MODELS FOR SPARSE INTERACTIONS: CNNs AND GNNs

S. Giagu / A. Ciardiello - Sapienza Università di Roma, INFN, and ISS

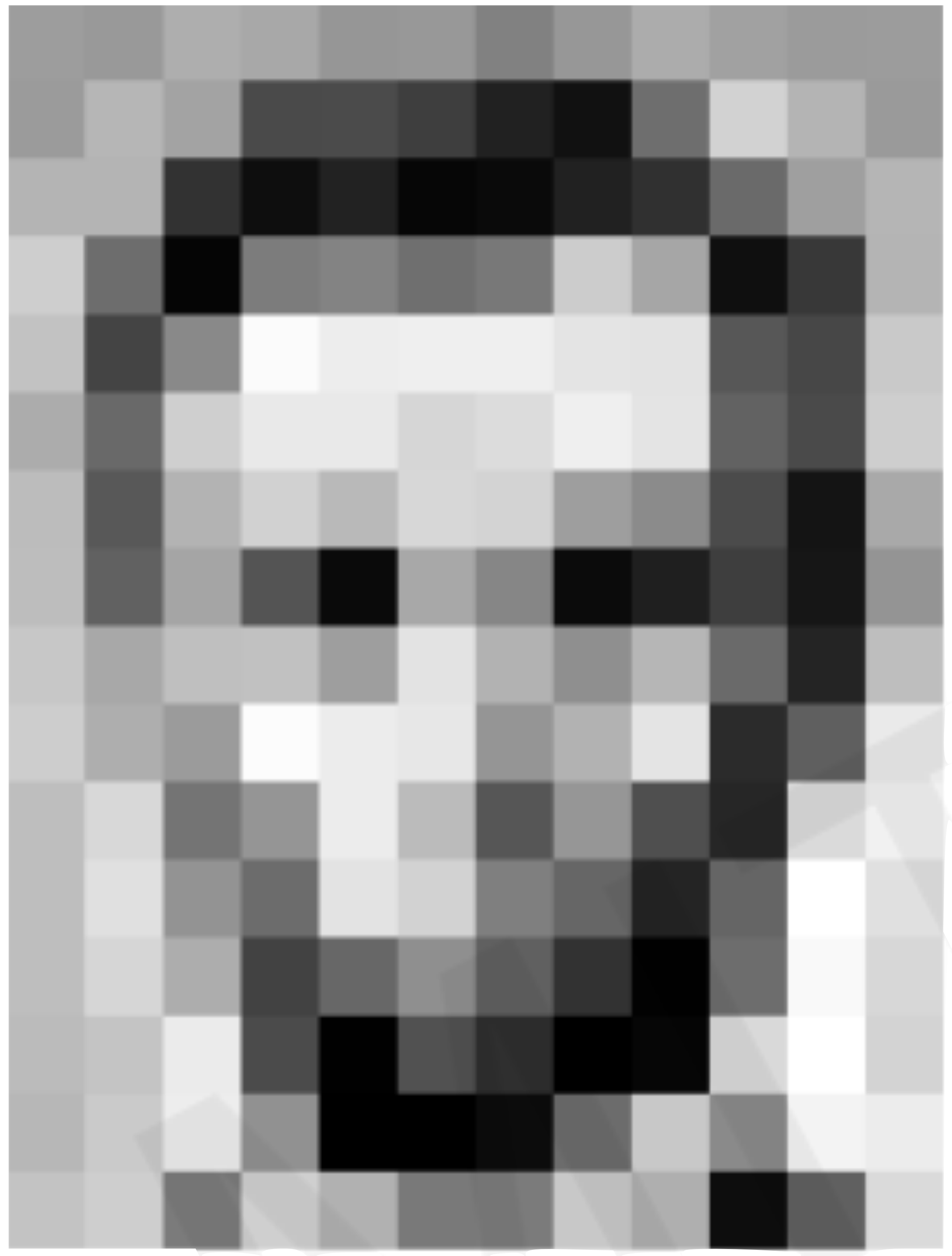
AIPHY 2024 - Monopoli - 29.9-5.10.2024

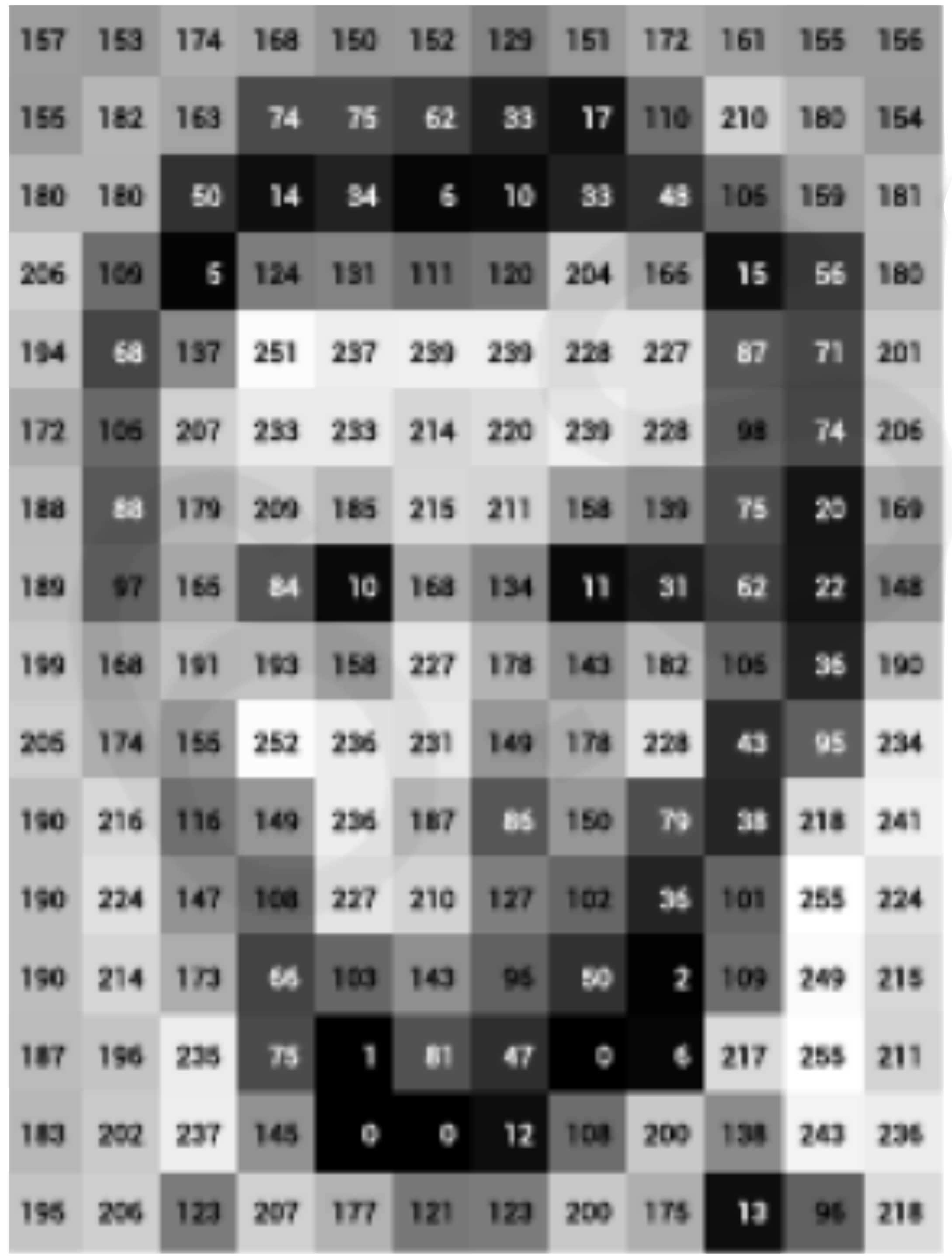
SAPIENZA
UNIVERSITÀ DI ROMA

CNNs

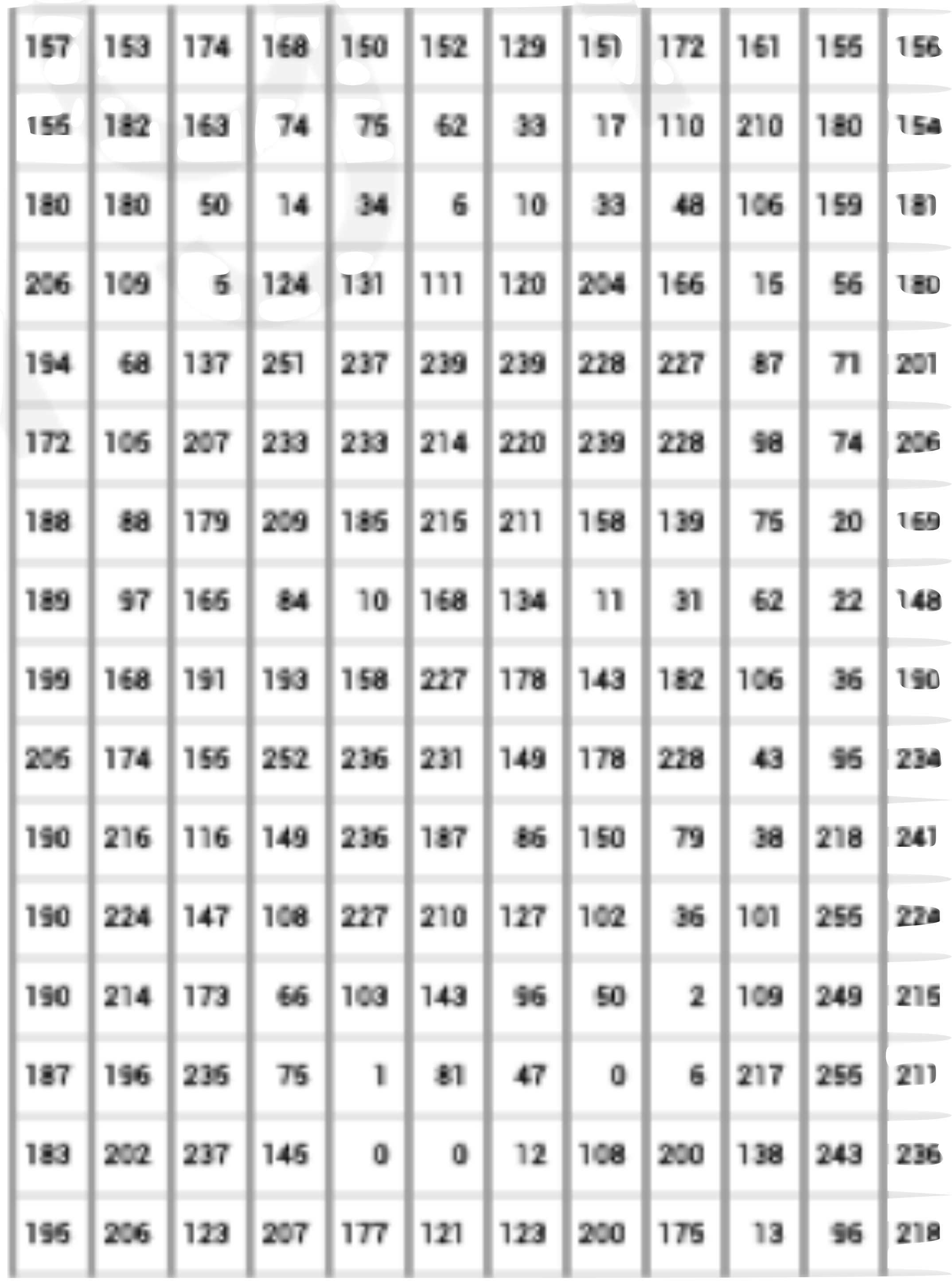
HOW AN ANN “SEE” AN IMAGE …

images for a computer are essentially meshes (tensors) of numbers





157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	85	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	95	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	106	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

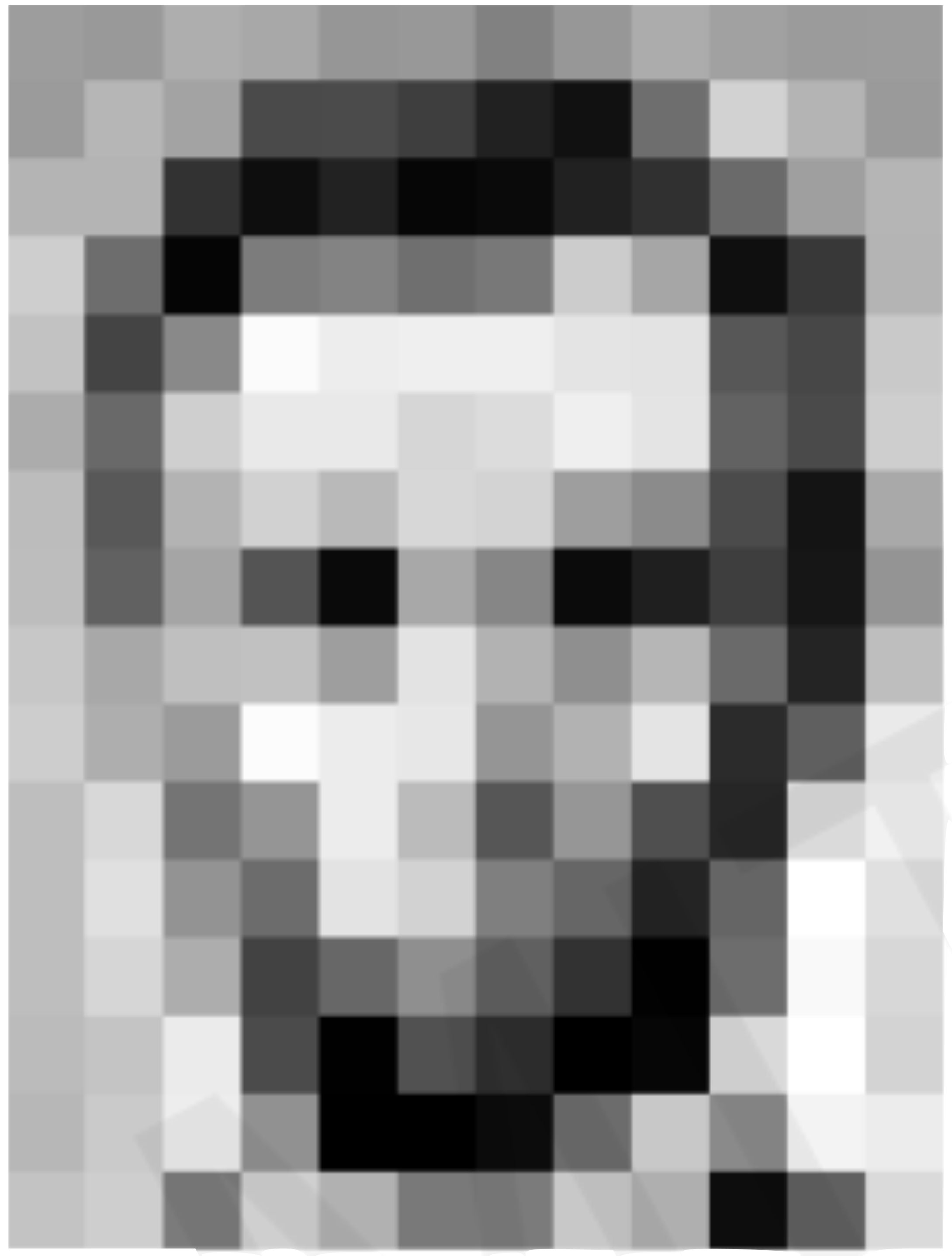
gray scale image with 8bit depth: 12x16x1 intensity ∈ [0,256]

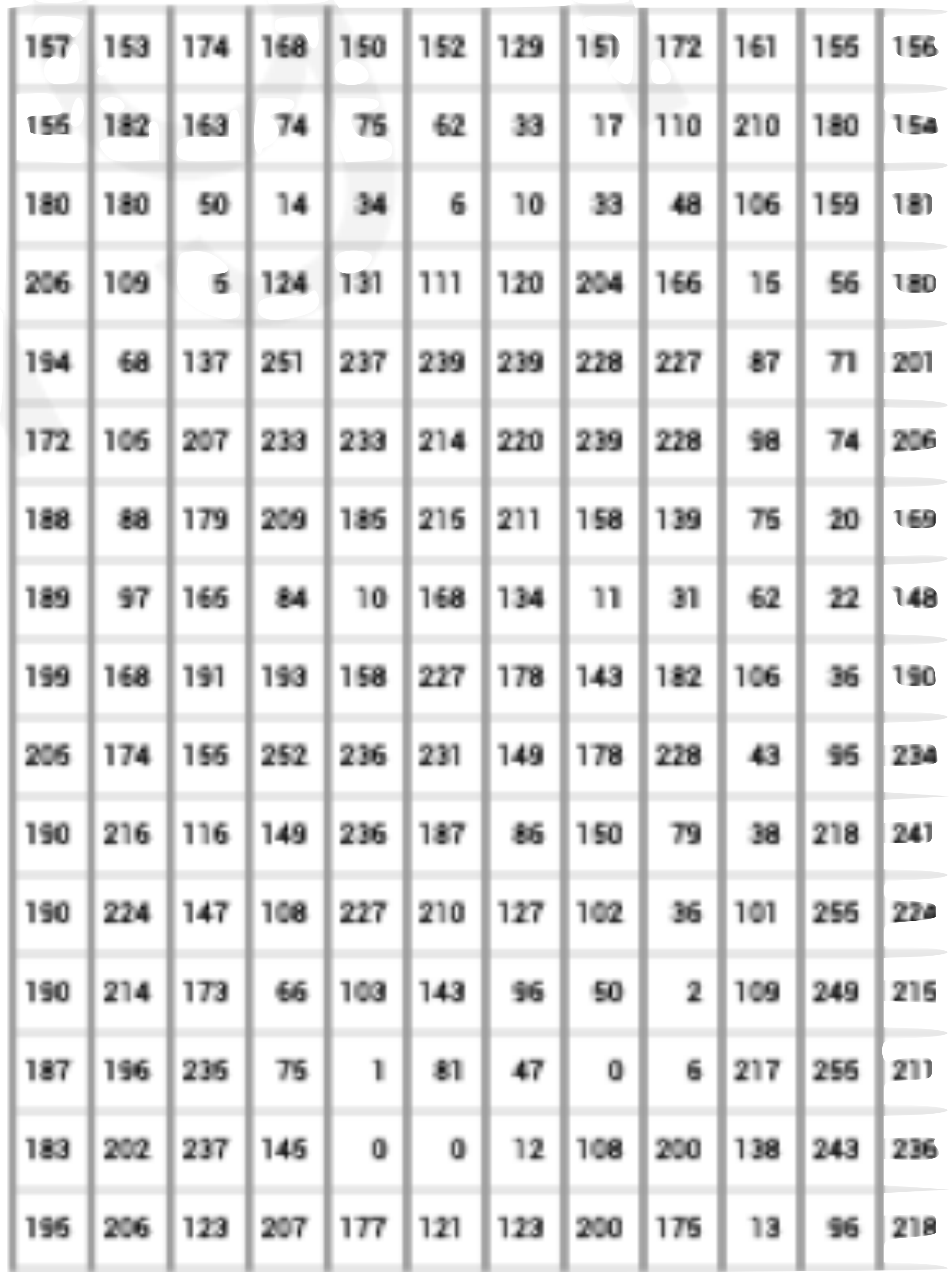
color image with n-bit depth: m_1xm_2x3 with each RGB intensity ∈ [0,2^n]

credit MIT AI course

HOW AN ANN “SEE” AN IMAGE …

Classification task:





157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

$$\begin{array}{lc} \text{Label} & \text{Probability} \\ \begin{array}{l} \text{Lincoln} \\ \text{Washington} \\ \text{Jefferson} \\ \text{Obama} \end{array} & \begin{bmatrix} 0.8 \\ 0.1 \\ 0.05 \\ 0.05 \end{bmatrix} \end{array}$$

credit MIT AI course

HOW AN ANN "SEE" AN IMAGE …

classification is performed by identifying as many local features as possible present in the specific image of the objects we want to recognize ...

NOSE,
MOUTH,
EYES,
…

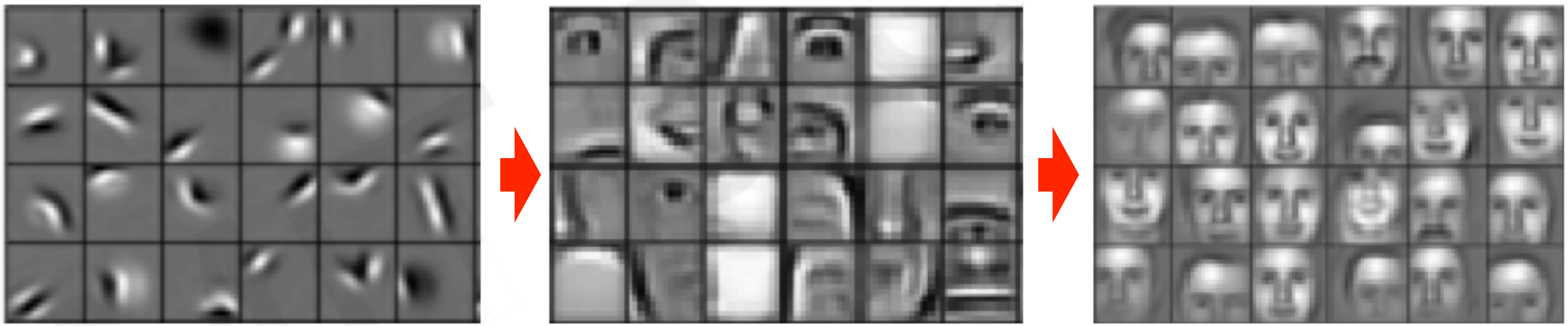
HEADLIGHTS,
WHEELS,
PLATE,
…

DOOR,
STAIRS,
WINDOWS,
…

credit MIT AI course

HOW AN ANN “SEE” AN IMAGE …

it is convenient to identify these characteristics in a hierarchical way ...



edges, spots, …

noses, eyes, ears, mouths, …

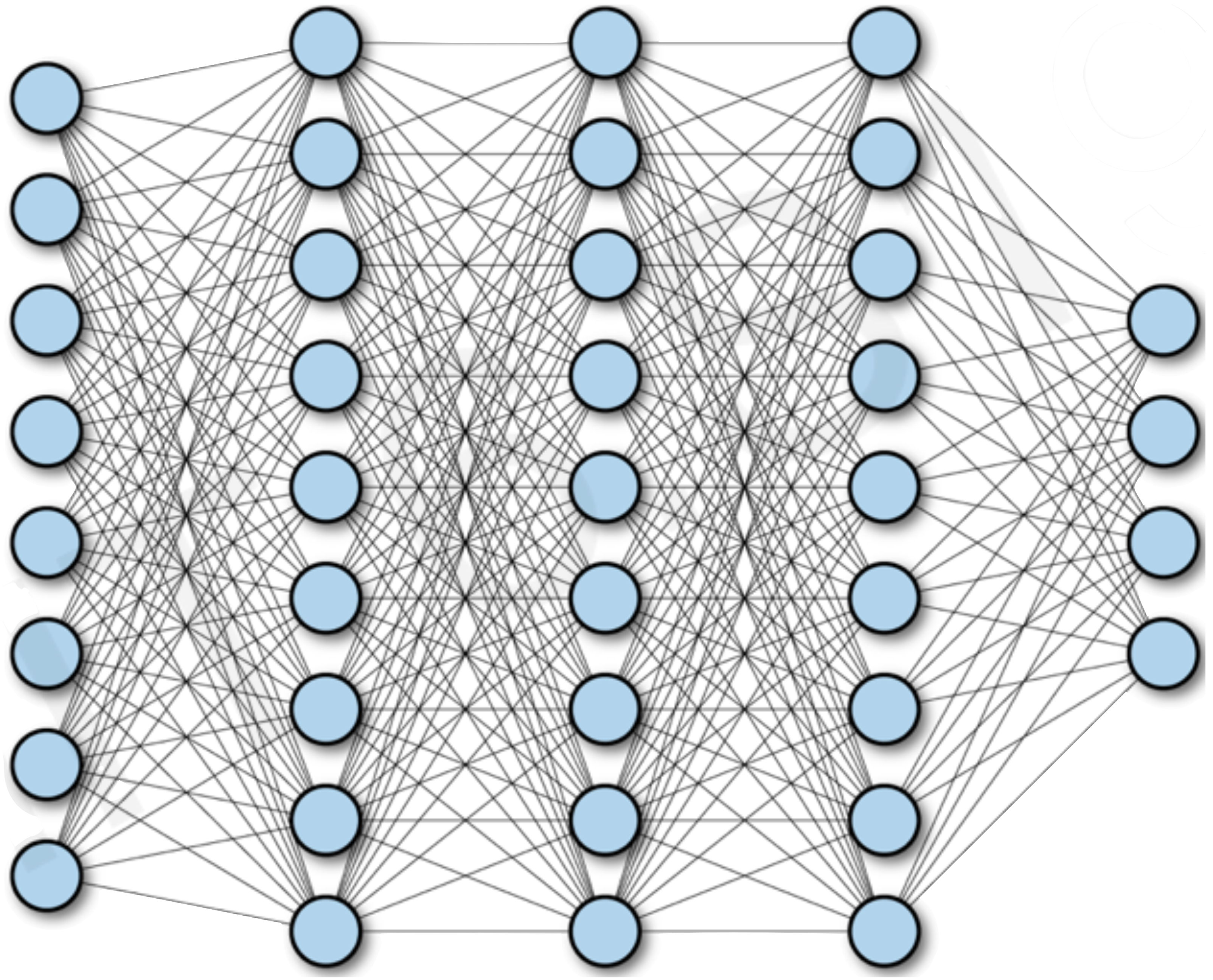
entire faces…

credit MIT AI course

HOW AN ANN “SEE” AN IMAGE …

we already know that a DNN is able to learn the desired hierarchical representations:

x 1D array of the pixel values (features)



P(label | **x**)

Issues:

- it needs a huge number of learnable parameters (weights) → hard to train, overfitting …
- it does not use any local spatial information
- too much flexibility results in arbitrarily complex models for which is very hard to achieve generalisation

A viable solution: Convolutional Neural Networks

ARCHITECTURES FOR VISION: CNN

- a remedy to facilitate the training of DNN is to introduce task independent priors, i.e. parts of the model that are not learnt (also called inductive relational biases), motivated by general properties and structures observed in data

- Convolutional NN is a specific DNN architecture designed to excel in image recognition tasks
  - acts directly on the images (raw "pixel" information organised in a fixed size mesh)
  - the inductive bias is based on assumptions on the properties of the input data:
    - translation equivariance: sub-features in the image remain the same in different points of the image
    - self-similarity: two or more identical sub-features can be recognised with a single filter that identifies one of the sub-features
    - compositionality: a complex feature made of several sub-features can be recognised by identifying only few sub-features
    - locality of the features: to identify a sub-feature it takes just a few pixels concentrated in a small portion of the image itself

CNN

- CNN implementation idea: apply NN layers called convolutional filters that operate on the input by recognising the local sub-features present there
  - several groups of filters use shared parameters (weights) and sequentially analyse all portions of the image
  - weights of the filters are not fixed but are learned

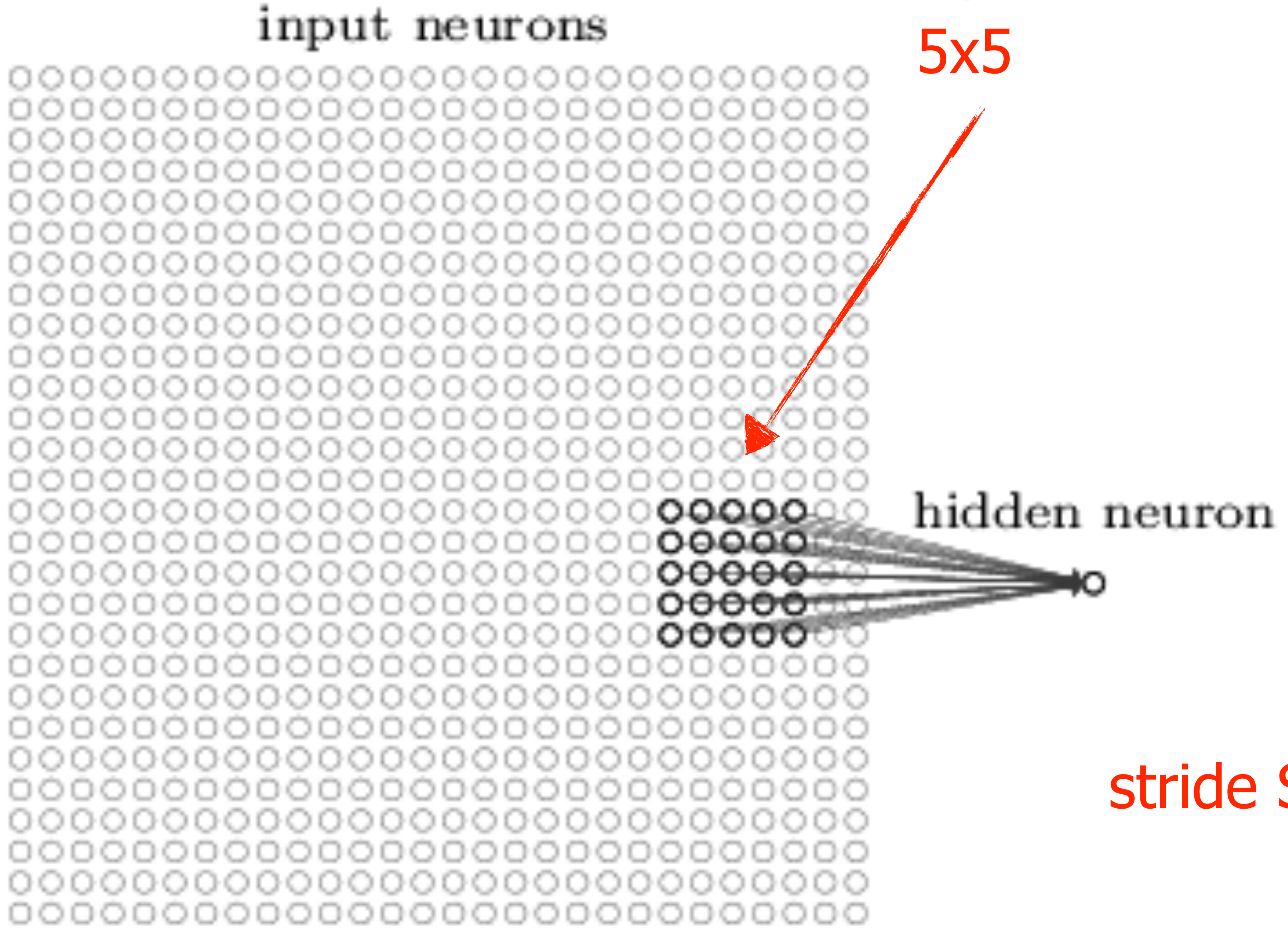
CNNs learn from the training data sample the best set of filters to solve the task at hand given the chosen architecture

CONVOLUTIONAL FEATURE EXTRACTION LAYER

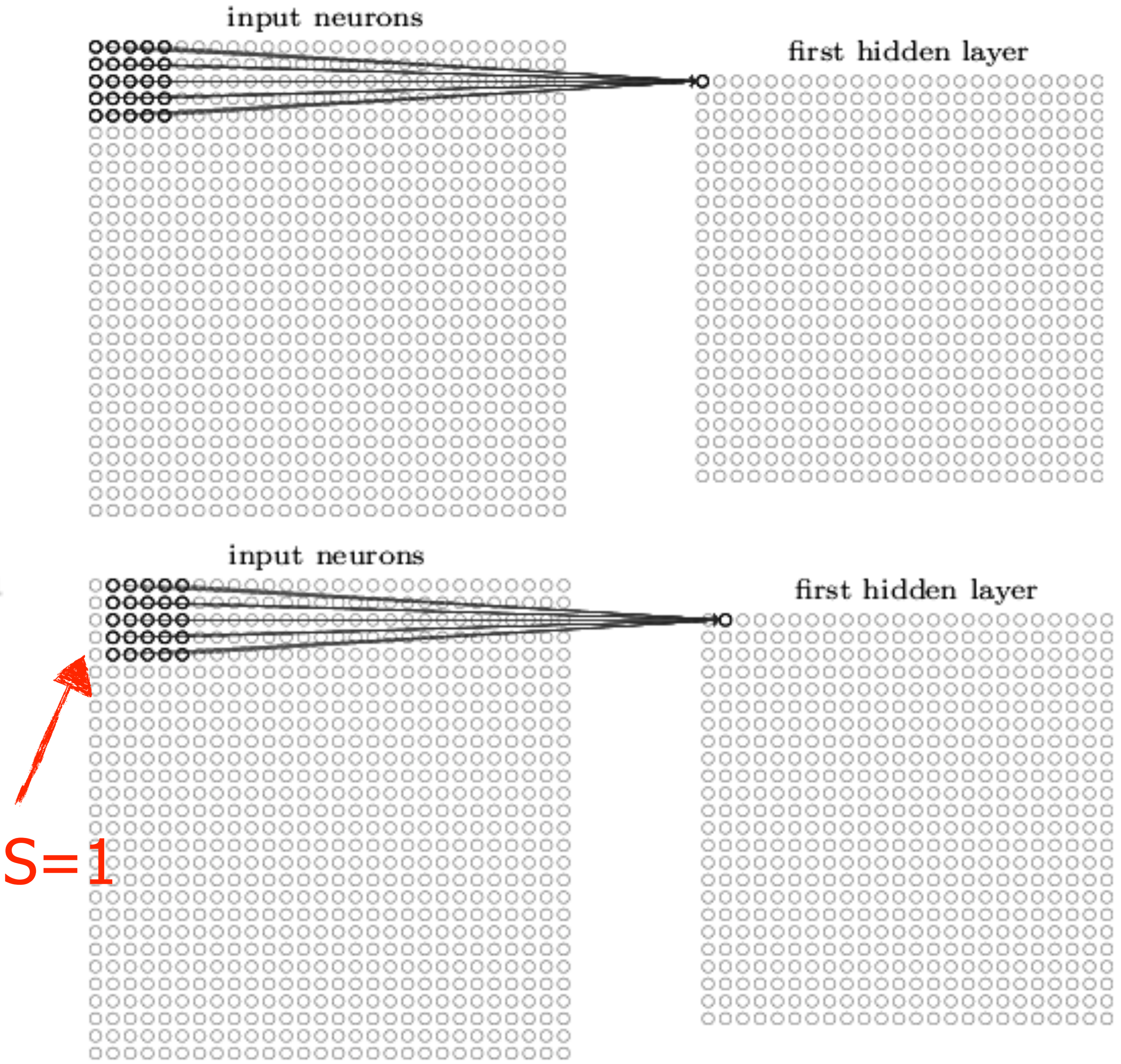
- used to identify similar features that are present in different position of the image

- based on three basic ideas:
  - local receptive field (filters or convolutional kernels)
  - shared-weights kernels
  - pooling layers

- input neurons (one for each NxN pixels of the image) are NOT fully connected with all the neurons of the first hidden layer. Connections exist only for localised and small regions of the image called local receptive fields
- the local receptive field is shifted through the whole image: for each shifted receptive field there will be an hidden neuron in the hidden layer
- multiple filters are used in order to identify different features, and combined in a stack of layers in output

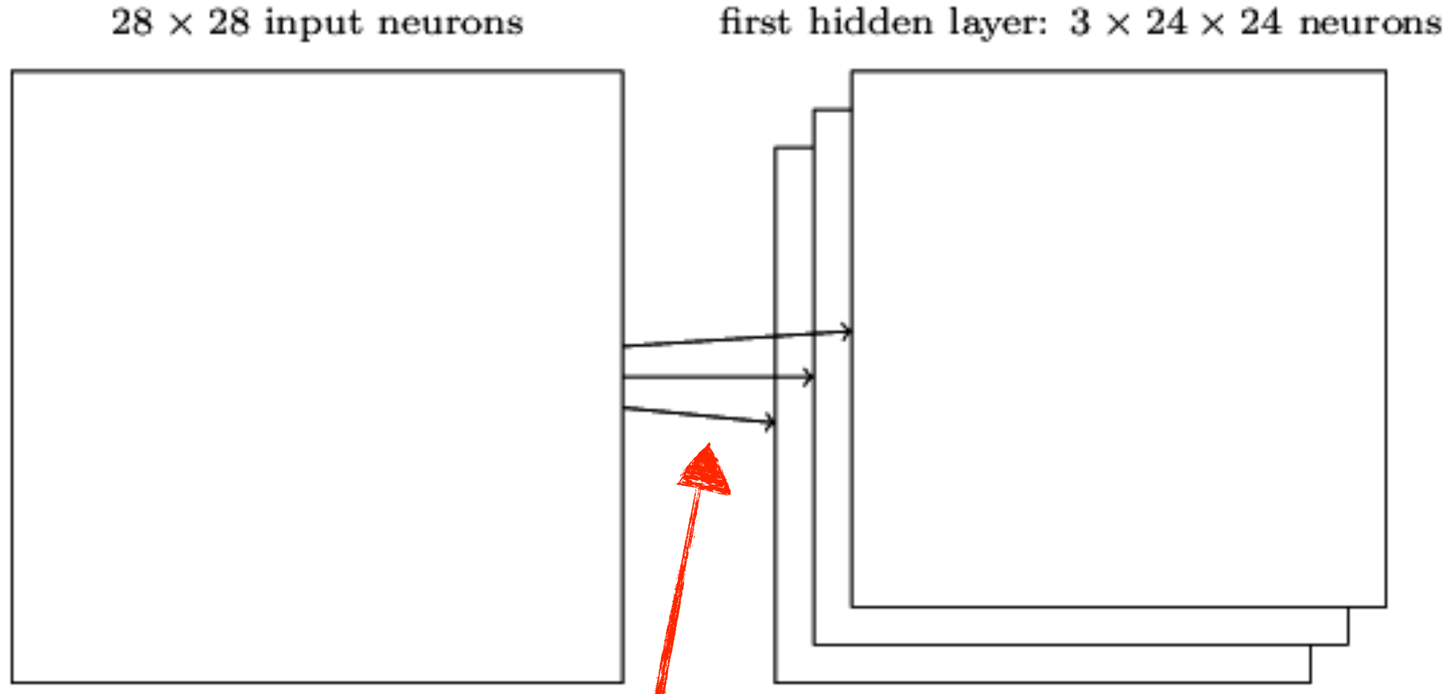


local receptive field
5x5



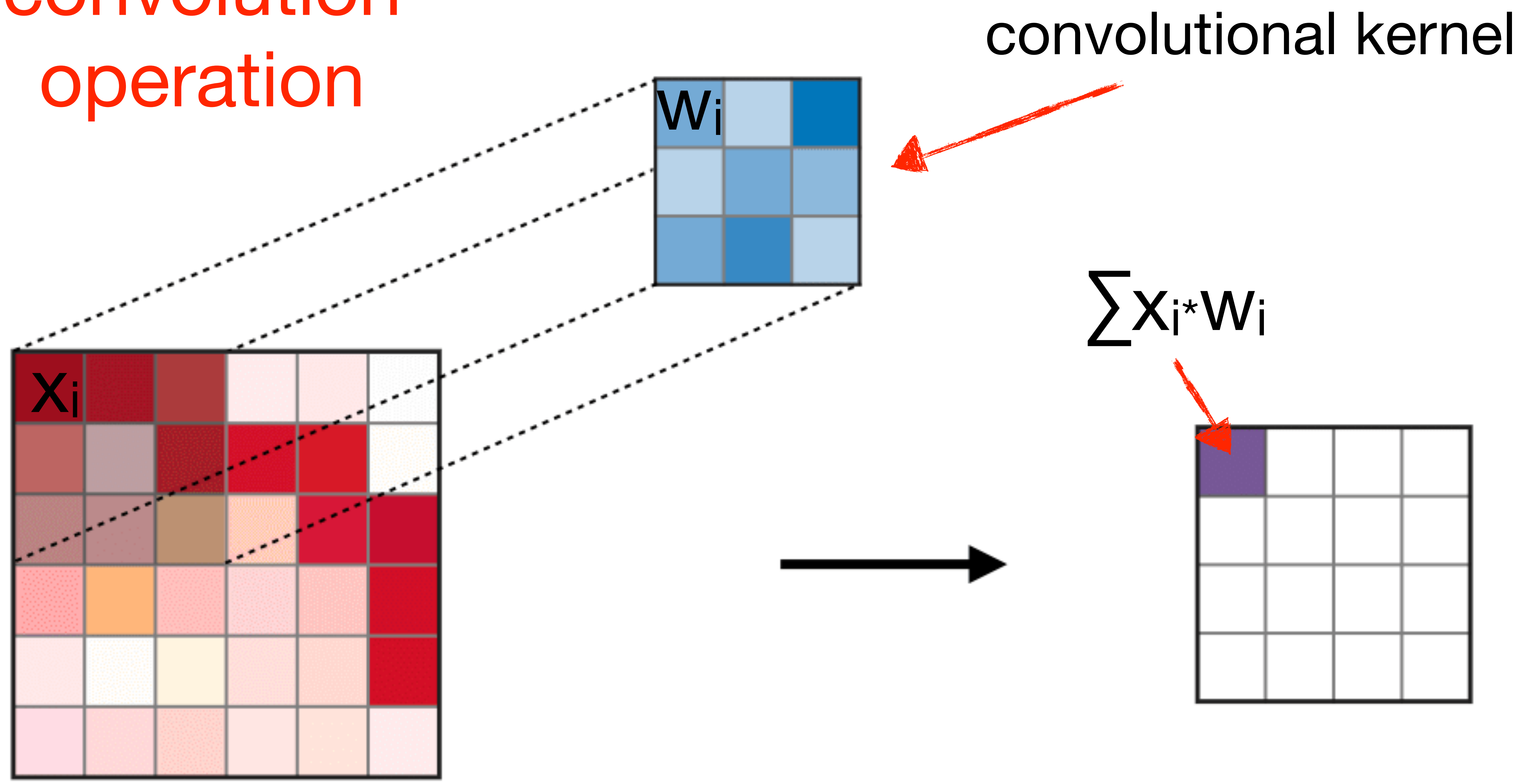
stride S=1

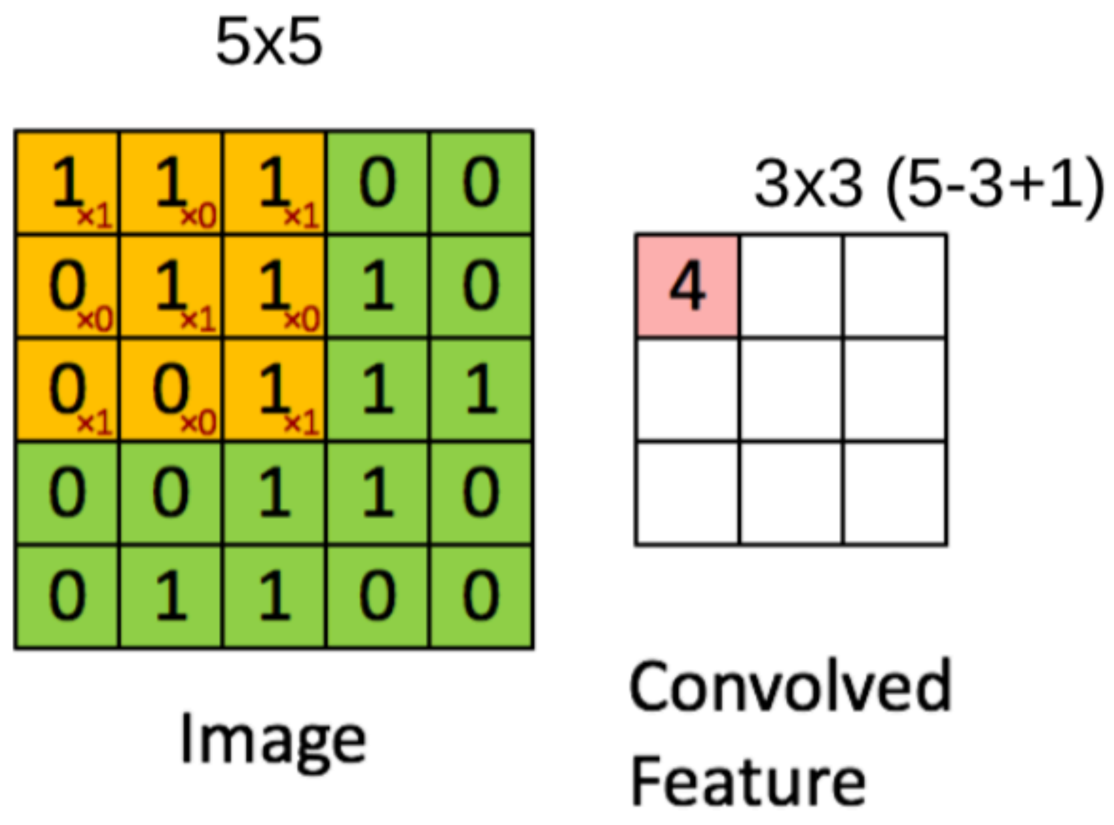
24 = 28-5+1

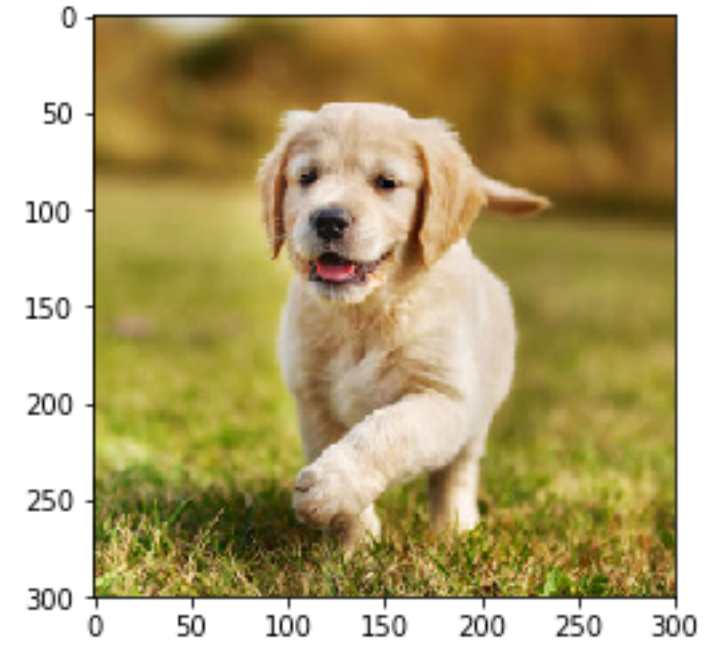


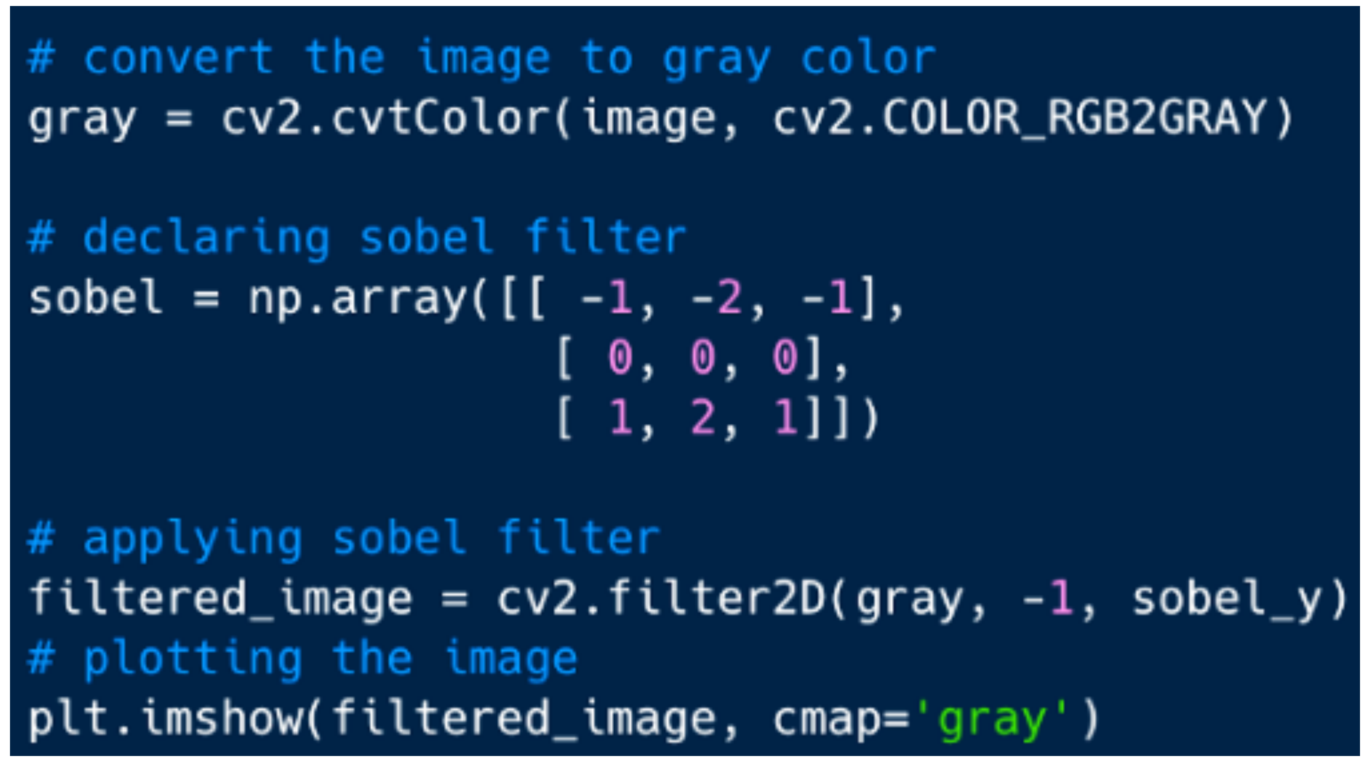
3 local receptive fields 5x5

convolution operation







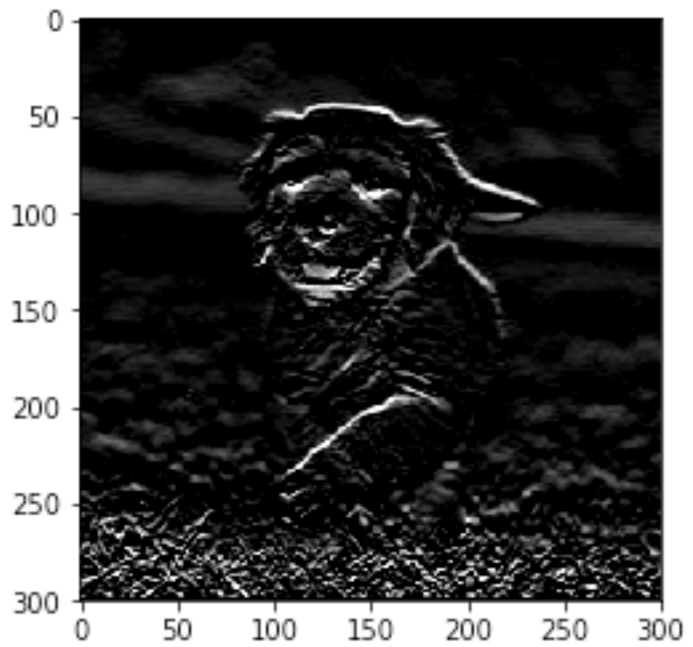


```
# convert the image to gray color
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

# declaring sobel filter
sobel = np.array([[ -1, -2, -1],
                  [ 0, 0, 0],
                  [ 1, 2, 1]])

# applying sobel filter
filtered_image = cv2.filter2D(gray, -1, sobel_y)
# plotting the image
plt.imshow(filtered_image, cmap='gray')
```

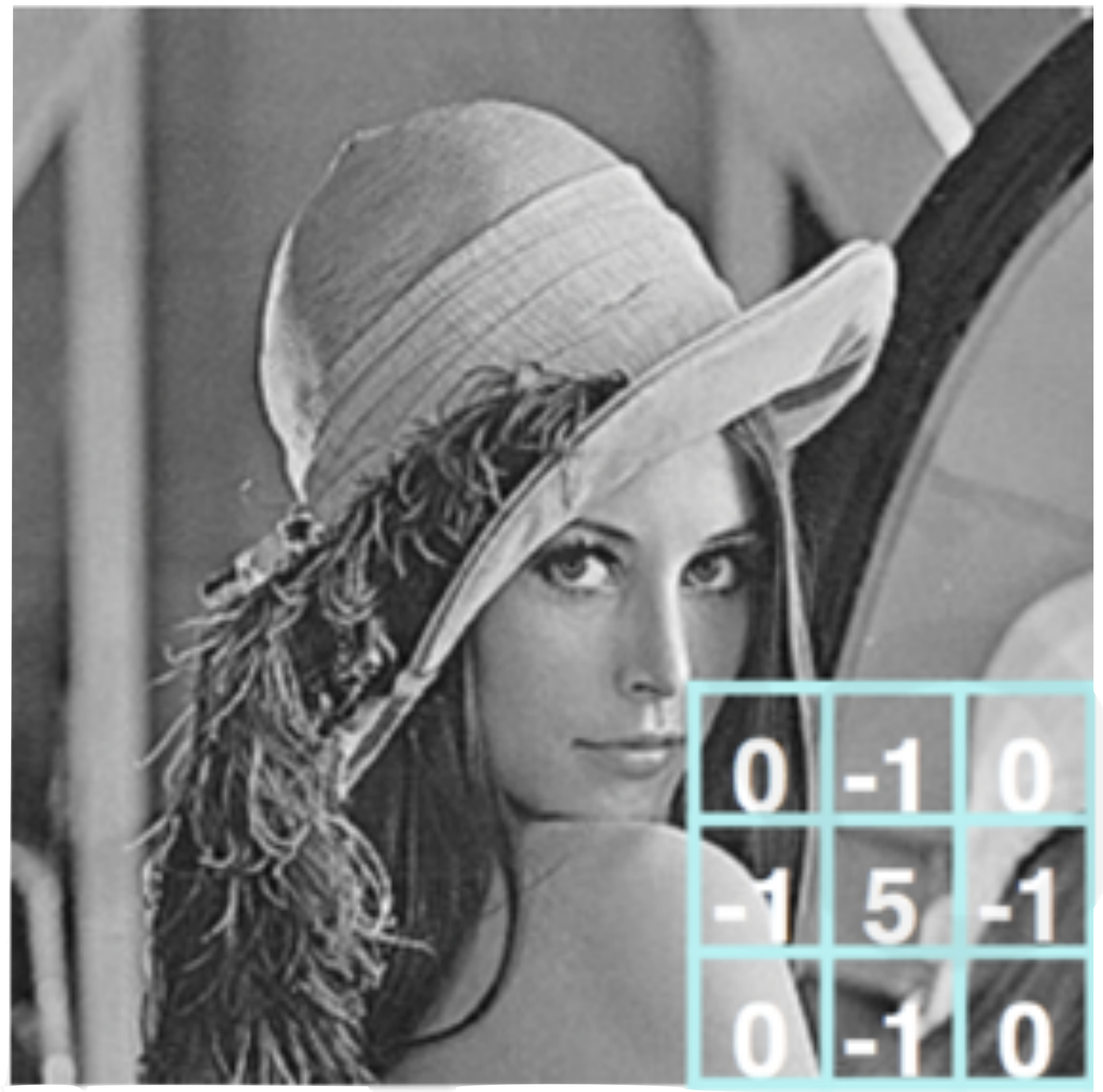
- shared-weights:
  - all the neurons produced by a filter share the same weights → all neurons of the hidden layer detect the same sub-feature, only in different regions of the image
  - as the CNN has to identify many sub-features: there are many convolutional kernels each one with an associated hidden layer: image input dimension (n,m,3) → output after a convolutional layer (k,l,d)
  - huge advantage wrt DNN: a much smaller number of weights to learn …



after the convolution operation, non linearity is applied to each neuron of the filtered image via an activation layer (ex. ReLU)

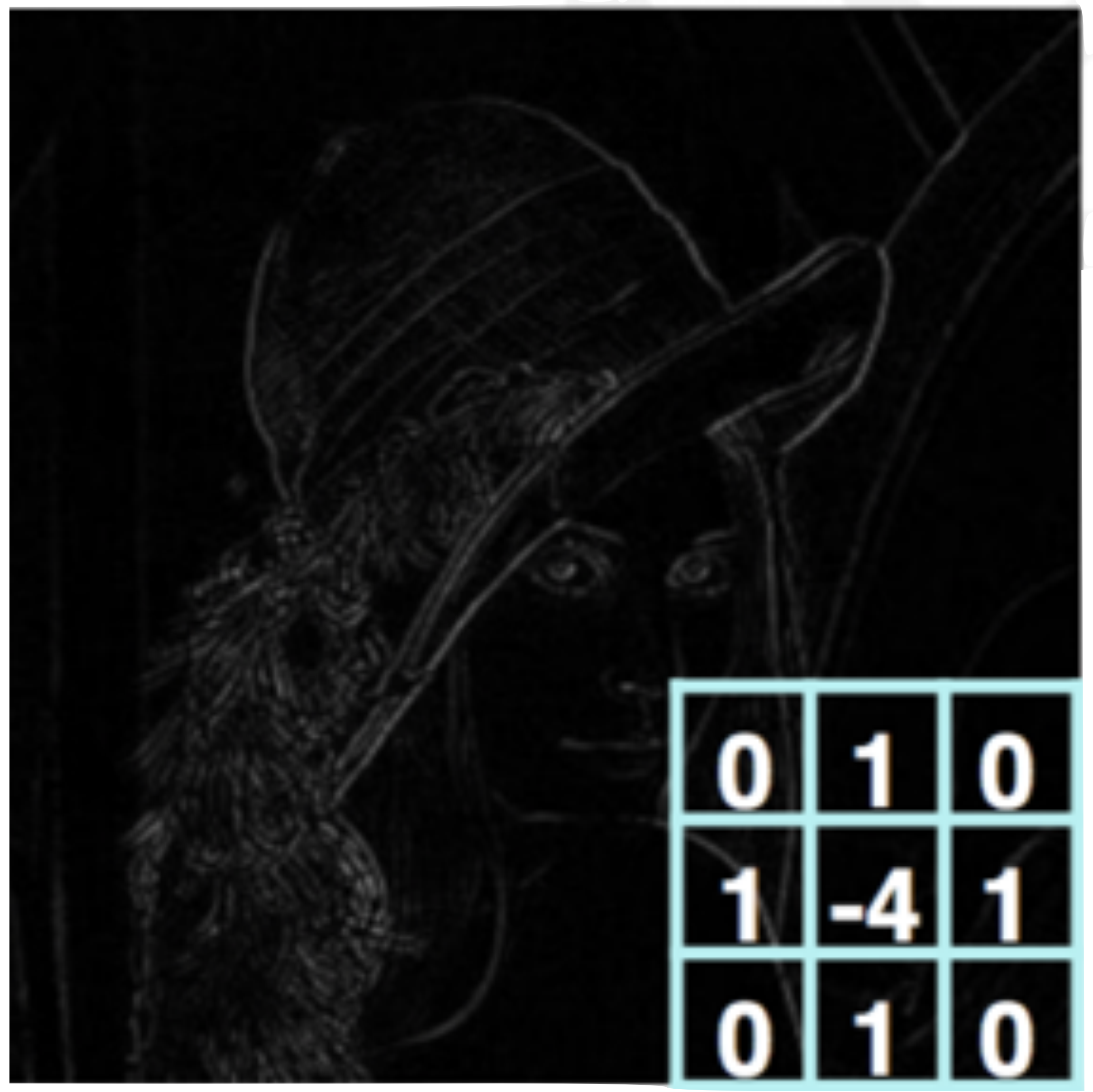
PRODUCED FEATURE MAPS VS KERNEL WEIGHTS

ORIGINAL



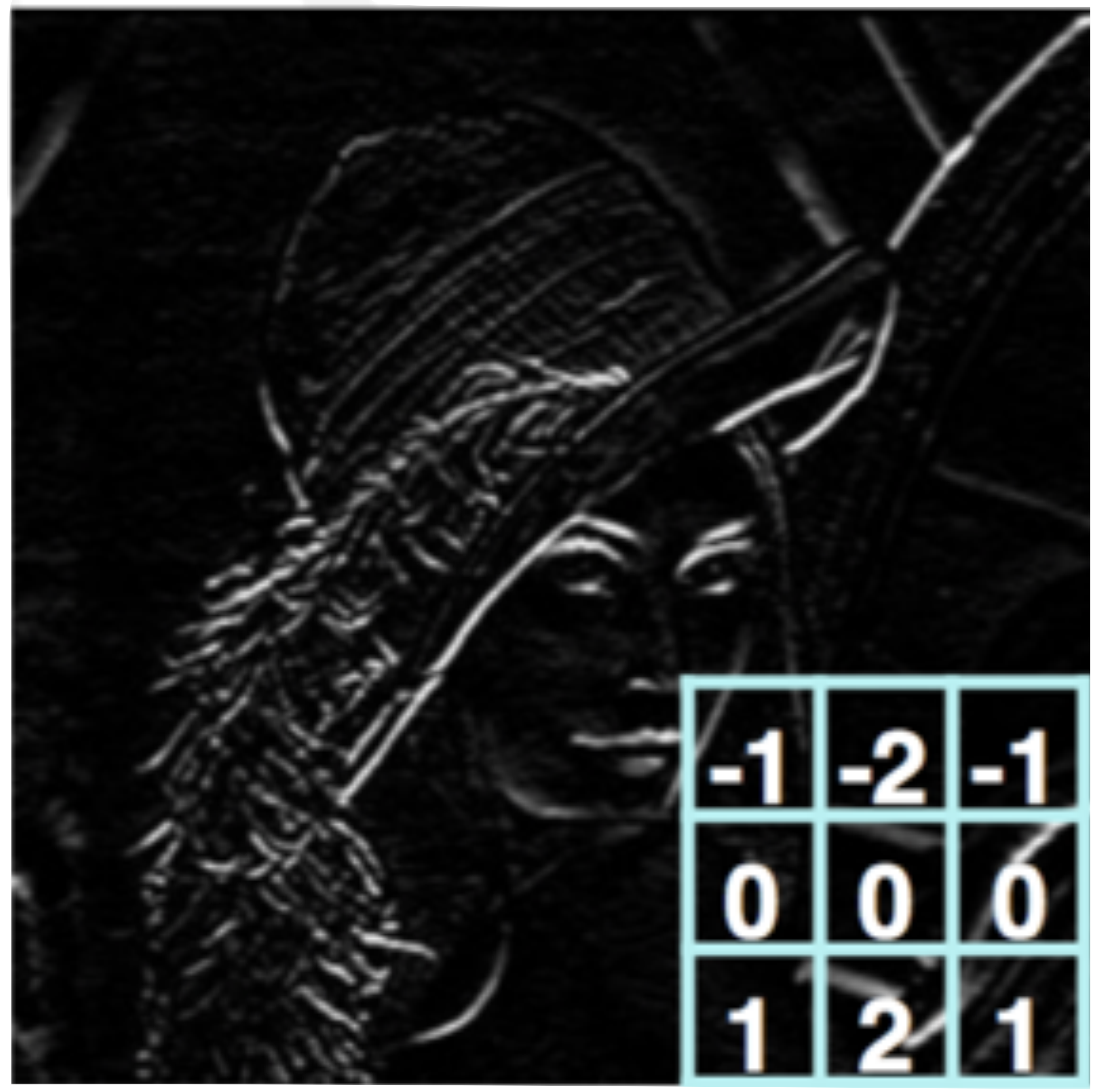
0	-1	0
-1	5	-1
0	-1	0

SHARPEN



0	1	0
1	-4	1
0	1	0

EDGE DETECT



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

STRONG EDGE DETECT

different weight values emphasize different characteristics of the image

DIGRESSION: NOTION OF SPARSITY OF INTERACTIONS

- the assumption of locality used in CNN is connected with the concept of sparsity of interactions
- this can be intuitively understood by considering the k-NN algorithm:

$$g(x) = \frac{1}{k} \sum_{i \in k-nn(x)} y_i$$

in a regression task returns the average of the values of the closest k-points according to a defined distance *d(x,xi)*

- g(x) is considered **"sparse" as only depends on k points of the entire dataset** (in a CNN k is the number of pixels involved in the convolution operation)

- the Nadaraya-Watson kernel estimator generalize the operation of selection the k-nn points extending the sum over all points weighting them with the distance d, and make it differentiable so usable with SGD:

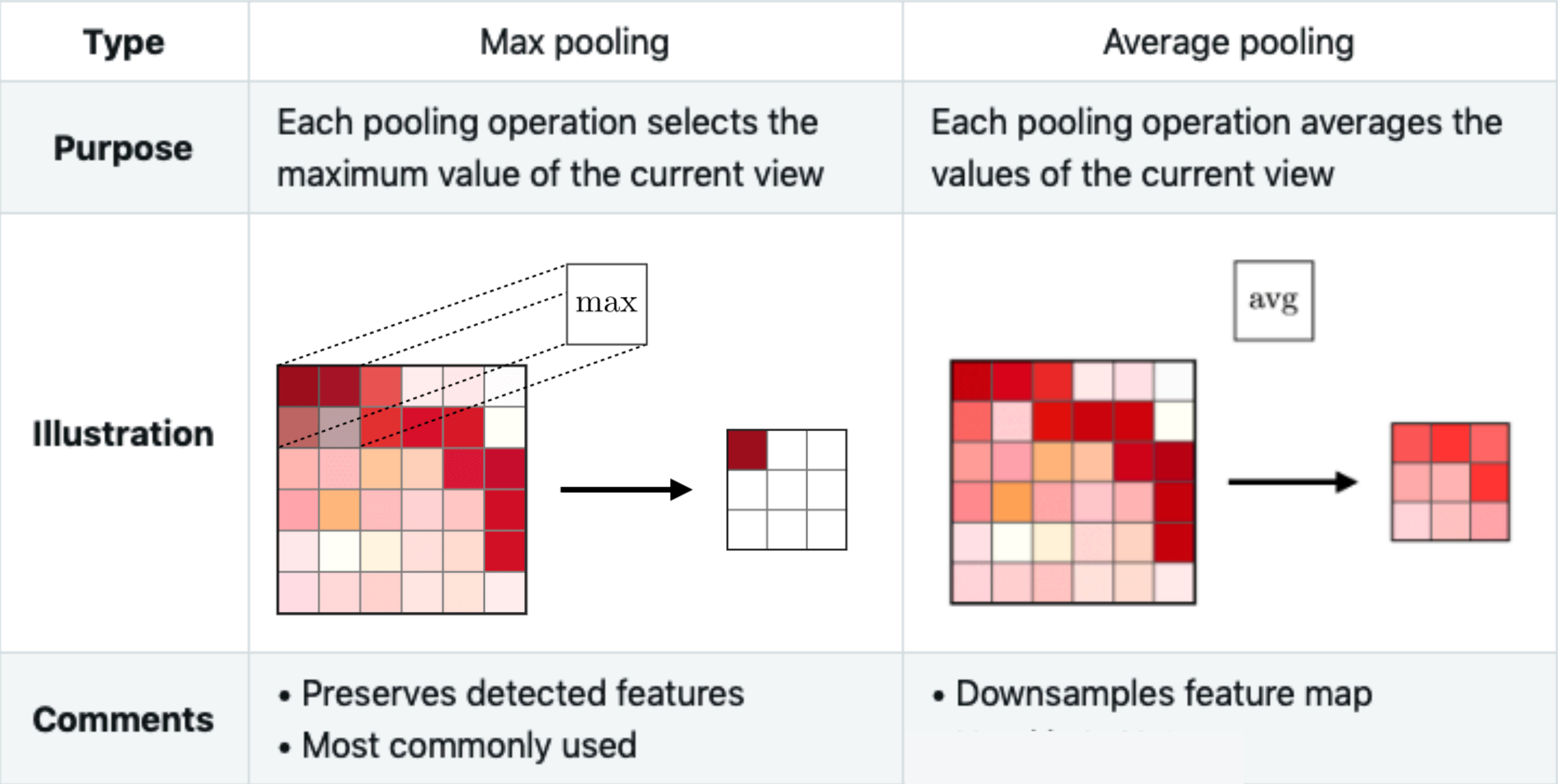
$$g(x) = \sum_i d(x, x_i)\, y_i = \sum_i e^{-\beta \|x - x_i\|_2}\, y_i$$

Nadaraya-Watson kernel estimator

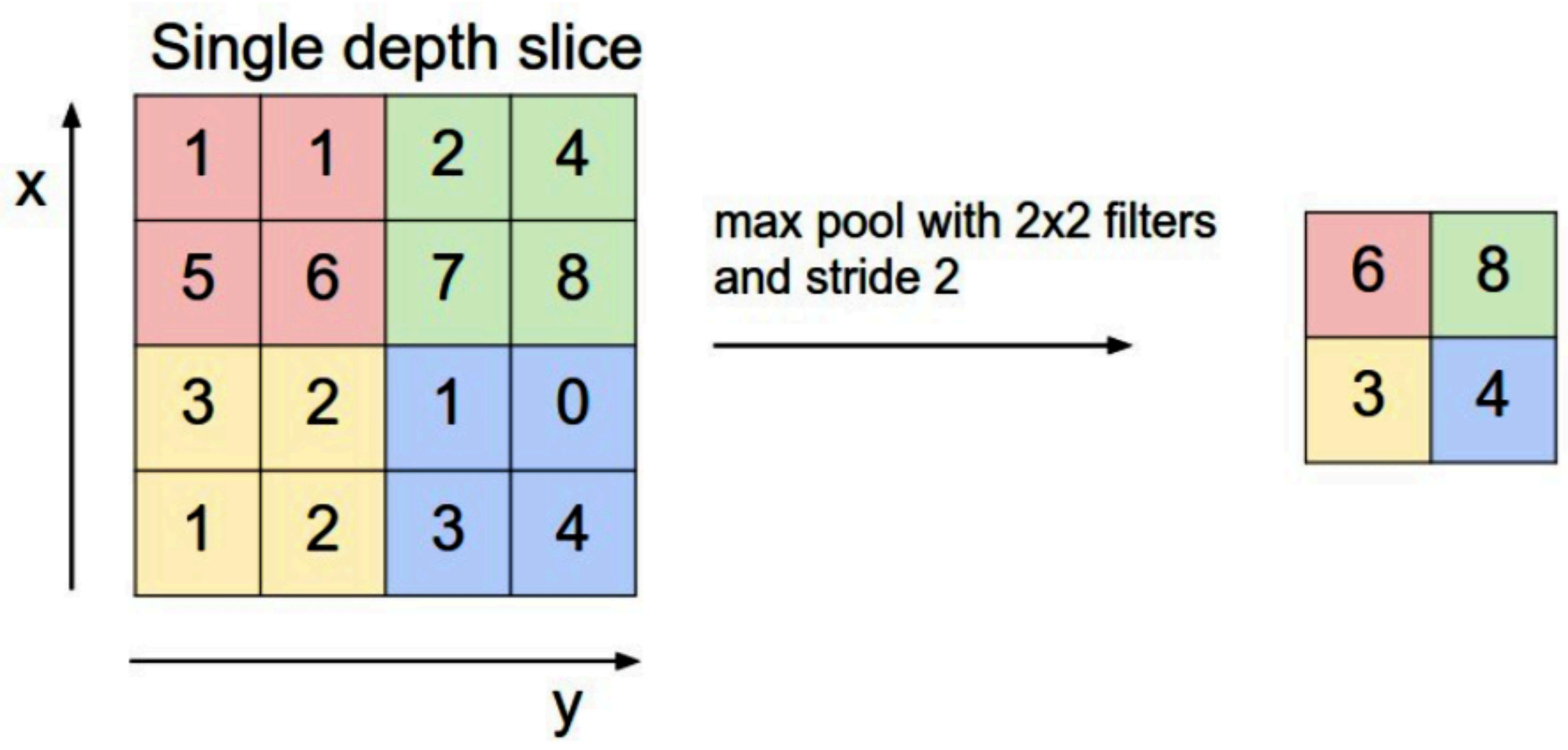
making the N-W a convex estimator by using softmax to weight the entries is called attention mechanism

• pooling layers:

- in addition of the convolution layers a CNN has also other layers called pooling layers, usually used after each convolution layer. They performs a downsampling operation: simplifying the information in output from the convolutional layer (less weights) and making the model less sensitive to small translations of the image
- motivated by the fact that once a sub-feature is found, to know the exact position is not as important as to know the relative position wrt the other sub-feature in the image



Type	Max pooling	Average pooling
Purpose	Each pooling operation selects the maximum value of the current view	Each pooling operation averages the values of the current view
Illustration	max	avg
Comments	• Preserves detected features • Most commonly used	• Downsamples feature map



Single depth slice

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

max pool with 2x2 filters and stride 2

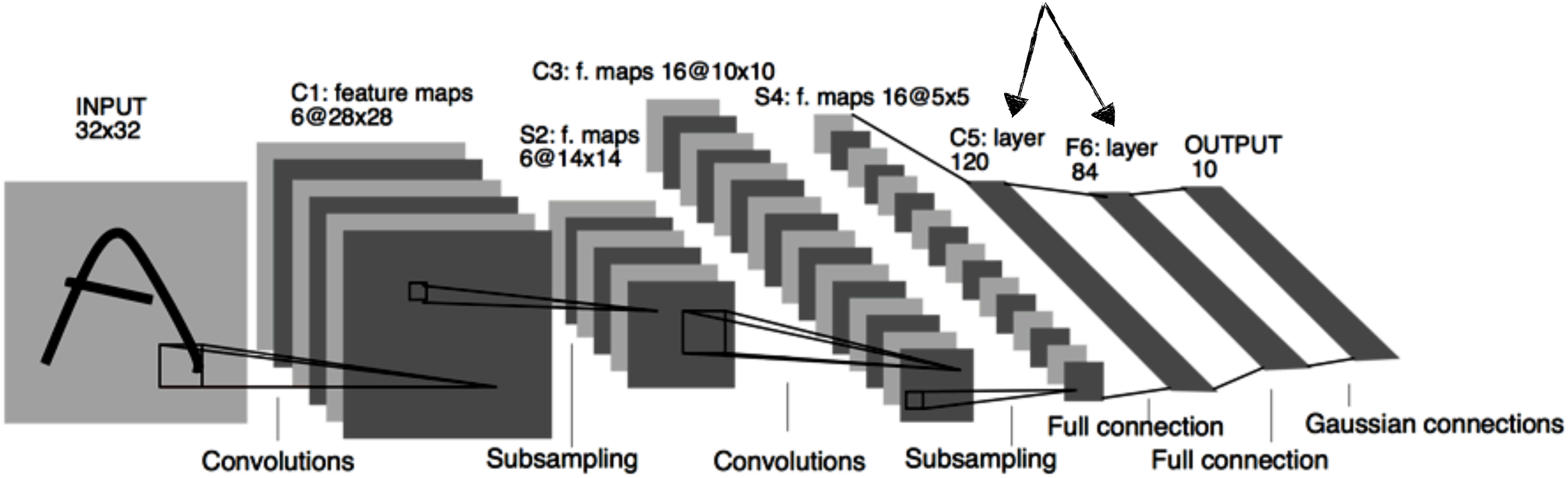
6	8
3	4

FULL CNN: CONV BLOCKS + DENSE MLP STAGE

- after the convolution blocks, the output of the convolutional layers can be connected via a flattening layer with one or more dense layers (DNN), that are used to optimise objectives: class scores (classification), mapping (regression), etc…

Example: LeNet (Yan LeCun 1989)

multi-staged CNN for classification: (Conv2D+MaxPooling)x2 + 2xDense + output layer (soft max)



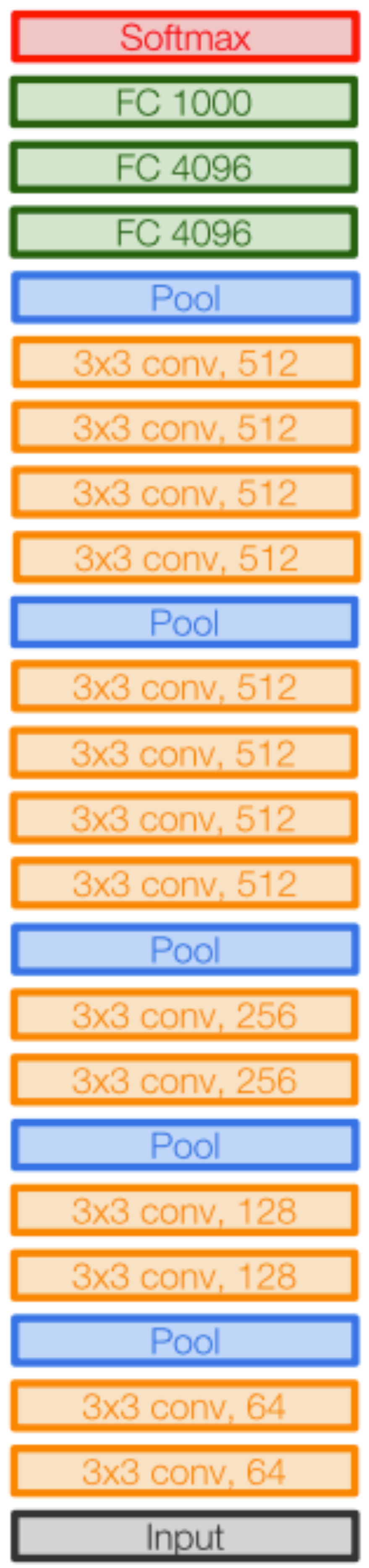
detects details (segments, arcs, …)

focus on overall shapes

maps high level representations to targets

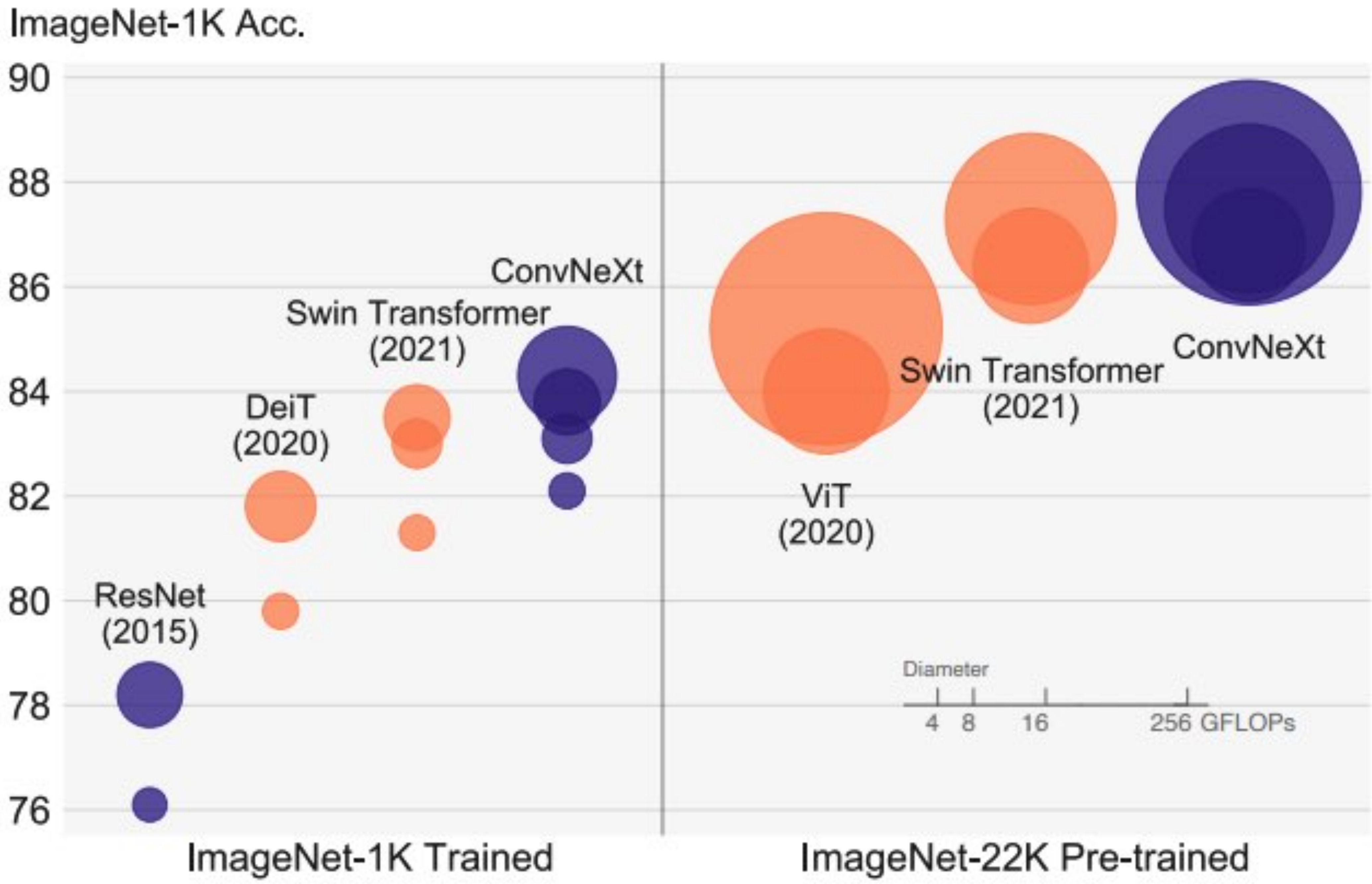
MODERN CNNs

philosophy: deeper is better



VGG19

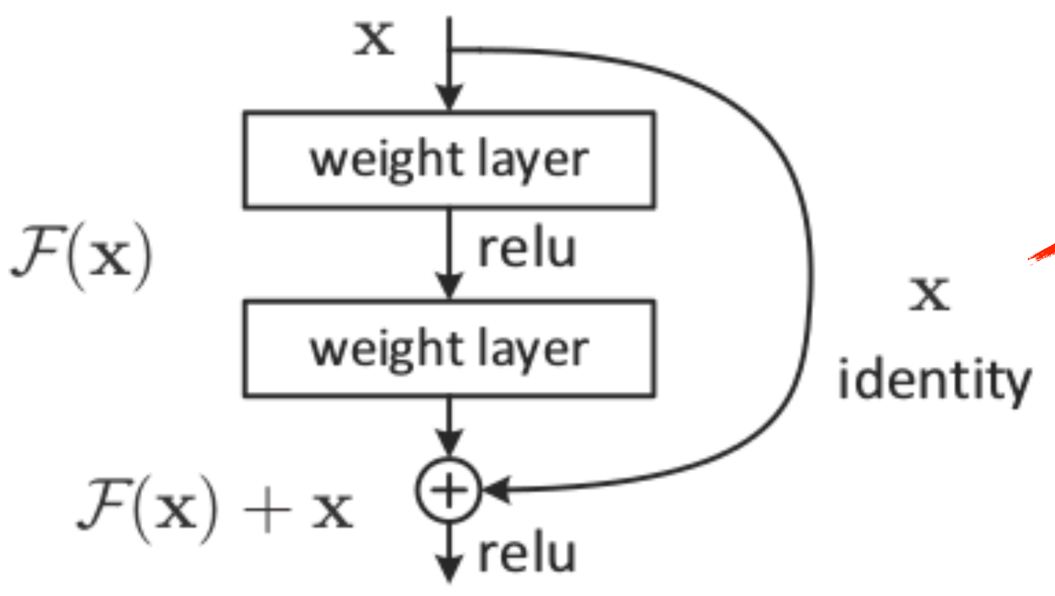
- VGG Net: small size kernels limits the number of parameters and induce more non-linearity and so more degree of freedom for the network
  - stacking them together change the receptive field: ex. C3x3-C3x3: works as a C5x5, C3x3-C3x3-C3x3: works as a C7x7



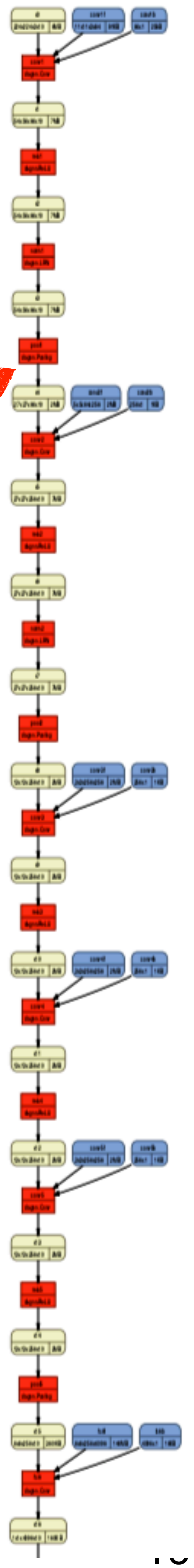
ResNet-152

.... continue for 152 layers

60 MPar



going deeper increase the vanishing gradient problem
residual learning in ResNet help mitigating it and to go deeper



GNNs

Graph Neural Networks (GNN) are an interesting and rapidly developing field in DL with several applications in natural science (particle physics, complex systems, statistical physics, combinatorial optimization, fluid dynamic, molecules modelling, …), and in industry and society (recommending systems, traffic forecasting systems, drug discovery, …)

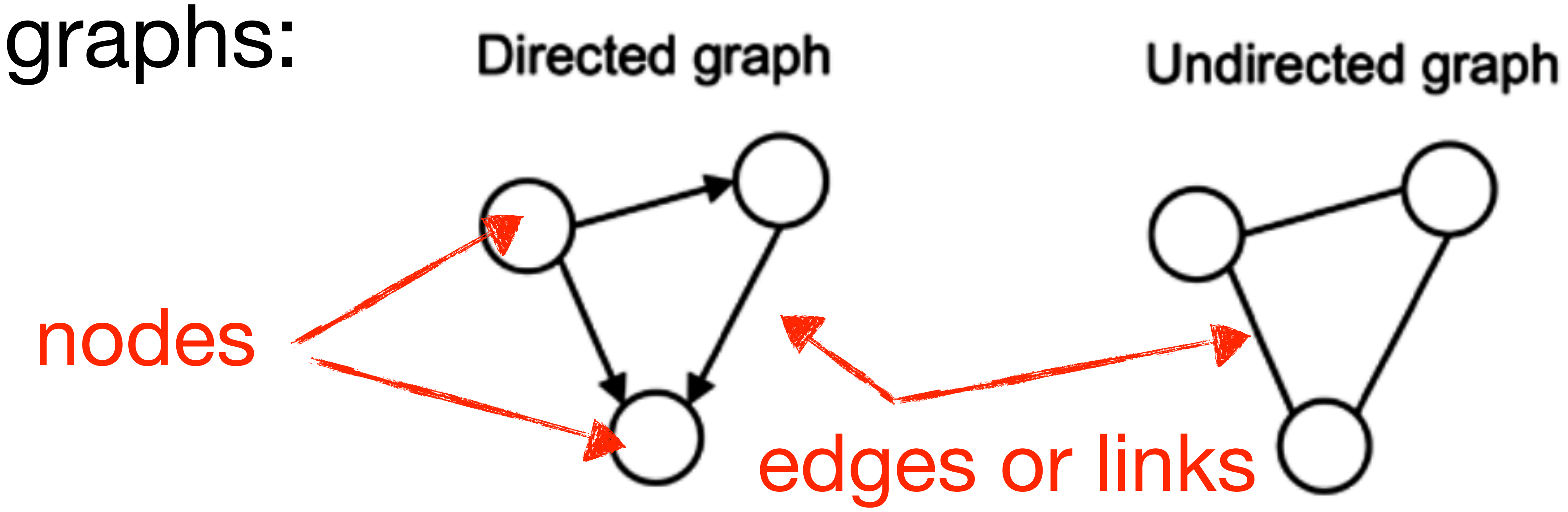
GNNs

- CNNs specialise in processing fixed mesh arrays of data (images), RNNs, specialise in processing sequences (text, …), Graph Neural Networks are neural architectures specialised in processing graphs
- processing graphs pose new challenges in the design of a neural network:
  - graph topology is variable event by event → hard to design networks that are both expressive and can cope with this variation
  - graphs may be extremely large (example: a graph representing connections between users of a social network has billions of nodes) → computational resource challenges (especially memory)
  - some dataset may be formed by only a single monolithic graph → in these cases the usual procedure of training and testing with many examples cannot be applied and different approaches have to be devise
  - …

GRAPHS

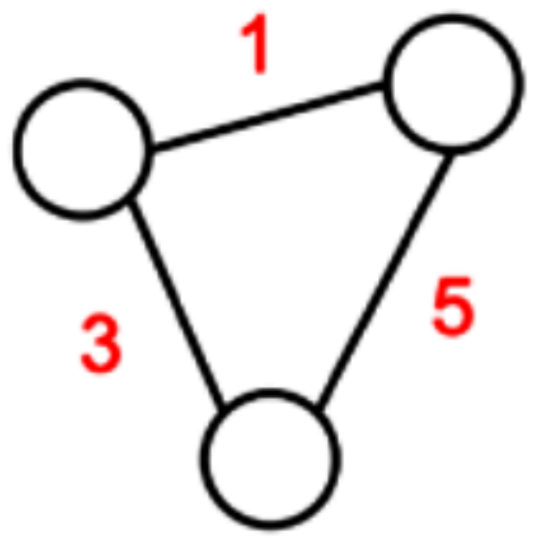
- graphs are a kind of data structure that is nonlinear and abstract, able to adapt to powerful representation of different physical and social systems

- common types of graphs:

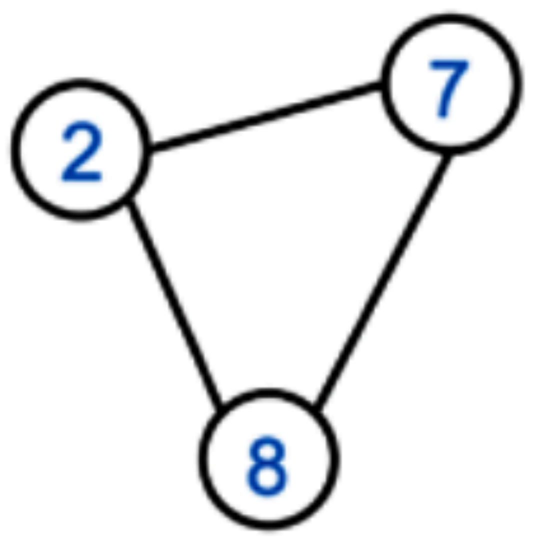


- graph are typically sparse: only a small subset of all possible edges are present
- fully connected graphs are called sets (unordered lists)

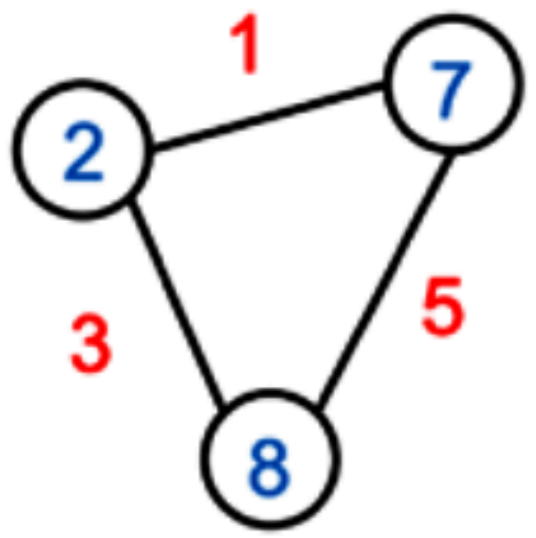
Edge-labeled undirected graph



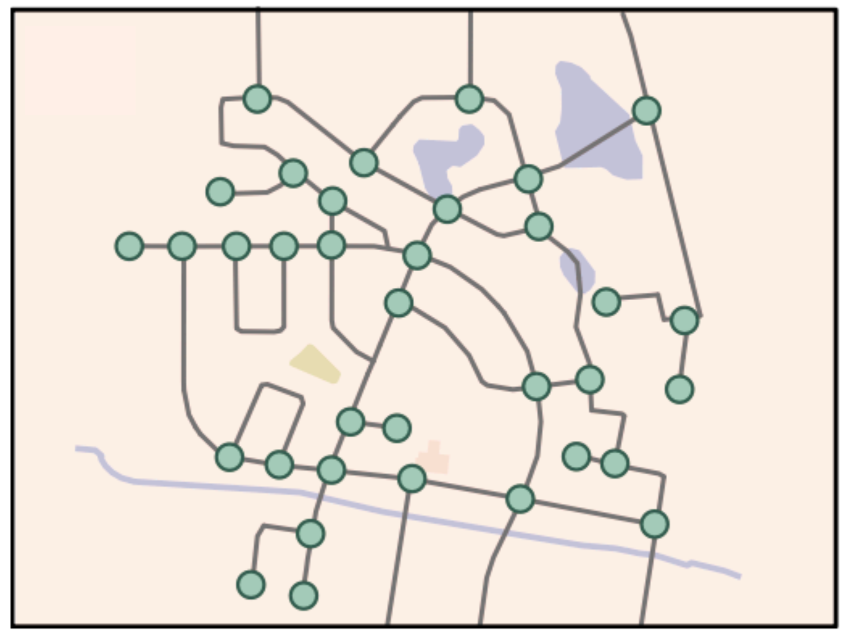
Node-labeled undirected graph

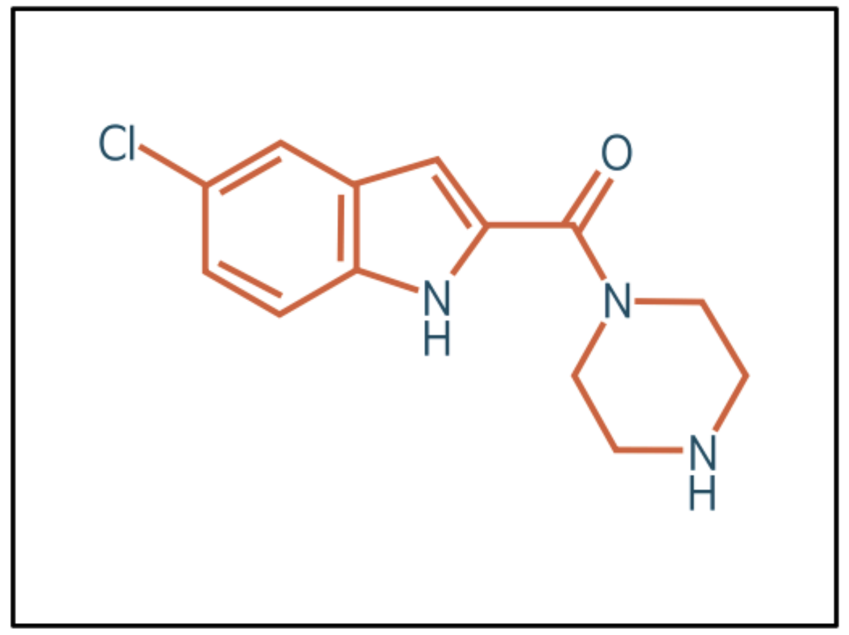


Node- and edge-labeled undirected graph

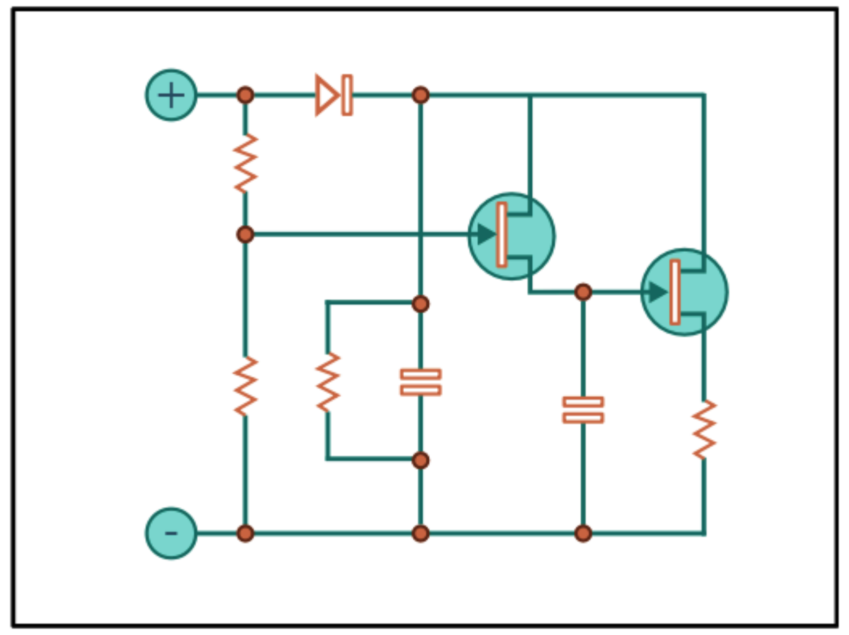


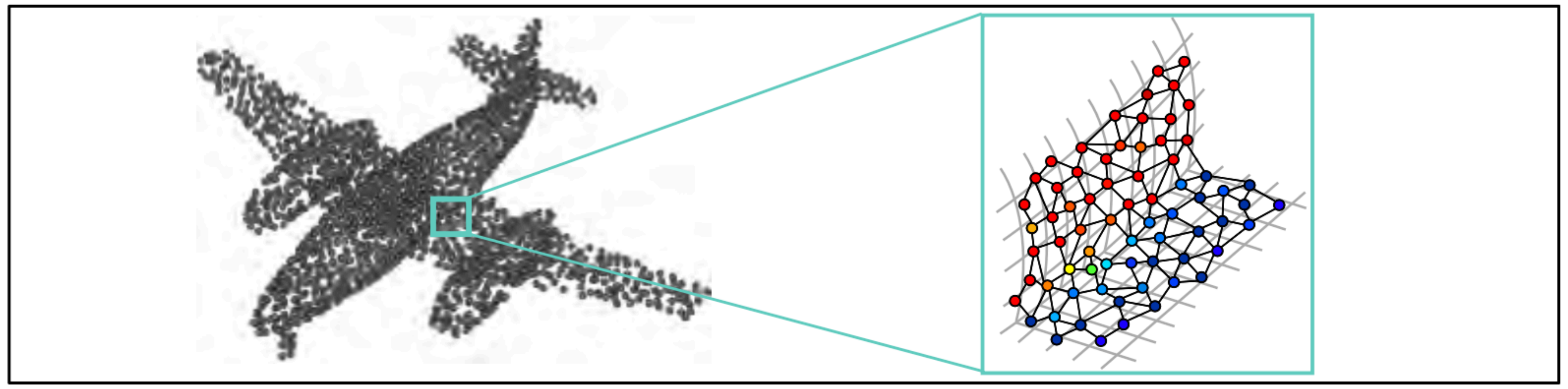
EXAMPLES OF GRAPHS





- many objects in the real world are naturally represented by graphs
    - road networks
    - electrical circuits or electrical networks
    - chemical molecules
    - physics objects measured in HEP detectors (jets reconstructed in calorimeters, charged particle tracks reconstructed in tracking detectors, …)
    - interactions between objects (particles, molecules, proteins, humans, planets, gravitational objects, …) can be expressed as graphs, where the nodes are the objects, and there is an edge between two objects if they interact
    - geometrical 2D/3D point-clouds (geometrical deep learning)
    - social networks: are graphs where nodes are people and the edges represent friendships between them
    - wikipedia can be thought of as a graph in which the nodes are articles and the edges represent hyperlinks between articles
    - computer programs can be represented as graphs in which the nodes are syntax tokens (variables at different points in the program flow) and the edges represent computations involving these variables (autograd used in backprop is implemented via computational graph)
    - …

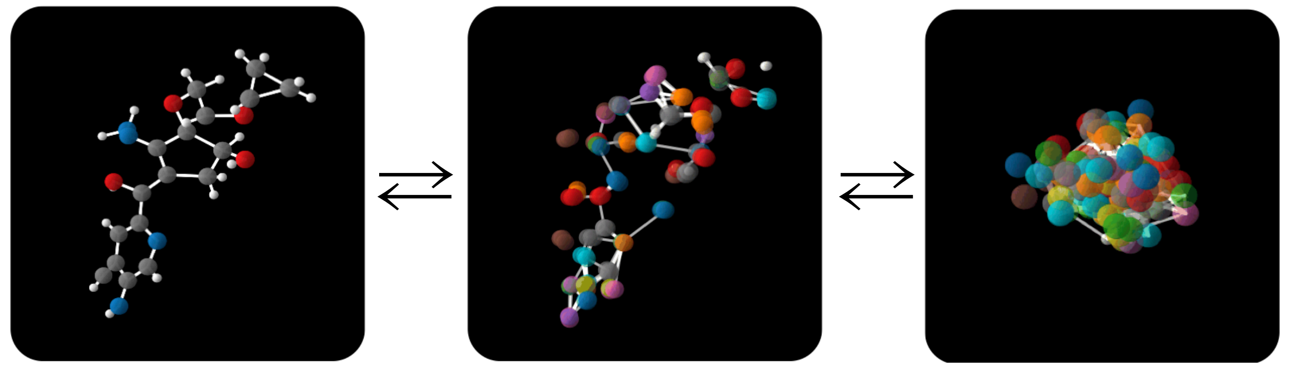


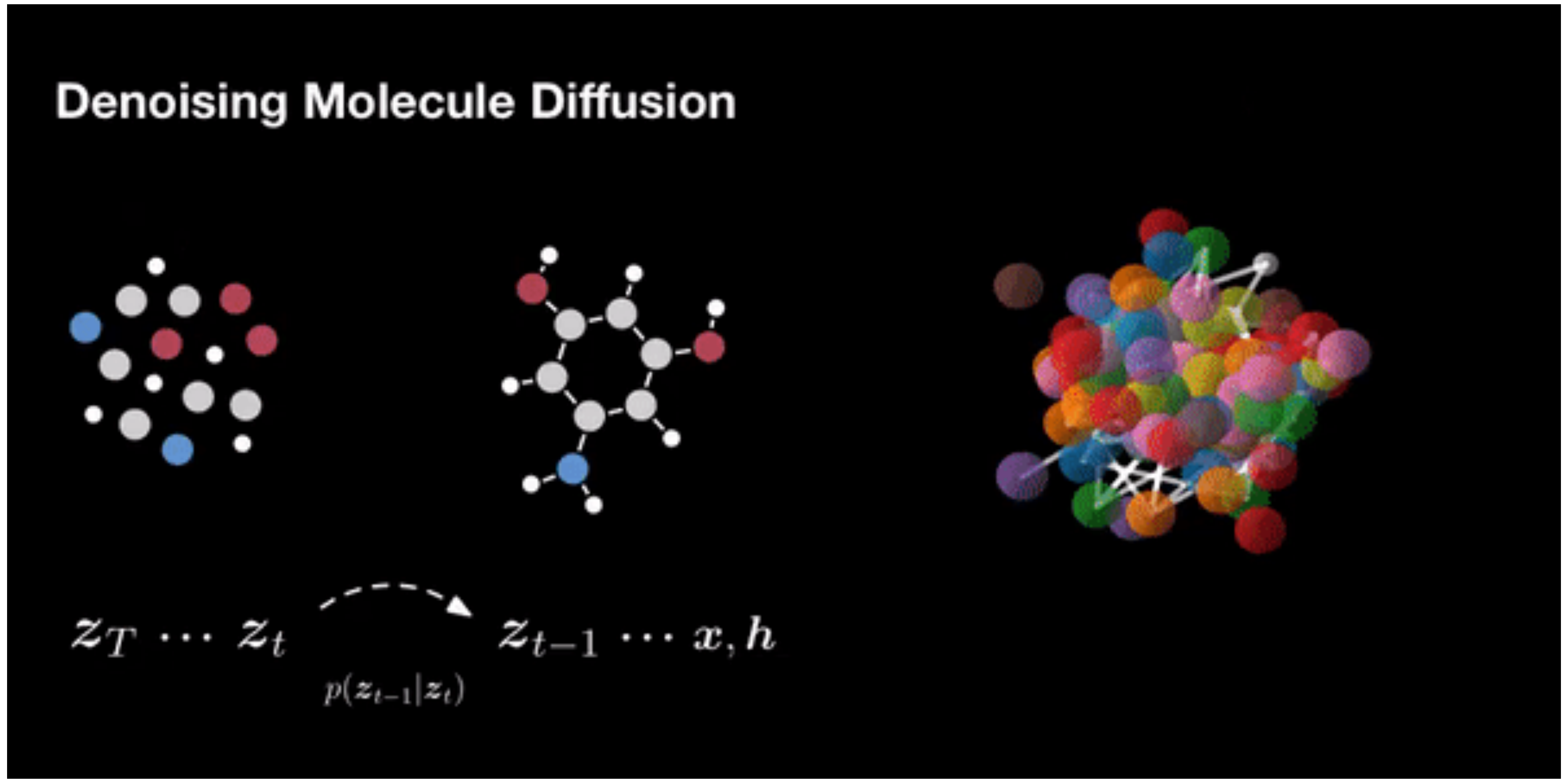


GRAPH-BASED PHYSICS-INSPIRED DIFFUSION MODEL FOR DRUG DESIGN

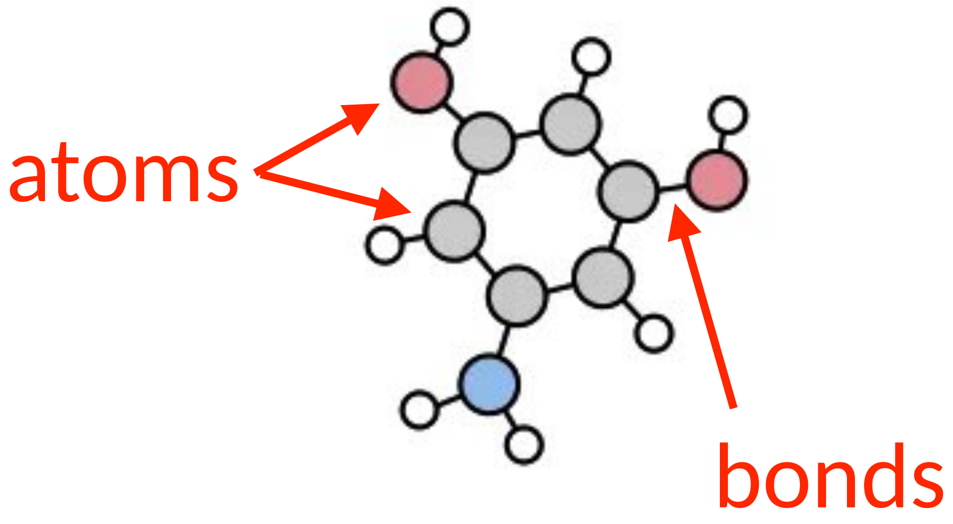
- new ways to the molecular design implemented via an iterative refinement process that allows methodic design of molecules and correction of possible errors arising during the process

noise is gradually injected until the molecule becomes a cluster of random atoms. At each step, an inverse process $p(z_{t-1}, z_t)$ is learned, which tries to predict the noise that was added to the data and then removes it to obtain clean data





graph neural network architecture: to naturally represent the intrinsic topological structures of molecules.



physics priors: incorporated into the neural network model in such a way that it respects the physical symmetries of molecules (rotations, translations, and reflections), to facilitate learning

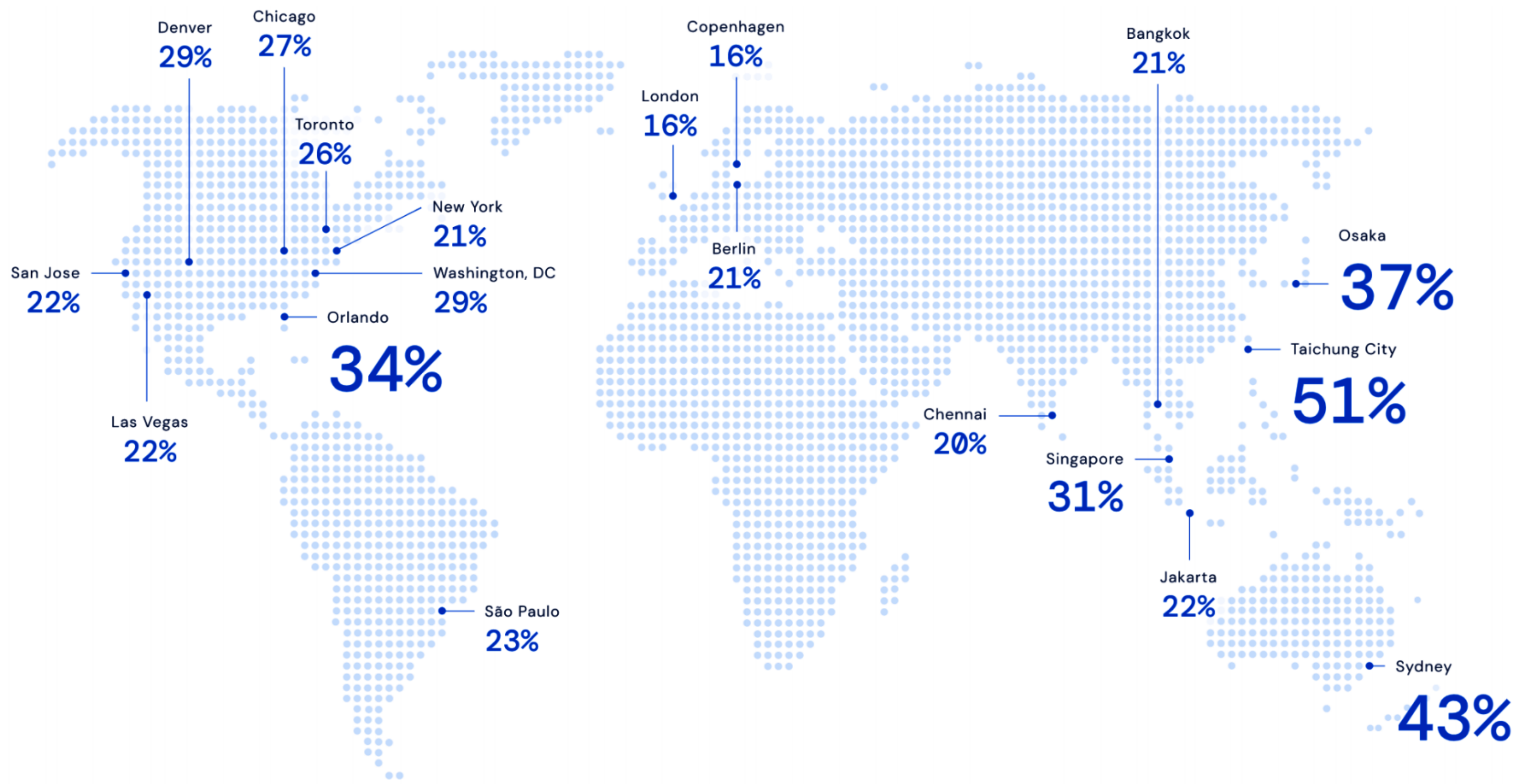
E.Hoogeboom et al., arXiv:22203.17003 [cs.LG]

GNN FOR TRAFFIC FORECASTING

Graph Neural Networks applications: improving ETA predictions in Google Maps

Accurately predicting the estimated time of arrival (ETA) for a given route requires a complex understanding of the spatiotemporal interactions taking place on the road. GNNs are well suited for this task because roads and their intersections naturally form a graph network. A GNN-based system reduced negative ETA outcomes between 16% and 51% around the world in live production.

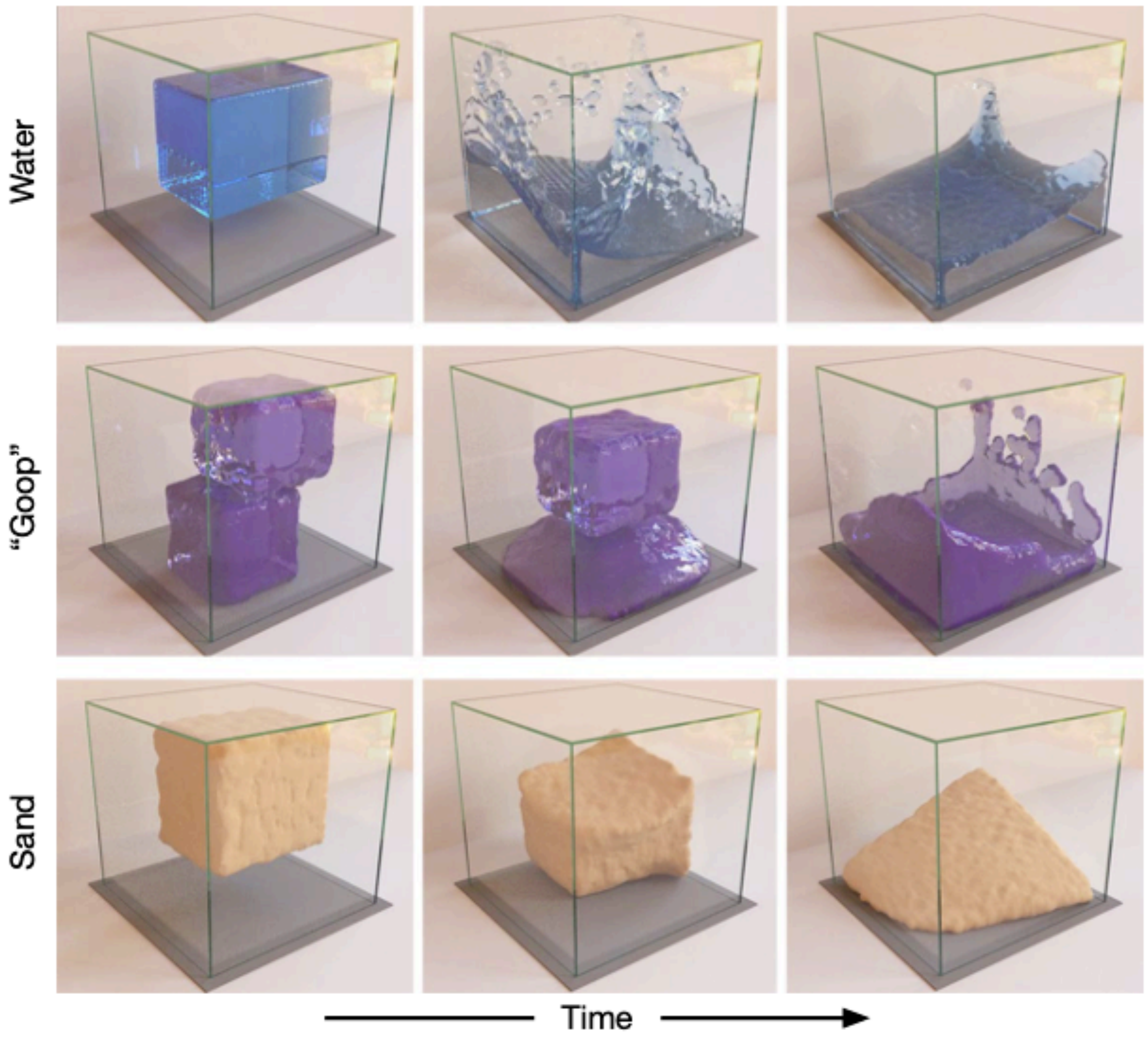
- first, roads are chunked into connected segments that follow typical traffic routes and form longer super-segments
- the world is divided into regions that have similar driving behaviors and train region-specific GNNs
- data represents the actual traversal times across segments and super-segments, which are used as node-level and graph-level labels for prediction, respectively
- for a given starting time, the GNN learns the travel time of each supersegment at specific points in the future.



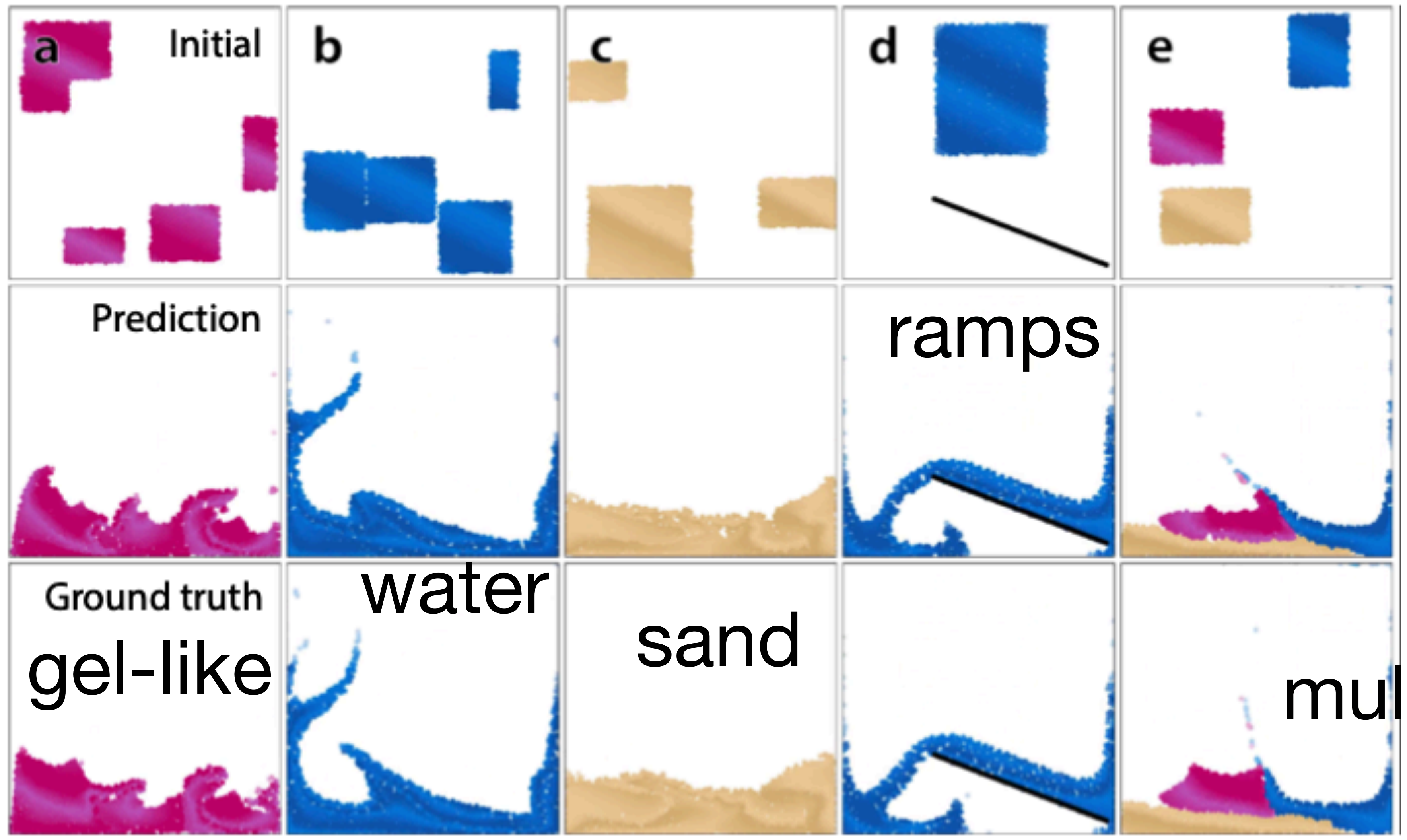
DeepMind Google AI

COMPLEX FLUID DYNAMIC SIMULATIONS

based on message passing
Graph-NN



A.Sanchez-Gonzalez et al, [arXiv:2002.09405 [cs.LG]](arXiv:2002.09405)



gel-like

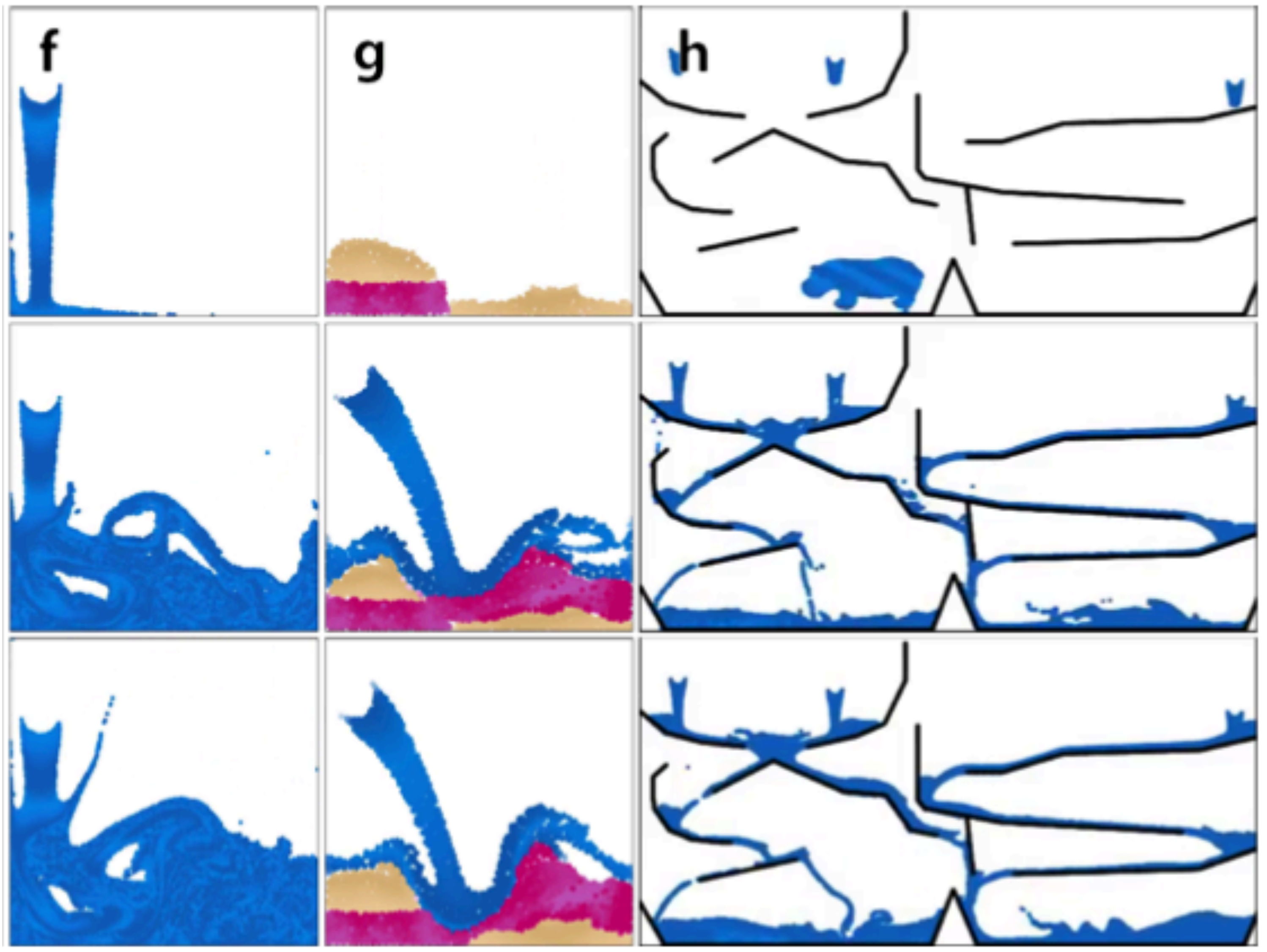
water

sand

ramps

multi-materials

prediction of
evolutions at
future states



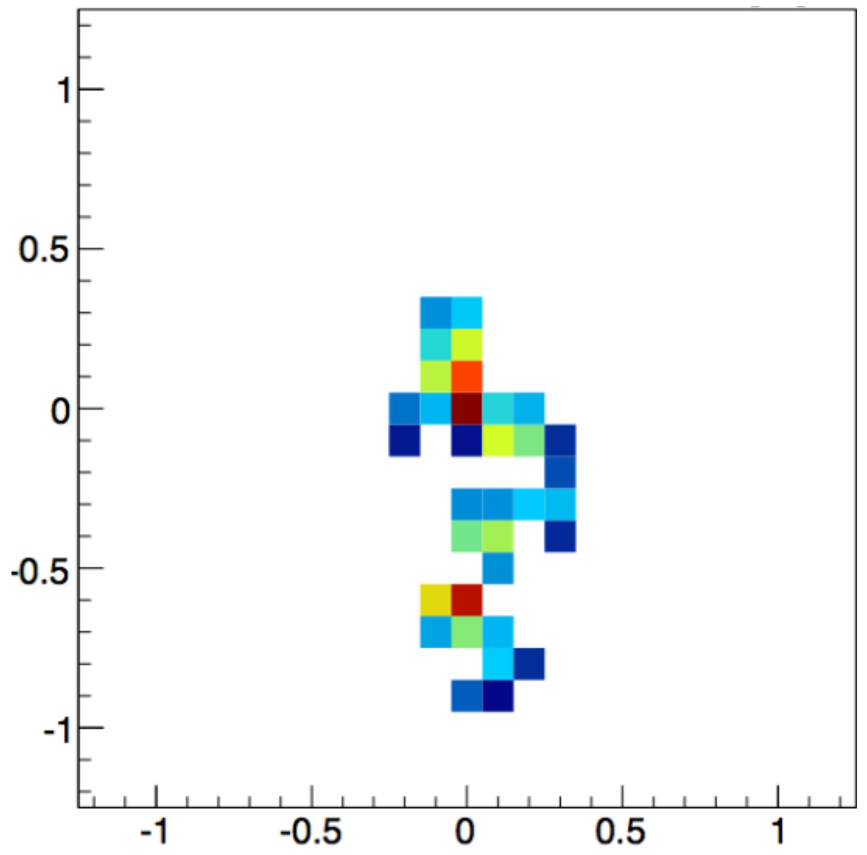
turbulent and complex
regimes (out of training)

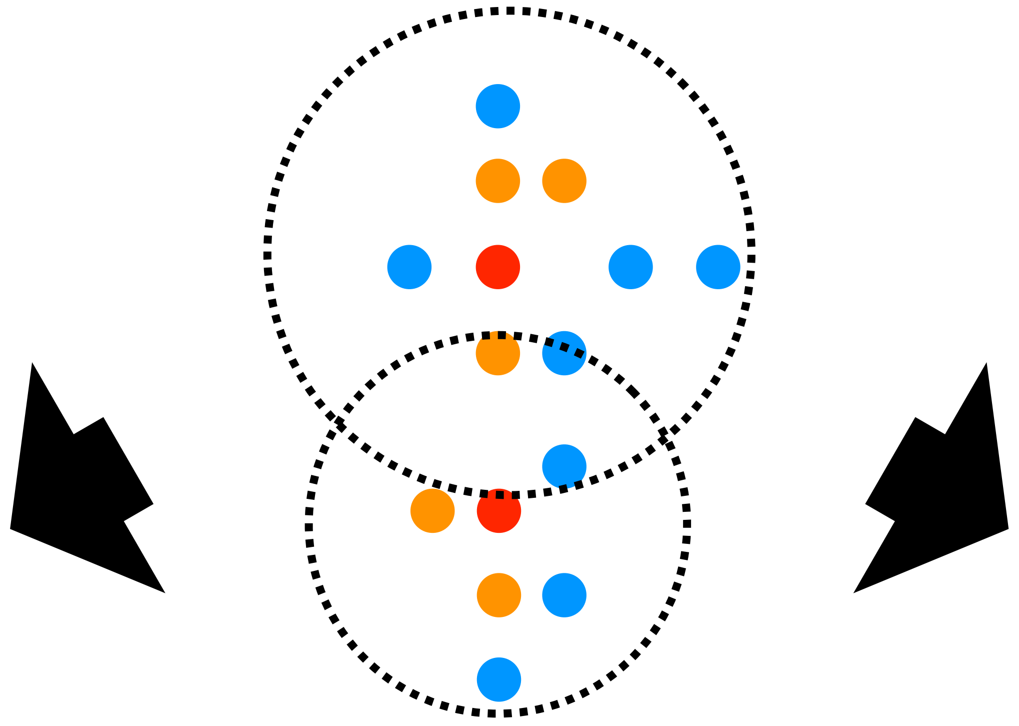
HEP ENHANCED DATA REPRESENTATION WITH GRAPHS

boosted W→qq’

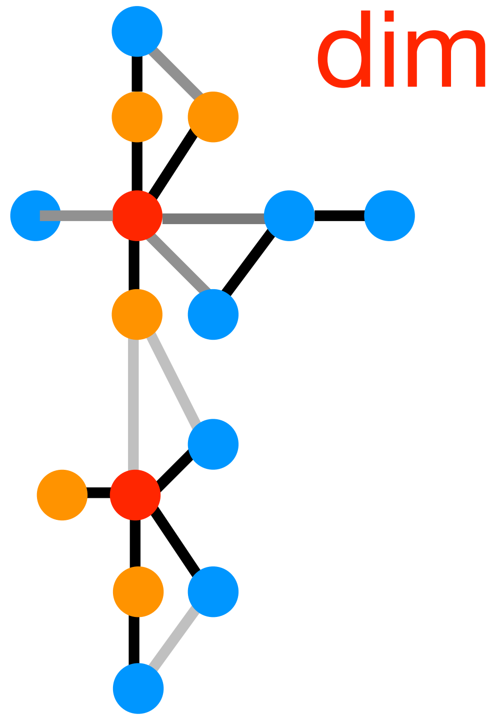
fixed-mesh image

dim: 28x28 = 784





graph



dim: 15x(3+1)+18 = 78

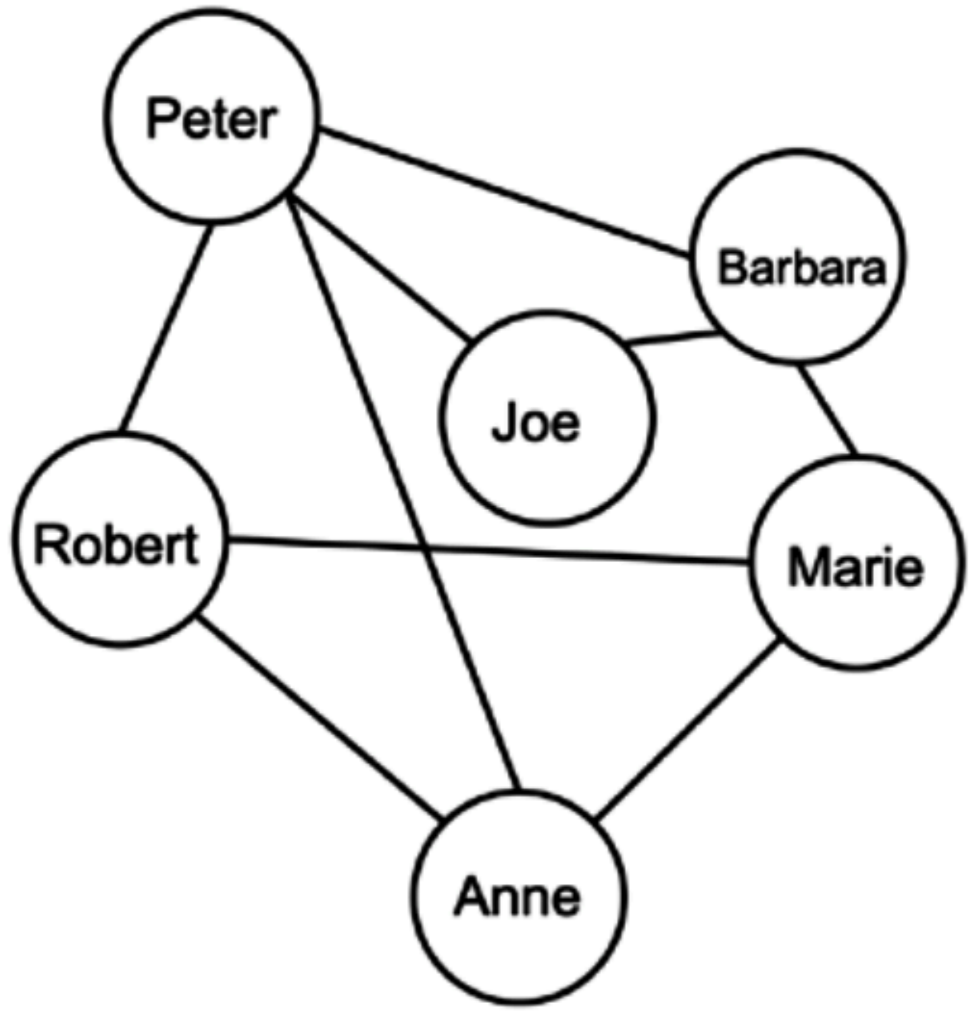
nodes edges

- Image-based: treating the energy deposition on each calorimetric cell as a pixel intensity creating an image of the event in fixed-shape mesh
  - natural representation for Convolutional Neural Networks
  - unclear how to incorporate additional information associated to each energy deposit
  - very sparse and inefficient representation: jets decays have O(10) to O(100) particles → more than 90% of the pixels are empty

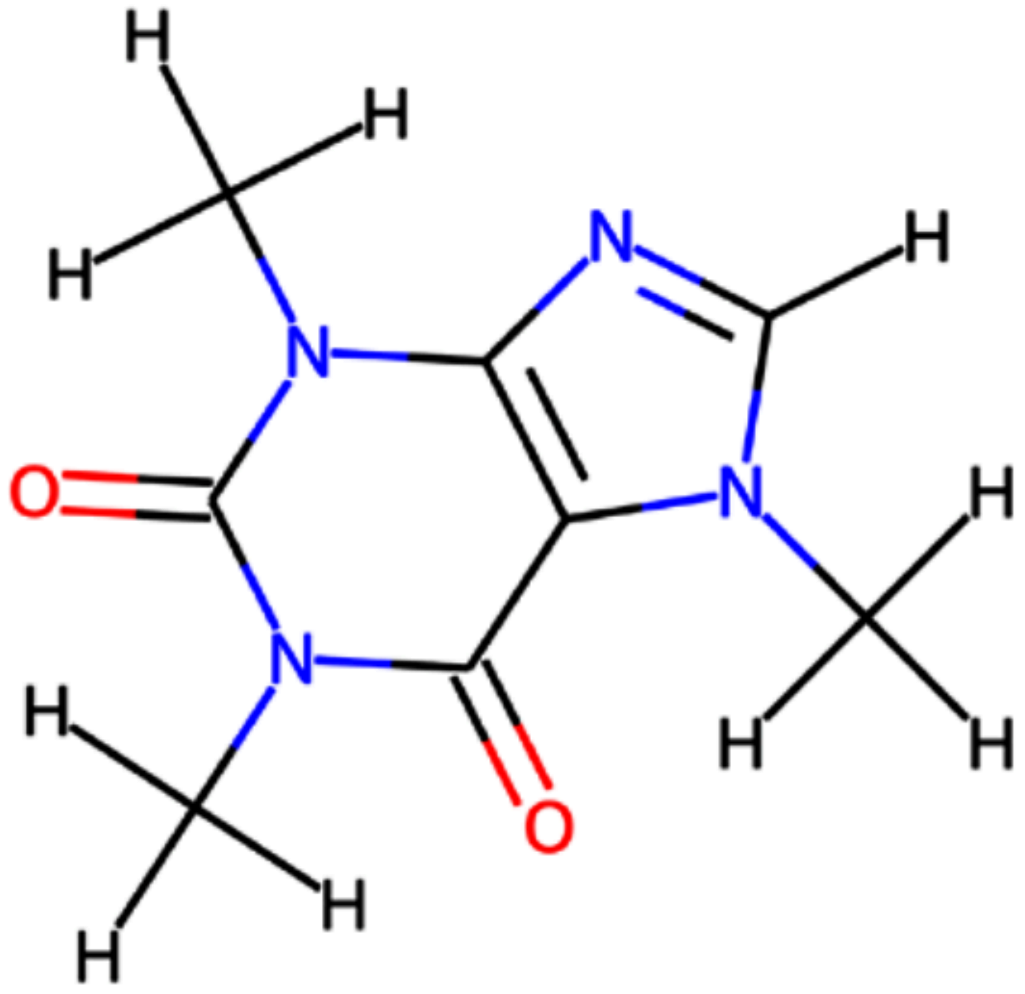
- Point cloud-based: unordered sets of entities distributed irregularly in space, analogous to the point cloud representation of 3D shapes
  - clouds allow rich internal structures
  - easy to incorporate additional information associated to the energy deposit (cell type, energy, timing information, …)
  - the architecture of the neural network has to be carefully designed to fully exploit the potential of this representation → Graph Neural Networks

UNDIRECTED GRAPHS

- an undirected graph consists of nodes that are connected via edges where the order of the nodes and their connection does not matter



Friend graph



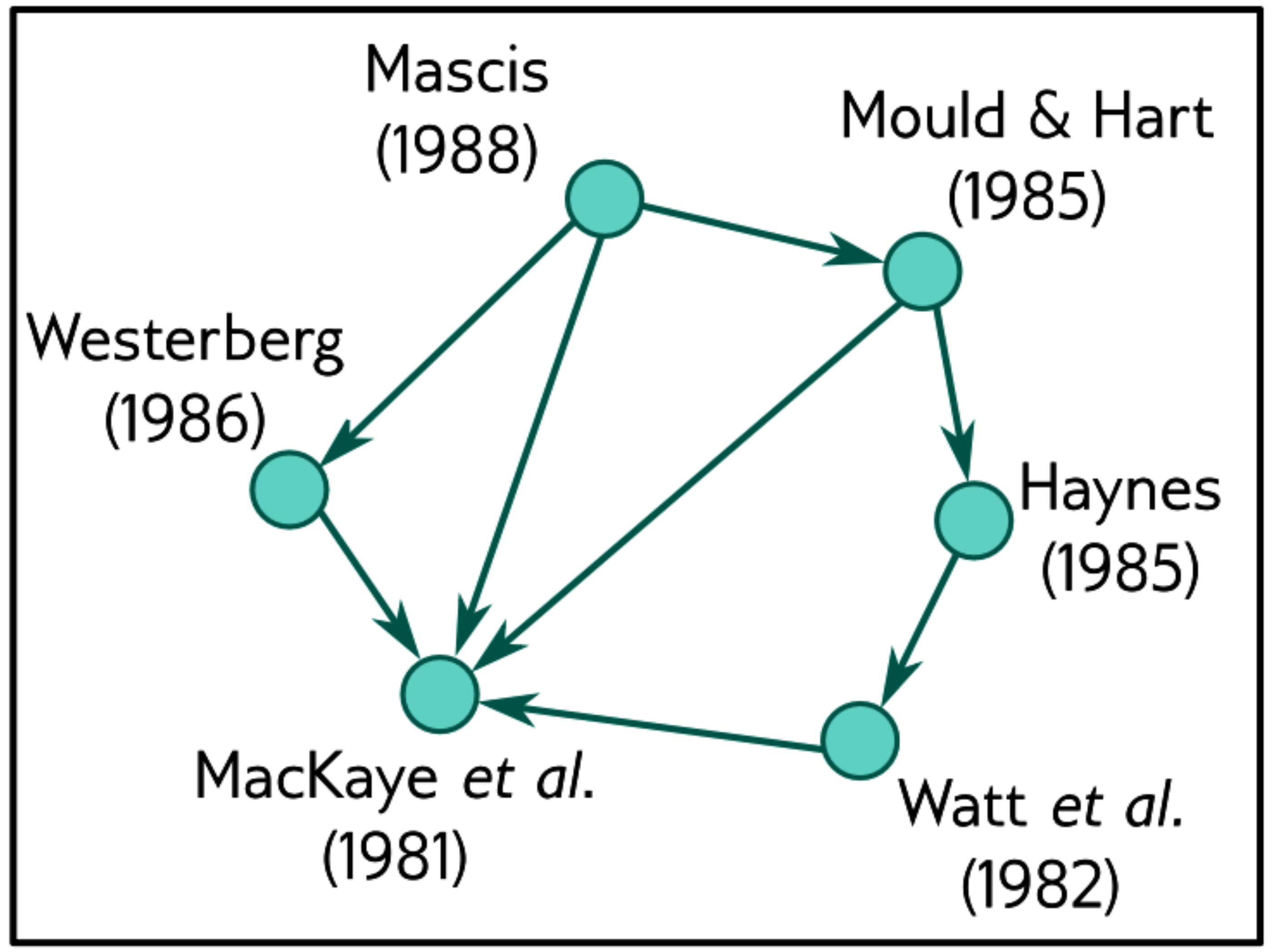
Molecular graph of caffeine

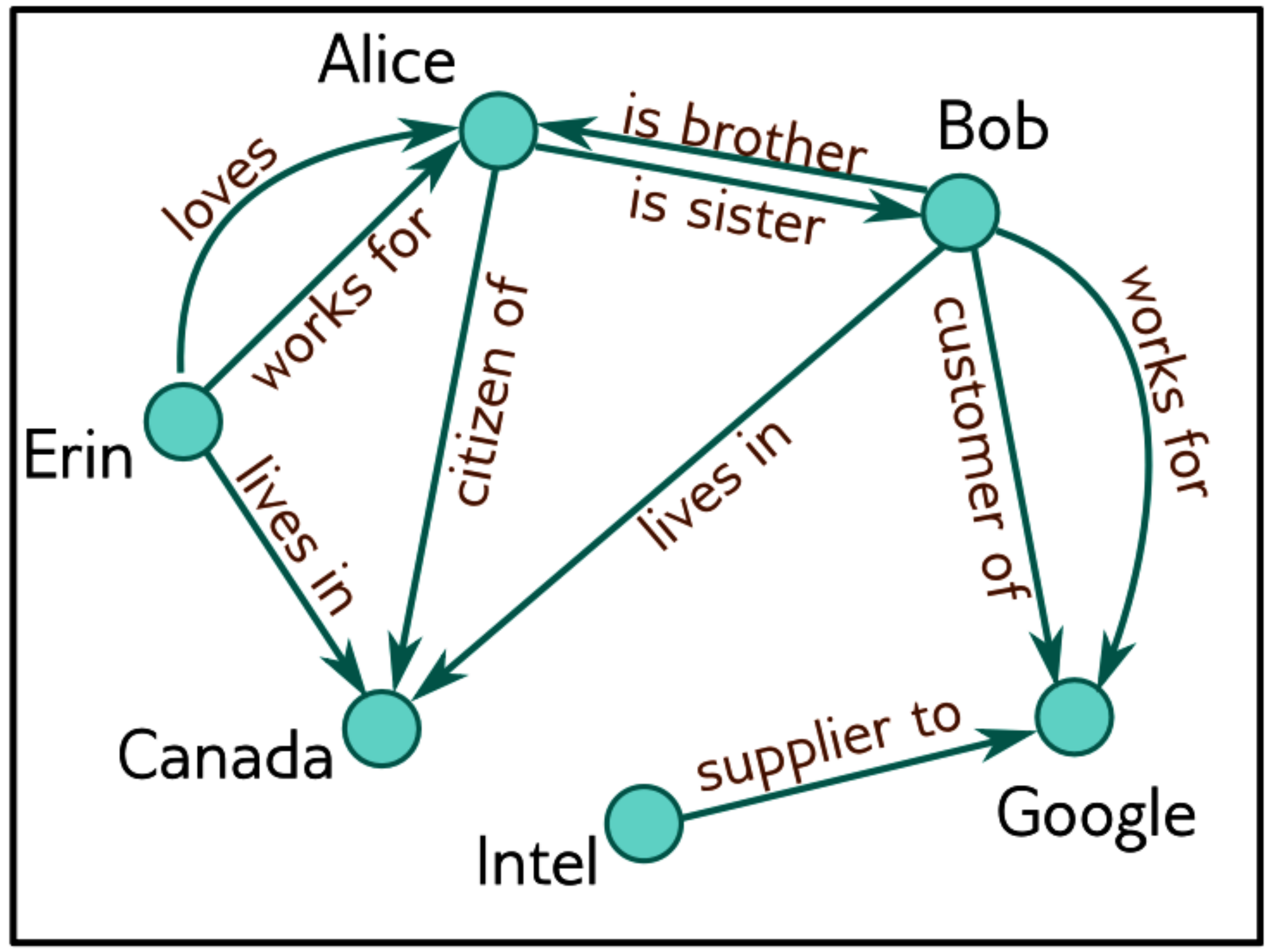
- G is defined as a pair (V, E), where V is a set of the graph's nodes, and E is the set of edges making up the paired nodes
- the graph structure can be encoded as a $|V| \times |V|$ adjacency matrix A. Each element x_{ij} in the matrix A is either a 1 or a 0, with 1 denoting an edge between nodes i and j (vice versa, 0 denotes the absence of an edge)
- undirected graph means A is symmetric ($x_{ij} = x_{ji}$)

- common examples of data that can be represented as undirected graphs include images, protein-protein interaction networks, point clouds, …

DIRECTED GRAPHS

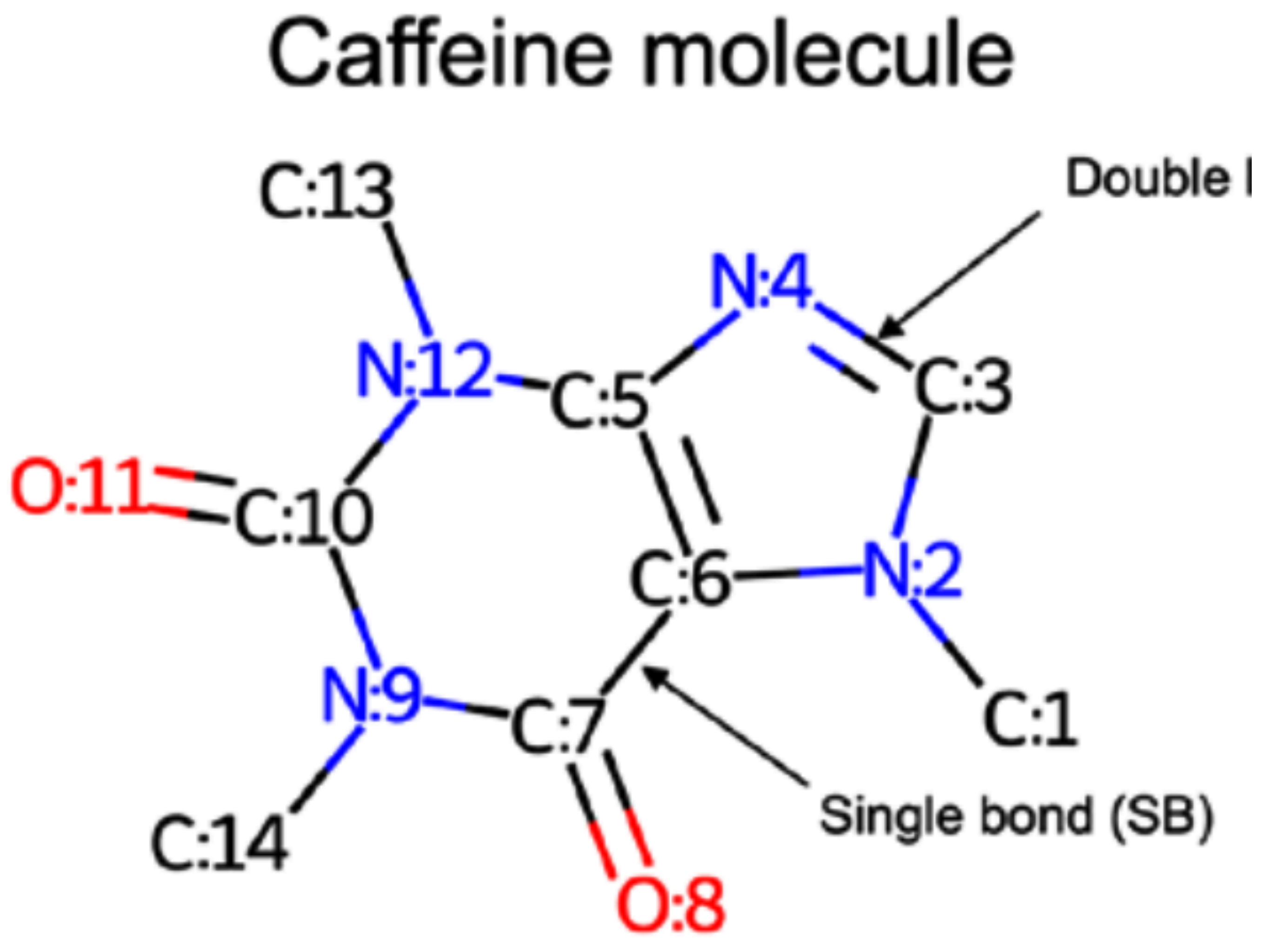
- in contrast to undirected graphs, directed graphs connect nodes via directed (directional) edges
- they are defined in the same way as an undirected graph, except that E, the set of edges, is a set of ordered pairs, e.g. A elements: $x_{ij} \neq x_{ji}$
- example of a directed graph is a **citation network**, where nodes are publications and edges from a node are directed toward the nodes of papers that a given paper cite, or a **knowledge graph** (a directed heterogeneous (nodes represent different object types (people, place, companies) multigraph (there may be multiple edges representing different relations between each node)





- in addition to the graph structure itself represented by **A**, typically there are also information associated with each node and/or each edge of the graph

- for example: we can represent a molecule as an undirected graph with a node label matrix, where each row is a one-hot encoding of the associated node's atom type. Additionally, there can be an edge label matrix where each row is a one-hot encoding of the associated edge's bond type. Considering the caffeine molecule a graph representation (with implicit hydrogen atoms) could be:



$$A = \begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

Adjacency matrix

One-hot encoding of the atom type

$$X = \begin{array}{c}
\begin{matrix} \mathbf{C} & \mathbf{N} & \mathbf{O} \end{matrix} \\
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 0 & 0 \\
1 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 0
\end{bmatrix}
\end{array}$$

Node label matrix

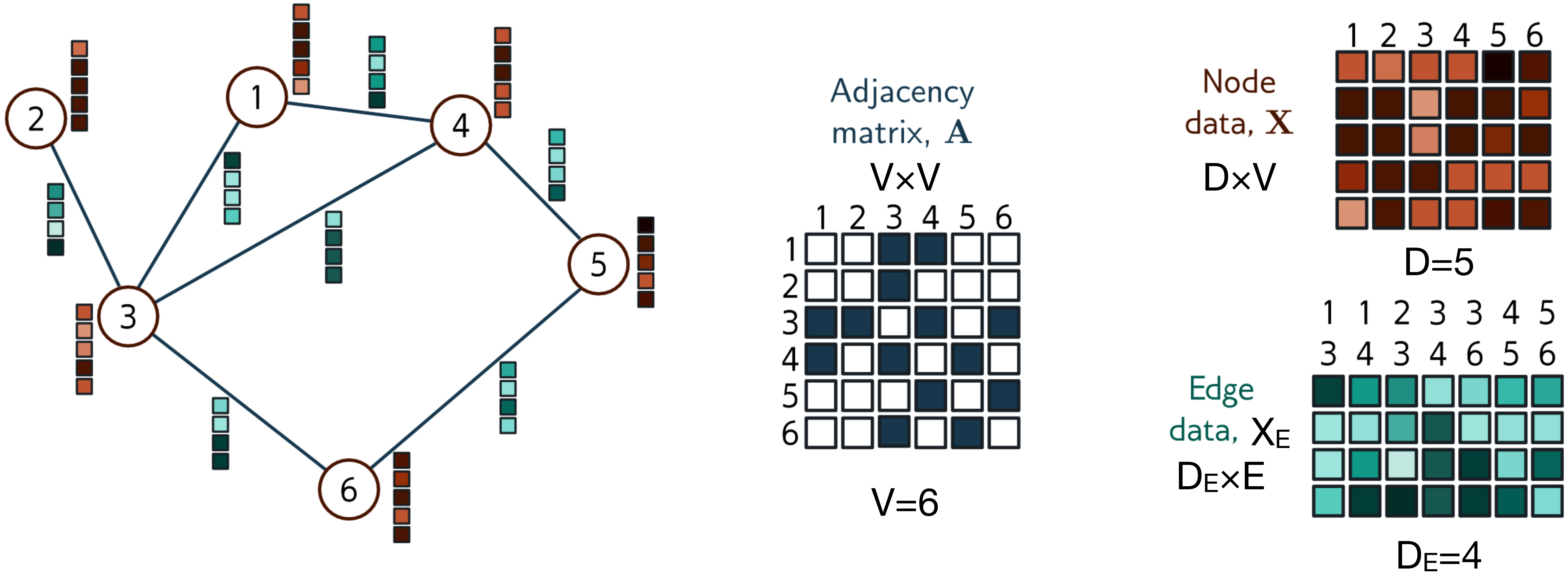
One-hot encoding of the bond type

$$X_E = \begin{array}{c|cc}
 & \mathbf{SB} & \mathbf{DB} \\
(1,2) & 1 & 0 \\
(2,3) & 1 & 0 \\
(2,6) & 1 & 0 \\
(3,4) & 0 & 1 \\
(4,5) & 1 & 0 \\
(5,6) & 0 & 1 \\
(5,12) & 1 & 0 \\
(6,7) & 1 & 0 \\
(7,8) & 0 & 1 \\
(7,9) & 1 & 0 \\
(9,10) & 1 & 0 \\
(9,14) & 1 & 0 \\
(10,11) & 0 & 1 \\
(10,12) & 1 & 0 \\
(12,13) & 1 & 0
\end{array}$$

Edge label matrix

- to summarise:
  - a graph G=(V,E) is represented by 3 matrices:
    - adjecency matrix A (dim: VxV) representing graph structure
    - node label matrix X (dim: DxV) representing the node embedding
    - edge label matrix X_E (dim: D_ExE) representing the edge embedding (often simpler graph don’t use X_E)

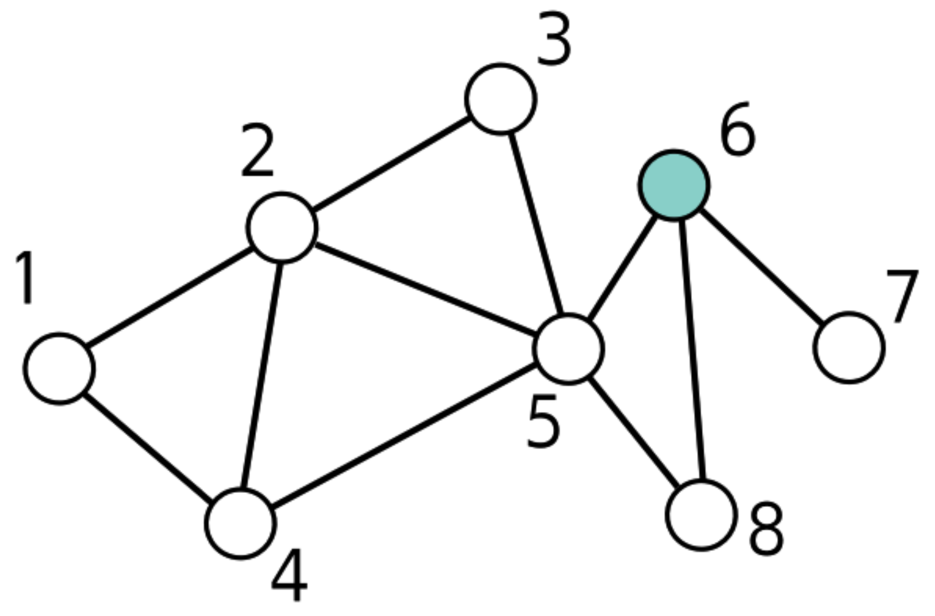
- example:



	1	2	3	4	5	6
1	□	□	■	■	□	□
2	□	□	■	□	□	□
3	■	■	□	■	□	■
4	■	□	■	□	■	□
5	□	□	□	■	□	■
6	□	□	■	□	■	□

1	1	2	3	3	4	5
3	4	3	4	6	5	6

- the adjacency matrix can be used to to find the neighbors of a node using linear algebra. For example if we encode the j-th node as a one-hot V-size column vector with all zeros but the j-th element:

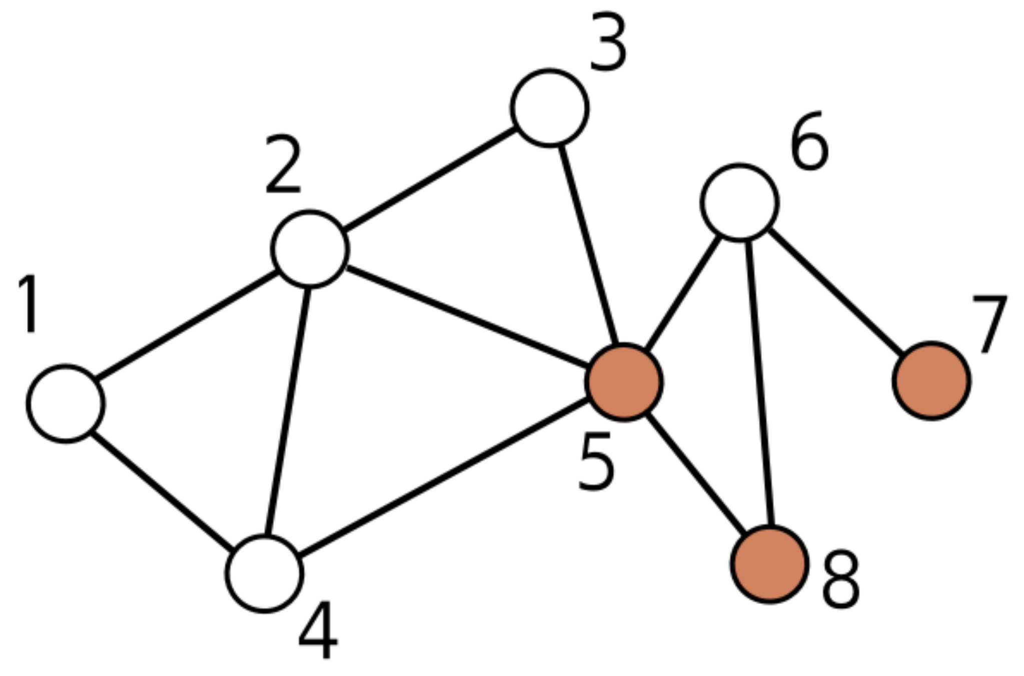


$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

$$\mathbf{x} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

6th node

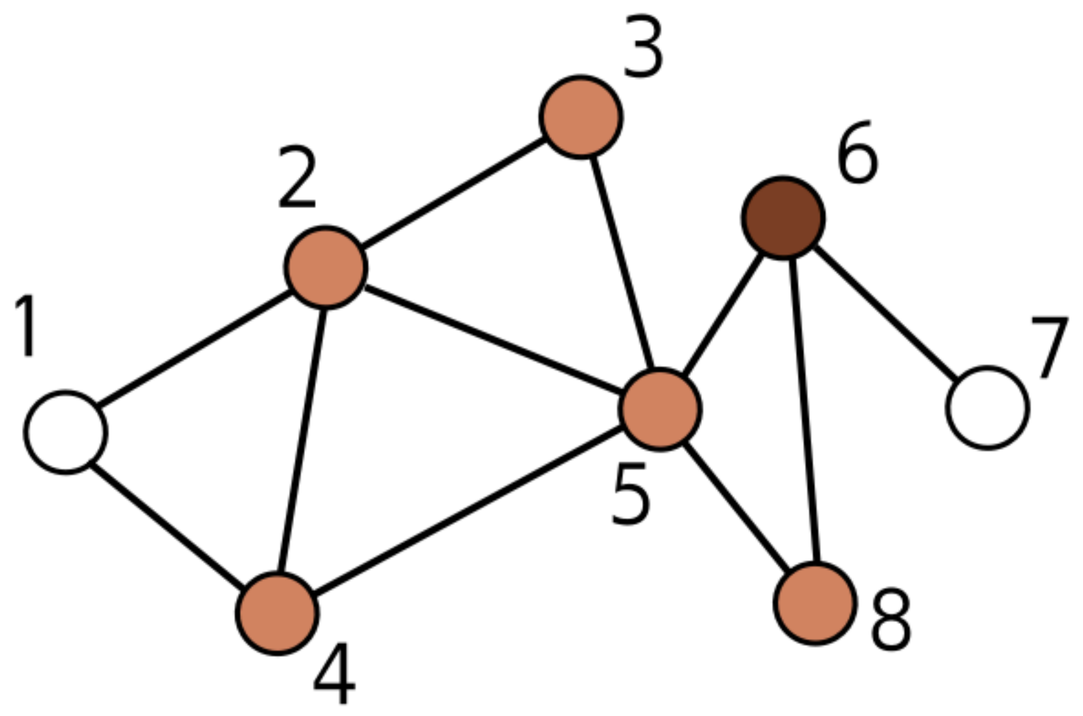
$$\mathbf{Ax} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$



6th node - neighbours

$$\mathbf{A}^2 = \begin{bmatrix} 2 & 1 & 1 & 1 & 2 & 0 & 0 & 0 \\ 1 & 4 & 1 & 2 & 2 & 1 & 0 & 1 \\ 1 & 1 & 2 & 2 & 1 & 1 & 0 & 1 \\ 1 & 2 & 2 & 3 & 1 & 1 & 0 & 1 \\ 2 & 2 & 1 & 1 & 5 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 3 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 2 \end{bmatrix}$$

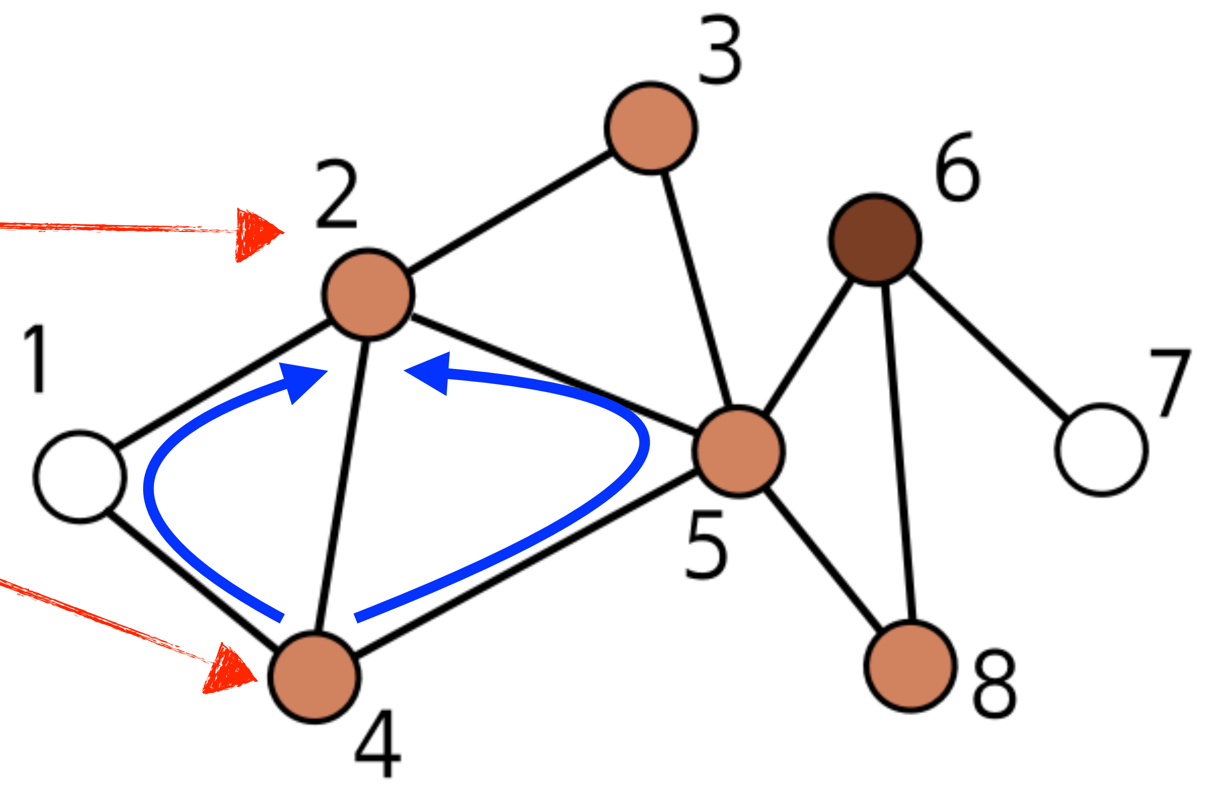
$$\mathbf{A}^2\mathbf{x} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 3 \\ 0 \\ 1 \end{bmatrix}$$



6th node walks of length 2 to the other nodes of the graph

- in general the element (i,j) of A^m contains the number of unique “walks” of length m from node i to node j

$$\mathbf{A}^2 = \begin{array}{c|cccccccc|}
 & & & & \text{i=4} & & & & \\
 & 2 & 1 & 1 & 1 & 2 & 0 & 0 & 0 \\
\text{j=2} & 1 & 4 & 1 & 2 & 2 & 1 & 0 & 1 \\
 & 1 & 1 & 2 & 2 & 1 & 1 & 0 & 1 \\
 & 1 & 2 & 2 & 3 & 1 & 1 & 0 & 1 \\
 & 2 & 2 & 1 & 1 & 5 & 1 & 1 & 1 \\
 & 0 & 1 & 1 & 1 & 1 & 3 & 0 & 1 \\
 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 2
\end{array}$$

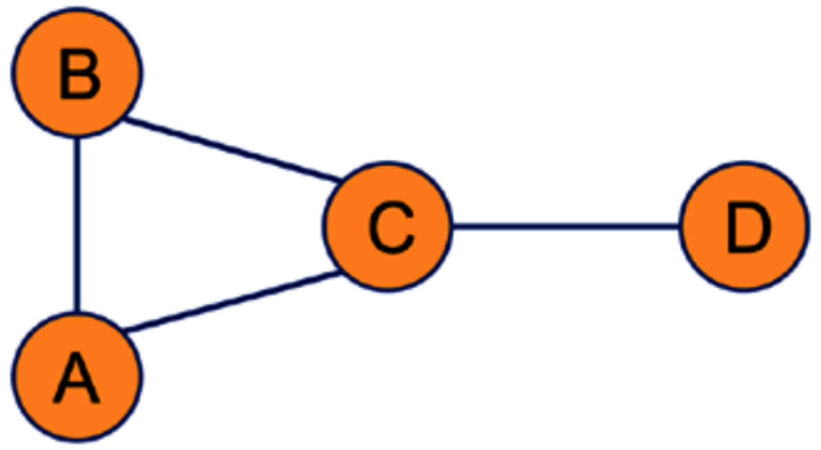


SIMILAR AND DIFFERENT INDUCTIVE BIASES BETWEEN IMAGES AND GRAPHS

- images and graphs share the **locality prior**, however the definition of locality is different:
  - images: locality in 2D space
  - graphs: structural locality
  - intuitively, this means that a node that is 1 edge away is more likely to be related than a node 5 edges away from a give node
  - example: in a citation graph, a directly cited publication, which would be one edge away, is more likely to have similar subject matter than a publication with multiple degrees of separation

- a strict prior true only for graph data is **permutation invariance**, which means that the ordering of the nodes does not affect the output

- changing the ordering of a graph's nodes does not change the graph's structure: since the same graph can be represented by multiple adjacency matrices, consequently, any operation on a graph needs to be permutation invariant



Adjacency matrix 1:

$$\begin{array}{c|cccc} \mathbf{D} & 0 & 0 & 0 & 1 \\ \mathbf{B} & 0 & 0 & 1 & 1 \\ \mathbf{A} & 0 & 1 & 0 & 1 \\ \mathbf{C} & 1 & 1 & 1 & 0 \end{array}$$

Adjacency matrix 2:

$$\begin{array}{c|cccc} \mathbf{A} & 0 & 1 & 0 & 1 \\ \mathbf{C} & 1 & 0 & 1 & 1 \\ \mathbf{D} & 0 & 1 & 0 & 0 \\ \mathbf{B} & 1 & 1 & 0 & 0 \end{array}$$

Adjacency matrix 3:

$$\begin{array}{c|cccc} \mathbf{A} & 0 & 1 & 1 & 0 \\ \mathbf{B} & 1 & 0 & 1 & 0 \\ \mathbf{C} & 1 & 1 & 0 & 1 \\ \mathbf{D} & 0 & 0 & 1 & 0 \end{array}$$

they represent the same graph

GNN

A GNN is a model that takes the node embeddings *X* and the adjacency matrix *A* as inputs and passes the graph through a series of K layers. At each layer the node embeddings are updated to create intermediate "hidden" representations H_k, before finally computing output embeddings H_K

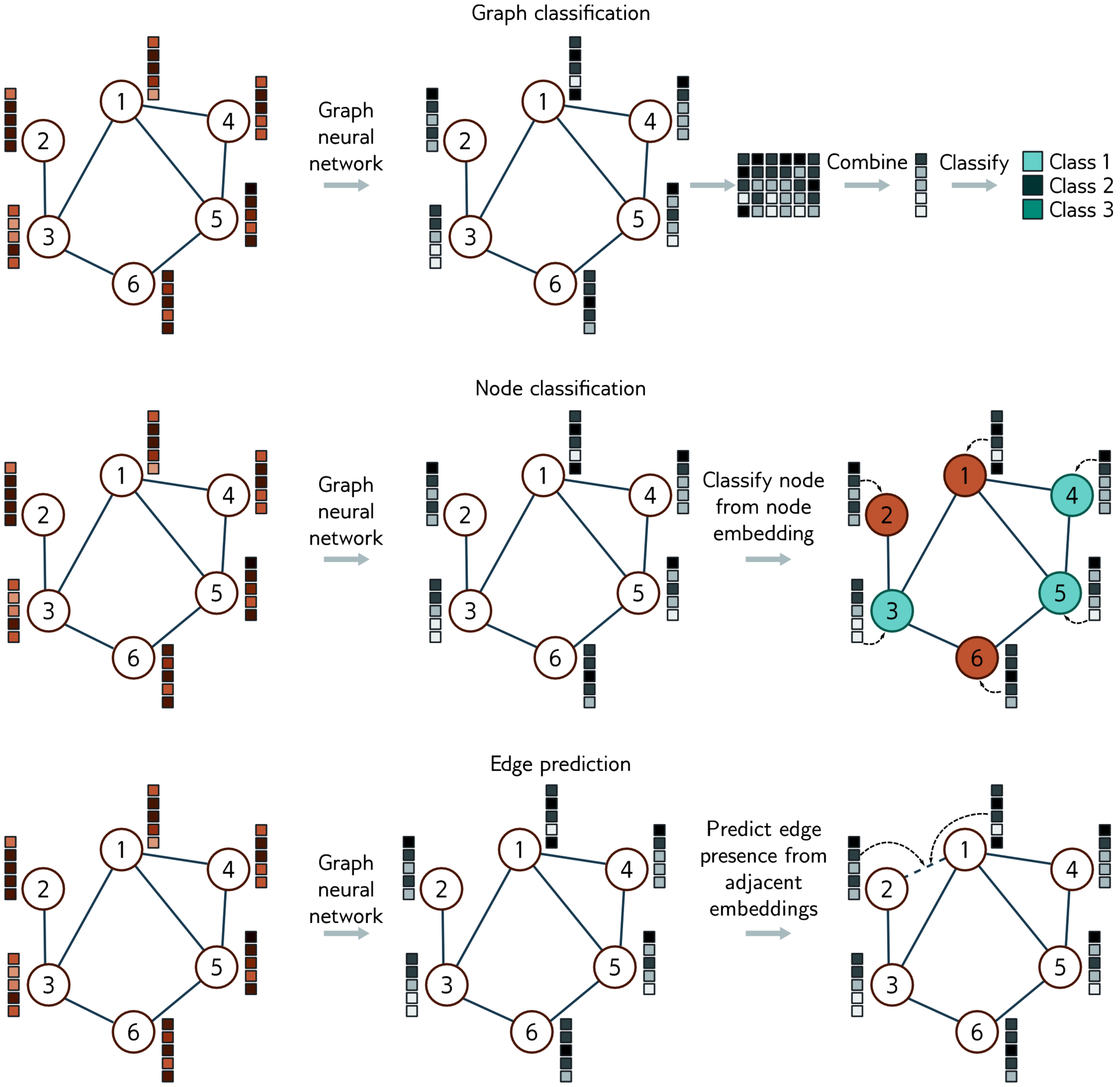
$$
\begin{aligned}
(H_1, A) &= f_1(X, A) \\
(H_2, A) &= f_2(H_1, A) \\
&\vdots \\
(H_k, A) &= f_k(H_{k-1}, A) \\
&\vdots \\
\end{aligned}
$$

$$\boxed{(H_K, A) = f_K(H_{K-1}, A) = f_K(f_{K-1}(\cdots f_1(X, A)))))}$$

$f_k = f_{W_k}$ w/ W_k learnable weights

- at the start, each column of the input node embeddings *X* contains only the information about the node itself
- at the end, each column of the model output H_K contains information about the node and its context within the graph

LEARNING PROBLEMS ON GRAPHS



- whole graph classification: obtain an embedding for the entire graph by performing a global pooling operation and use the embedding on a MLP for example (example: $y = \text{softmax}[b_K + W_K H_K \mathbf{1}/N]$)

mean pooling

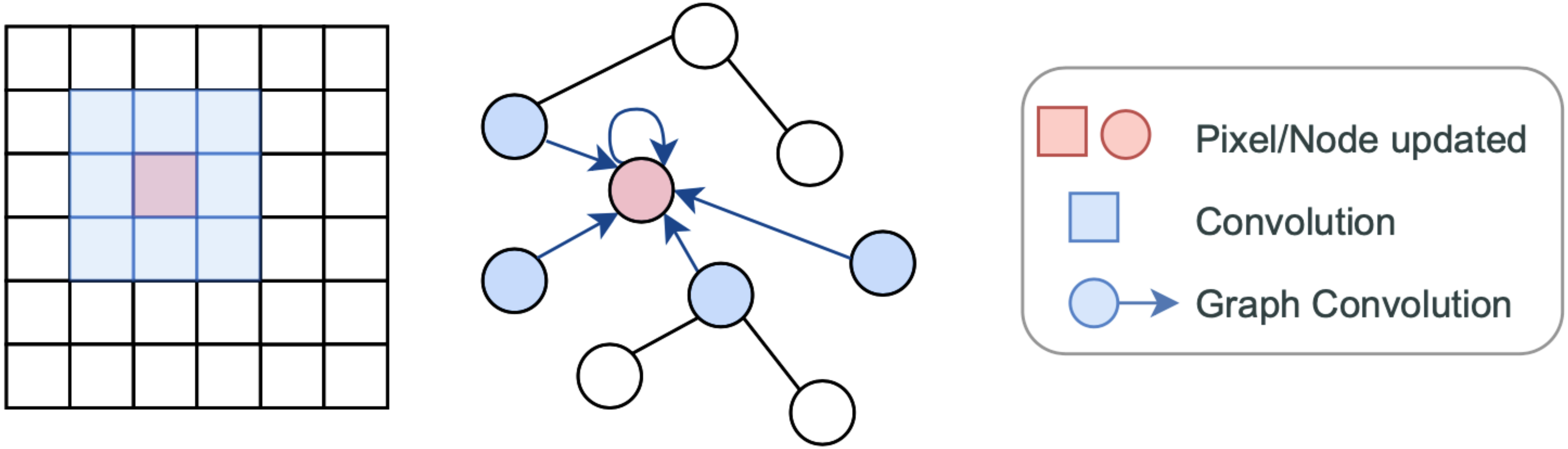
- node classification: is it possibile to classify a single or every node, or any combination of it (example: $y^{(i)} = \text{softmax}[b_K + W_K h_K^{(i)}]$)

- edge classification/link prediction: a prediction of an edge can be obtained by a prediction for an edge by combining the features of the corresponding nodes, and pass this information to an MLP (example: $y^{(i,j)} = \text{softmax}[h_K^{(i)T} h_K^{(j)}]$)

LEARNING OVER A GRAPH: GRAPH CONVOLUTIONS

- a possibility to effectively use graph representations is to generalise the concept of convolution of a CNN
- in a CNN, convolution means to slide a filter over the image, where, at each step, a weighted sum is computed between the filter and the receptive field
- the generalisation to a graph is implemented in the so called spatial-based graph convolutional neural networks (GCN)
    - are convolutional in the sense that they update each node by aggregating information from nearby nodes, and as such, they induce a relational inductive bias (i.e., a bias towards prioritising information from neighbours)
    - are spatial-based because they do this in a straightforward manner using the original graph structure (in contrast to spectral-based methods that apply convolutions on graphs in the Fourier domain)
    - the convolutional approach is even more important for graphs than images because it can function with graphs of different sizes (there are many image datasets with a fixed resolution, while most graph datasets contain graphs of varying sizes)

GRAPH VS CNN CONVOLUTION



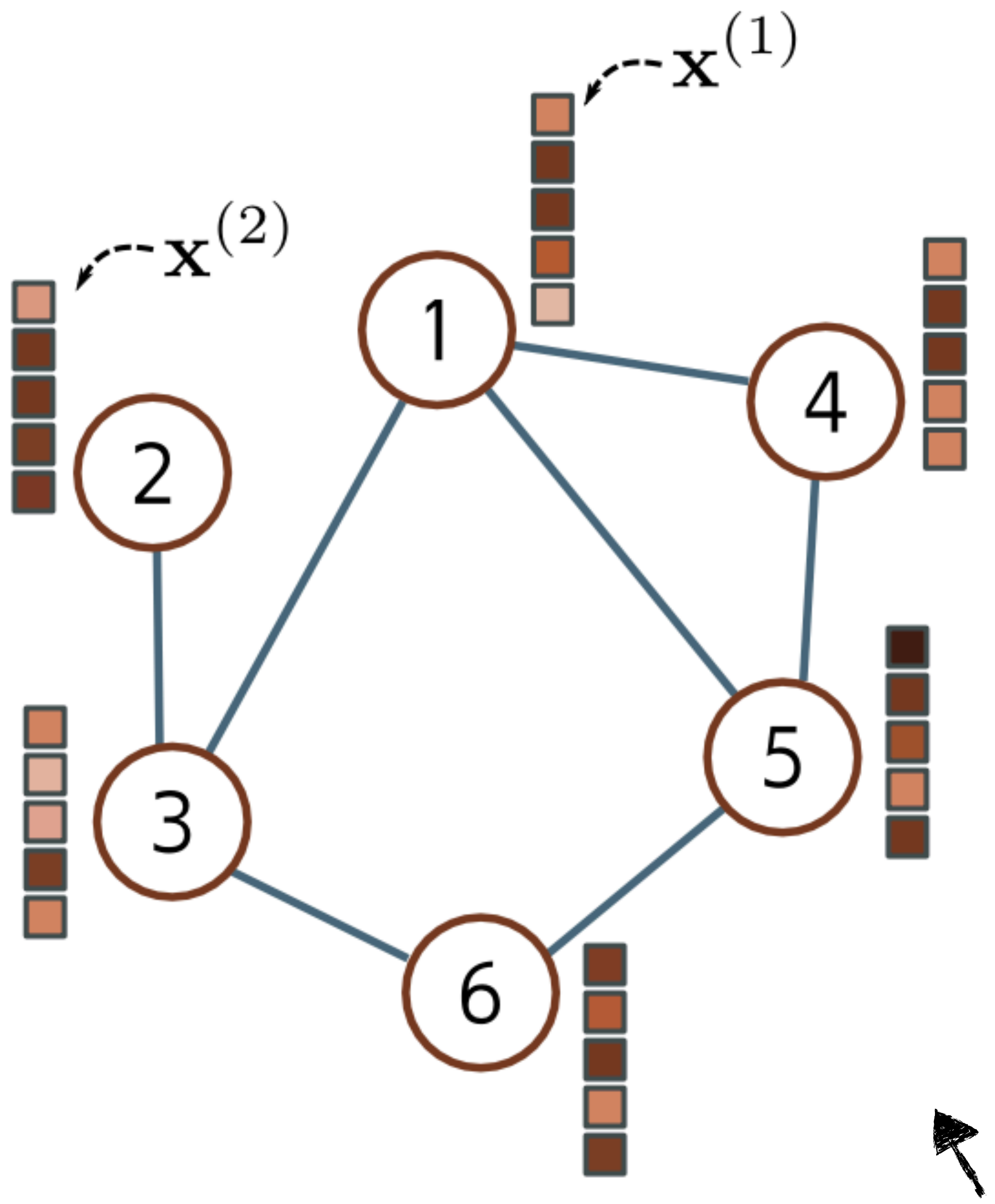
Pixel/Node updated

Convolution

Graph Convolution

BASIC GRAPH CONVOLUTION

- example of implementation of a basic graph convolution over an undirected graph with node labels specified by an V×V adjacency matrix A and V×D node feature matrix X:



- graph convolutions update the embeddings at each node based on the embeddings of their neighbors and themselves via a message passing operation that consists in two phases:

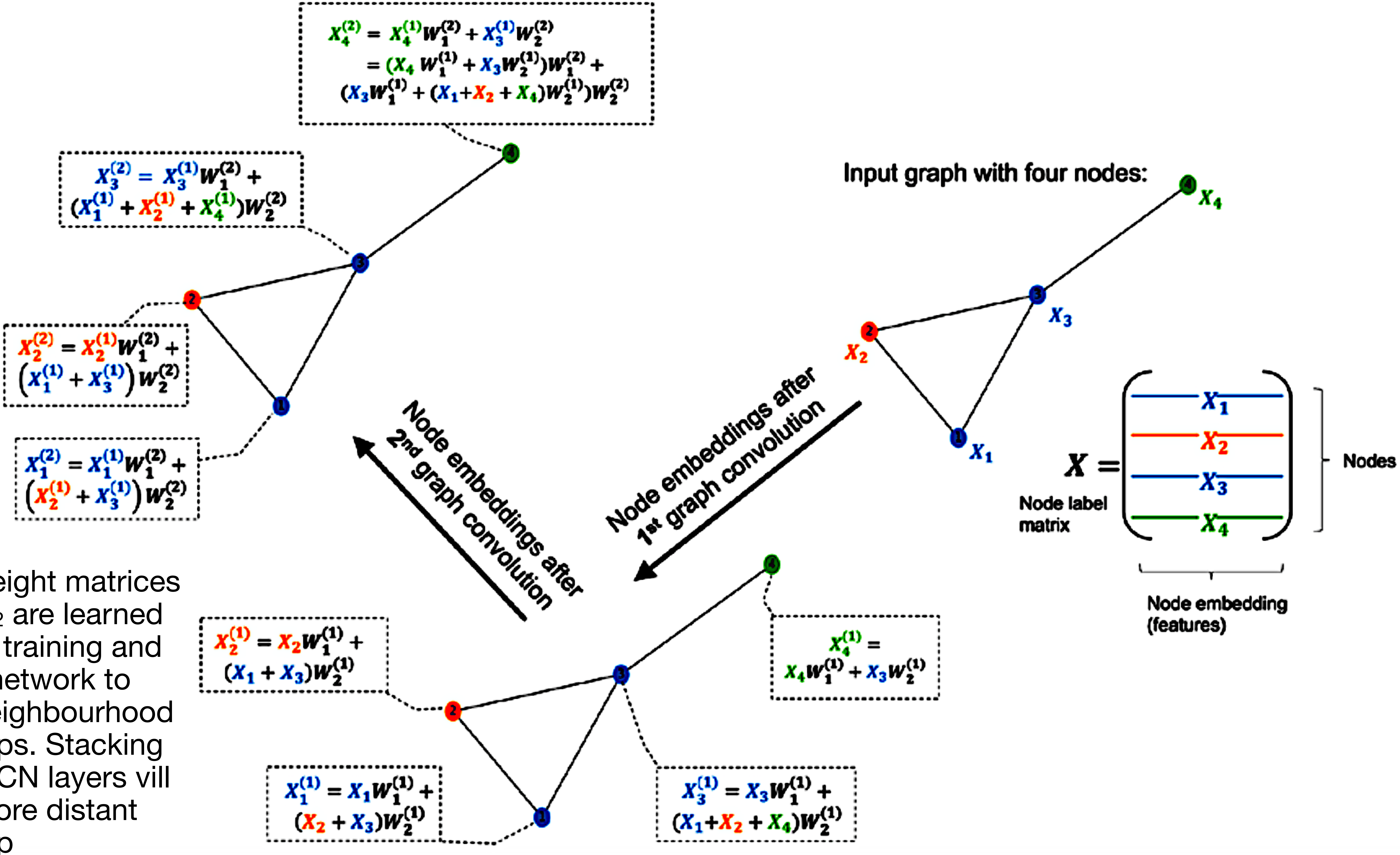
- aggregation phase: at each node *i* in layer *k*, information from neighboring nodes is aggregated by summing their node embeddings h:

$$AGG_i^{(k)} = \sum_{j \in N(i)} W_2^{(k)} h_j^{(k)}$$

- update phase: applies a linear transformation to the current node and to the aggregate message and pass the result through a non linear activation:

$$h_i^{(k+1)} = \phi[b^{(k)} + W_1^{(k)} h_i^{(k)} + AGG_i^{(k)}]$$

non linearity induced via the activation function ϕ


$X_4^{(2)} = X_4^{(1)} W_1^{(2)} + X_3^{(1)} W_2^{(2)}$
$= (X_4 W_1^{(1)} + X_3 W_2^{(1)}) W_1^{(2)} +$
$(X_3 W_1^{(1)} + (X_1 + X_2 + X_4) W_2^{(1)}) W_2^{(2)}$
4
$X_3^{(2)} = X_3^{(1)} W_1^{(2)} +$
$(X_1^{(1)} + X_2^{(1)} + X_4^{(1)}) W_2^{(2)}$
3
2
$X_2^{(2)} = X_2^{(1)} W_1^{(2)} +$
$\left(X_1^{(1)} + X_3^{(1)}\right) W_2^{(2)}$
1
$X_1^{(2)} = X_1^{(1)} W_1^{(2)} +$
$\left(X_2^{(1)} + X_3^{(1)}\right) W_2^{(2)}$
Input graph with four nodes:
4
X_4
3
X_3
2
X_2
1
X_1
Node embeddings after 2nd graph convolution
Node embeddings after 1st graph convolution
$X =$
Node label matrix
X_1
X_2
X_3
X_4
Nodes
Node embedding (features)
4
$X_4^{(1)} =$
$X_4 W_1^{(1)} + X_3 W_2^{(1)}$
$X_2^{(1)} = X_2 W_1^{(1)} +$
$(X_1 + X_3) W_2^{(1)}$
2
3
1
$X_1^{(1)} = X_1 W_1^{(1)} +$
$(X_2 + X_3) W_2^{(1)}$
$X_3^{(1)} = X_3 W_1^{(1)} +$
$(X_1 + X_2 + X_4) W_2^{(1)}$

- the two weight matrices W_1 and W_2 are learned during the training and allow the network to capture neighbourhood relationships. Stacking multiple GCN layers vill capture more distant relationship

BATCHING MULTIPLE GRAPHS

- MLP, CNN and transformers all leverage the parallelism of modern GPU hardware to concurrently process an entire batch simultaneously. However in GNN each graph may have a different number of nodes. Hence, the matrices X_i and A_i have different sizes and there is no way to concatenate them into 3D tensors

- solved by a simple trick: the graphs in the batch are treated as disjoint components of a single large graph. The network can then be run as a single instance of the network equations. The mean pooling is carried out only over the individual graphs to make a single representation per graph that can be fed into the loss function

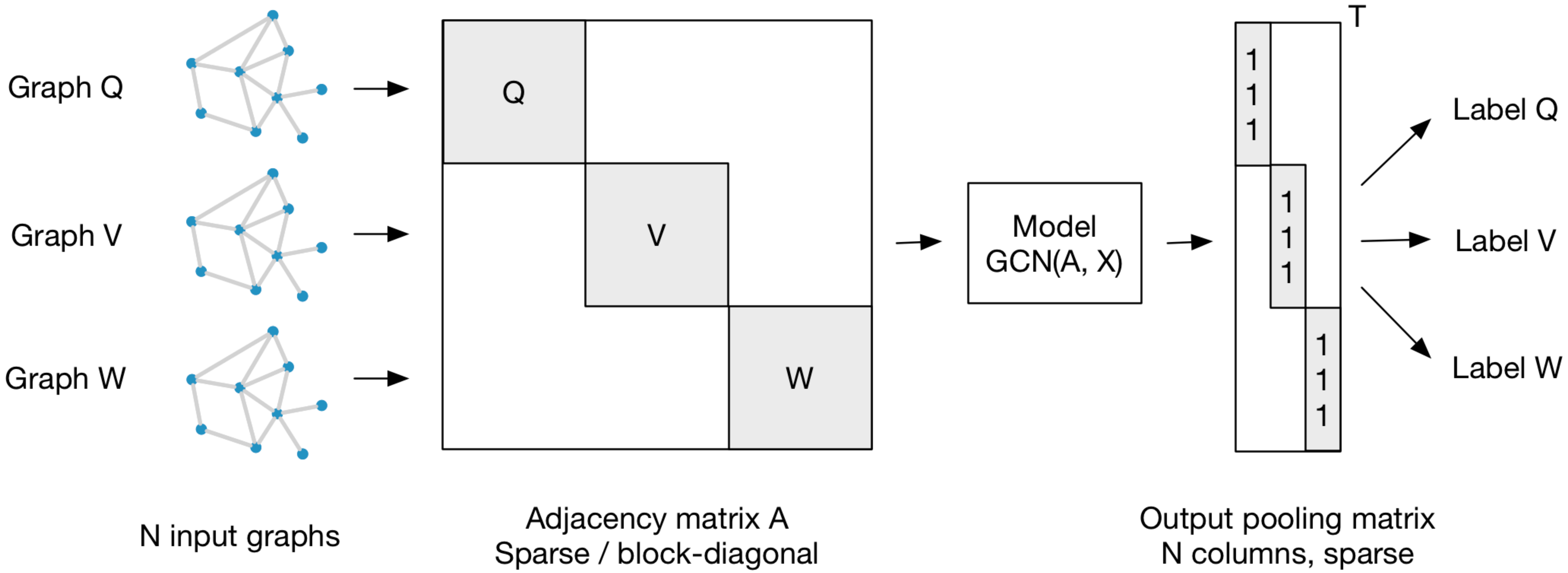
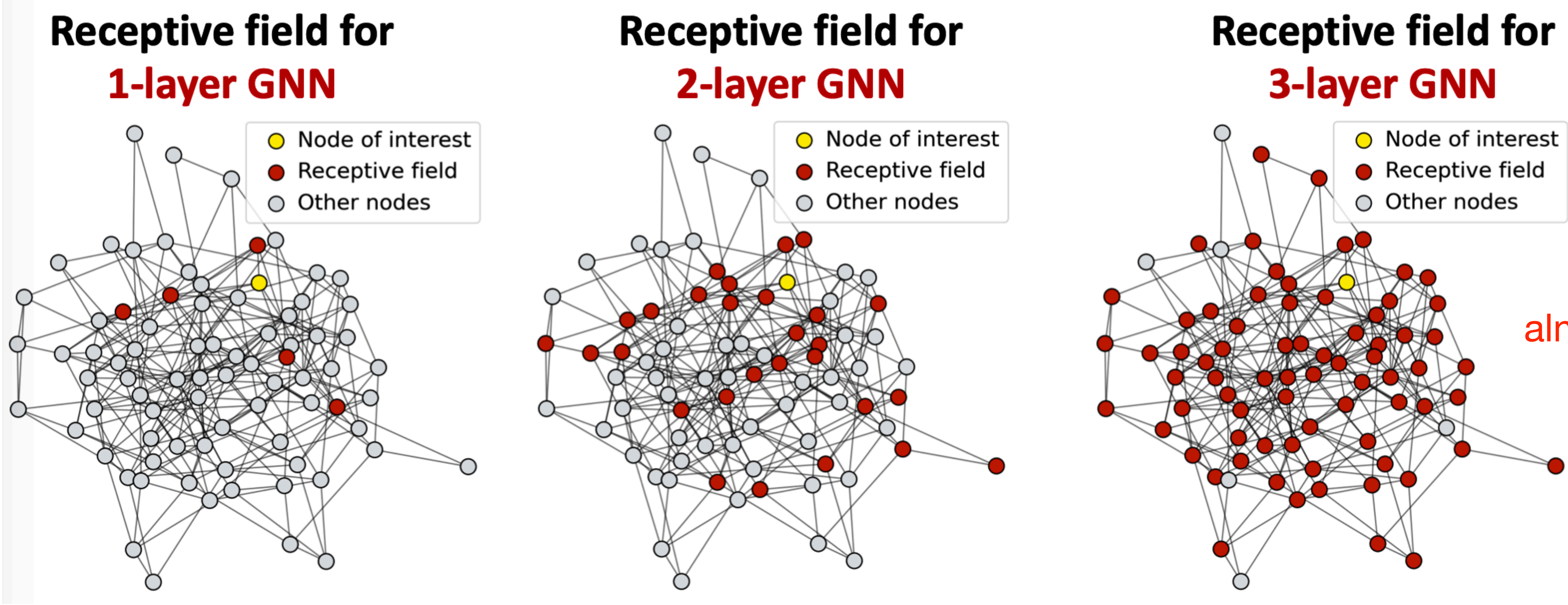


figure from https://github.com/tkipf/gcn

OVER-SMOOTHING IN GNNs

- a common issue with GNNs is the so called over-smoothing: after several iterations of GNN message passing all the nodes embedding converge to the same value
- tendency especially common in basic GNN models that makes it impossible to build very deep GNN models which leverage longer-term dependencies in the graph
    - over smoothing is related to the receptive field of the GNN (e.g. the set of nodes that determine the embedding of a node of interest)
    - if two nodes have highly-overlapped receptive fields, then their embeddings are highly similar

in a K-layers GNN, each node has a receptive field of K-hop neighborhood



almost all the nodes!

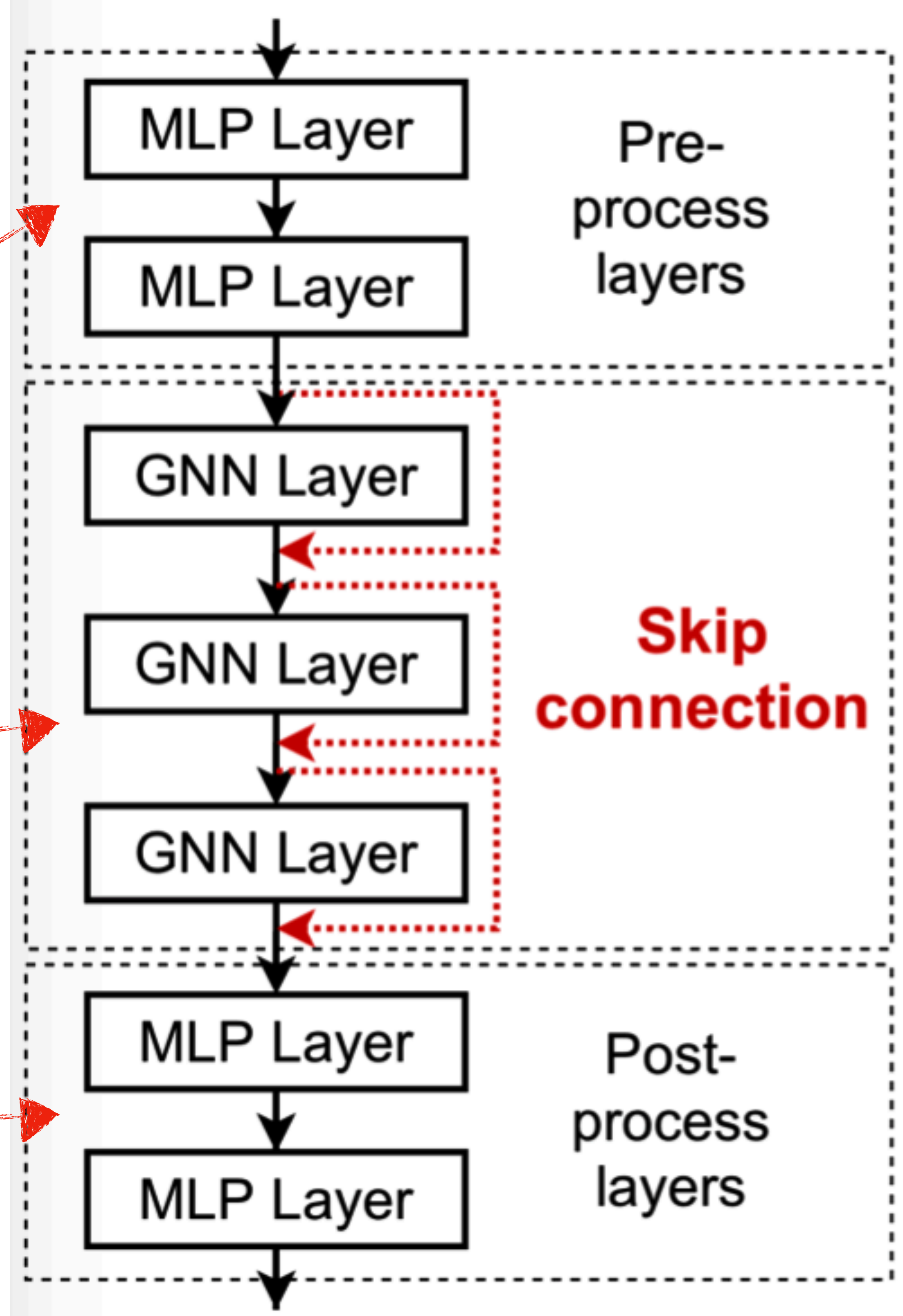
HOW TO AVOID OVER-SMOOTHING PROBLEMS

- analyze the necessary receptive field to solve your problem. e.g., by computing the diameter of the graph
- set number of GNN layers just slightly larger than the needed receptive, refrain to set it to unnecessarily large values
- to increase the expressive power of the GNN with a small number of layers:
  - increase the expressive power of each single layer: add layers that do not pass messages, message passing based on deep MLP networks or/and use attention or multi-head attention mechanisms and/or use of skip connections

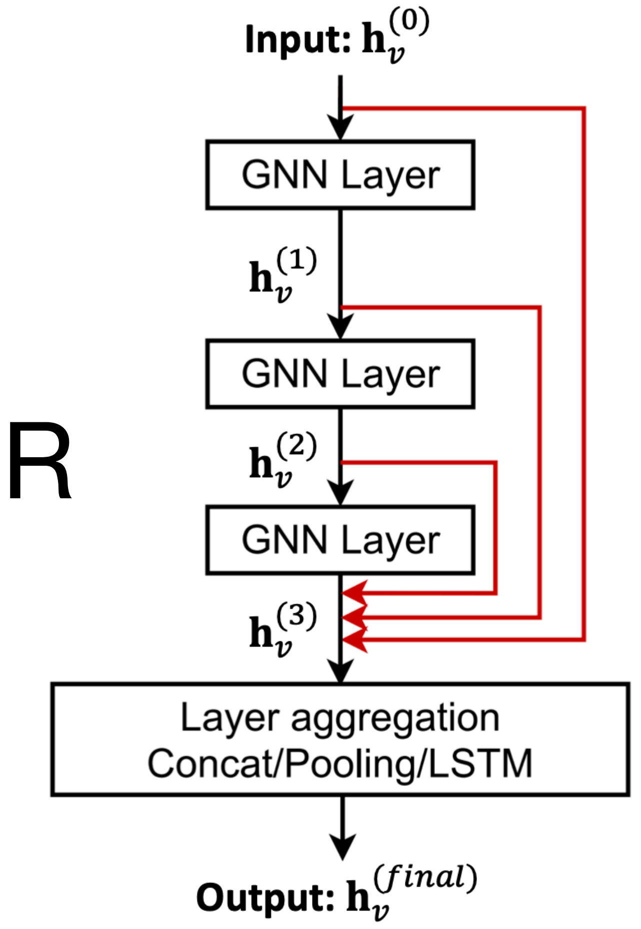
pre-processing layers: used when encoding node features is necessary (as ex. when nodes represent images/text)

skip connections: mix shallow and deep GNNs, so that residual networks works like ensemble of shallow networks

post-processing layers: used when reasoning / transformation over node embeddings are needed
example: graph classification, knowledge graphs



OR


Input: $\mathbf{h}_v^{(0)}$
GNN Layer
$\mathbf{h}_v^{(1)}$
GNN Layer
$\mathbf{h}_v^{(2)}$
GNN Layer
$\mathbf{h}_v^{(3)}$
Layer aggregation Concat/Pooling/LSTM
Output: $\mathbf{h}_v^{(final)}$