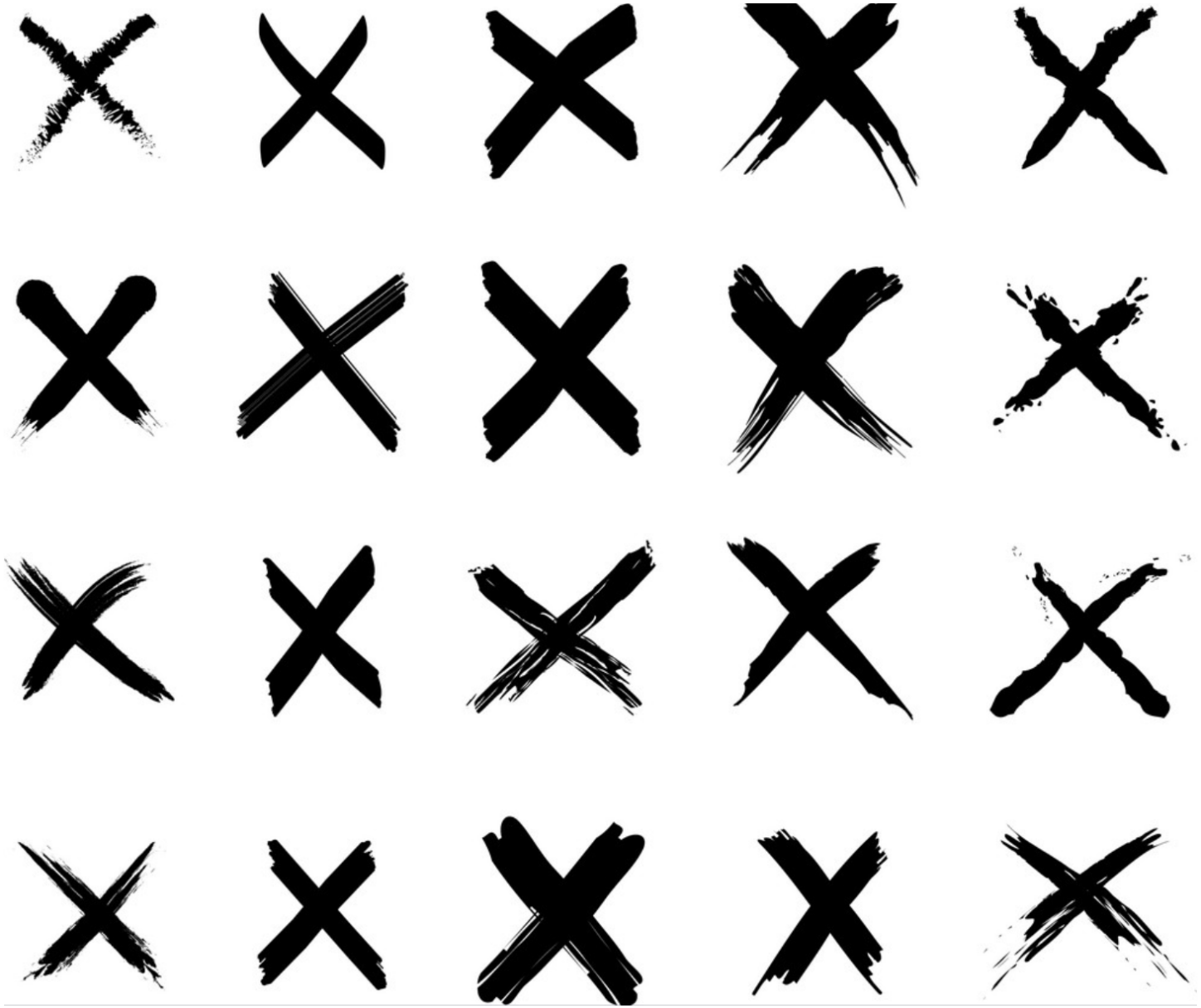


MaxPooling: reduce image size, generalizes result



MaxPooling: reduce image size, generalizes result

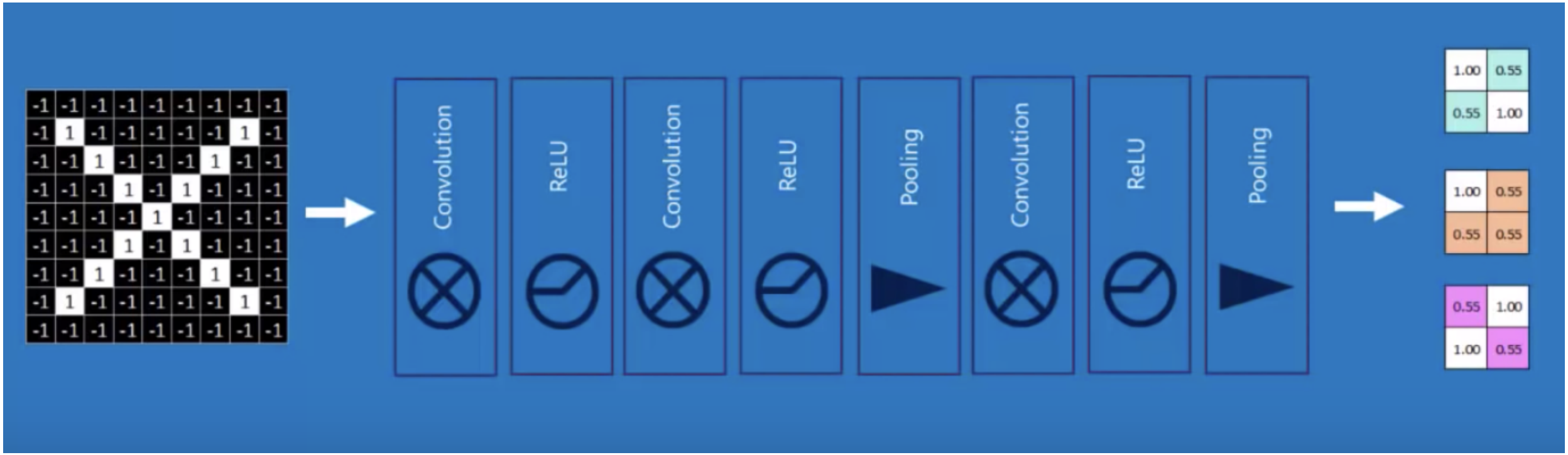




MaxPooling: reduce image size *& generalizes result*

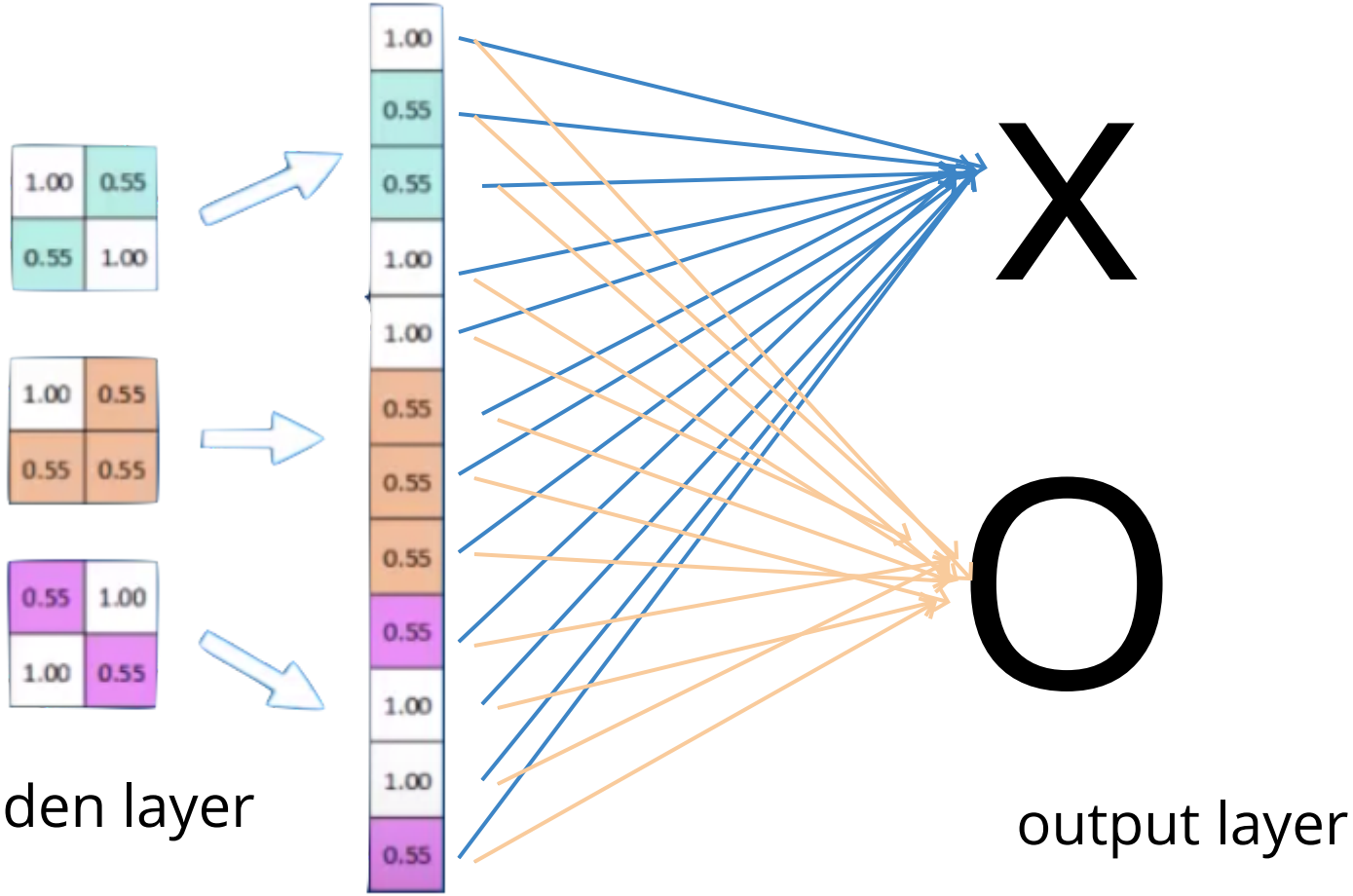
By reducing the size and picking the maximum of a sub-region we make the network less sensitive to specific details

CNN


-1 -1 -1 -1 -1 -1 -1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 -1 -1 1 -1 -1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1
Convolution
ReLU
Convolution
ReLU
Pooling
Convolution
ReLU
Pooling
1.00 0.55
0.55 1.00
1.00 0.55
0.55 0.55
0.55 1.00
1.00 0.55

final layer:

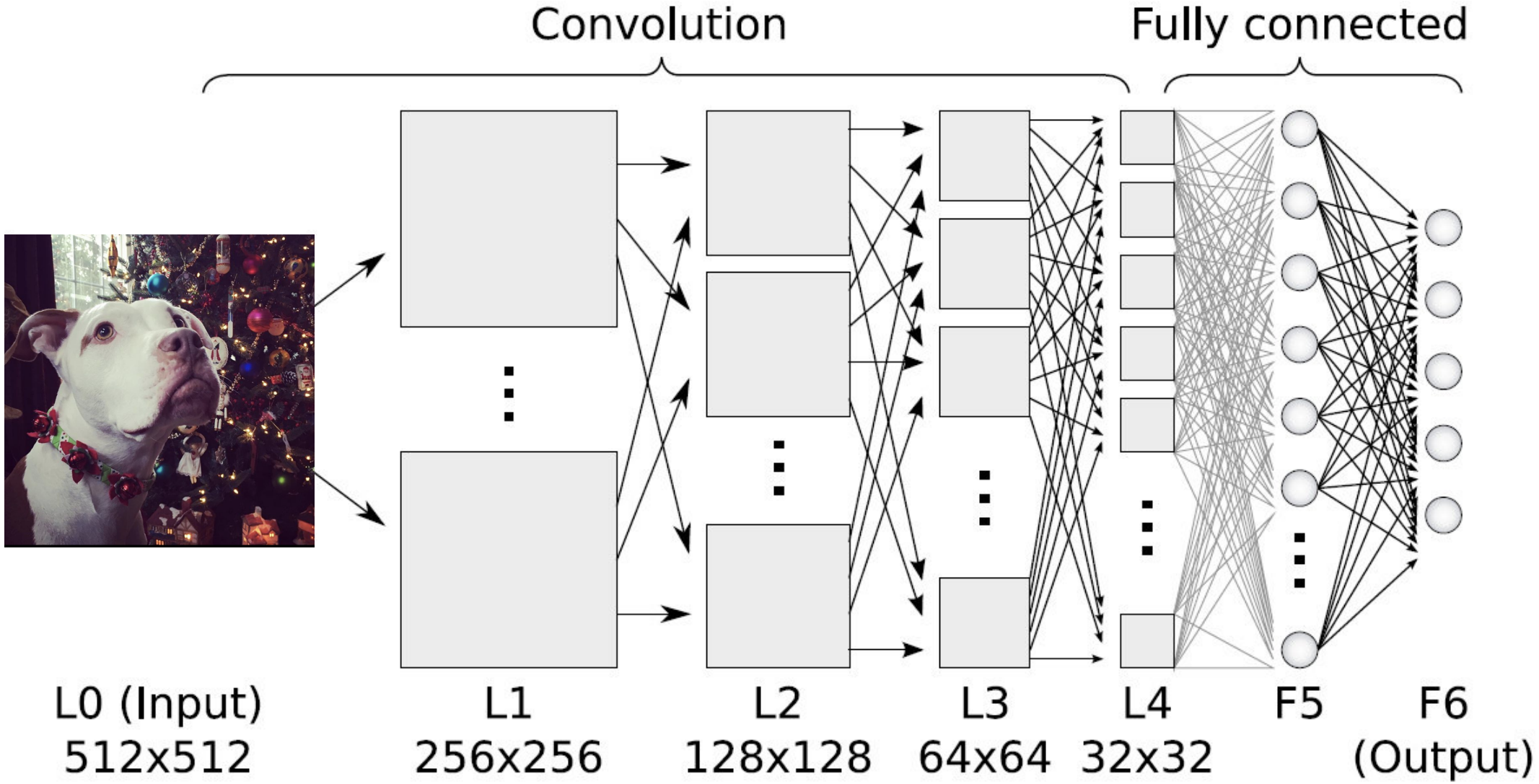
the final layer is fully connected



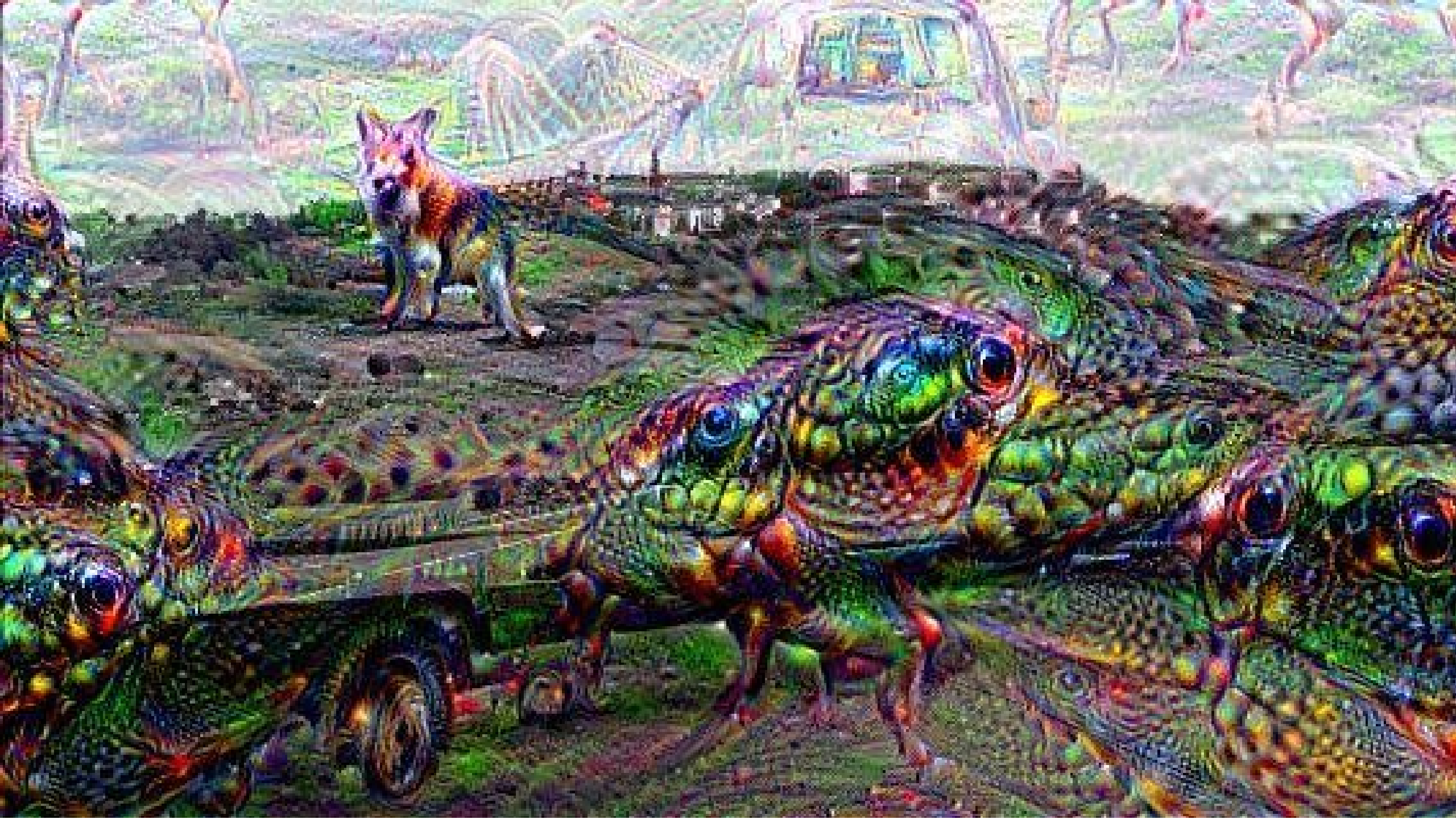
last hidden layer

output layer

Stack multiple convolution layers




deep dreams



what is happening in DeepDream?

Deep Dream (DD) is a google software, a pre-trained NN (originally created on the Cafe architecture, now imported on many other platforms including tensorflow).

The high level idea relies on training a convolutional NN to recognize common objects, e.g. dogs, cats, cars, in images. As the network learns to recognize those objects is develops its convolutional layers to pick out "features" of the NN, like lines at a certain orientations, circles, etc.

Each neuron, is a filters: e.g. edge finders.

The DeepDream software runs this NN on an image you give it, and it loops on some hidden layers, thus "manifesting" the things it knows how to recognize in the image. The output of an inner layer (input of the next inner layer) is called a "feature map". We are taking a peek into the feature maps of a deep neural network trained to recognized common onbjects.

excellent blog post on BP: http://colah.github.io/posts/2015-08-Backprop/

Training a DNN

Training models with this many parameters requires a lot of care:

. defining the metric

. optimization schemes

. training/validation/testing sets

But just like our simple linear regression case, the fact that small changes in the parameters leads to small changes in the output for the right activation functions.

define a cost function, e.g.

$$C = \frac{1}{2}|y - a^L|^2 = \frac{1}{2}\sum_j (y_j - a_j^L)^2$$

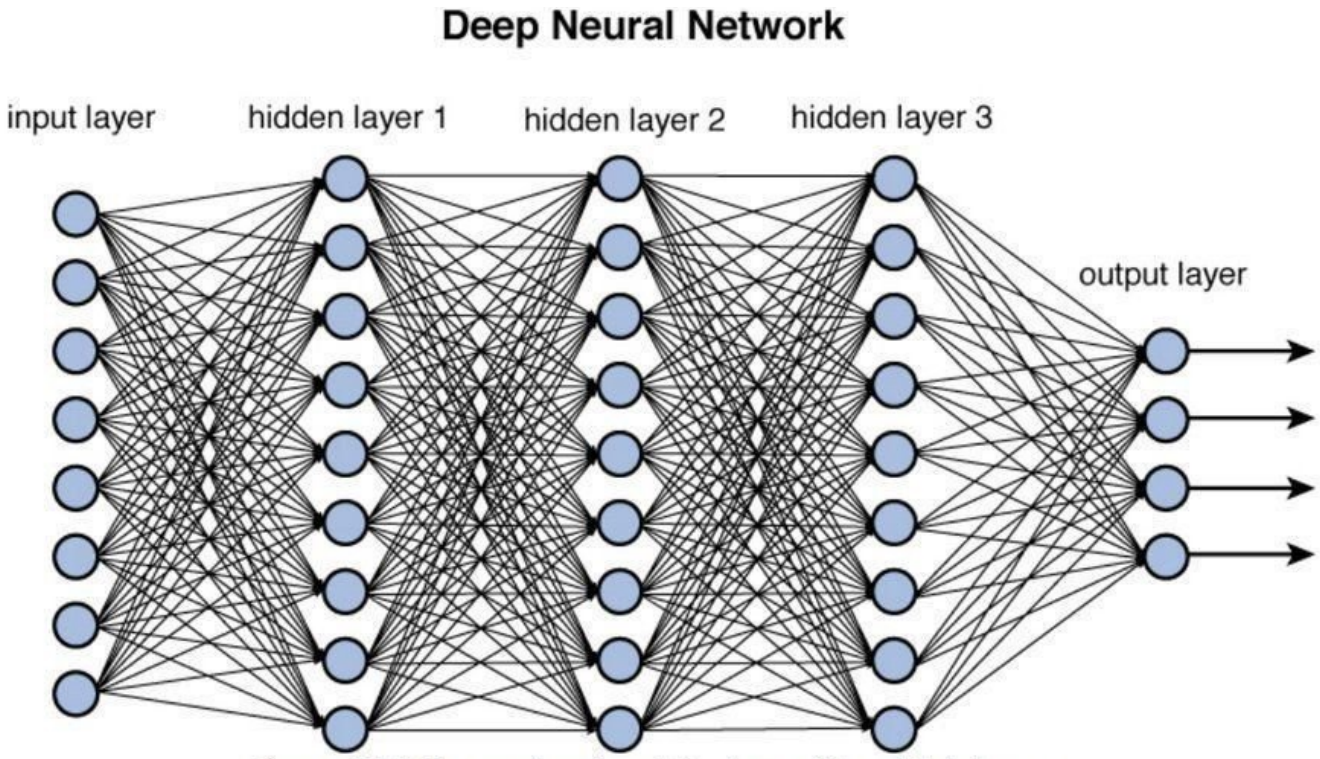


Figure 12.2 Deep network architecture with multiple layers.

feed data forward through network and calculate cost metric

for each layer, calculate effect of small changes on next layer

$$\vec{y} = f_N(....(f_1(\vec{x}W_i + b_1 ... W_N + b_N)))$$

overfitting

2

Minibatch & Dropout

overfitting

Split your training set into many smaller subsets and train on each small set separately

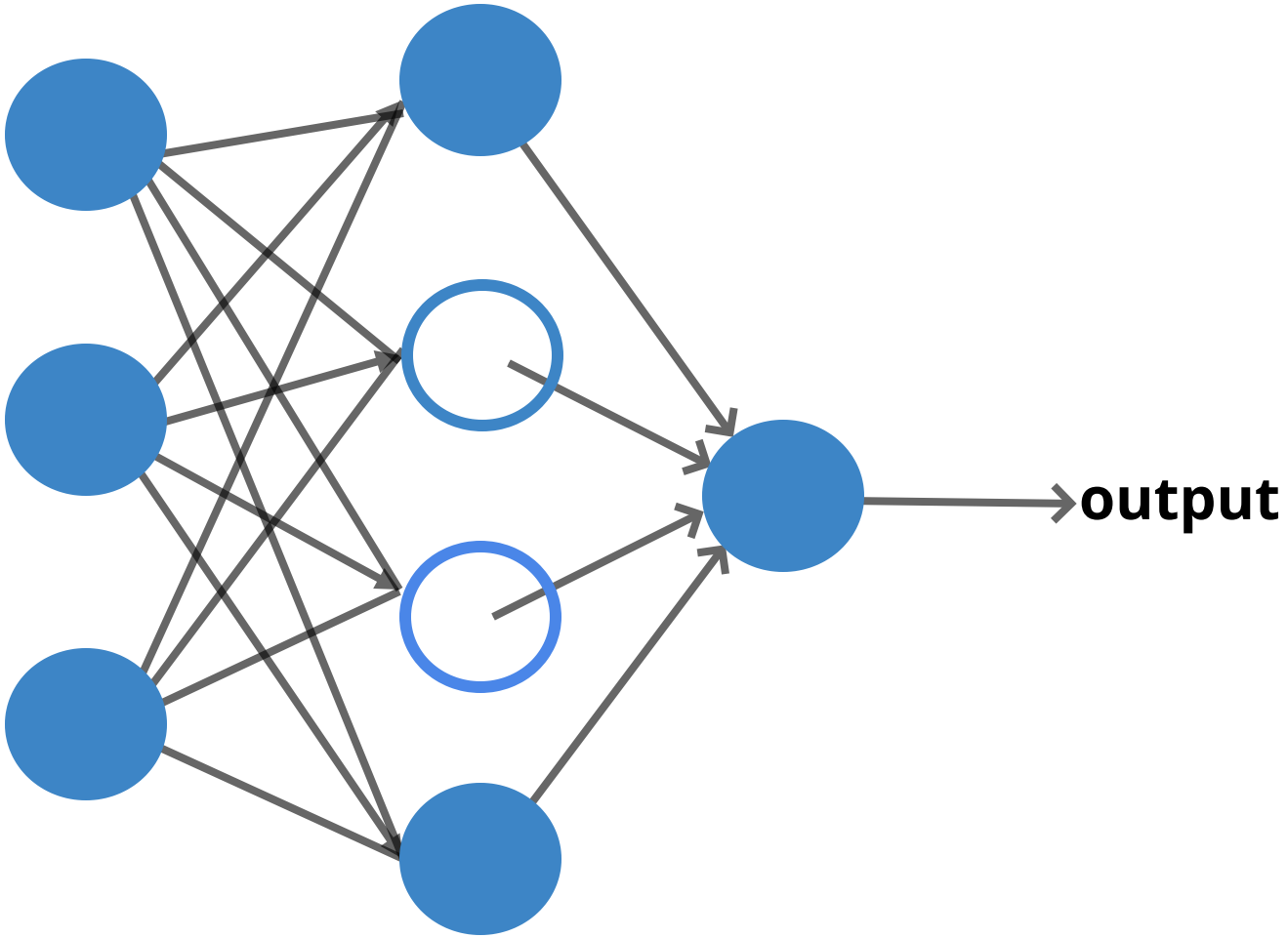
- If one updates model parameters after processing the whole training data (i.e., epoch), it would take too long to get a model update in training, and the entire training data probably won't fit in the memory.
- If one updates model parameters after processing every instance (i.e., stochastic gradient descent), model updates would be too noisy, and the process is not computationally efficient.
- Therefore, minibatch gradient descent is introduced as a trade-off between fast model updates (memory efficiency) and accurate model updates (computational efficiency).

https://www.quora.com/What-is-a-minibatch-in-a-neural-network

overfitting

Dropout

Artificially remove some neurons for different minibatches to avoid overfitting



Architecture components: neurons, activation function

- basically each neuron is a multivariate regression with an activation function that turns the output into a probability
- changing the weights and biases in the linear regression gives different results

Single layer NN: perceptrons

- perceptrons were developed in the 50s but a long time passed since then till people figured out how to build complex layered architectures and especially how to train them

Deep NN:

- DNN are multi-layer architectures of neurons. They can be fully connected (each neuron goes to each neuron of the next layer) or not (a neuron goes only to some neurons in the next layer)
- DNN have a lot of parameters (thousands!) which makes the interpretability and feature extraction of NN difficult.

Convolutional NN

- convolutional NN are DNN with three types of layers:
  - convolutional layers: run filters through an image to detect features like edges or colors
  - maxpool layers: decrease the size of the previous layer outputs and removes some details
  - ReLU : rectified linear units: normalizes the output of conv layers so that it is all positive (sets negatives to 0)
- CNNs are great for the stud of structure in large datasets (images are large datasets)

Training an NN:

- most ML methods are trained by gradient descent: change weights and biases based

Lots of parameters and lots of hyperparameters! What to choose? cheatsheet

1. **architecture - wide networks tend to overfit, deep networks are hard to train**

2. number of epochs - the sweet spot is when learning slows down, but before you start overfitting... it may take DAYS! jumps may indicate bad initial choices

3. loss function - needs to be appropriate to the task, e.g. classification vs regression

4. activation functions - needs to be consistent with the loss function

5. optimization scheme - needs to be appropriate to the task and data

6. learning rate in optimization - balance speed and accuracy

7. batch size - smaller batch size is faster but leads to overtraining

An article that compars various DNNs

AN ANALYSIS OF DEEP NEURAL NETWORK MODELS FOR PRACTICAL APPLICATIONS

Alfredo Canziani & Eugenio Culurciello
Weldon School of Biomedical Engineering
Purdue University
{canziani,euge}@purdue.edu

Adam Paszke
Faculty of Mathematics, Informatics and Mechanics
University of Warsaw
a.paszke@students.mimuw.edu.pl

ABSTRACT

Since the emergence of Deep Neural Networks (DNNs) as a prominent technique in the field of computer vision, the ImageNet classification challenge has played a major role in advancing the state-of-the-art. While accuracy figures have steadily increased, the resource utilisation of winning models has not been properly taken into account. In this work, we present a comprehensive analysis of important metrics in practical applications: accuracy, memory footprint, parameters, operations count, inference time and power consumption. Key findings are: (1) power consumption is independent of batch size and architecture; (2) accuracy and inference time are in a hyperbolic relationship; (3) energy constraint is an upper bound on the maximum achievable accuracy and model complexity; (4) the number of operations is a reliable estimate of the inference time. We believe our analysis provides a compelling set of information that helps design and engineer efficient DNNs.

https://arxiv.org/pdf/1605.07678.pdf

An article that compars various DNNs

AN ANALYSIS OF DEEP NEURAL NETWORK MODELS FOR PRACTICAL APPLICATIONS

Alfredo Canziani & Eugenio Culurciello
Weldon School of Biomedical Engineering
Purdue University
{canziani,euge}@purdue.edu

Adam Paszke
Faculty of Mathematics, Informatics and Mechanics
University of Warsaw
a.paszke@students.mimuw.edu.pl

ABSTRACT

Since the emergence of Deep Neural Networks (DNNs) as a prominent technique in the field of computer vision, the ImageNet classification challenge has played a major role in advancing the state-of-the-art. While accuracy figures have steadily increased, the resource utilisation of winning models has not been properly taken into account. In this work, we present a comprehensive analysis of important metrics in practical applications: accuracy, memory footprint, parameters, operations count, inference time and power consumption. Key findings are: (1) power consumption is independent of batch size and architecture; (2) accuracy and inference time are in a hyperbolic relationship; (3) energy constraint is an upper bound on the maximum achievable accuracy and model complexity; (4) the number of operations is a reliable estimate of the inference time. We believe our analysis provides a compelling set of information that helps design and engineer efficient DNNs.

https://arxiv.org/pdf/1605.07678.pdf

accuracy comparison

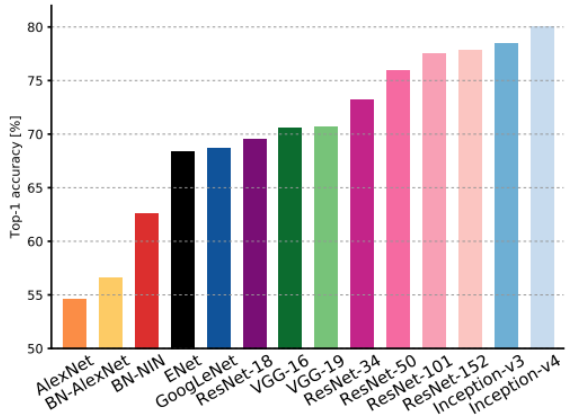


Figure 1: **Top1 *vs.* network.** Single-crop top-1 validation accuracies for top scoring single-model architectures. We introduce with this chart our choice of

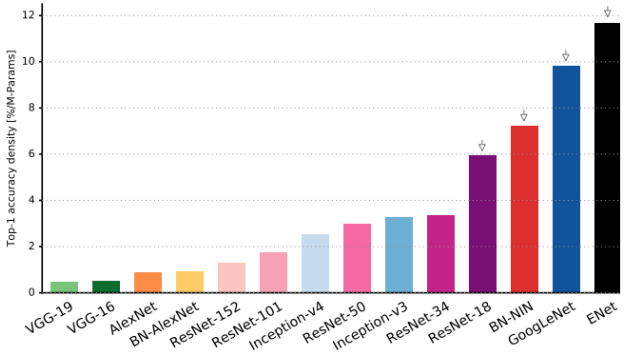


Figure 10: **Accuracy per parameter *vs.* network.** Information density (accuracy per parameters) is an efficiency metric that highlight that capacity of a specific architecture to better utilise its parametric space. Models like VGG and AlexNet are clearly oversized, and do not take fully advantage of their potential learning ability. On the far right, ResNet-18, BN-NIN, GoogLeNet and ENet (marked by grey arrows) do a better job at "squeezing" all their neurons to learn the given task, and are the winners of this section.

An article that compars various DNNs

AN ANALYSIS OF DEEP NEURAL NETWORK MODELS FOR PRACTICAL APPLICATIONS

Alfredo Canziani & Eugenio Culurciello
Weldon School of Biomedical Engineering
Purdue University
{canziani,euge}@purdue.edu

Adam Paszke
Faculty of Mathematics, Informatics and Mechanics
University of Warsaw
a.paszke@students.mimuw.edu.pl

ABSTRACT

Since the emergence of Deep Neural Networks (DNNs) as a prominent technique in the field of computer vision, the ImageNet classification challenge has played a major role in advancing the state-of-the-art. While accuracy figures have steadily increased, the resource utilisation of winning models has not been properly taken into account. In this work, we present a comprehensive analysis of important metrics in practical applications: accuracy, memory footprint, parameters, operations count, inference time and power consumption. Key findings are: (1) power consumption is independent of batch size and architecture; (2) accuracy and inference time are in a hyperbolic relationship; (3) energy constraint is an upper bound on the maximum achievable accuracy and model complexity; (4) the number of operations is a reliable estimate of the inference time. We believe our analysis provides a compelling set of information that helps design and engineer efficient DNNs.

https://arxiv.org/pdf/1605.07678.pdf

accuracy comparison

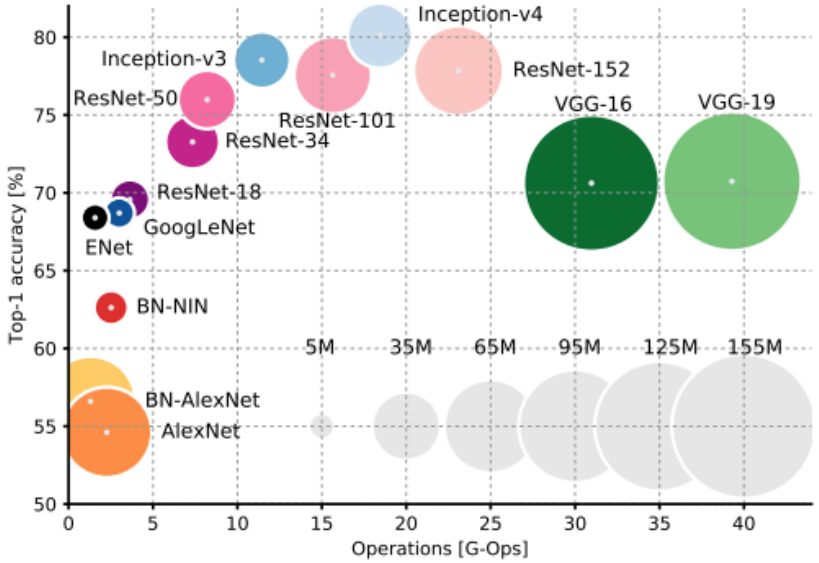


Figure 2: **Top1 *vs.* operations, size $\propto$ parameters.** Top-1 one-crop accuracy versus amount of operations required for a single forward pass. The size of the blobs is proportional to the number of network parameters; a legend is reported in the bottom right corner, spanning from 5×10^6 to 155×10^6 params. Both these figures share the same y-axis, and the grey dots highlight the centre of the blobs.

An article that compars various DNNs

AN ANALYSIS OF DEEP NEURAL NETWORK MODELS FOR PRACTICAL APPLICATIONS

Alfredo Canziani & Eugenio Culurciello
Weldon School of Biomedical Engineering
Purdue University
{canziani,euge}@purdue.edu

Adam Paszke
Faculty of Mathematics, Informatics and Mechanics
University of Warsaw
a.paszke@students.mimuw.edu.pl

ABSTRACT

Since the emergence of Deep Neural Networks (DNNs) as a prominent technique in the field of computer vision, the ImageNet classification challenge has played a major role in advancing the state-of-the-art. While accuracy figures have steadily increased, the resource utilisation of winning models has not been properly taken into account. In this work, we present a comprehensive analysis of important metrics in practical applications: accuracy, memory footprint, parameters, operations count, inference time and power consumption. Key findings are: (1) power consumption is independent of batch size and architecture; (2) accuracy and inference time are in a hyperbolic relationship; (3) energy constraint is an upper bound on the maximum achievable accuracy and model complexity; (4) the number of operations is a reliable estimate of the inference time. We believe our analysis provides a compelling set of information that helps design and engineer efficient DNNs.

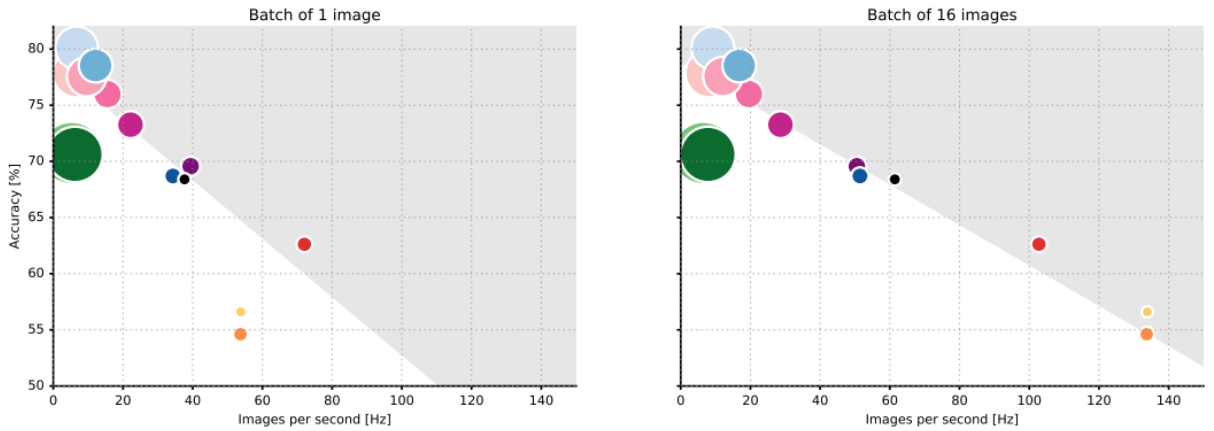


Figure 9: **Accuracy *vs.* inferences per second, size $\propto$ operations.** Non trivial linear upper bound is shown in these scatter plots, illustrating the relationship between prediction accuracy and throughput of all examined architectures. These are the first charts in which the area of the blobs is proportional to the amount of operations, instead of the parameters count. We can notice that larger blobs are concentrated on the left side of the charts, in correspondence of low throughput, *i.e.* longer inference times. Most of the architectures lay on the linear interface between the grey and white areas. If a network falls in the shaded area, it means it achieves exceptional accuracy or inference speed. The white area indicates a suboptimal region. *E.g.* both AlexNet architectures improve processing speed as larger batches are adopted, gaining 80 Hz.

https://arxiv.org/pdf/1605.07678.pdf

batch size

Lots of parameters and lots of hyperparameters! What to choose? cheatsheet

1. architecture - wide networks tend to overfit, deep networks are hard to train

2. **number of epochs - the sweet spot is when learning slows down, but before you start overfitting... it may take DAYS!** jumps may indicate bad initial choices

3. **loss function - needs to be appropriate to the task, e.g. classification vs regression**

4. activation functions - needs to be consistent with the loss function

5. optimization scheme - needs to be appropriate to the task and data

6. learning rate in optimization - balance speed and accuracy

7. batch size - smaller batch size is faster but leads to overtraining

Lots of parameters and lots of hyperparameters! What to choose?

https://github.com/fedhere/MLTSA_FBianco/blob/master/autoencode_digits.ipynb

3.0.1 regression

- loss='mean_squared_error' L2: default loss to use for regression problems. => linear activation funct in output layer, one node out

alternatives: loss='mean_squared_logarithmic_error', 'mean_absolute_error' (which is L1 instead of L2)

3.0.2 binary classification

- loss='binary_crossentropy' => sigmoid activation function in output layer, one node out

alternatives: 'hinge'

3.0.3 multiclass classification

categorical encoded as numerical

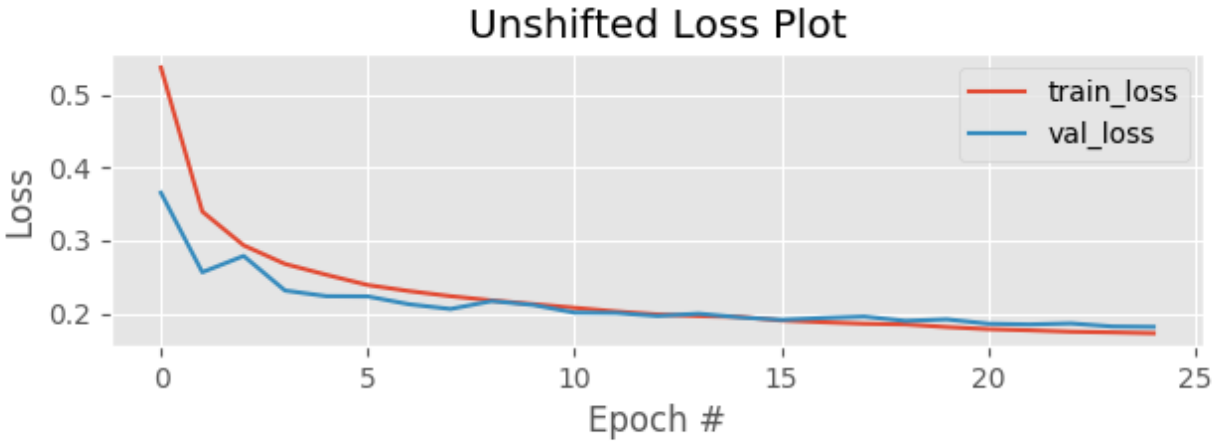
- loss='categorical_crossentropy' => softmax n nodes out

onehot encoded categoridal

- 'parse_categorical_crossentropy' => softmax n nodes out
- 'kullback Leibler Divergence Loss' => probabilistic categorical classification; log(P/Q)

What should I choose for the loss function and how does that relate to the activation functiom and optimization?

Lots of parameters and lots of hyperparameters! What to choose? cheatsheet



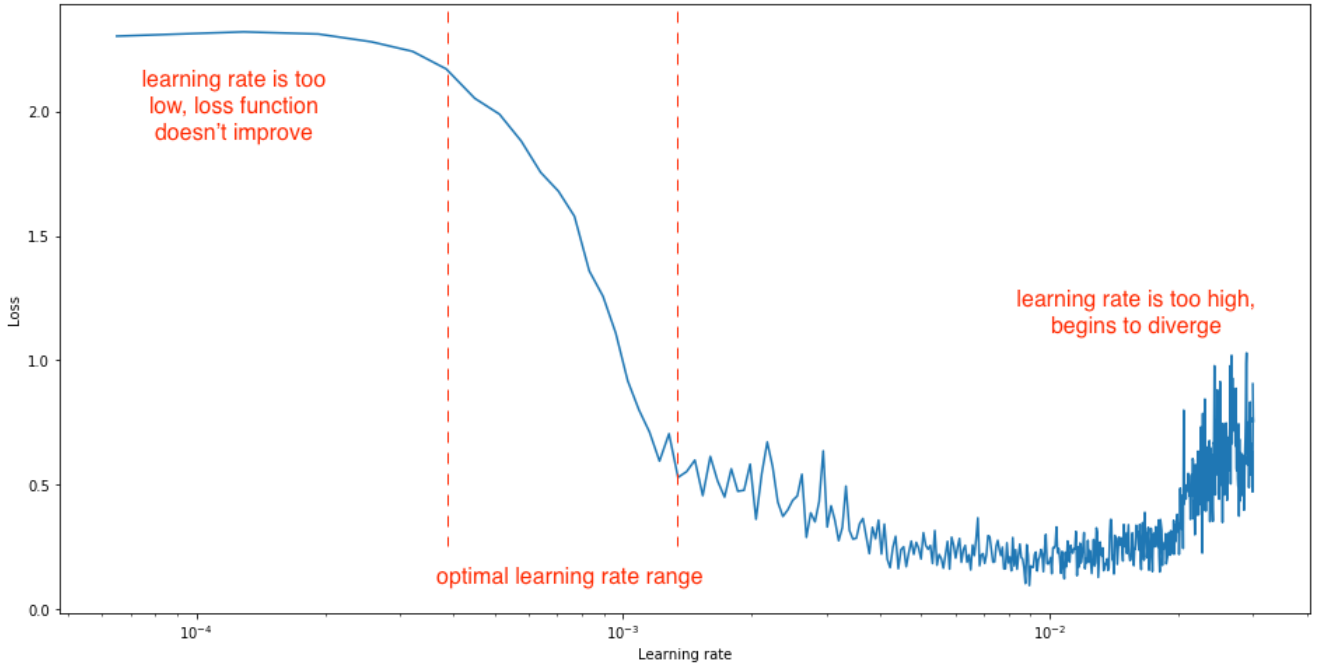
always check your loss function! it should go down smoothly and flatten out at the end of the training.

not flat? you are still learning!

too flat? you are overfitting...

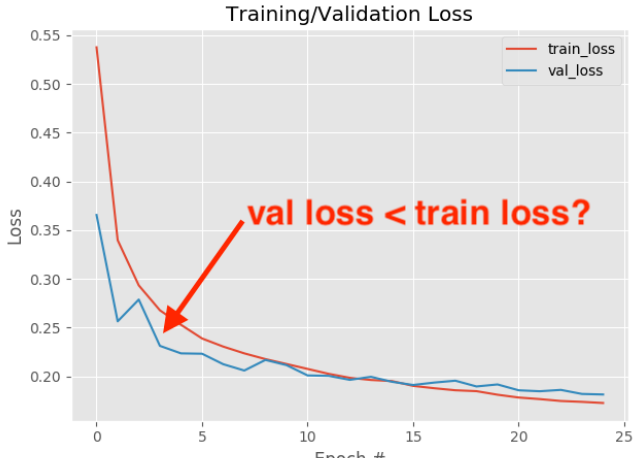
loss (gallery of horrors)

https://github.com/fedhere/MLTSA_FBianco/blob/master/autoencode_digits.ipynb



loss and learning rate (not that the appropriate learning rate depends on the chosen optimization scheme!)

jumps are not unlikely (and not necessarily a problem) if your activations are discontinuous (e.g. relu)



when you use validation you are introducing regularizations (e.g. dropout) so the loss can be smaller than for the training set

Building a DNN

with keras and tensorflow

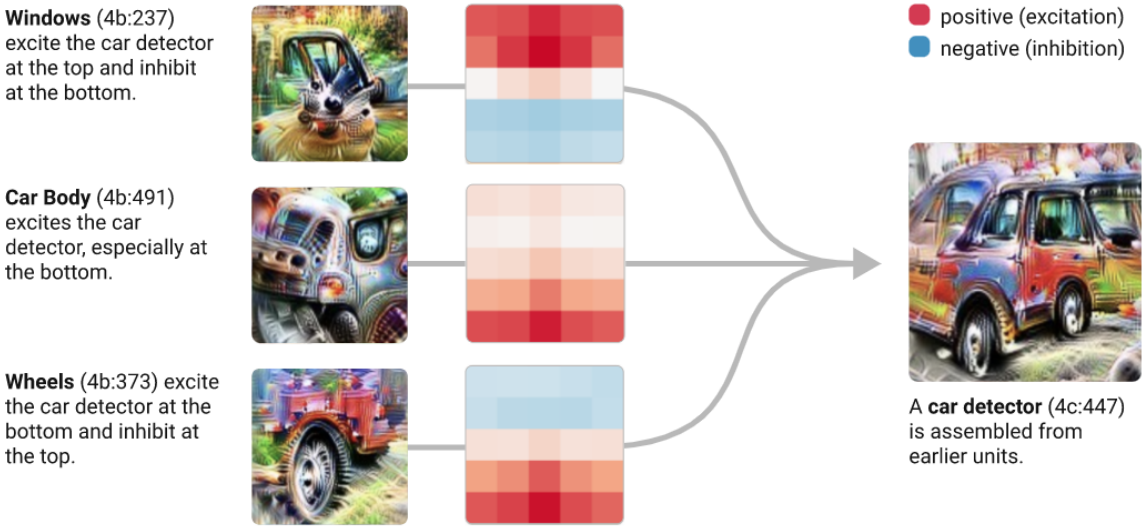
autoencoder for image recontstruction

https://github.com/fedhere/MLTSA_FBianco/blob/master/autoencode_digits.ipynb

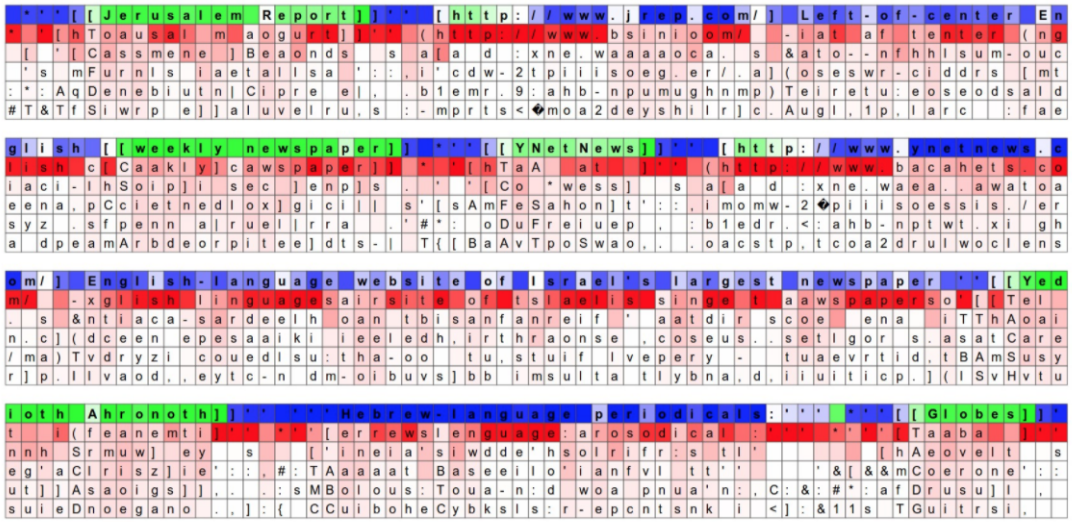
What should I choose for the loss function and how does that relate to the activation functiom and optimization?

loss	**good for**	**activation last layer**	**size last layer**
mean_squared_error	regression	linear	one node
mean_absolute_error	regression	linear	one node
mean_squared_logarithmit_error	regression	linear	one node
binary_crossentropy	binary classification	sigmoid	one node
categorical_crossentropy	multiclass classification	sigmoid	N nodes
Kullback_Divergence	multiclass classification, probabilistic inerpretation	sigmoid	N nodes

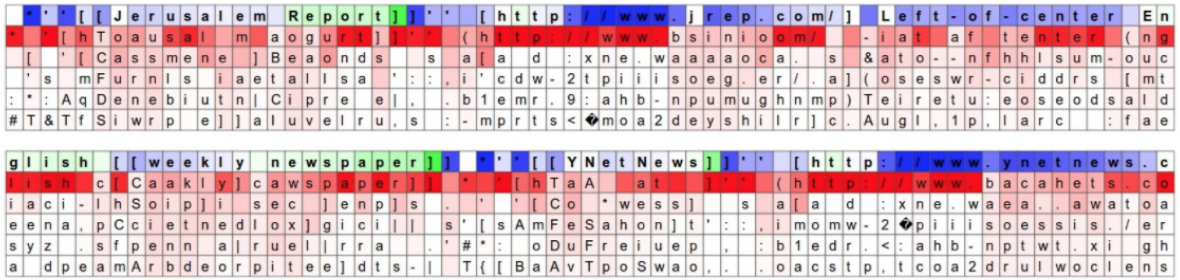
On the interpretability of DNNs



https://distill.pub/2020/circuits/zoom-in/



The highlighted neuron here gets very excited when the RNN is inside the [[ ]] markdown environment and turns off outside of it. Interestingly, the neuron can't turn on right after it sees the character "[", it must wait for the second "[" and then activate. This task of counting whether the model has seen one or two "[" is likely done with a different neuron.

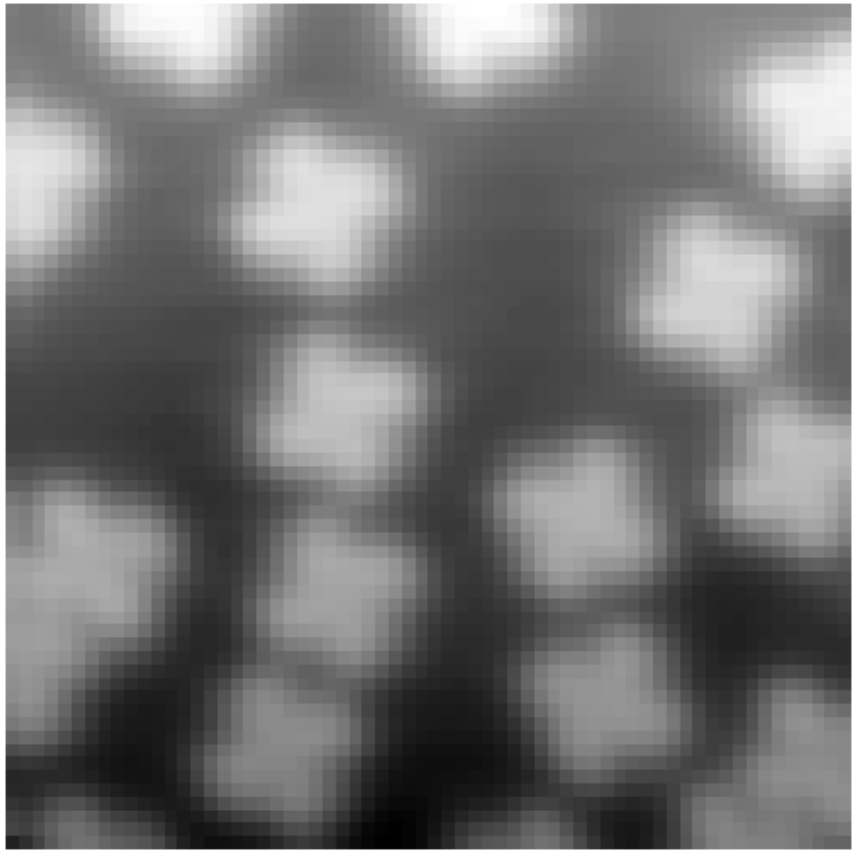


Here we see a neuron that varies seemingly linearly across the [[ ]] environment. In other words its activation is giving the RNN a time-aligned coordinate system across the [[ ]] scope. The RNN can use this information to make different characters more or less likely depending on how early/late it is in the [[ ]] scope (perhaps?).

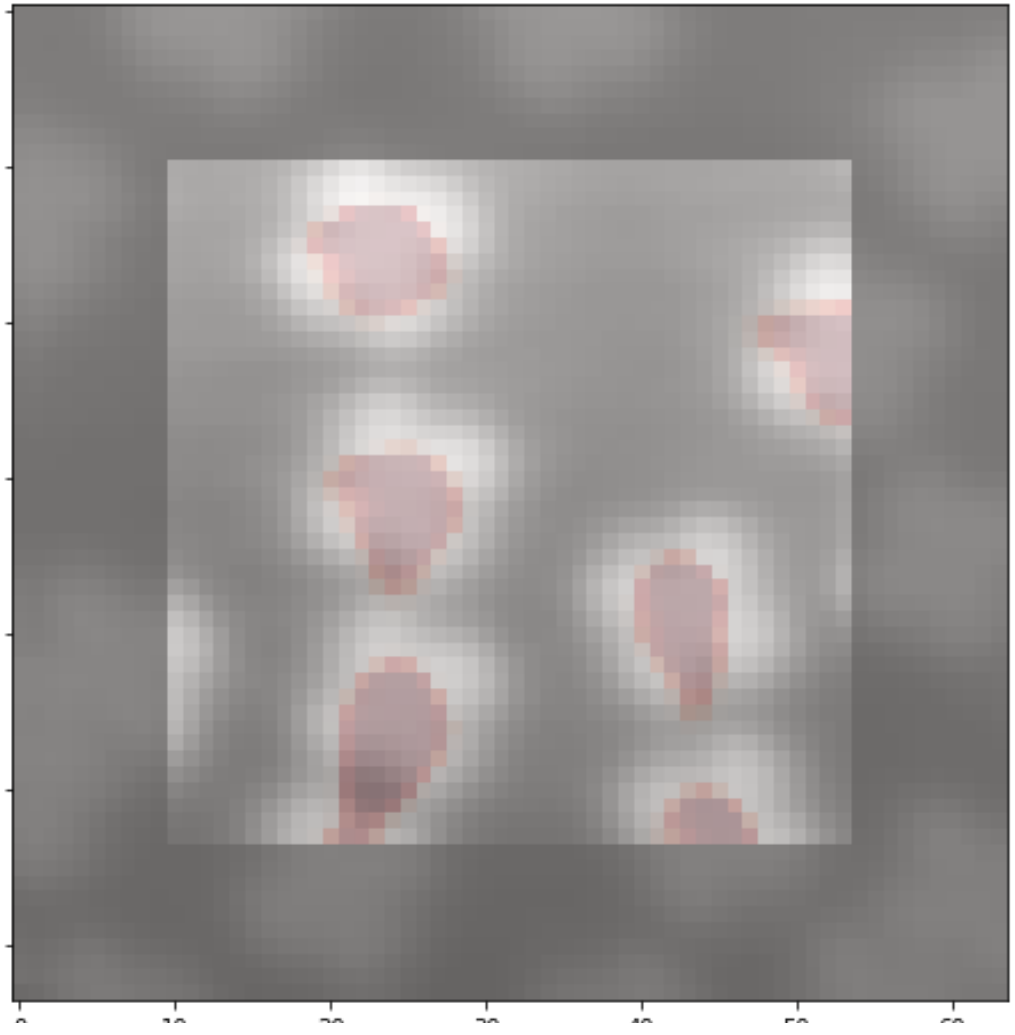
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

YOLO and R-CNN

Object detection

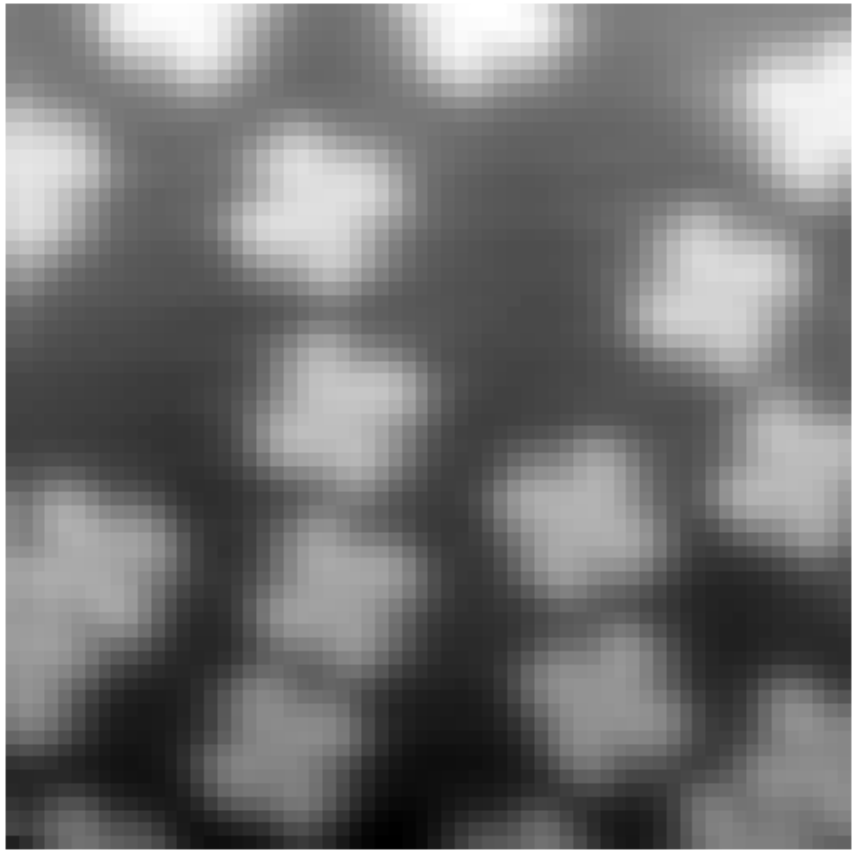


Naive model: we took different region of the image and measured the probability of presence of the object in that region

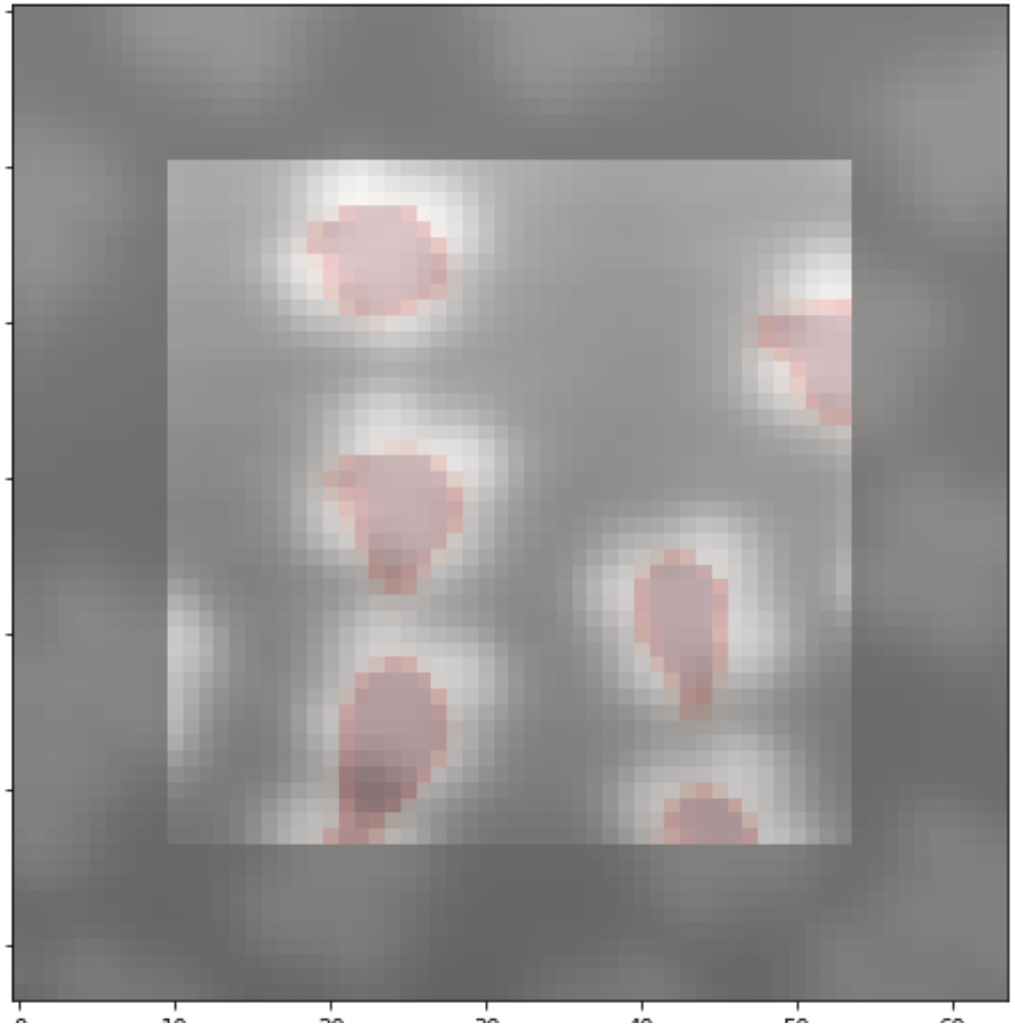


YOLO and R-CNN

Object detection

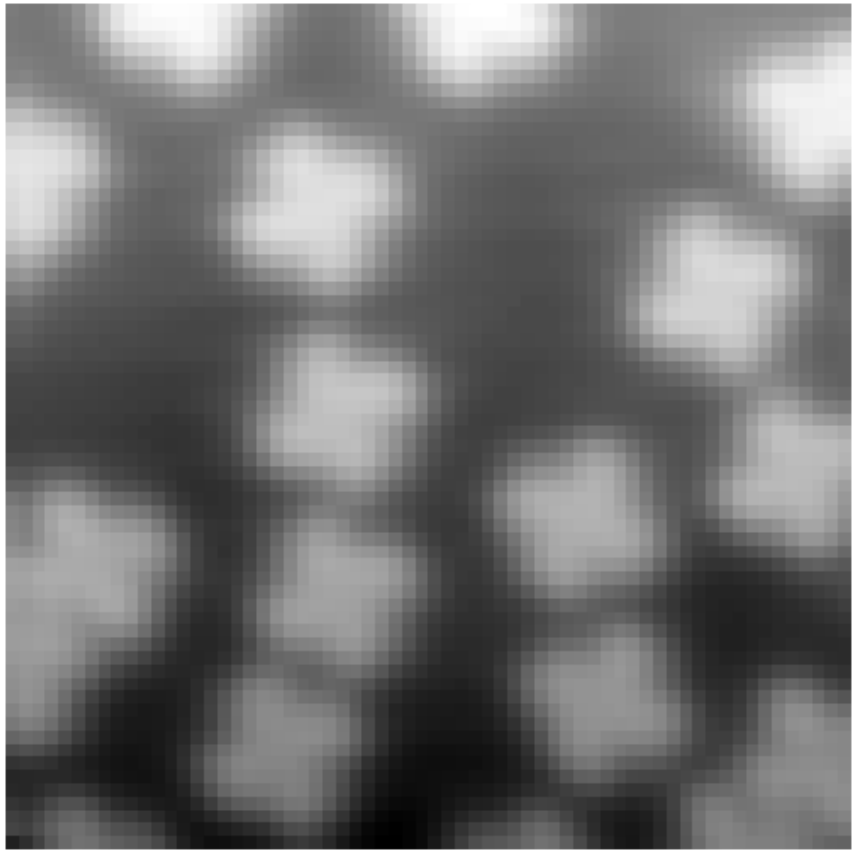


Problem: we had to search the whole image which is time consuming, **we could only find 1 kind of object at 1 scale**

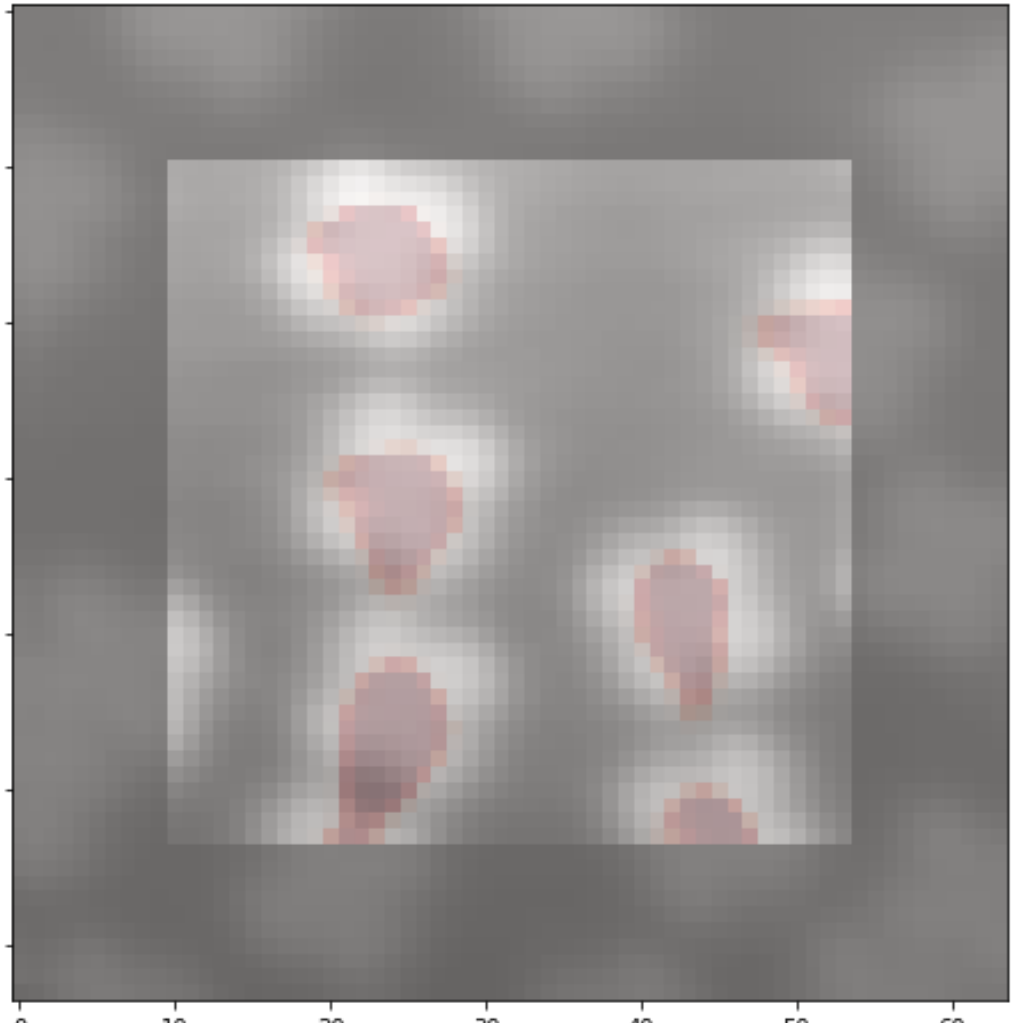


YOLO and R-CNN

Object detection



Problem: we had to search the whole image which is time consuming, **we could only find 1 kind of object at 1 scale**



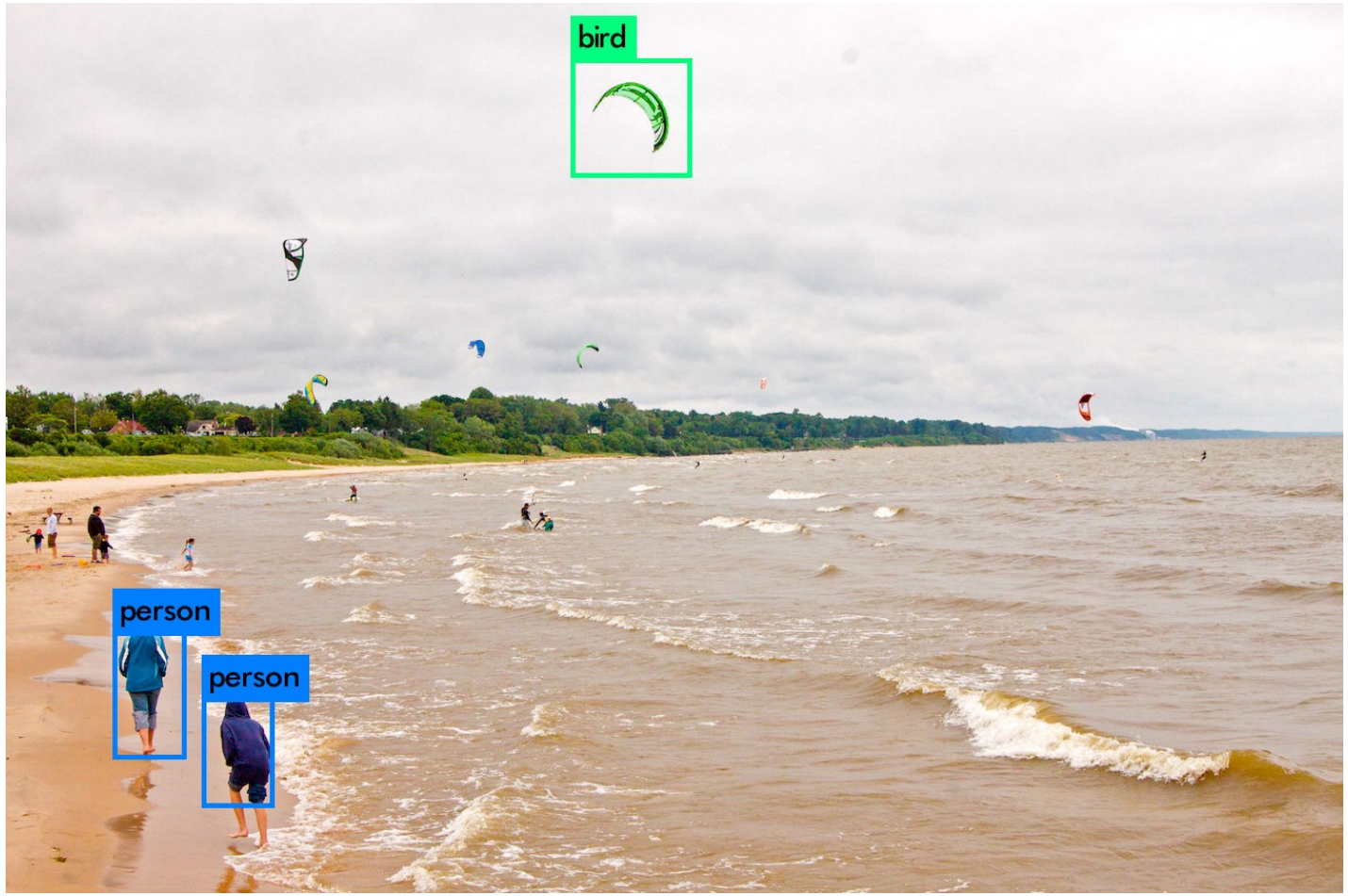
YOLO and R-CNN

Object detection

What if you do not know what is in the mage?

Final Dense layer has undefined size (one per kind of object in the region)

Objects can have different scale or axis ration: how many regions can you search before the problem blows up computationally??



R-CNN

Girshick et al. 2013

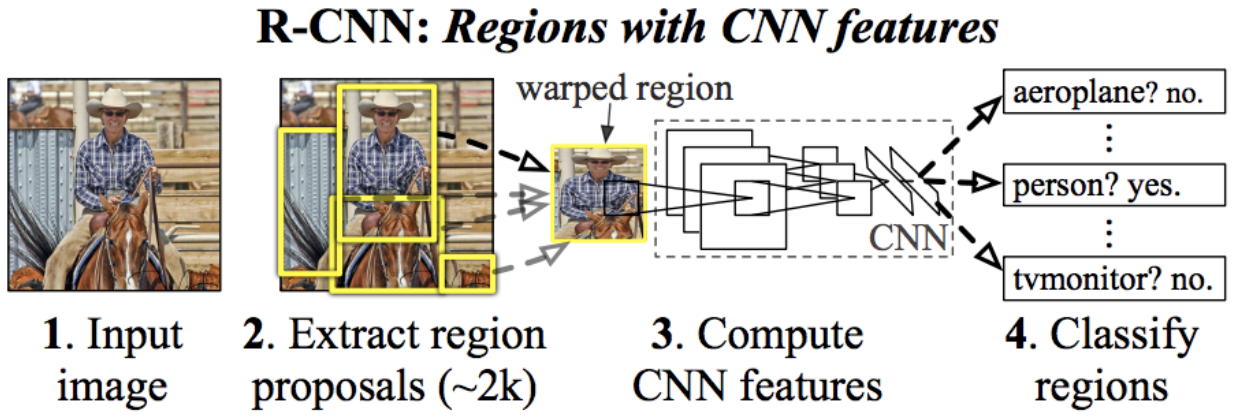
Extract 2000 regions from the image "region proposals."

Feature Extraction CNN produces a 4096-dimensional feature vector in an output dense layer

SVM classify the presence of the object within that candidate region proposal.

TOO SLOW (47 sec to test 1 image)

```
1. Generate initial sub-
segmentation, we generate many
candidate     regions
2. Use greedy algorithm to
recursively combine similar
regions into larger ones
3. Use the generated regions to
produce the final candidate
region proposals
```



Fast R-CNN

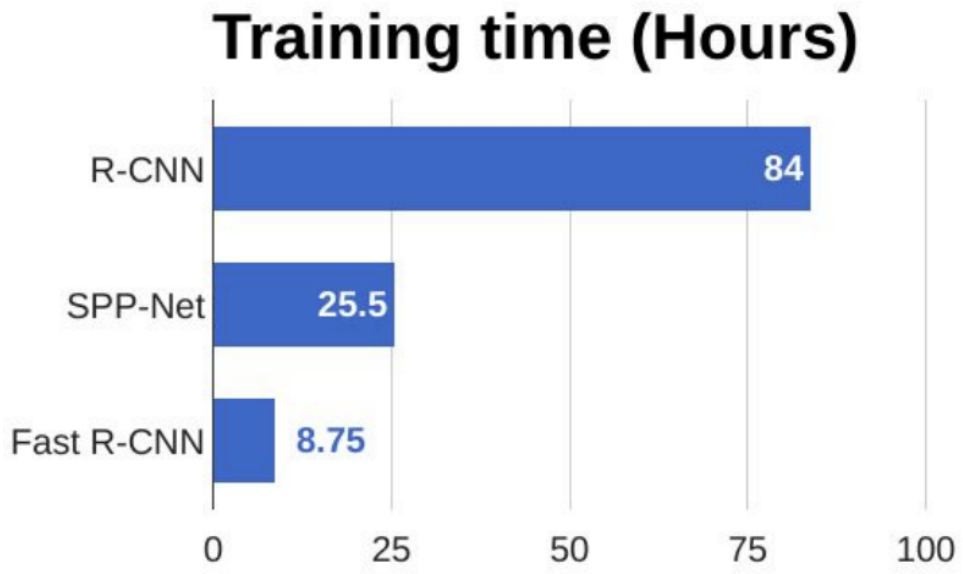
Girshick et al. 2015

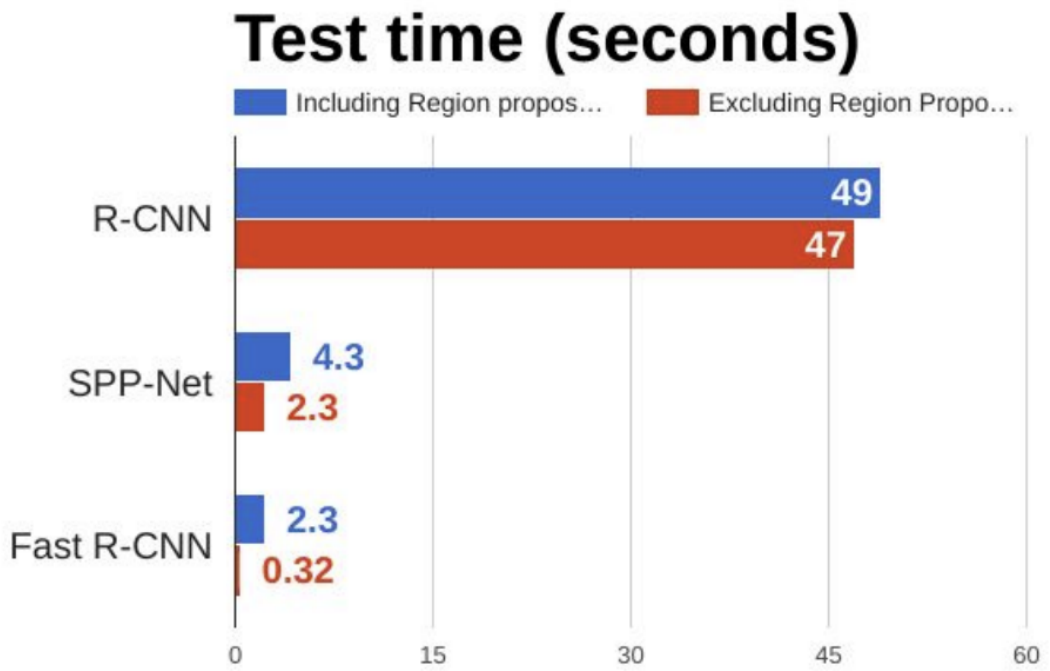
Use a CNN to generate convolutional feature maps

Use Selective Search Algorithm to tdentify the RPs and warp them into squares

Using an RoI ***pooling layer*** to reshape them into a fixed size so that it can be fed into a fully connected layer - predict box offset

Softmax layer to predict the class of the proposed





Fast R-CNN

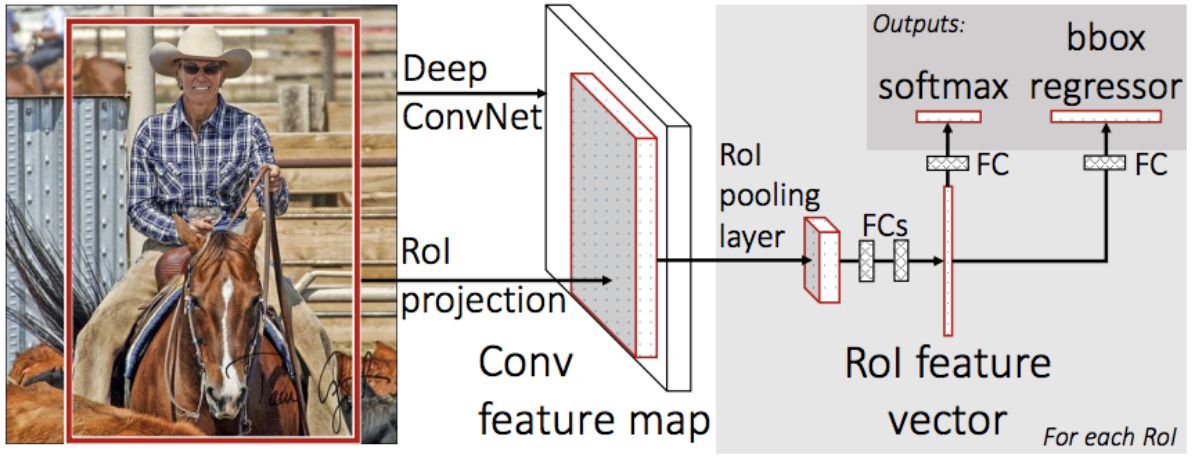
Girshick et al. 2015

Use a CNN to generate convolutional feature maps

Use Selective Search Algorithm to tdentify the RPs and warp them into squares

Using an RoI ***pooling layer*** to reshape them into a fixed size so that it can be fed into a fully connected layer - predict box offset

Softmax layer to predict the class of the proposed



Faster R-CNN

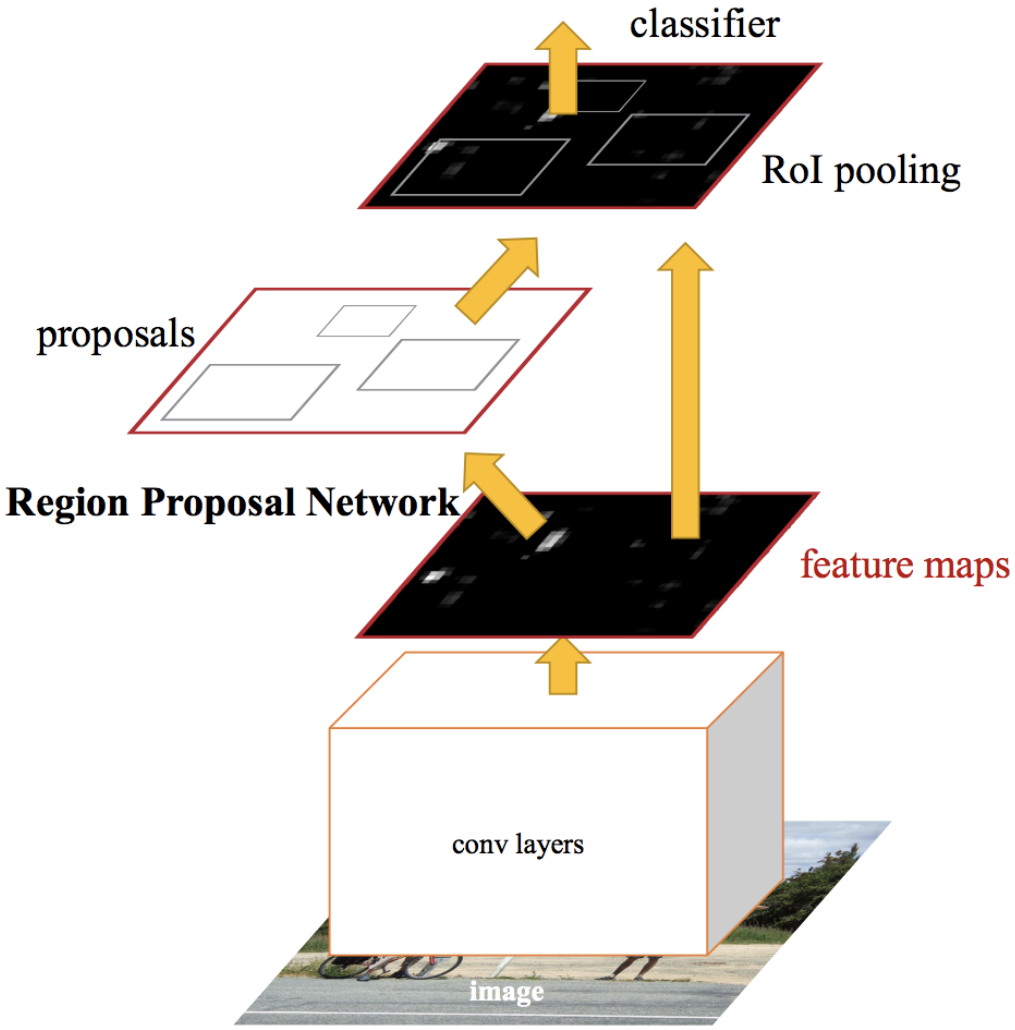
Ren et al. 2015

Use a CNN to generate convolutional feature maps

Use CNN to predict RPs and warp them into squares

Using an RoI ***pooling layer*** to reshape them into a fixed size so that it can be fed into a fully connected layer - predict box offset

Softmax layer to predict the class of the proposed



Faster R-CNN

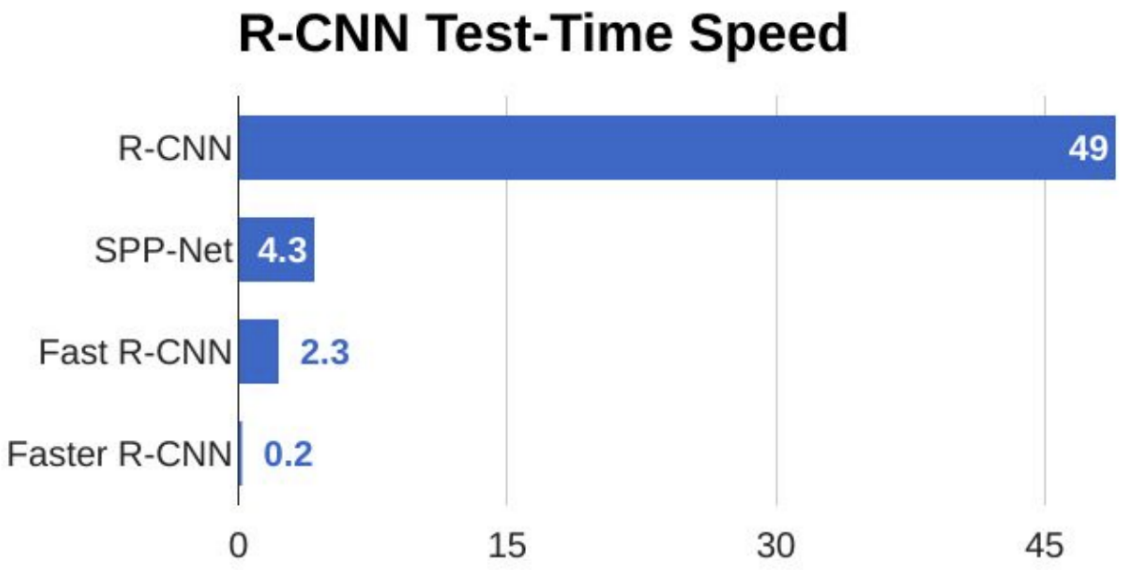
Ren et al. 2015

Use a CNN to generate convolutional feature maps

Use CNN to predict RPs and warp them into squares

Using an RoI ***pooling layer*** to reshape them into a fixed size so that it can be fed into a fully connected layer - predict box offset

Softmax layer to predict the class of the proposed



Yolo

Redmon et al 2016

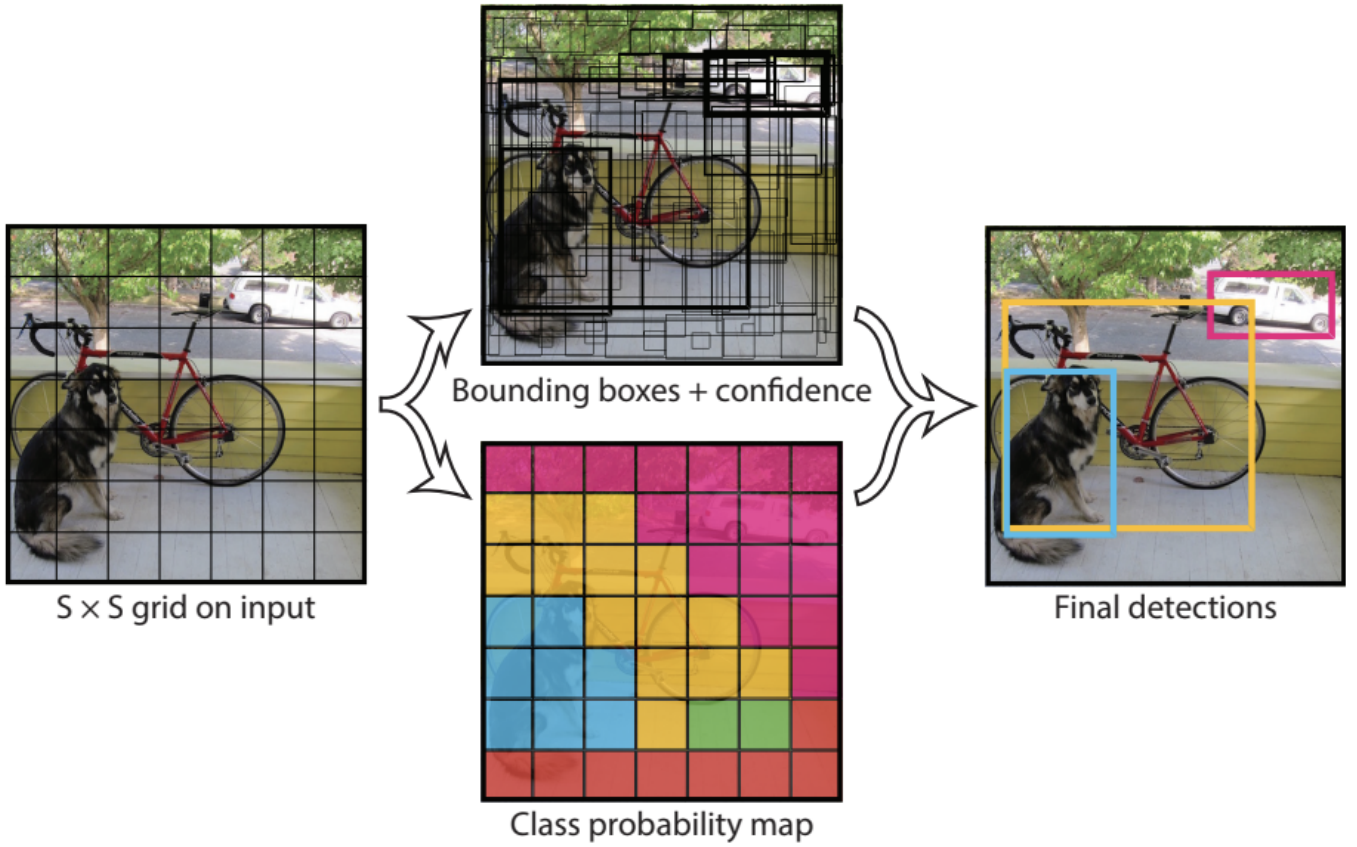
What if you looked at the whole image instead of RoIs in the image??

Split an image into a SxS grid

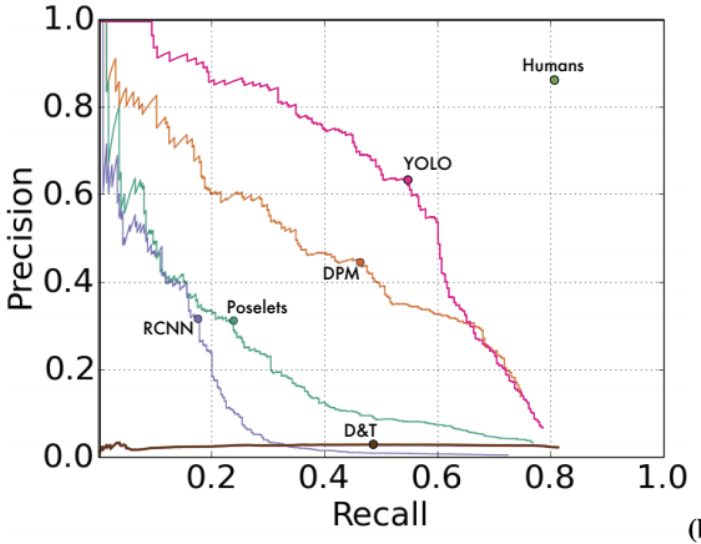
For each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities.

CNN outputs probability that BB has an object (+ offset)

High prob BBs are classified



Redmon et al 2016



(a) Picasso Dataset precision-recall curves.

	VOC 2007	Picasso		People-Art
	AP	AP	Best F_1	AP
YOLO	**59.2**	**53.3**	**0.590**	**45**
R-CNN	54.2	10.4	0.226	26
DPM	43.2	37.8	0.458	32
Poselets [2]	36.5	17.8	0.271	
D&T [4]	-	1.9	0.051	

(b) Quantitative results on the VOC 2007, Picasso, and People-Art Datasets. The Picasso Dataset evaluates on both AP and best F_1 score.

Figure 5: Generalization results on Picasso and People-Art datasets.

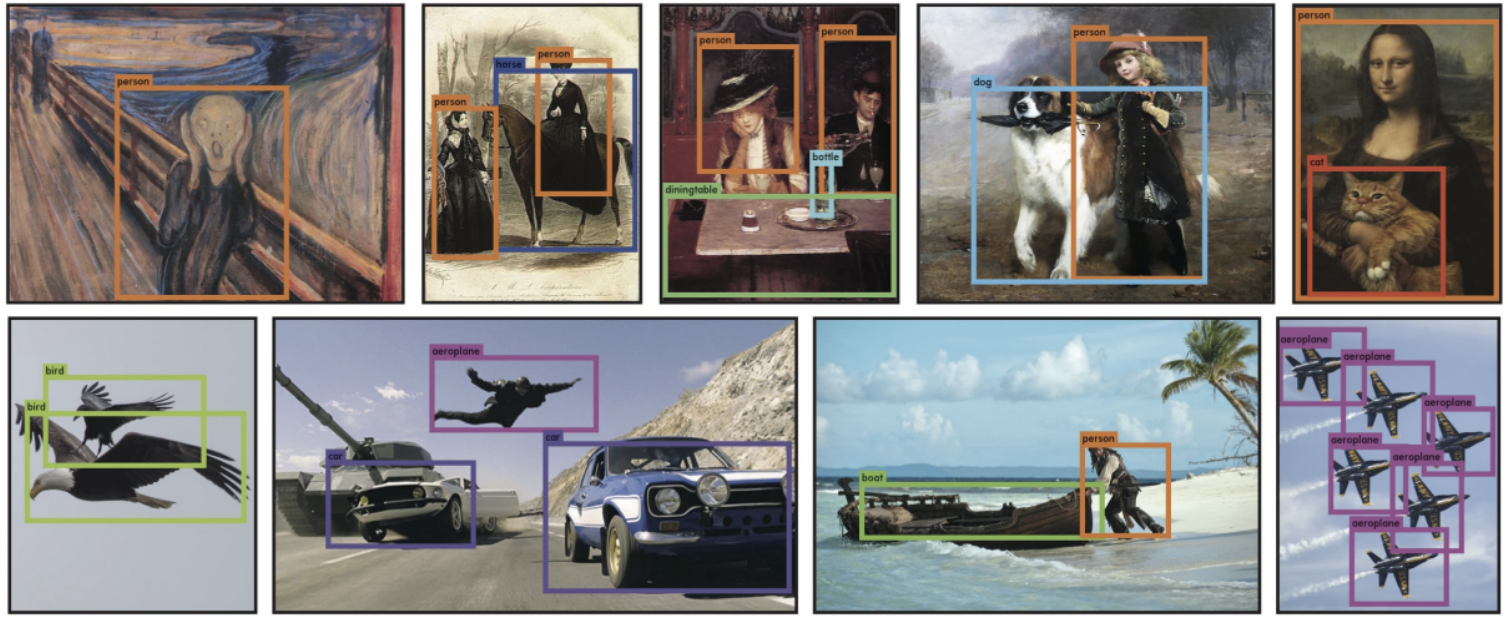


Figure 6: Qualitative Results. YOLO running on sample artwork and natural images from the internet. It is mostly accurate although it does think one person is an airplane.

Labling tools

https://www.v7labs.com/blog/best-image-annotation-tools

Neural Network and Deep Learning

an excellent and free book on NN and DL

http://neuralnetworksanddeeplearning.com/index.html

History of NN

https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history2.html

Gradient Descent

https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html

Backpropagation

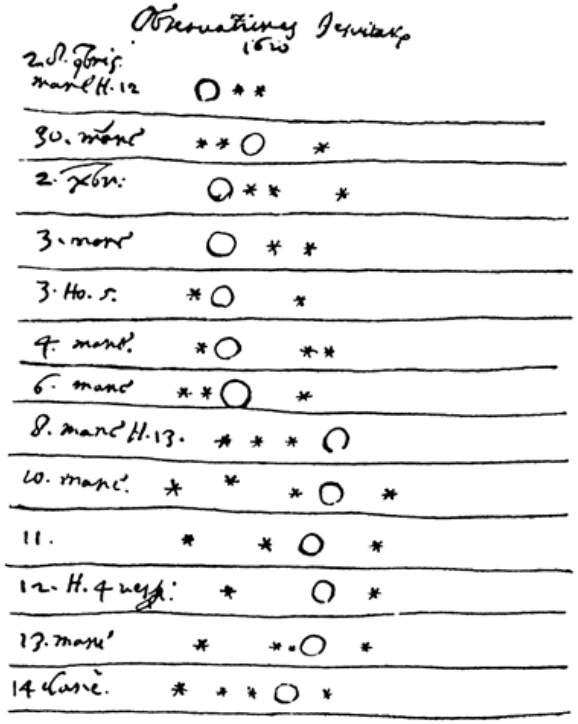
http://colah.github.io/posts/2015-08-Backprop/

Physics Informed NN

PINN

Application regime:

-infinity - 1950's

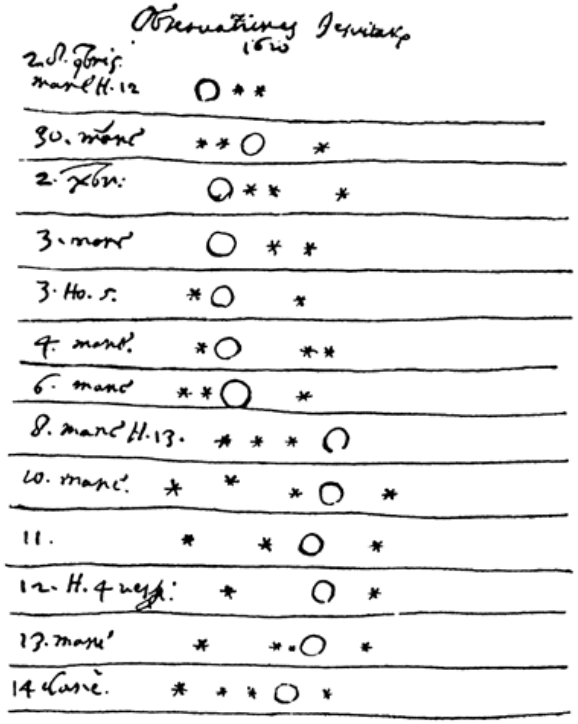


theory driven: *little data, mostly theory, falsifiability and all that...*

PiNN

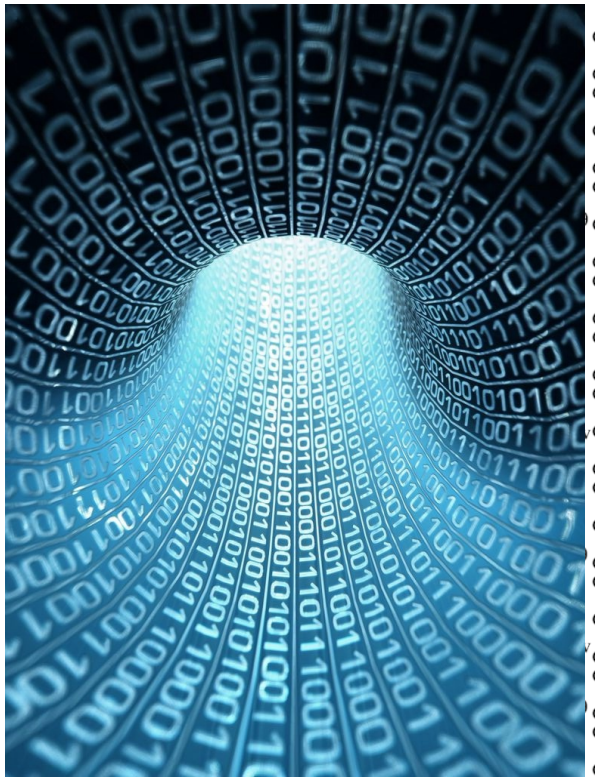
Application regime:

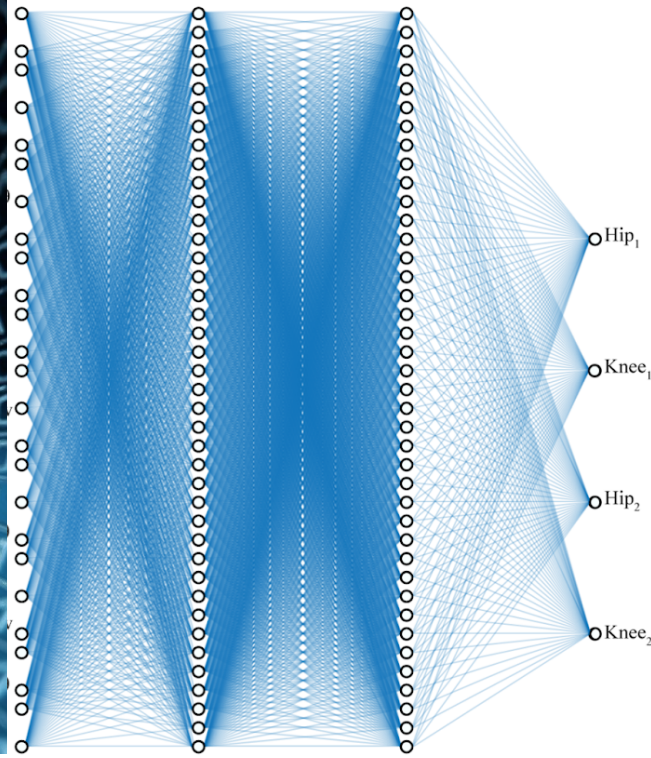
-infinity - 1950's



theory driven: *little data, mostly theory, falsifiability and all that...*

-1980's - today



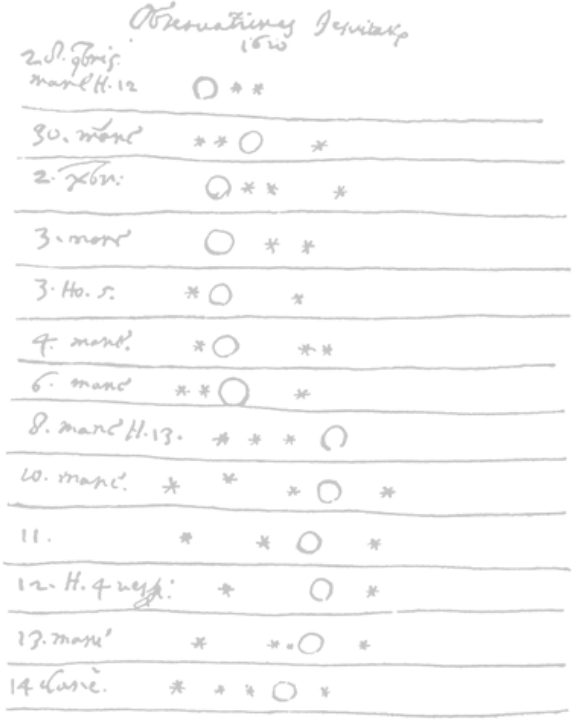


data driven: *lots of data, drop theory and use associations, black-box modles*

PINN

Application regime:

-infinity - 1950's

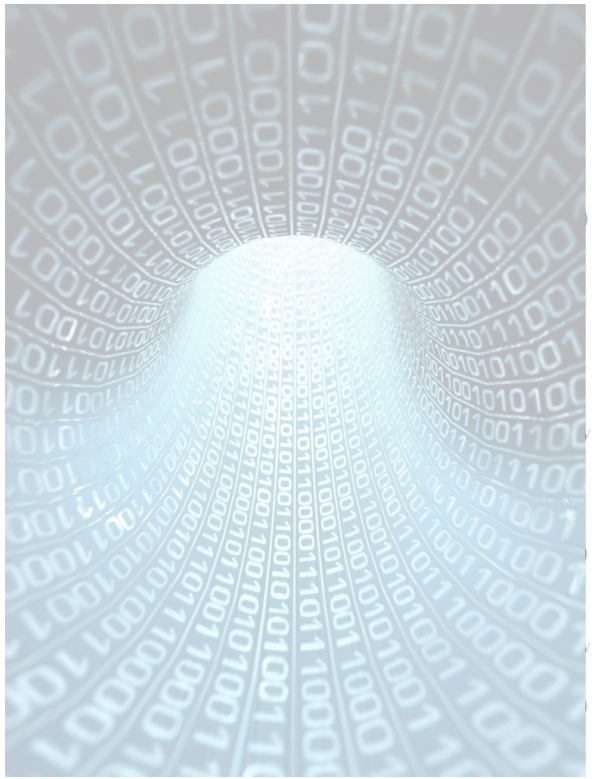


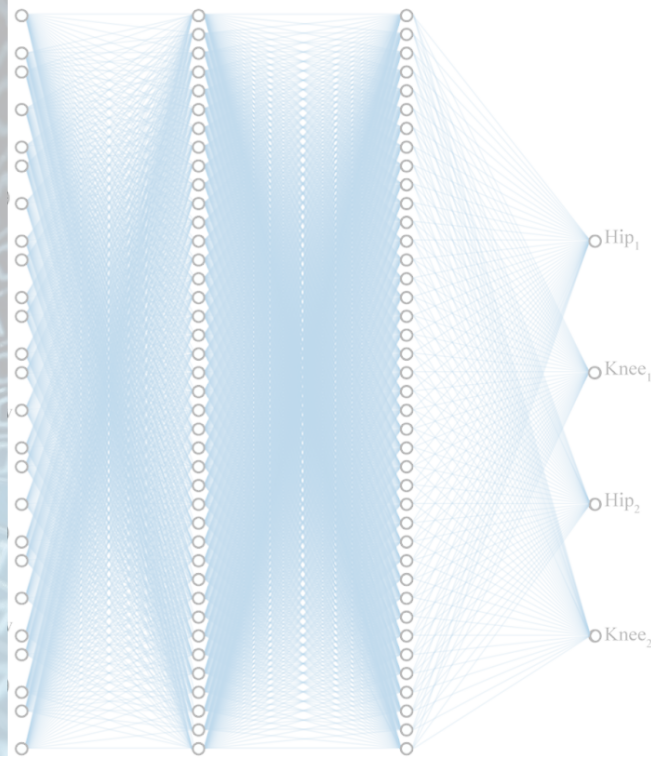
lots of data yet not enough for entirely automated decision making

complex theory that cannot be solved analytically

combine it with some theory

-1980's - today





theory driven: *little data, mostly theory, falsifiability and all that...*

data driven: *lots of data, drop theory and use associations, black-box modles*

PINN

$\mathcal{N}[\cdot]$ is a nonlinear differential operator

$$\partial_t u(t,x) + \mathcal{N}[u](t,x) = 0$$

$$u(0,x) = u_0(x)$$

$$u : [0,T] \times D => \mathbb{R}$$

$$(t,x) \in (0,T] x D$$

General conservation law

$$u(t) + \frac{\partial f(u)}{\partial x} = 0$$

e.g. flux function (linear)

$$f(u) = a \cdot u$$

Burgers eq (non-linear)

$$f(u) = \frac{1}{2}u^2$$

Non Linear PDEs are hard to solve!

Existence and uniqueness of solutions

A fundamental question for any PDE is the existence and uniqueness of a solution for given boundary conditions. open problem of existence (and smoothness) of solutions to the Navier–Stokes equations is one of the seven Millennium Prize problems in mathematics.

- Raissi et al. *Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations*. arXiv 1711.10561
- Raissi et al. *Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations*. arXiv 1711.10566
- Raissi et al. *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*. J. Comp. Phys. 378 pp. 686-707 DOI: 10.1016/j.jcp.2018.10.045

https://en.wikipedia.org/wiki/Nonlinear_partial_differential_equation

Non Linear PDEs are hard to solve!

Linear approximation

The solutions in a neighborhood of a known solution can sometimes be studied by linearizing the PDE around the solution. This corresponds to studying the tangent space of a point of the moduli space of all solutions.

- Raissi et al. *Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations*. arXiv 1711.10561
- Raissi et al. *Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations*. arXiv 1711.10566
- Raissi et al. *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*. J. Comp. Phys. 378 pp. 686-707 DOI: 10.1016/j.jcp.2018.10.045

https://en.wikipedia.org/wiki/Nonlinear_partial_differential_equation

Non Linear PDEs are hard to solve!

Exact solutions

It is often possible to write down some special solutions explicitly in terms of elementary functions (though it is rarely possible to describe all solutions like this). One way of finding such explicit solutions is to reduce the equations to equations of lower dimension, preferably ordinary differential equations, which can often be solved exactly.

- Raissi et al. *Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations*. arXiv 1711.10561
- Raissi et al. *Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations*. arXiv 1711.10566
- Raissi et al. *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*. J. Comp. Phys. 378 pp. 686-707 DOI: 10.1016/j.jcp.2018.10.045

https://en.wikipedia.org/wiki/Nonlinear_partial_differential_equation

Non Linear PDEs are hard to solve!

Numerical solutions

Numerical solution on a computer is almost the only method that can be used for getting information about arbitrary systems of PDEs. There has been a lot of work done, but a lot of work still remains on solving certain systems numerically, especially for the Navier–Stokes and other equations related to weather prediction.

- Raissi et al. *Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations*. arXiv 1711.10561
- Raissi et al. *Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations*. arXiv 1711.10566
- Raissi et al. *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*. J. Comp. Phys. 378 pp. 686-707 DOI: 10.1016/j.jcp.2018.10.045

https://en.wikipedia.org/wiki/Nonlinear_partial_differential_equation

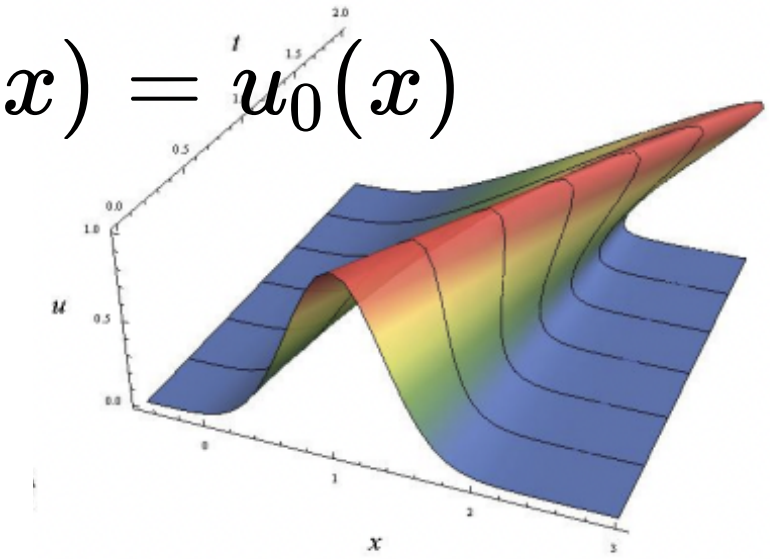
PINN

Burgers equation:

second order non-linear PDE

$$\partial_t u + u\,\partial_x u - \nu\,\partial_{xx} u = 0,$$

$$u(0, x) = u_0(x)$$



Applications of Burgers eq:

shock weave formation, turbulence, the weather problem, traffic flow and acoustic transmission

$$u : [0, T] \times D => \mathbb{R}$$

$$(t, x) \in (0, T] x D$$

x : spatial coordinate

t : temporal coordinate

$u(x, t)$: speef of fluid at x,t

ν : viscosity

Burgers equation:

second order non-linear PDE

$$\partial_t u + u\,\partial_x u - (0.01/\pi)\,\partial_{xx} u = 0,$$

How to solve analytically

https://www.youtube.com/watch?v=5ZrwxQr6aV4

Domain

$$(t, x) \in (0, 1] \times (-1, 1),$$

$$x \in [-1, 1],$$

$$t \in (0, 1]$$

Boundary Conditions

$$u(0, x) = -\sin(\pi\, x),$$

$$u(t, -1) = u(t, 1) = 0.$$

PINN

Burgers equation:

second order non-linear PDE

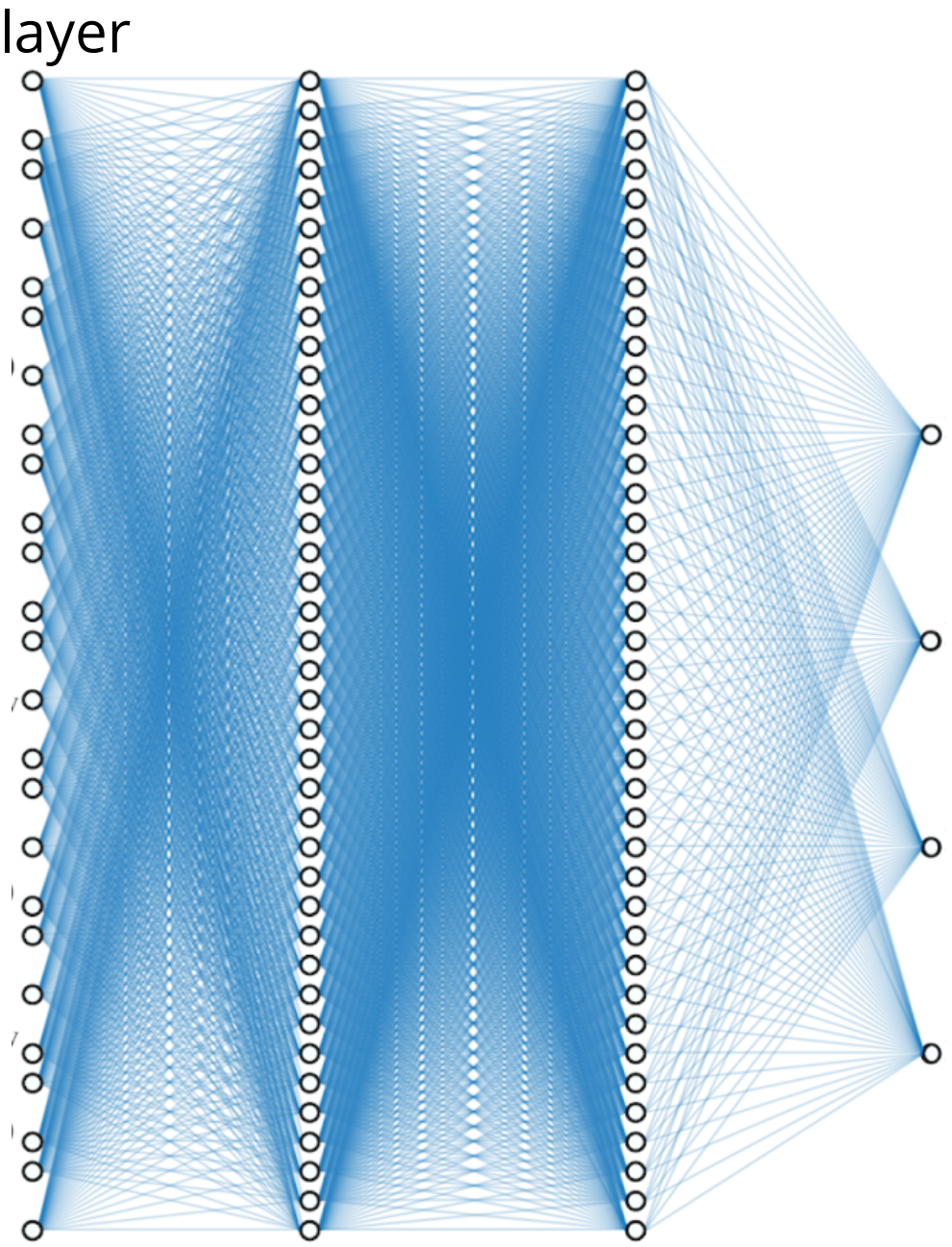
$$\partial_t u + u\,\partial_x u - (0.01/\pi)\,\partial_{xx} u = 0,$$

How to solve analytically

https://www.youtube.com/watch?v=5ZrwxQr6aV4

https://www.youtube.com/embed/nFBvAd81lds?enablejsapi=1

PINN

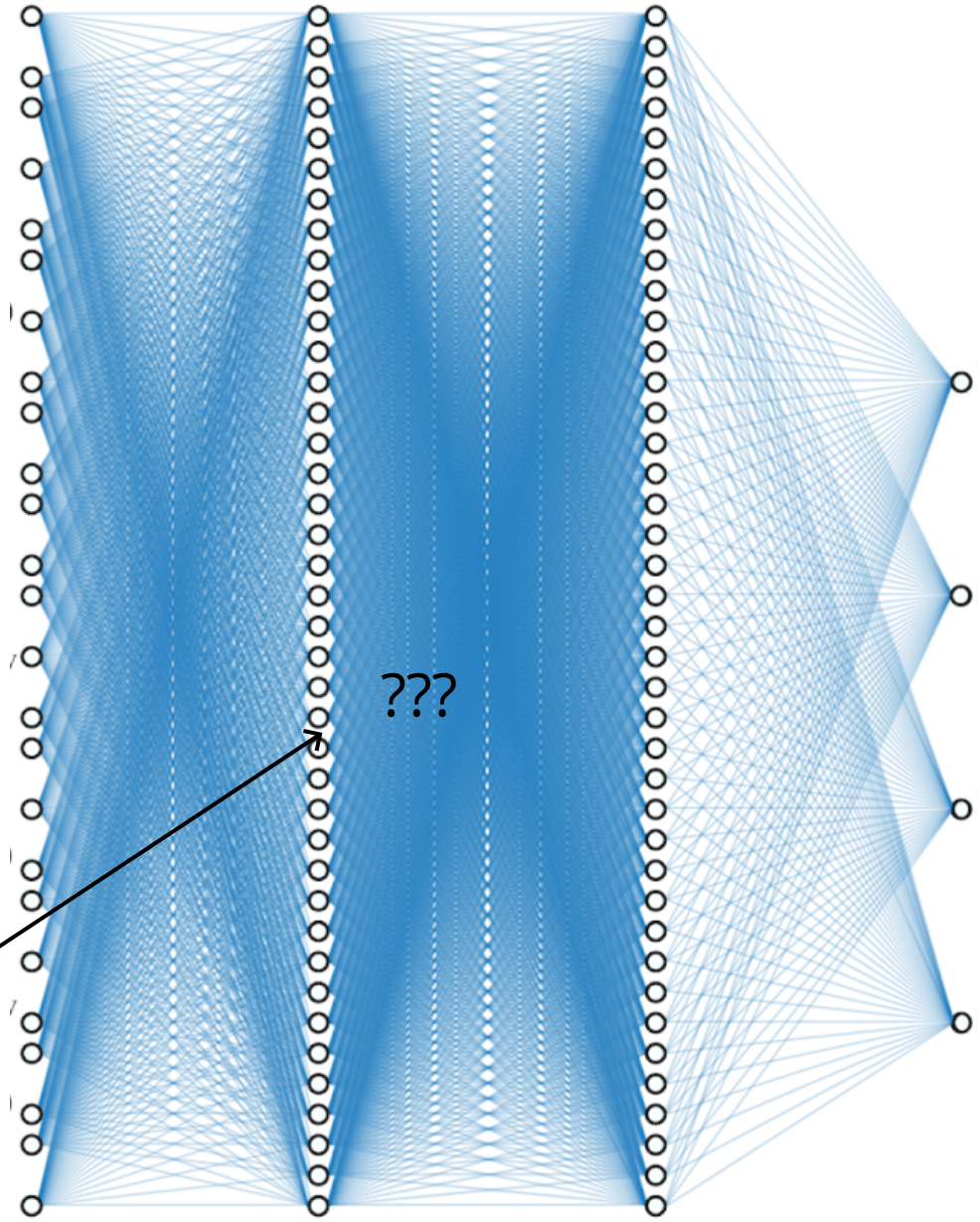


- Provide training points at the boundary with calculated solution (trivial cause we have boundary conditions)

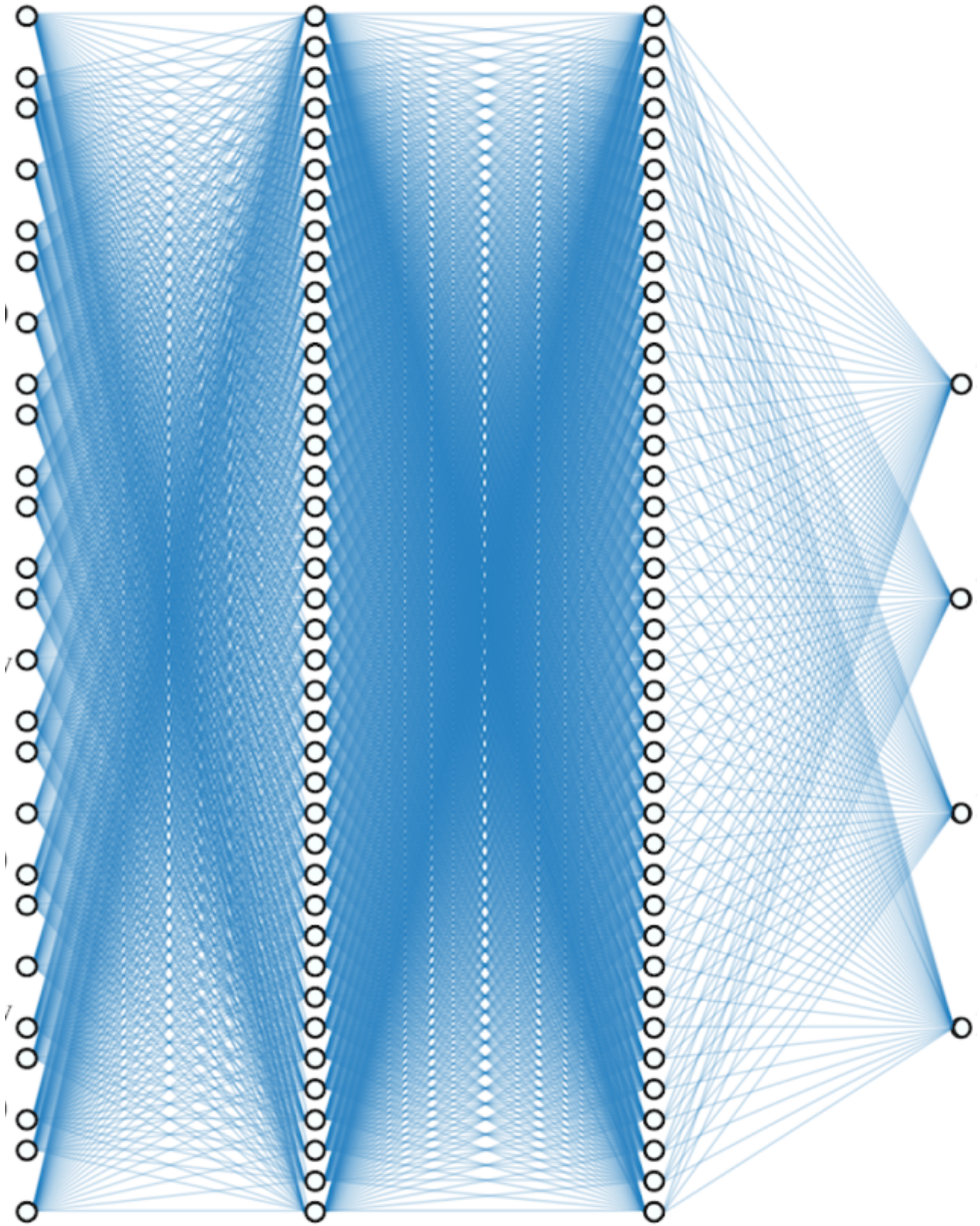
PINN

- Provide training points at the boundary with calculated solution (trivial cause we have boundary conditions)

- Provide the physical constraint: make sure the solution satisfies the PDE



PINN



- Provide training points at the boundary with calculated solution (trivial cause we have boundary conditions)

- Provide the physical constraint: make sure the solution satisfies the PDE

via a modified loss function that includes residuals of the prediction and residual of the PDE

PINN

- Provide training points at the boundary with calculated solution (trivial cause we have boundary conditions)

- Provide the physical constraint: make sure the solution satisfies the PDE

via a modified loss function that includes residuals of the prediction and residual of the PDE

$$\text{loss} = L2 + PDE =$$
$$\sum (u_\theta - u)^2 +$$
$$(\partial_t u_\theta + u_\theta \, \partial_x u_\theta - (0.01/\pi) \, \partial_{xx} u_\theta)^2$$

https://maziarraissi.github.io/PINNs/

PINN

https://github.com/fedhere/MLPNS_FBianco/blob/main/PINN/PINN_Burgers.ipynb

- Provide training points at the boundary with calculated solution (trivial cause we have boundary conditions)

- Provide the physical constraint: make sure the solution satisfies the PDE

via a modified loss function that includes residuals of the prediction and residual of the PDE

$$\text{loss} = L2 + PDE =$$
$$\sum (u_\theta - u)^2 +$$
$$\partial_t u_\theta + u_\theta \, \partial_x u_\theta - (0.01/\pi) \, \partial_{xx} u_\theta$$