POLITECNICO
MILANO 1863

IMAGE AND SOUND
ISPG
PROCESSING GROUP

CREATIVE PROGRAMMING AND COMPUTING

convNET and RNN

CONVOLUTIONAL NETWORK

CONV NET (CNN)

- Input data exhibits an abstraction of features and concepts
- E.g. in image recognition
  - Pixel → edge → texton → motif → part → object
- With MLP we let the network to learn the abstraction

- **However, how brain has specific neurons devoted the analysis and abstract the visual scene.**

CONV NET (CNN)

- Inspired by the human biology of the **visual cortex**

- CNN are variants of MLPs.

- From Hubel and Wiesel's early work: the **visual cortex** contains a complex arrangement of cells. These cells are sensitive to small sub-regions of the visual field, called a receptive field. The sub-regions are tiled to cover the entire visual field.
- These cells act as local filters over the input space
- Additionally, two basic cell types have been identified:
  - **Simple cells** respond maximally to specific **edge-like patterns** within their receptive field.
  - **Complex cells:** have **larger receptive fields** and are locally invariant to the exact position of the pattern.

FILTERS AND CONVOLUTION

- The general formulation of convolution is

$$g(x,y) = \omega * f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} \omega(s,t) f(x-s, y-t)$$

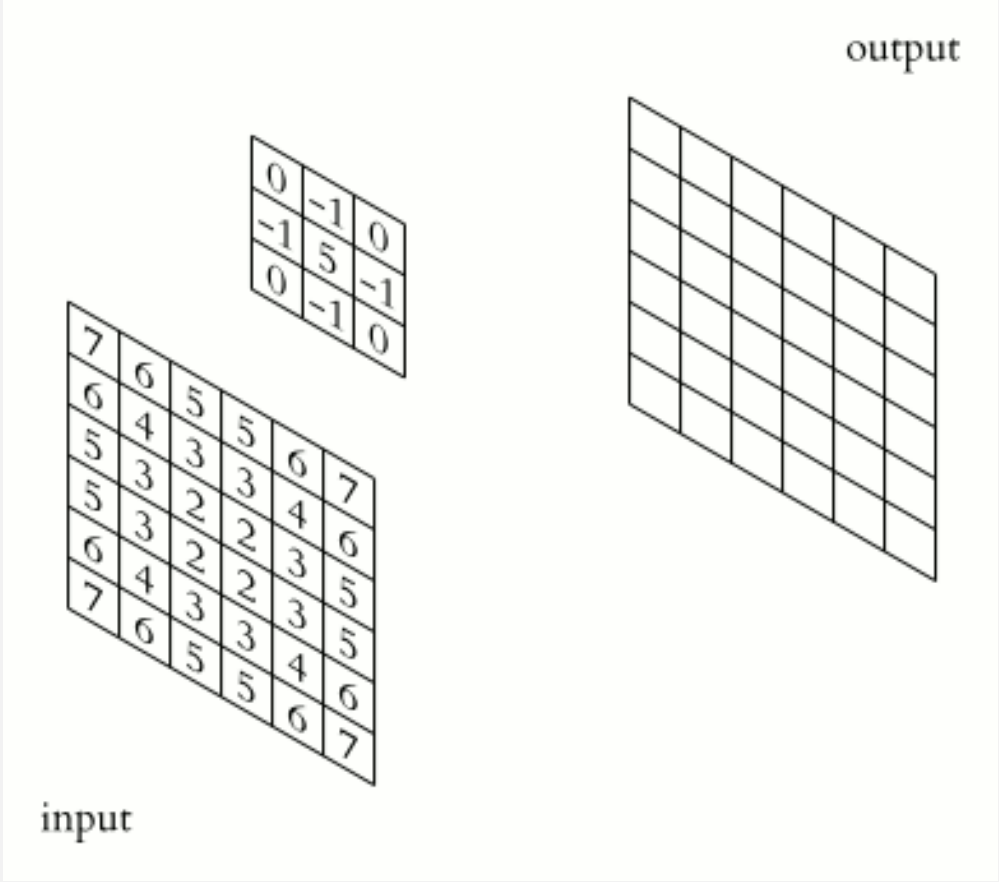
- Where g(x,y) is the filtered image, f(x,y) is the original image, ω is the filter kernel

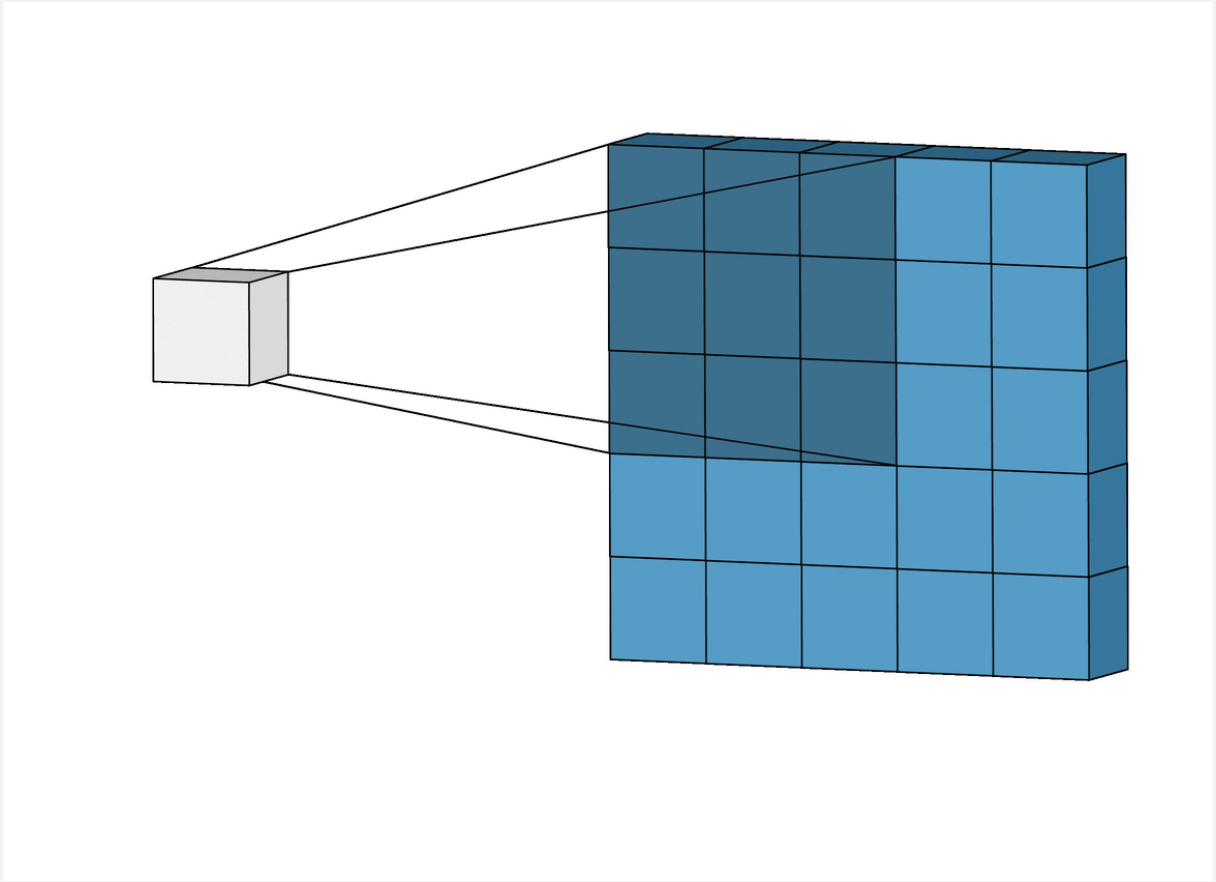
$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

- Kernel in the edge detection process
- Differently sized kernels containing different patterns of numbers produce different results under convolution
- Different kernels and different kernel sizes are able to **capture** different **patterns**

FILTERS AND CONVOLUTION

$$g(x,y) = \omega * f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} \omega(s,t) f(x-s, y-t)$$





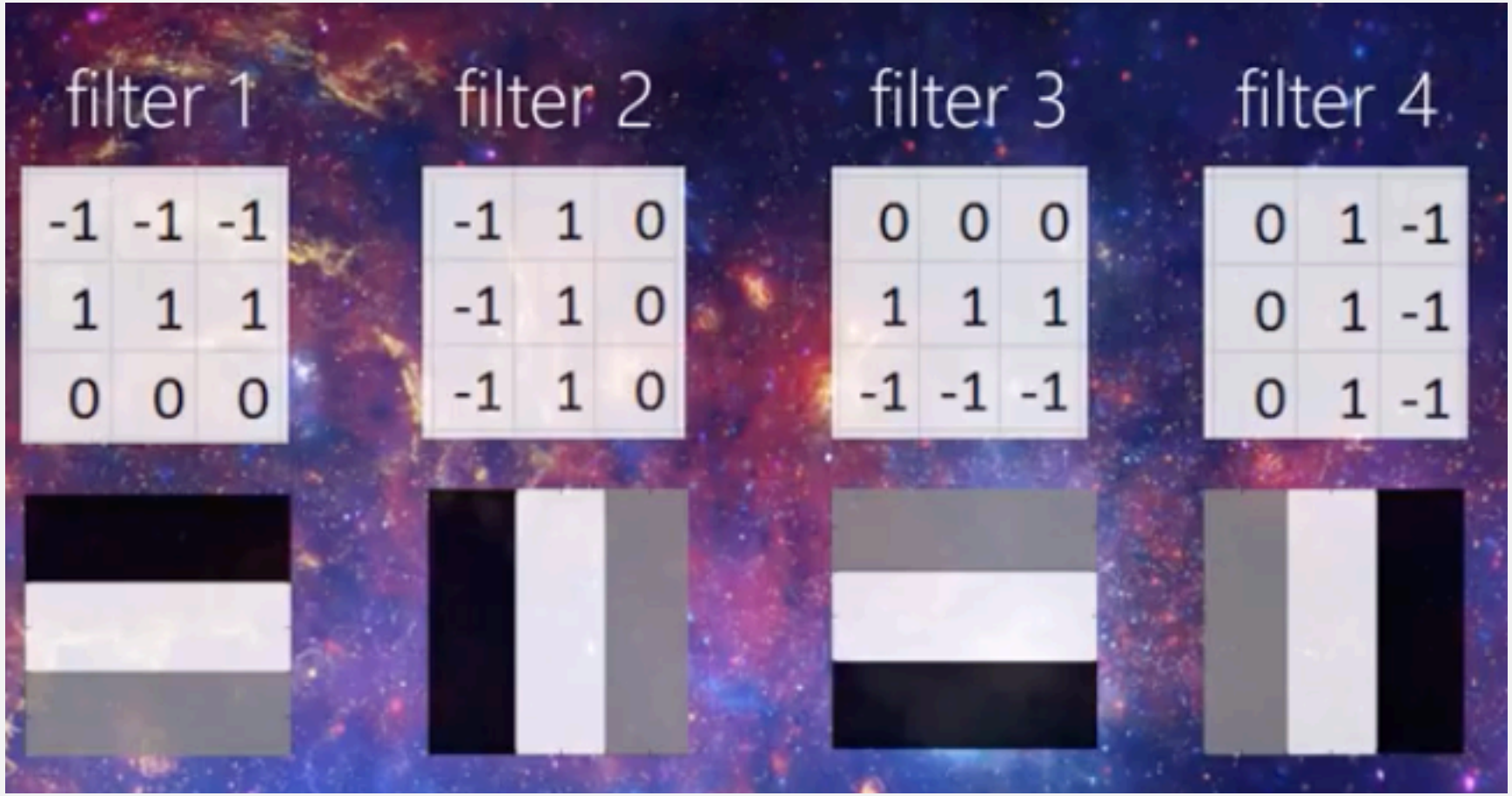
FILTERS AND CONVOLUTION

Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

https://setosa.io/ev/image-kernels/

FILTERS AND CONVOLUTION

- We can se the filters as the images of the patterns to capture
- The filters below detect edges



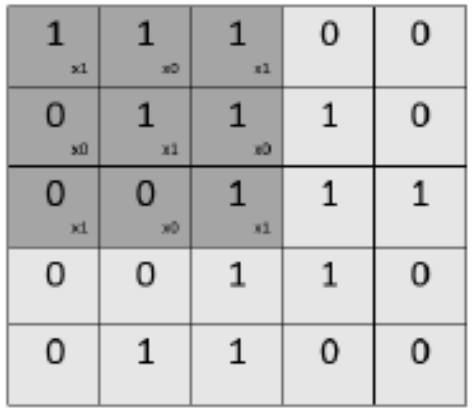
CONVOLUTION

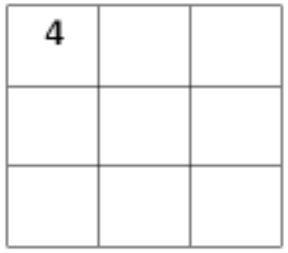
- Kernel =

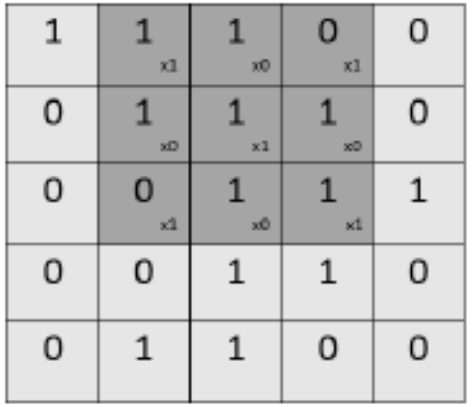
[1 0 1

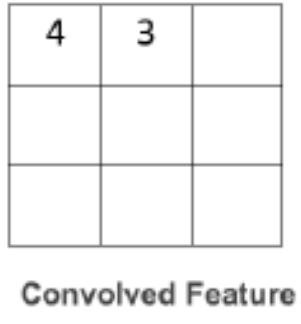
0 1 0

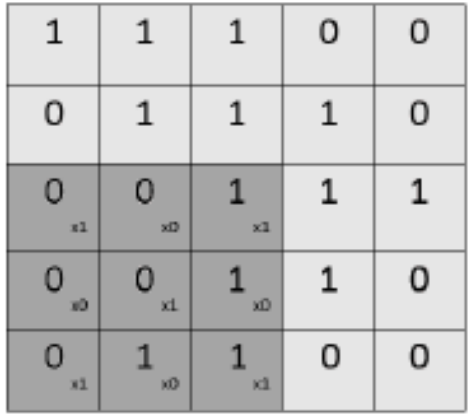
1 0 1]

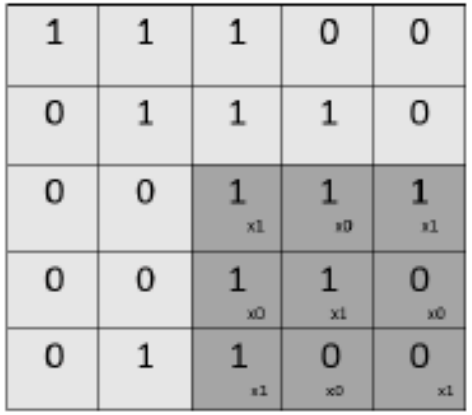


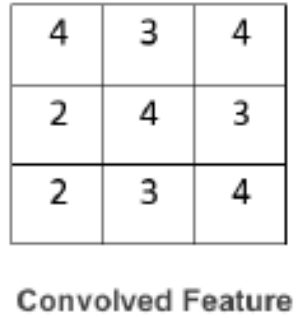












0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

In the specific case of images also the three channels are summed

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

Kernel Channel #1

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

Kernel Channel #2

0	1	1
0	1	0
1	-1	1

Kernel Channel #3

$308 + (-498) + 164 + 1 = -25$

Bias = 1

Output

-25				...
				...
				...
				...
...	...	...	...	...

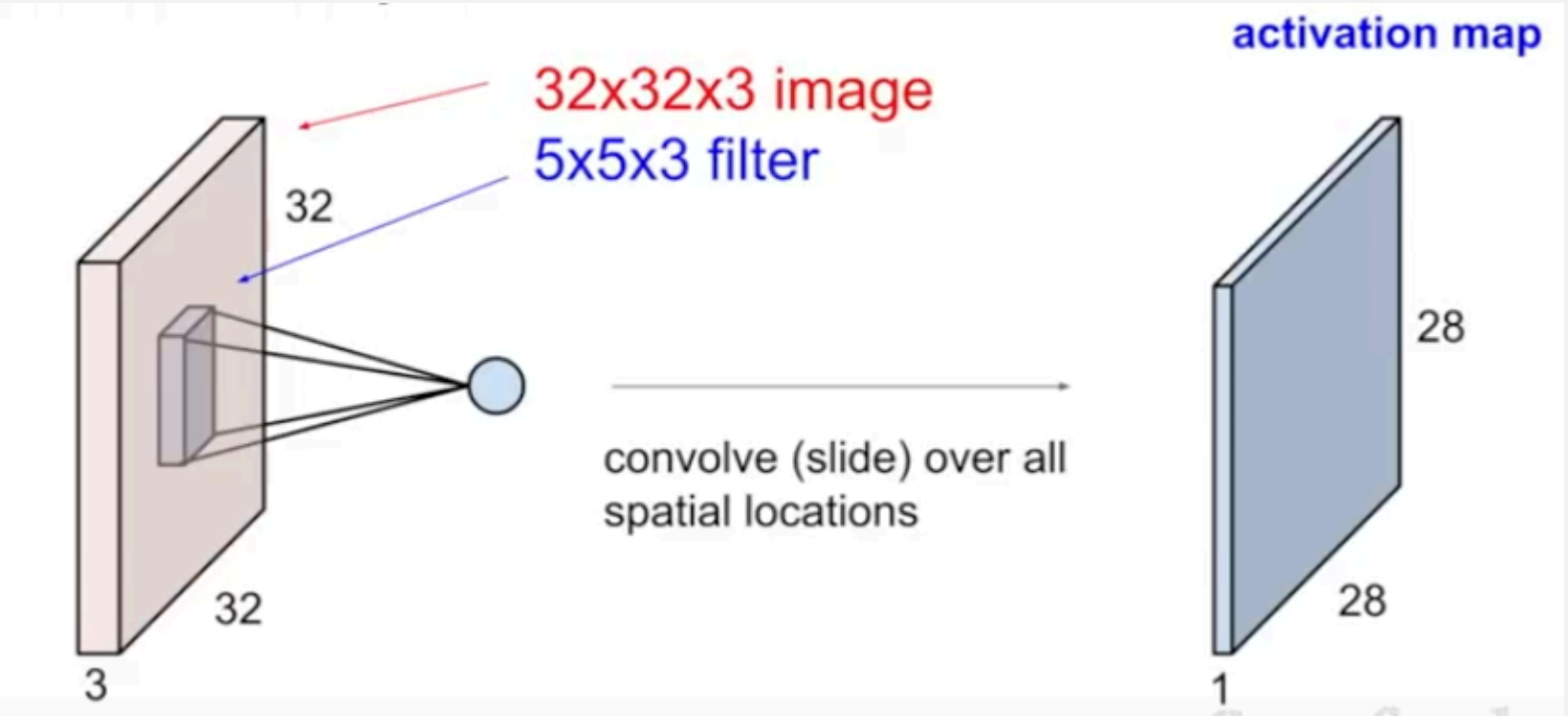
CONVOLUTIONAL LAYERS – IMAGES

- Let's think to have an image (32x32x3) – 3 -> RGB
- Given a filter 5x5x3 it produces an **activation map** or **feature maps**
- With the zero padding the activation maps is transformed into a 32x32x3 image
- **Pixels of the image are the inputs (each pixel is an input value)**

- In MLP the neuron

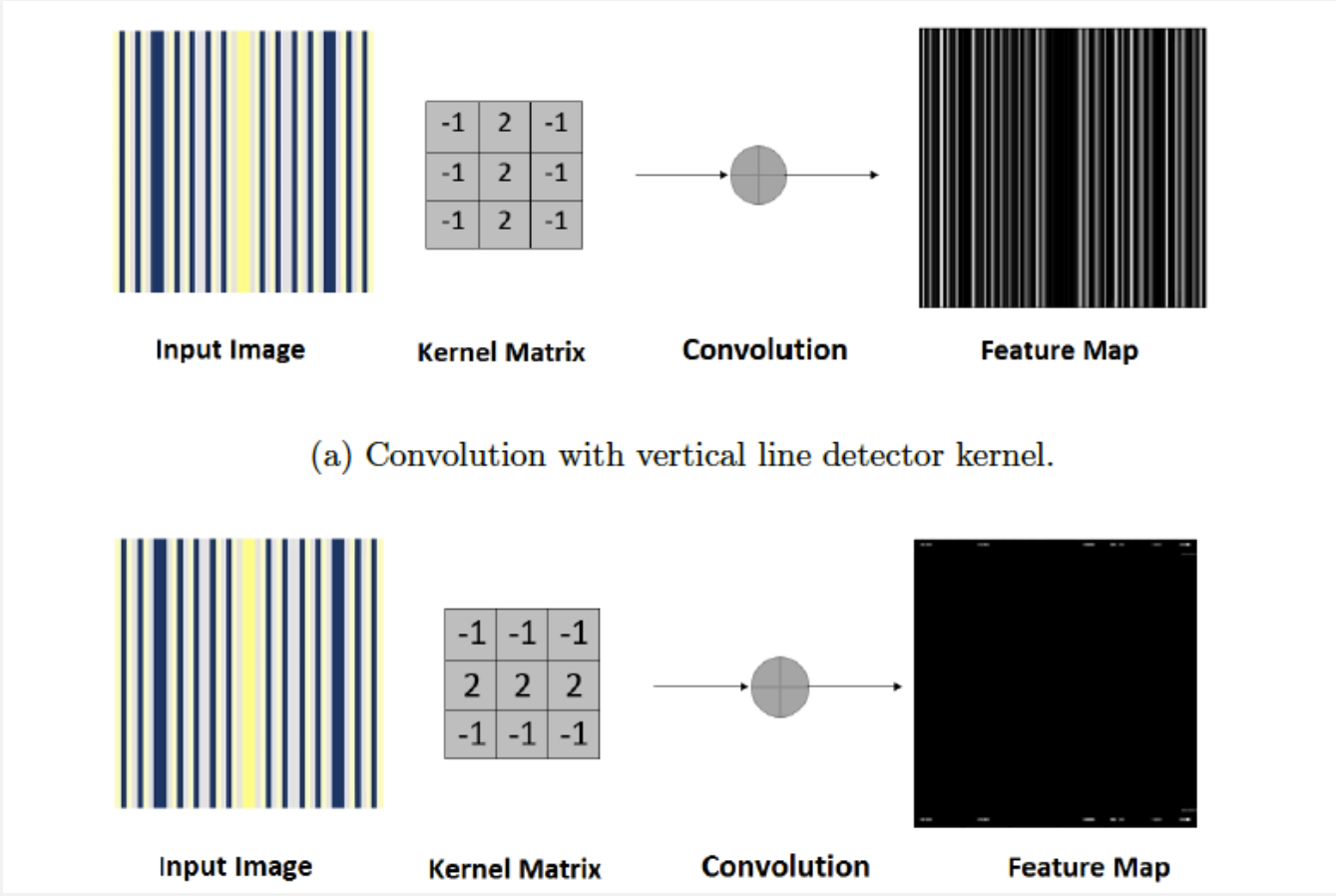
$$h(\sum_{i}^{I} w_{ji} * x_i)$$

- Here we substitute multiplication with the convolution
- **Filter values are the weights**



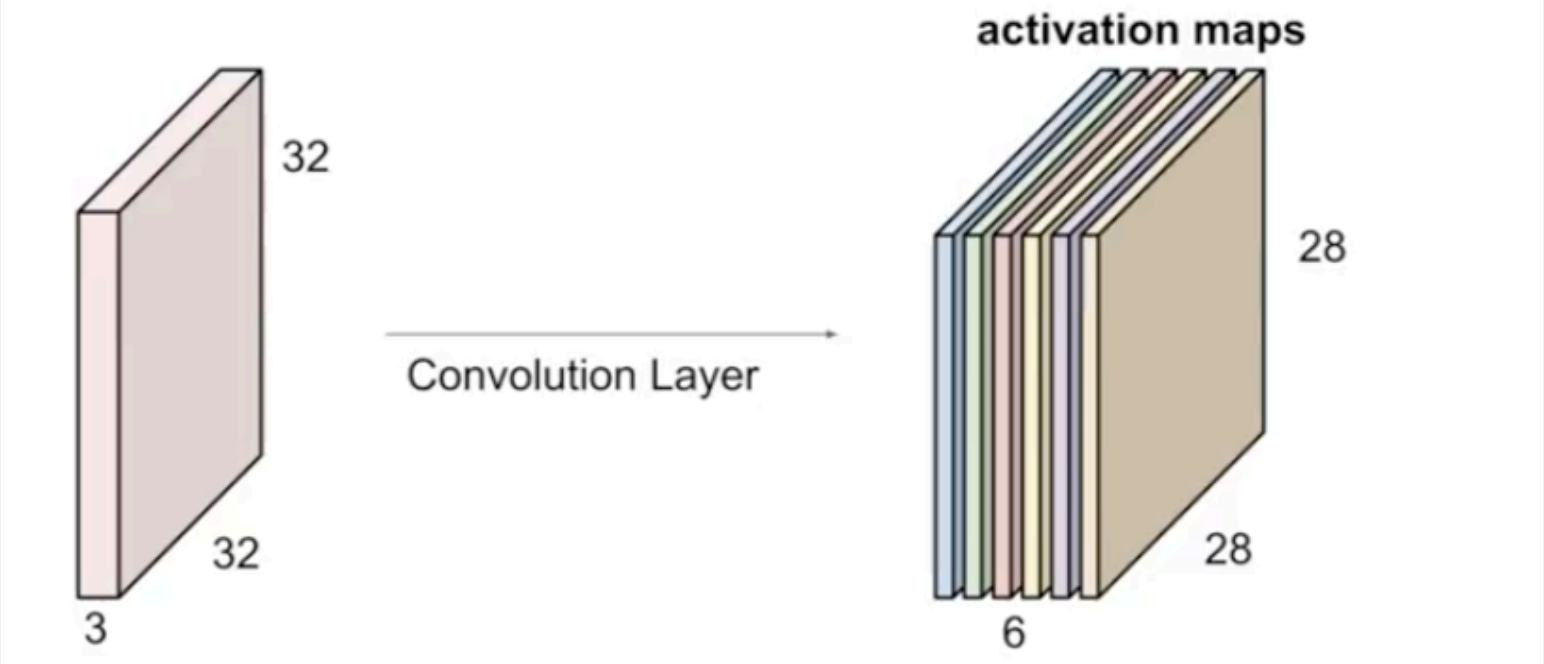
CONVOLUTIONAL LAYERS

- The intuition behind this process is that the CNN will learn to adjust the filters so that they will activate when they see some type of specific visual features, such as an edge of some orientation or a portion of some colour



CONVOLUTIONAL LAYERS – IMAGES

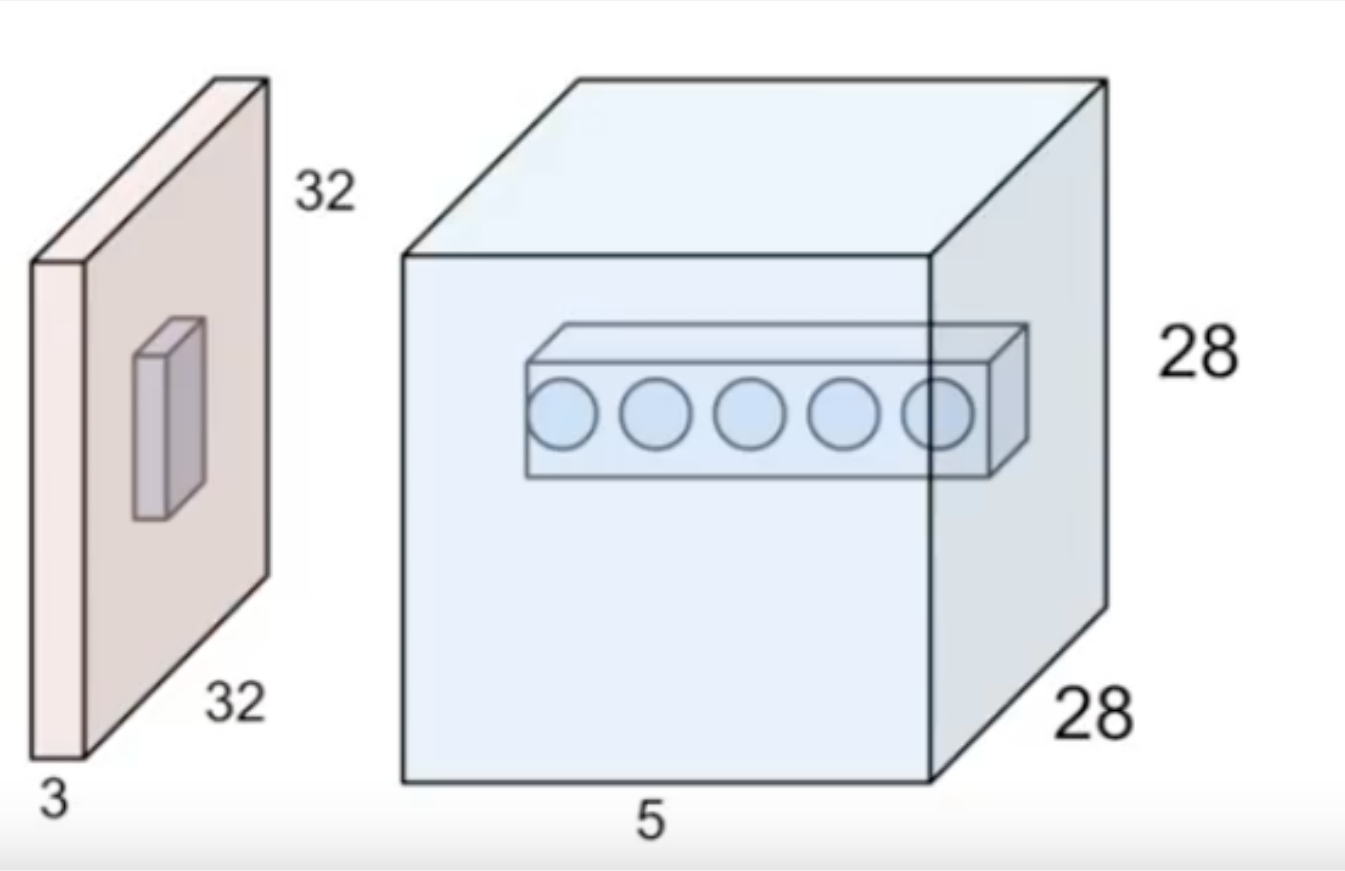
- For each convolutional layer we can have more filters – **filters** are called **neurons**
- If we have N filters in the convolutional layer, than we have N activation maps
- The number of filters in the layer is called **Depth** of the convolutional layer



- We can stack the activation maps to have a unique map of 28x28x6 values (activation values)

CONVOLUTIONAL LAYERS – IMAGES

- Given a specific region in the input image
- Let's have a layer with depth 5 (5 neurons) that produces 5 feature maps



- **We will have 5 different versions of the same region**

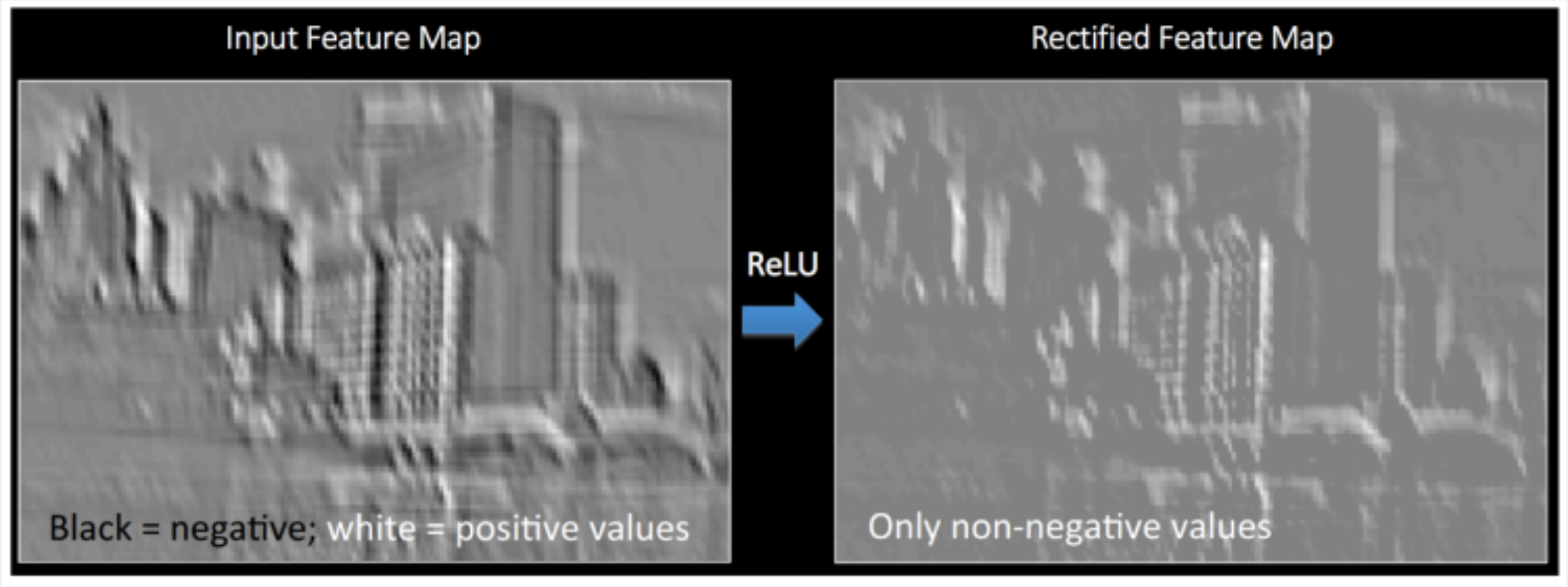
$$h\left(\sum_{i}^{I} w_{ji} * x_i\right)$$

Activation function

Activation map

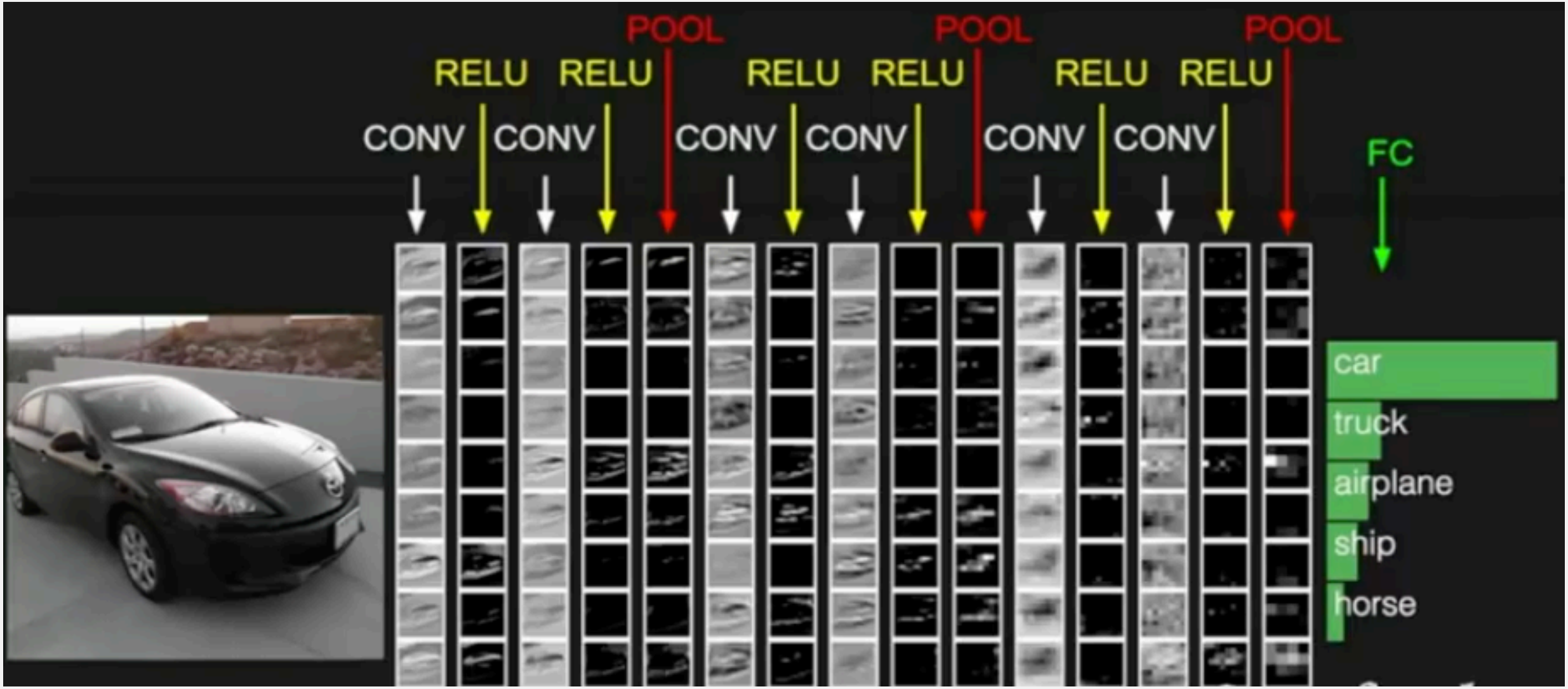
- Often it used in the combination as Convolutional Layer + ReLU

CONVOLUTIONAL LAYERS – IMAGES



CONVOLUTIONAL LAYERS – IMAGES

- Having multiple convolutional layers



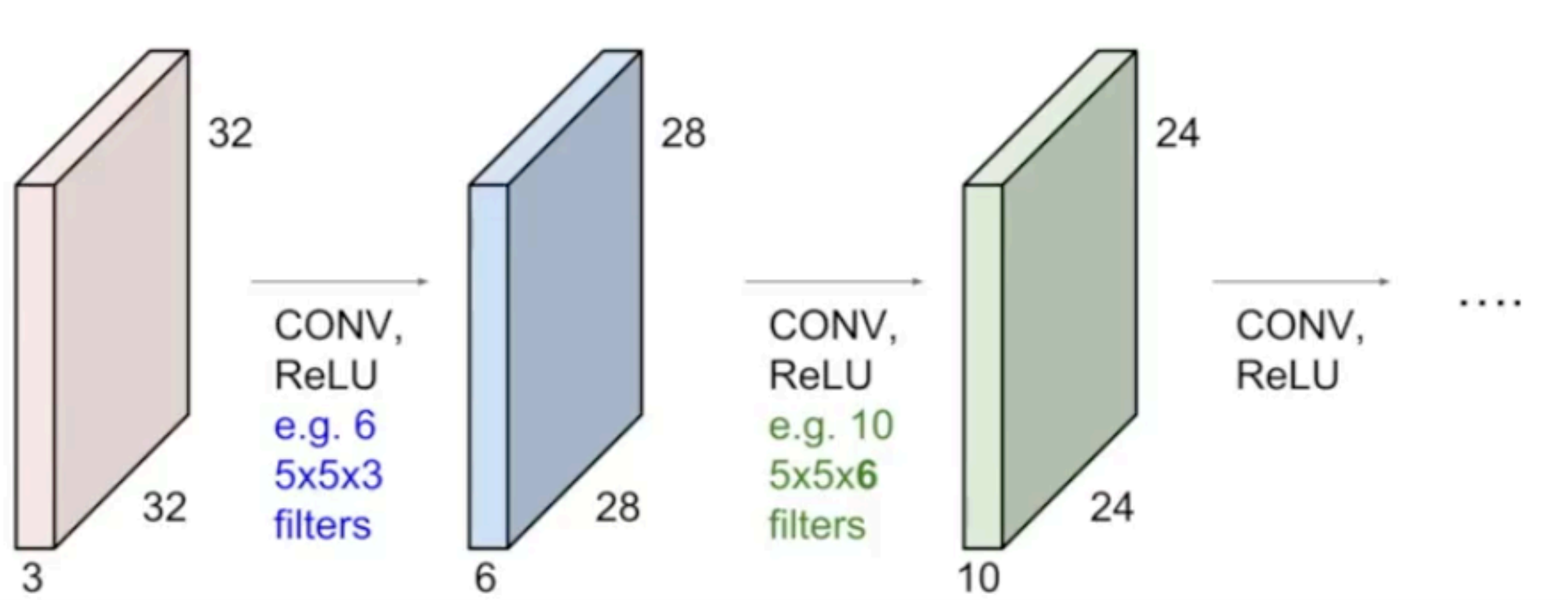
CONVOLUTIONAL LAYERS – IMAGES

- Having multiple convolutional layers

RGB = 3

Neurons = Filters = 6

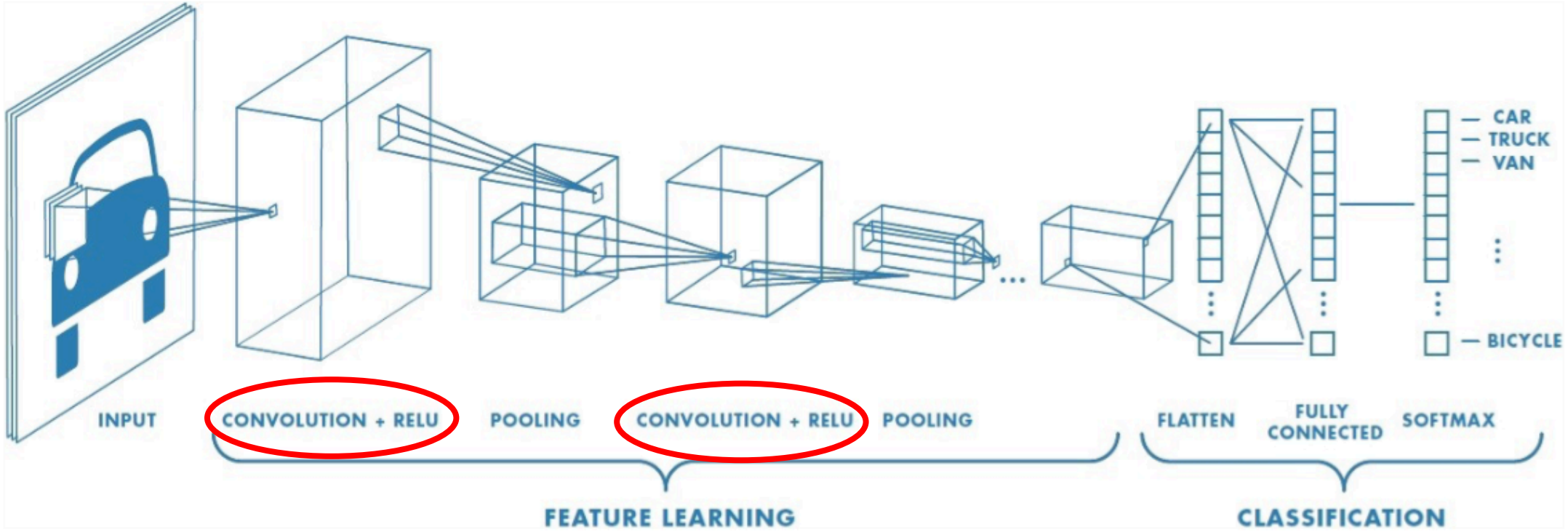
Neurons = Filters = 10



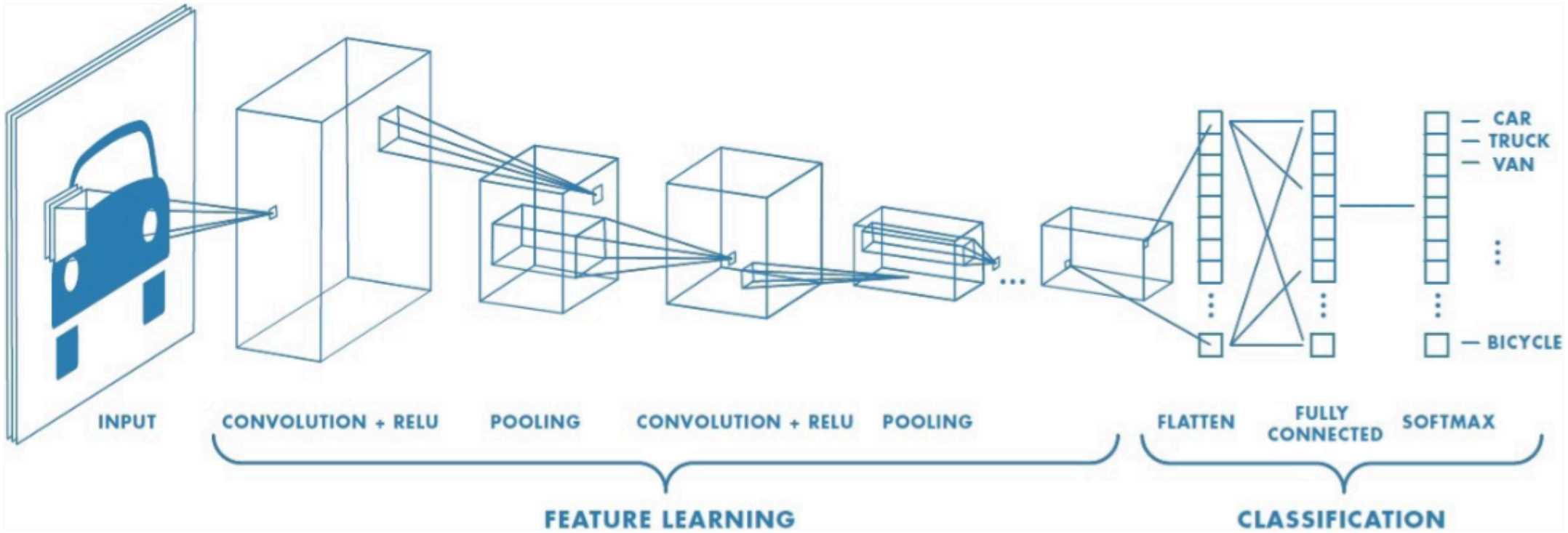
CONVOLUTIONAL LAYERS – IMAGES

- Given neuron to have an activation function here the most common activation function is ReLU

$$h(\sum_{i}^{I} w_{ji} * x_i)$$

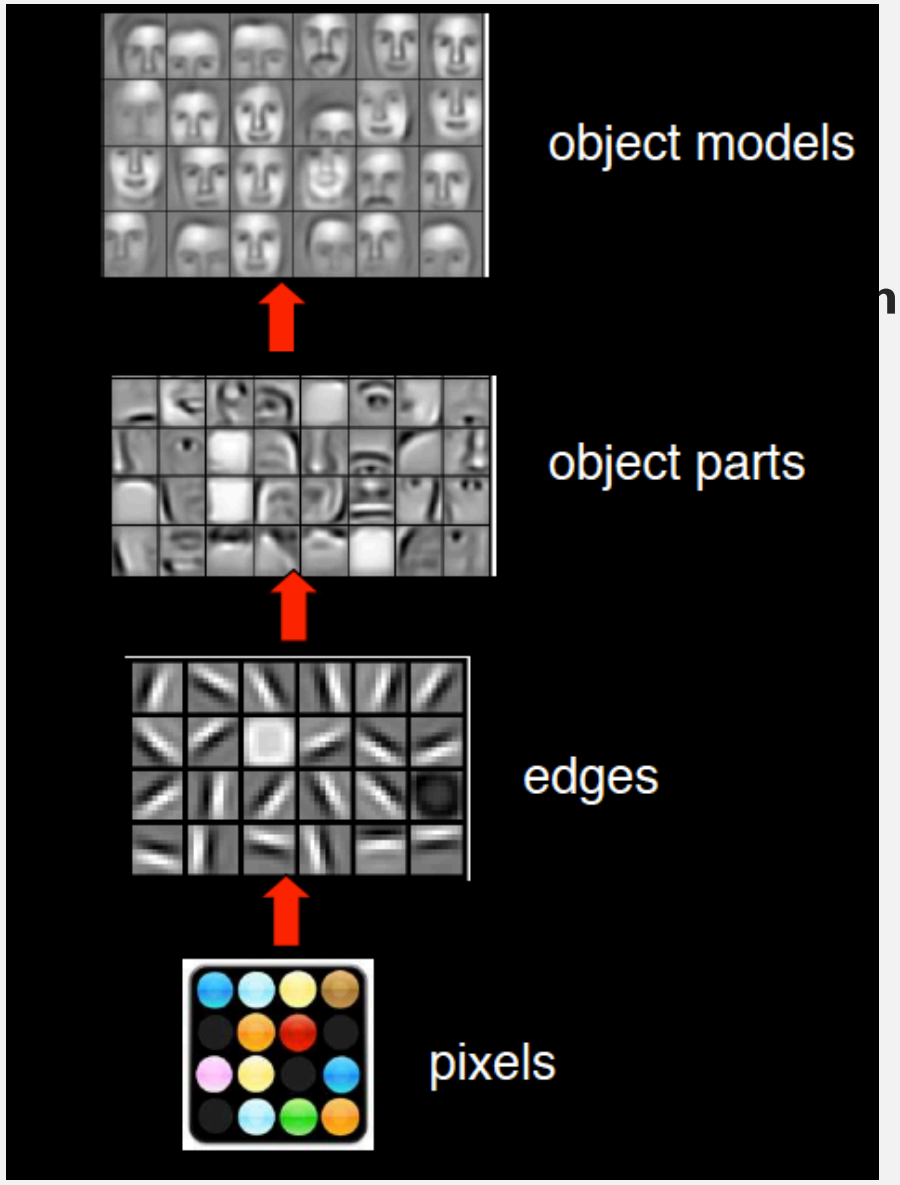


CONVOLUTIONAL NETWORK



CONVOLUTIONAL LAYERS – IMAGES

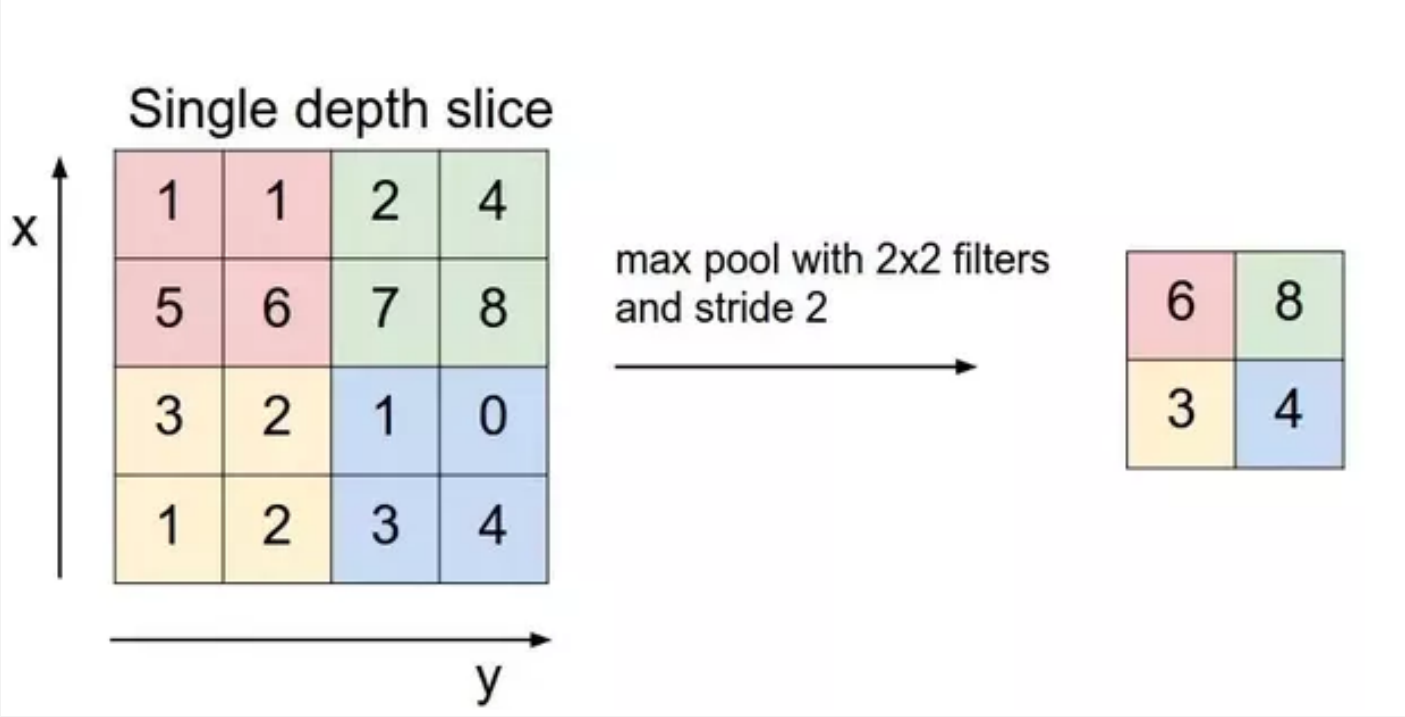
- Deeper convolutional layers are able to capture more abstract patterns



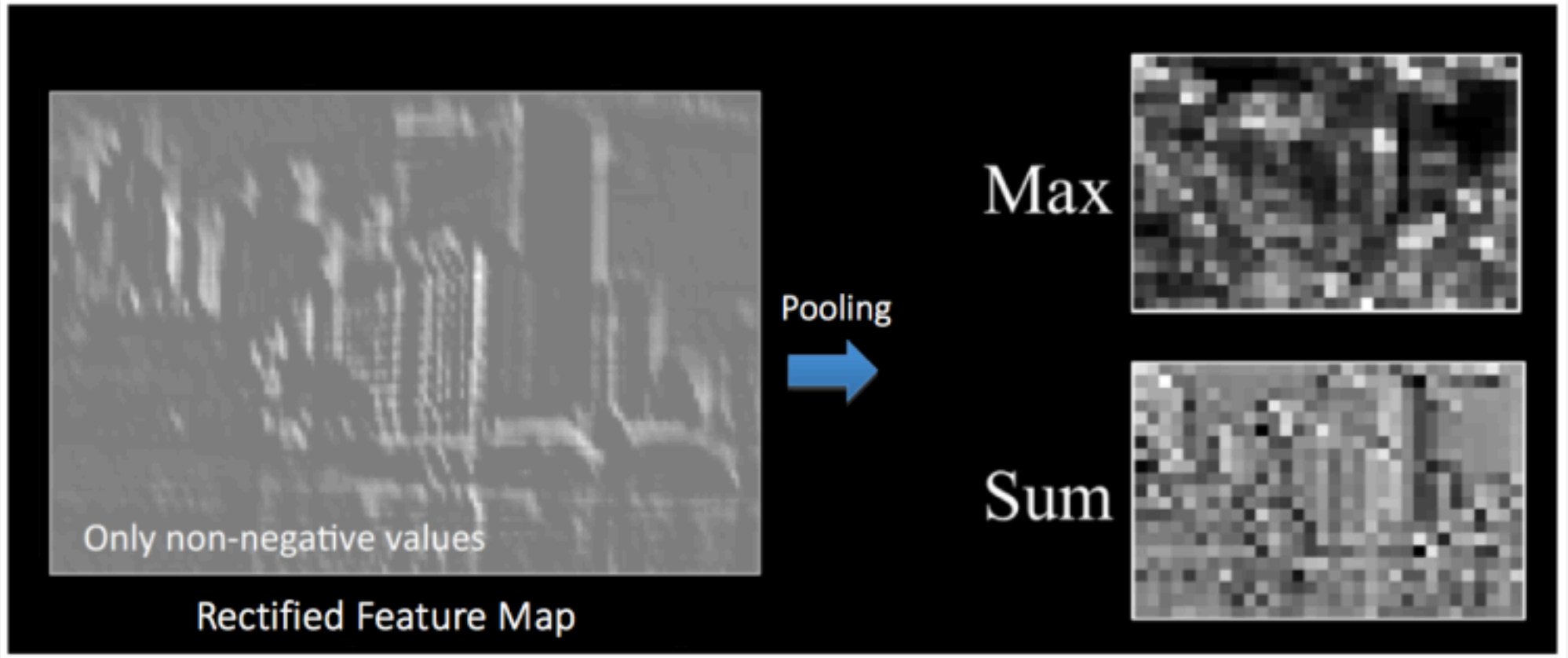
POOLING LAYER

- Pooling layers section would reduce the number of parameters when the images are too large
- Reduces the dimensionality of each activation map but retains important information. Spatial pooling can be of different types:
  - Max Pooling
  - Average Pooling
  - Sum Pooling

Max pooling takes the largest element from the rectified feature map

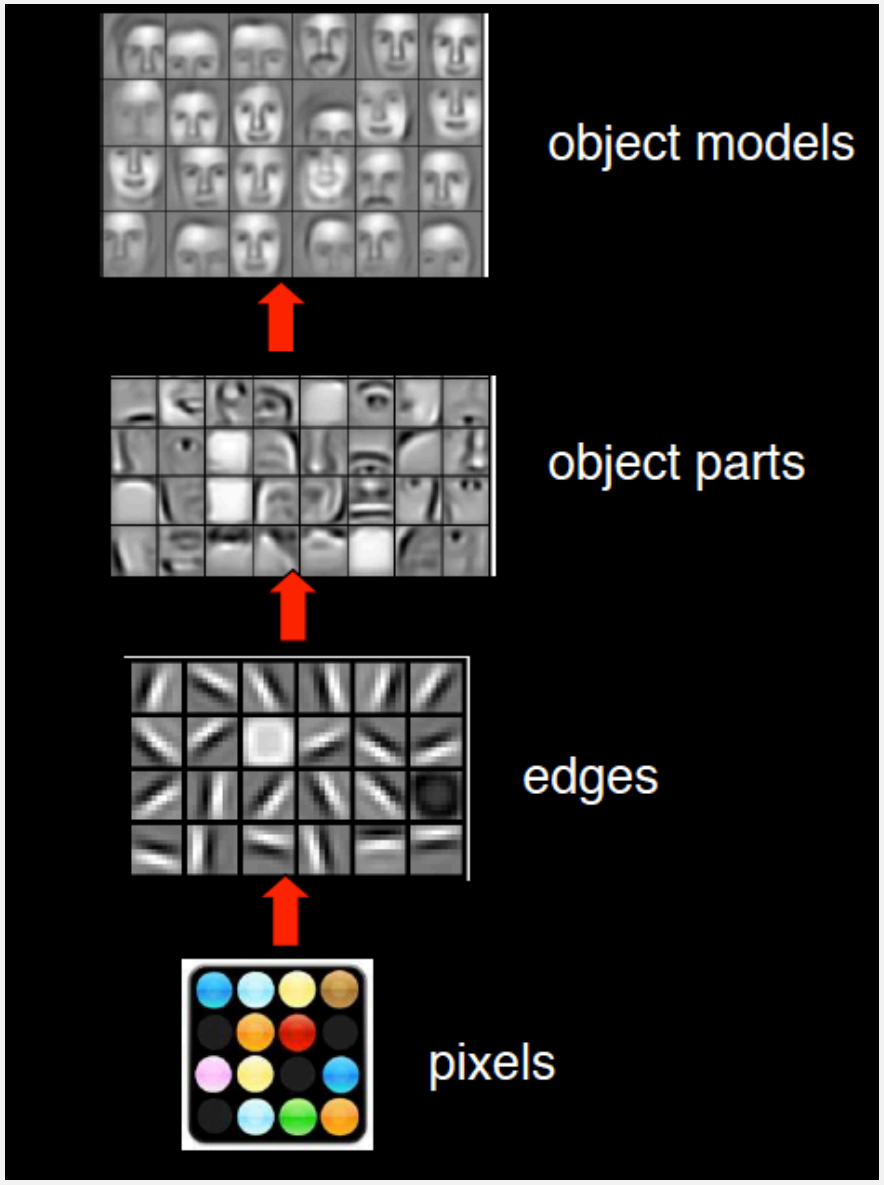


POOLING LAYER

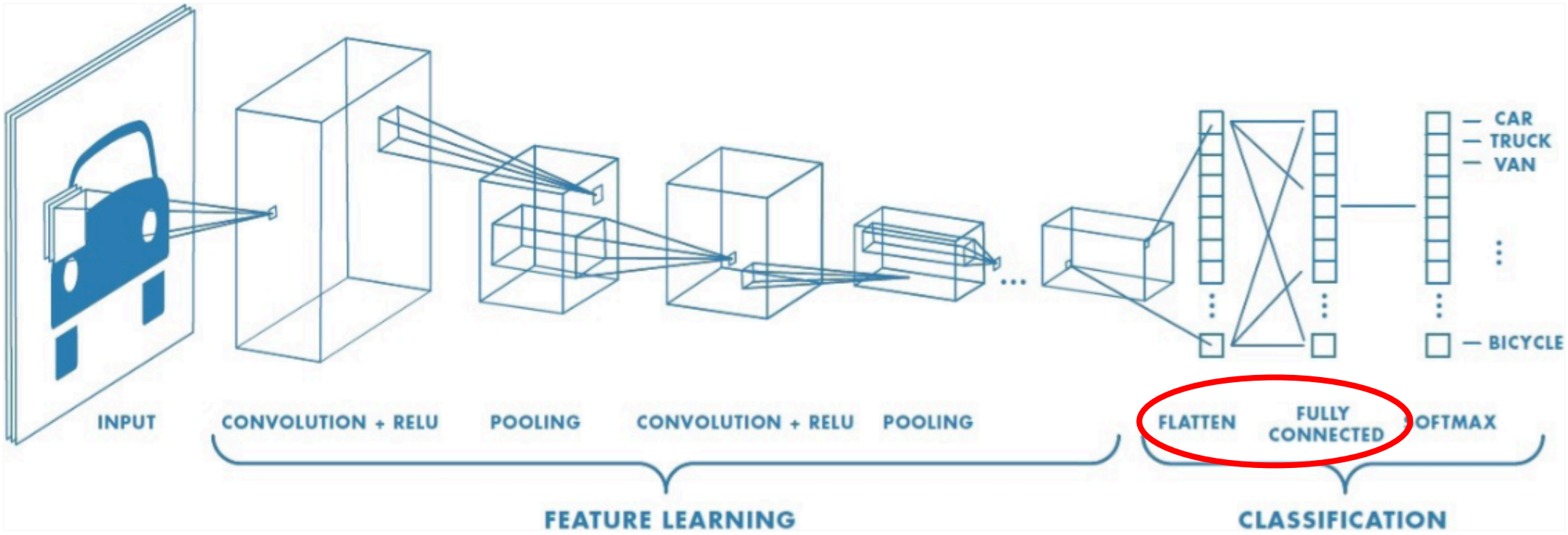


POOLING LAYER

- Max Pooling allows to perform a multi scale analysis
- It allows zoom out and capture different structures and patterns



CONVOLUTIONAL NETWORK



- Flatten layer transform the 3D martix into a vector to feed in input to the fully connected layer
- Fully connected layer is a MLP

- Some architecture are popular LeNet, AlexNet, VGGNet, GoogLeNet, ResNet, ZFNet

http://scs.ryerson.ca/~aharley/vis/conv/flat.html

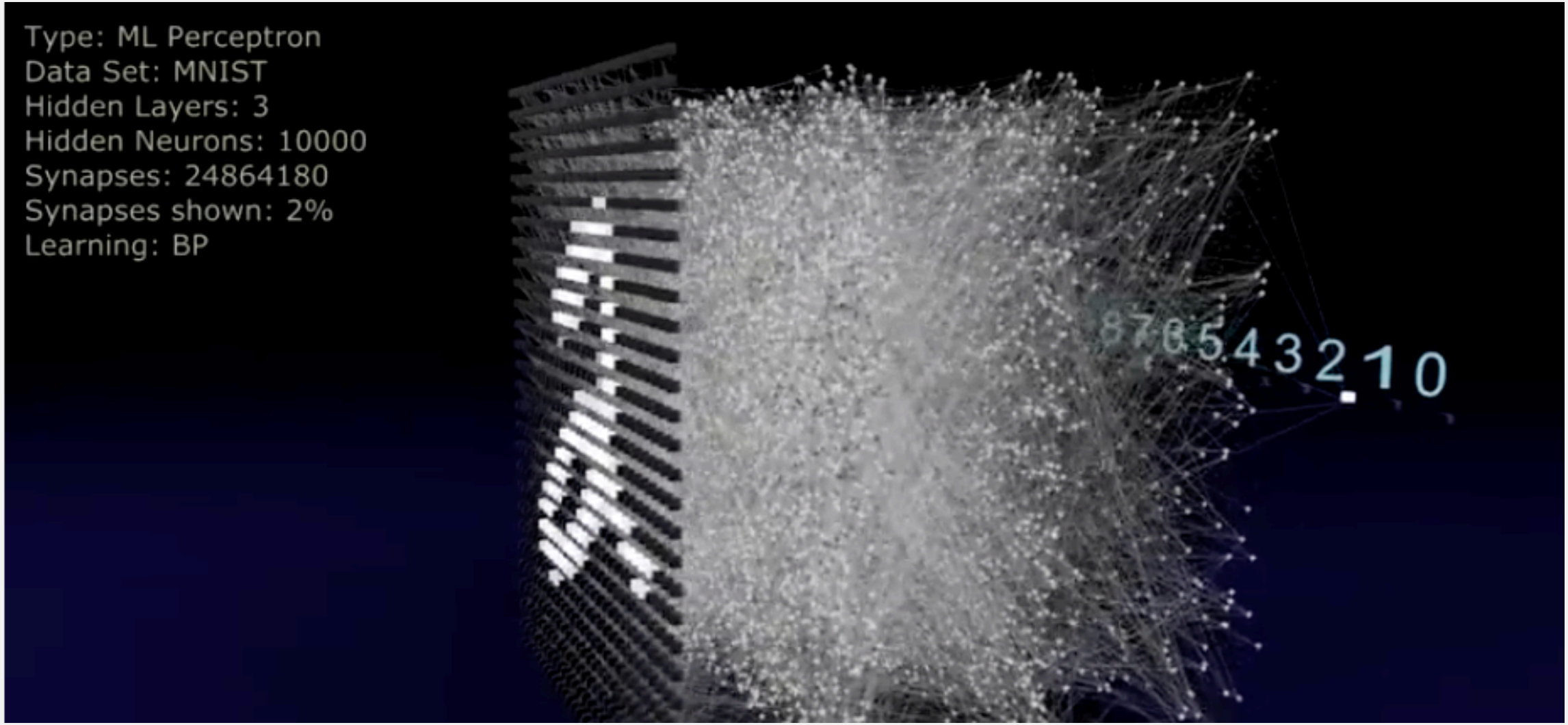
TRAINING

- Filter in the convolutional layers can be manually selected
- But the power of the convolutional network is to also learn the filters
- **Filters are learned using back propagation procedure**

- Features will be learned in order to be effective in capturing pattern relevant for the task and descriptive for the input data

Fashion MNIST classification

EXAMPLE



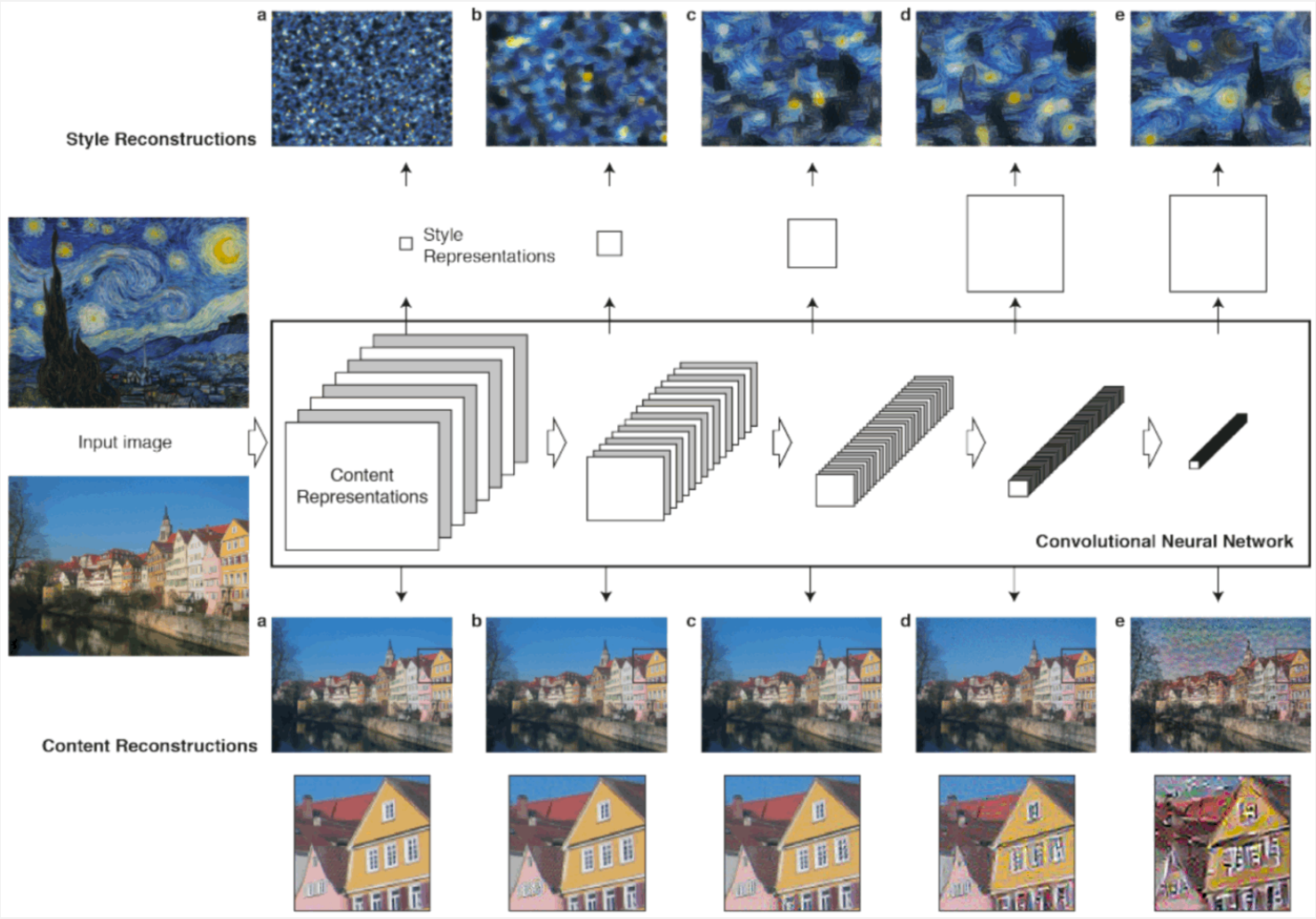
https://www.youtube.com/watch?v=3JQ3hYko51Y

NEURAL STYLE TRANSFER

NEURAL STYLE TRANSFER

- Deep Neural Network that creates artistic images of high perceptual quality
- Separates and recombines the content and style of an image
- rRepresentations of content and style in the Convolutional Neural Networks (CNNs) are separable.

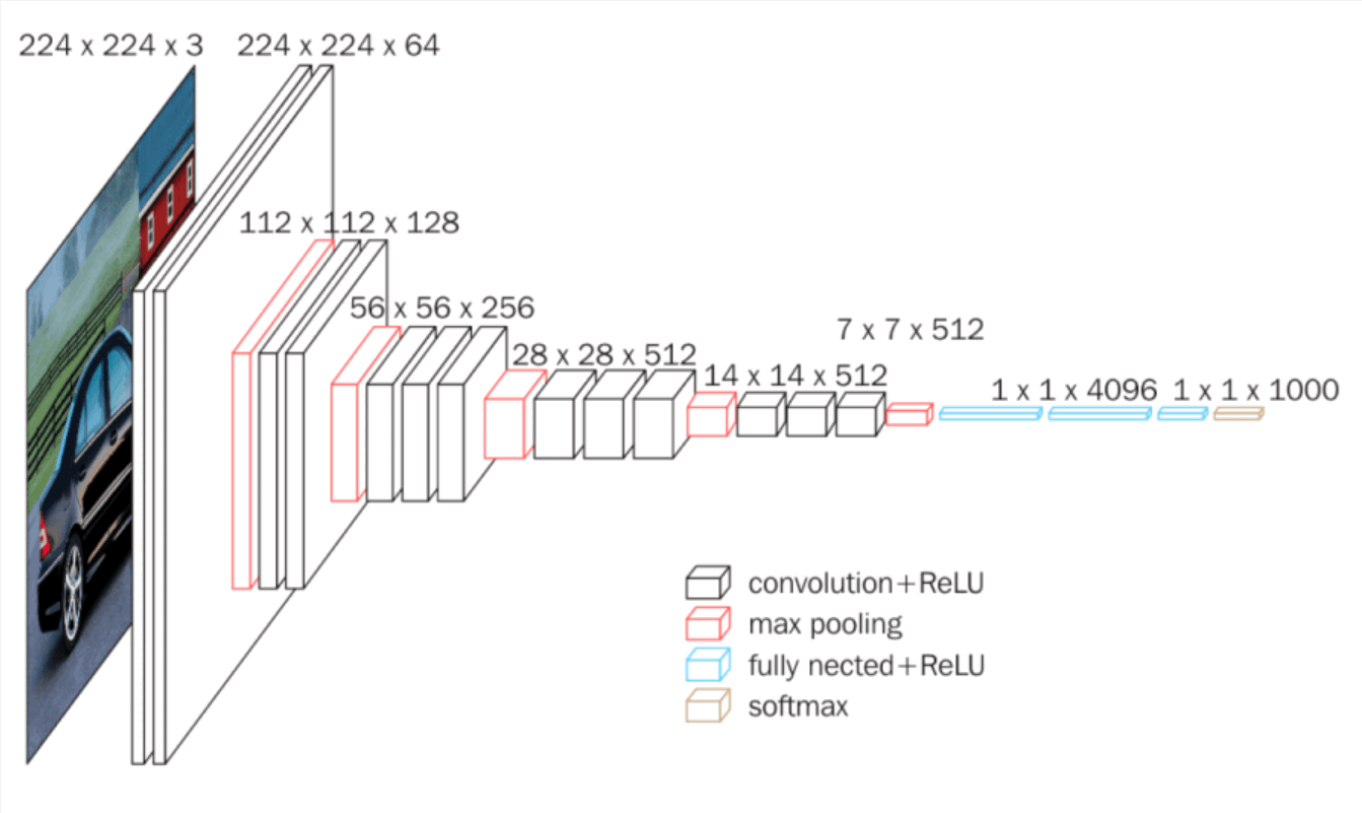
NEURAL STYLE TRANSFER



NEURAL STYLE TRANSFER

- images are synthesised by finding an image that simultaneously matches the content representation of the photograph and the style representation of the respective piece of art

- Features are extracted using the pre-trained classification model **VGG**
- Achieved 92.7 % accuracy in 2014 on the ImageNet dataset (1000 classes, 14 million images)



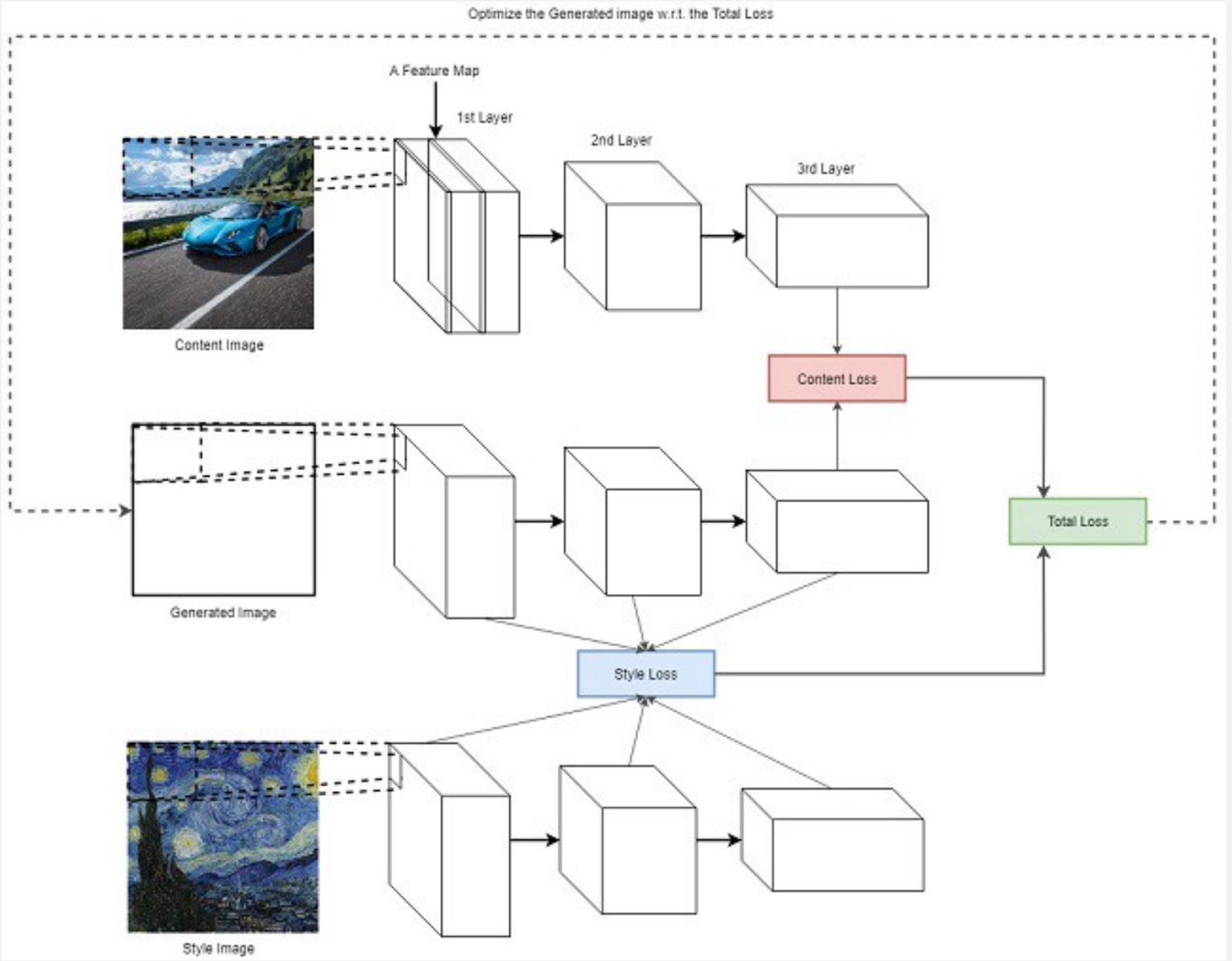
NEURAL STYLE TRANSFER

- Each layer in the network defines a non-linear filter bank whose complexity increases with the position of the layer in the network
- A layer with N_l filters has N_l feature maps, each of size M_l = height*width
- Response in layer l can be stored in a matrix

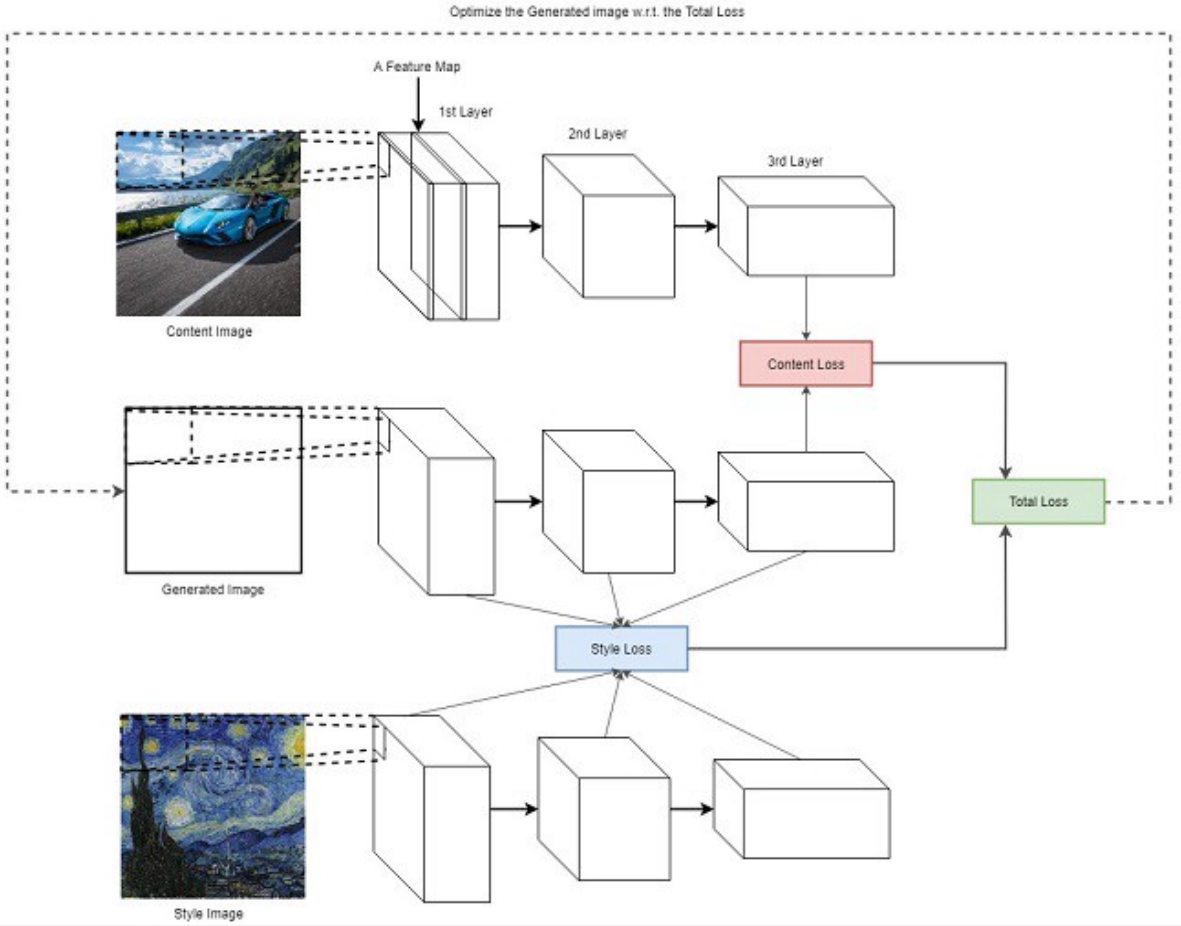
where F_{ij}^{l} contains the activation of the i-th filter at position j in layer l

$$F^{l} \in \mathcal{R}^{N_l \times M_l}$$

NEURAL STYLE TRANSFER



NEURAL STYLE TRANSFER



- No training process
- The generated image starts from a white noise image

NEURAL STYLE TRANSFER

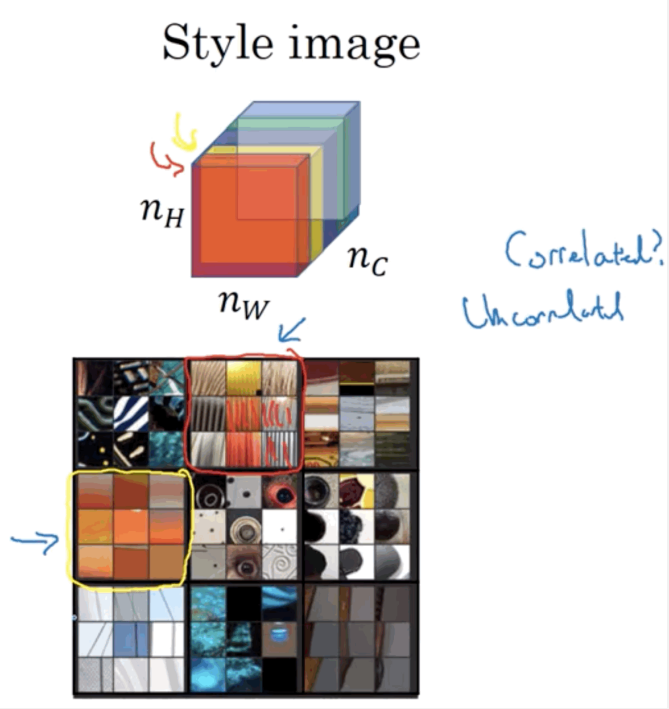
- ***Content loss***:
  - We use gradient descent to recover the content of the image
    - p -> original image P ->feature-maps
    - x -> generated image F ->feature-maps
  - We define the loss as the squared error between the two representations

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} (F_{ij}^{l} - P_{ij}^{l})$$

NEURAL STYLE TRANSFER

- ***Style loss***:
  - Compute the correlation between the different filter responses, where the expectation is taken over the spatial extent of the input image
  - We use the Gram matrix $$G^l = \sum_k F^l_{ik} F^l_{jk}$$
  - Through gradient descent we minimize the mean-squared distances between the Gram matrix of the original image A and of the generated one G

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \frac{1}{4N_l^2 M_l^2} \sum_{l=0}^{L} \sum_{i,j} (G^l_{ij} - A^l_{ij})$$



NEURAL STYLE TRANSFER

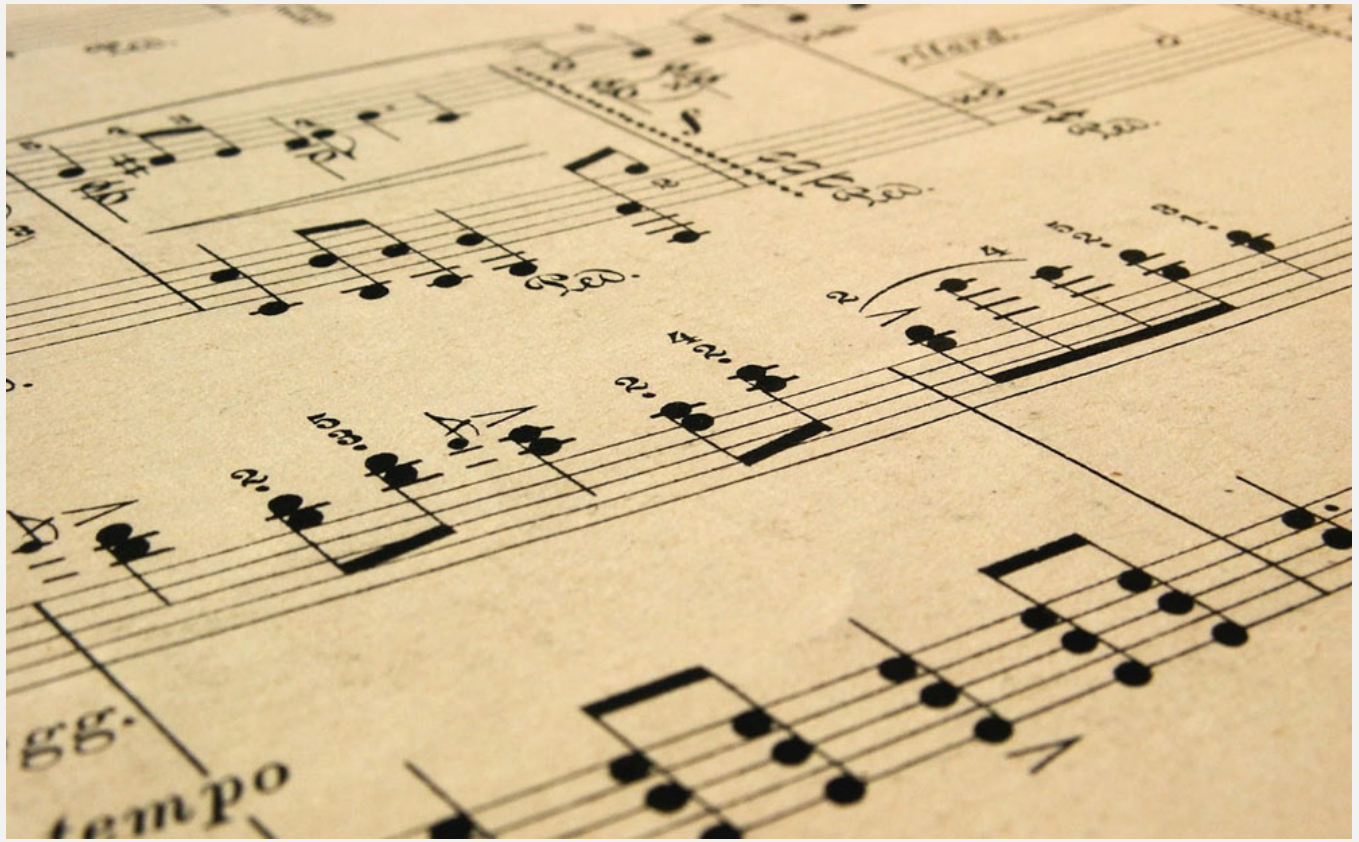
- ***Full loss***:
  - Compute the correlation between the different filter responses, where the expectation is taken over the spatial extent of the input image

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha\mathcal{L}(\vec{p}, \vec{x}) + \beta\mathcal{L}_{style}(\vec{a}, \vec{x})$$

RNN

TIME-AWARE NEURAL NETWORKS

- In the course we have seen two families of problem to model:
  - The output is dependent on the input and on the status
  - The output is dependent on the evolution of the events to models (sequence)

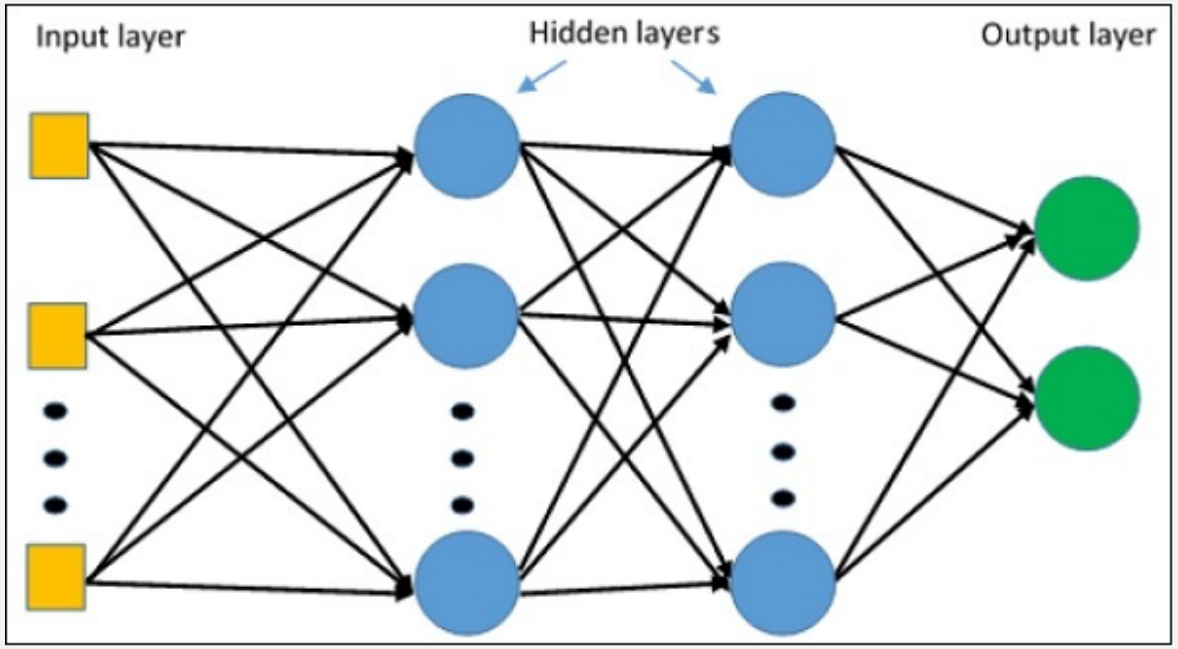


- In the second model there is the need of memory able to keep track of the past history

- Perceptron, MLP and CNN are not able to keep track of the past sequence of events

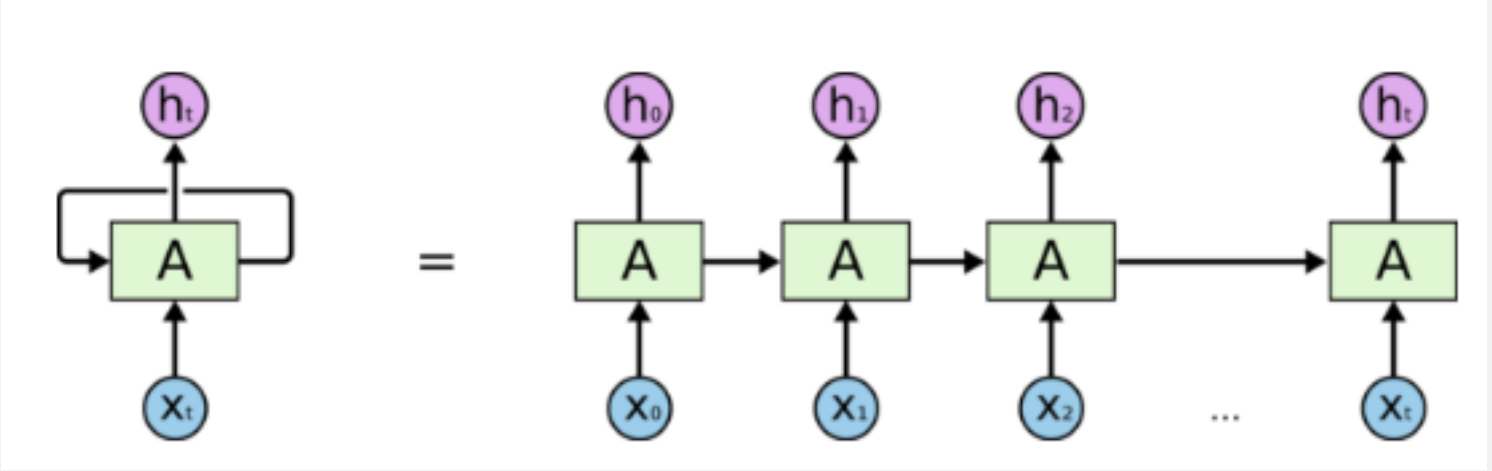
TIME-AWARE NEURAL NETWORKS

- It is possible to model a sequence providing the sequence data to the net
- Example: to model sequences of notes (melodies)
  - Each neuron is the code of a note (MIDI)
  - 10 input neurons allows to model sequences of 10 notes
- Cons:
  - It learns patterns, not transition functions (not effective in grammar-based problems)



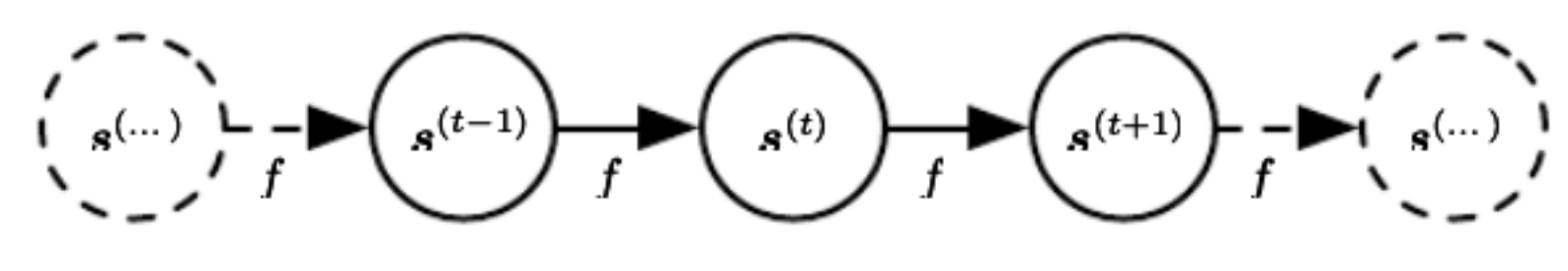
RECURRENT NEURAL NETWORKS (RNN)

- Recurrent neural networks, or RNNs (Rumelhart et al., 1986), are a family of neural networks for processing sequential data
- Most recurrent networks can also process sequences of variable length
- Recurrent Neural Network is a generalization of feedforward neural network that has an internal memory
- Given a sequence to model, RNN performs the same function for every input of data while the output of the current input depends on the past one computation
- After producing the output, it is copied and sent back into the recurrent network
- For making a decision, it considers the current input and the output that it has learned from the previous input.



COMPUTATIONAL GRAPH

- A computational graph is a way to formalize the structure of a set of computations



$$s^{(t)} = f(s^{(t-1)}; \theta),$$

state of the system

parameter of f

- The equation is recurrent because the definition of s at time t refers back to the same definition at time t − 1
- The same f and the same set of parameters is applied to each step

COMPUTATIONAL GRAPH

- For a finite number of time steps τ, the graph can be **unfolded** by applying the definition τ − 1 times

$$\begin{aligned} \boldsymbol{s}^{(3)} &= f(\boldsymbol{s}^{(2)}; \boldsymbol{\theta}) \\ &= f(f(\boldsymbol{s}^{(1)}; \boldsymbol{\theta}); \boldsymbol{\theta}). \end{aligned}$$

- Let's add an input

$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}),$$

- The input $x^{(t)}$ is a sequence of data to model
- s is the set of states dependent on the input and the previous state
- The unfolded version for a sequence $x^{(1)}$ and $x^{(2)}$

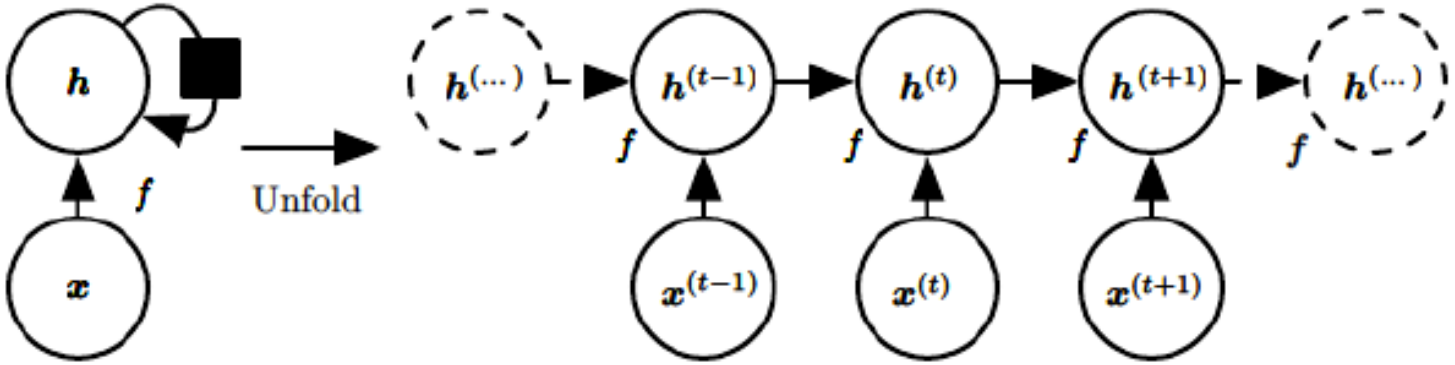
$$s^{(3)} = f(s^{(2)}, f(s^{(1)}, x^{(1)}; \Phi), x^{(2)}; \Phi)$$

RECURRENT NEURAL NETWORKS

- For a finite number

$$s^{(3)} = f(s^{(2)}, f(s^{(1)}, x^{(1)}; \Phi), x^{(2)}; \Phi)$$

- Let's see the states as a hidden neurons of a network

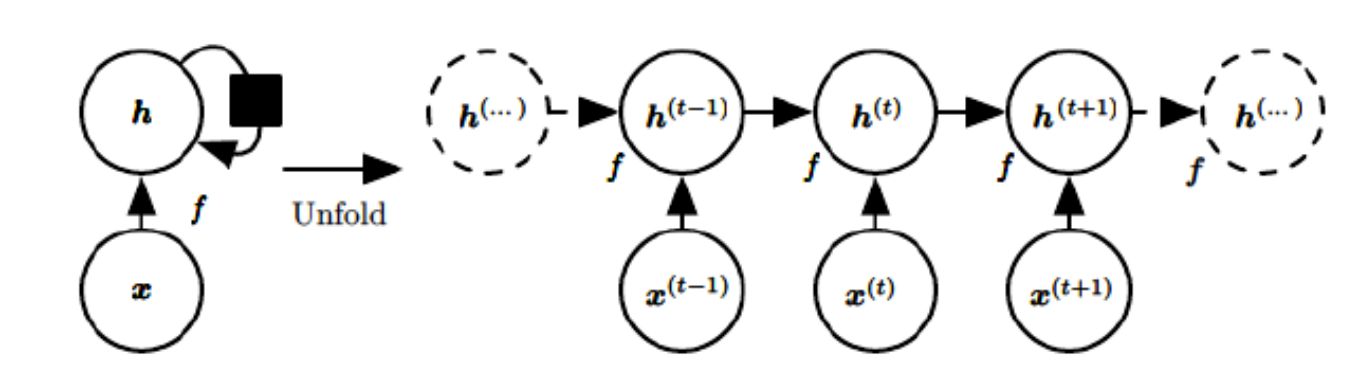


$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}),$$

- (Left) Circuit diagram. The black square indicates a delay of a single timestep.
- (Right) The same network seen as an unfolded computational graph, where each node is now associated with one particular time instance

RECURRENT NEURAL NETWORKS

- The RNN is trained to perform a task that requires predicting the future from the past,
- The RNN typically learns to use $h^{(t)}$ as a kind of lossy summary of the task-relevant aspects of the past sequence of inputs up to t



$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta),$$

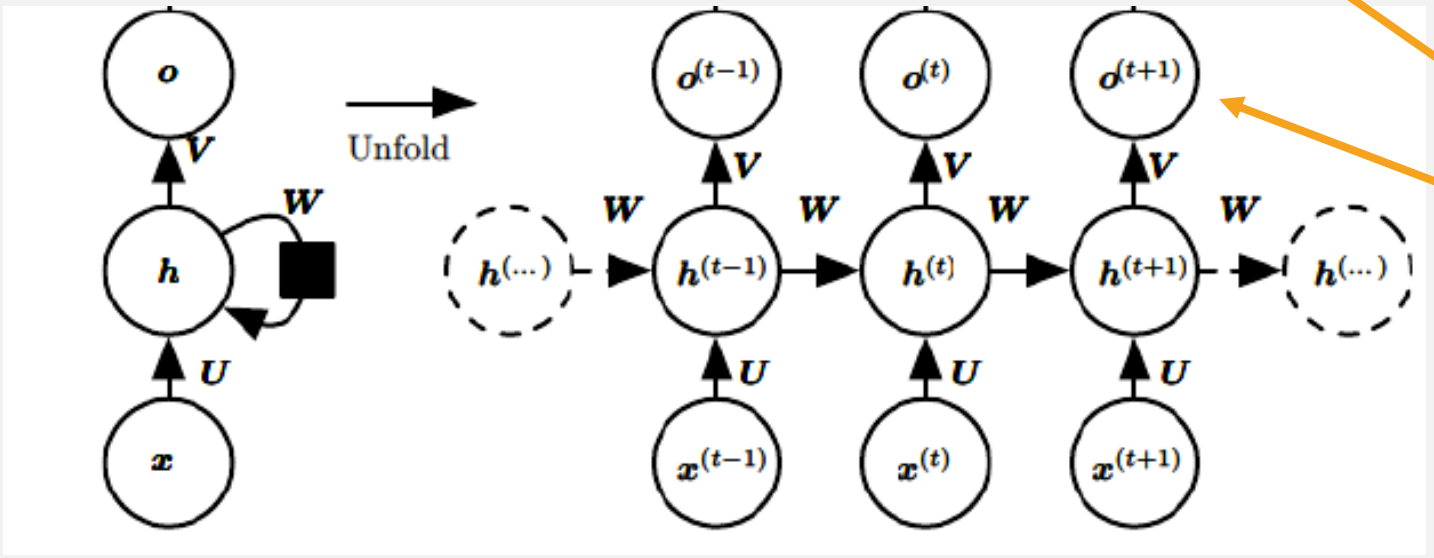
- The unfolded graph has a size that depends on the sequence length to model
- The train will model a sequence of input data of length t
- The train will learn a unique transition function f that will be applied to all the steps
- Learning a single shared model allows generalization to sequence lengths that did not appear in the training set

RECURRENT NEURAL NETWORKS

Transform into the forward propagation equation

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}),$$

$$\begin{aligned} \boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\ \boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\ \boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \end{aligned}$$

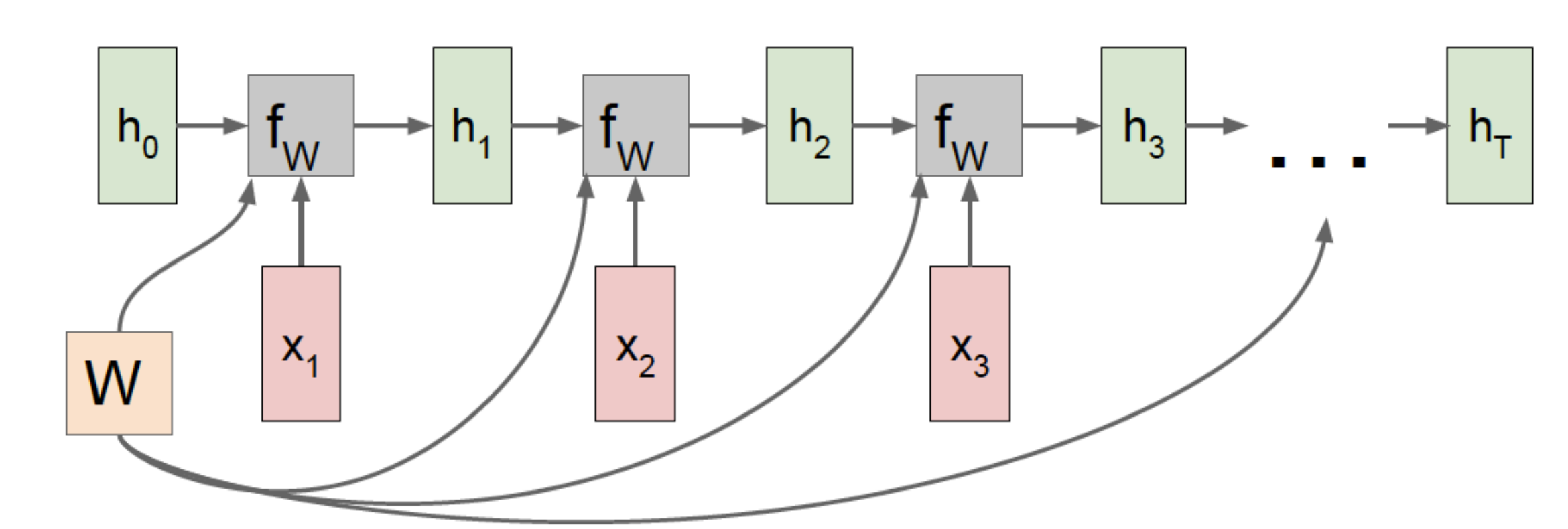


- W, U, V are weight matrixes
- b and c are bias values

Add the output layer

RECURRENT NEURAL NETWORKS

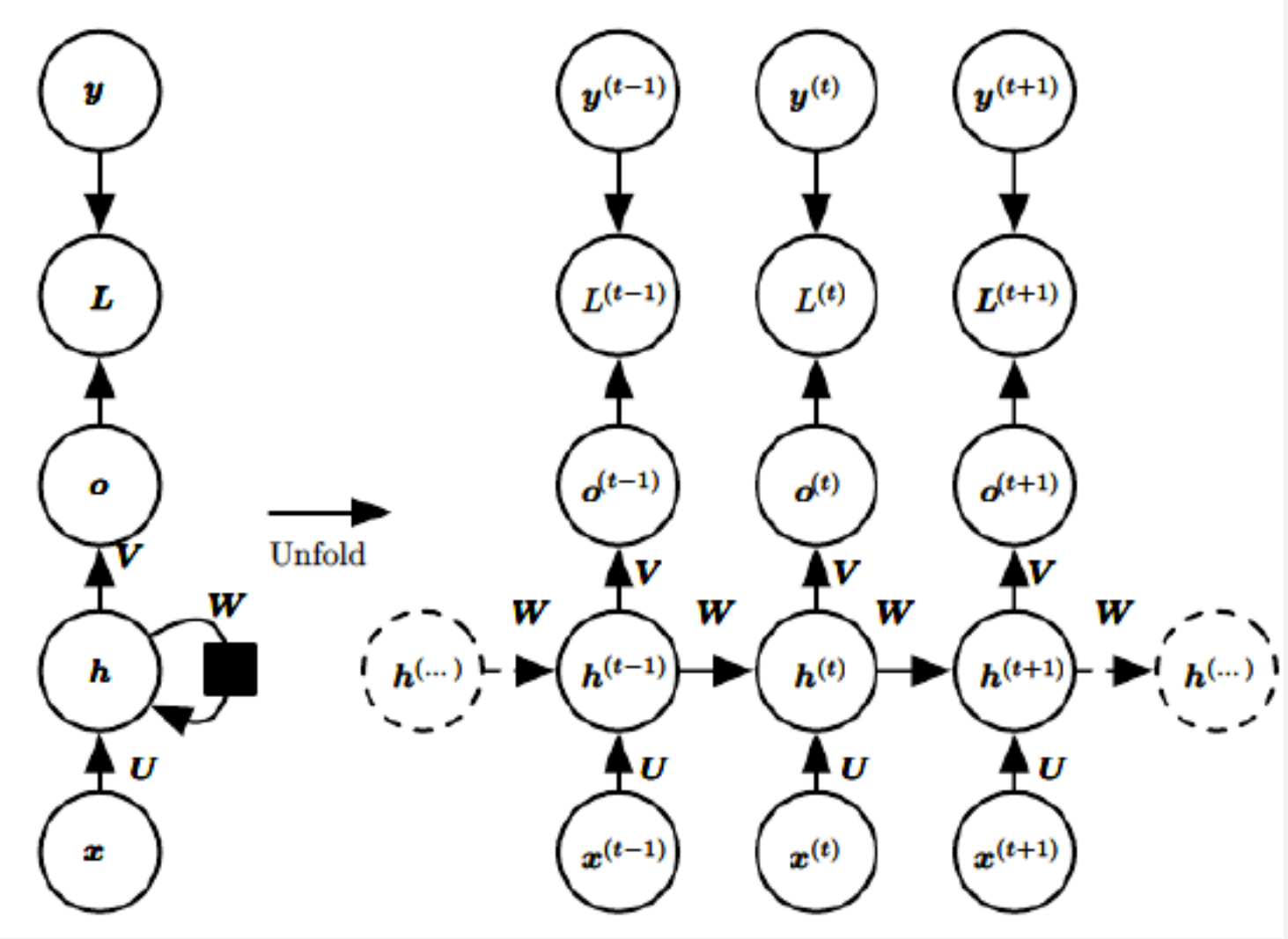
- It is possible to use the same transition function for all the states
- W and U are same for all the states



RNN - TRAINING

- In the training, the network learns the weights W, U, V
- Given a desired output sequence

  $y^{(1)} \dots y^{(t)}$
- Compute a loss function $L^{(t)}$ between $o^{(t)}$ and $y^{(t)}$
- The total loss of the system is the sum of the single loss values



RNN - TRAINING

- Often the output $o^{(t)}$ is normalized using **softmax function**

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}(\boldsymbol{o}^{(t)})$$

- Softmax takes as input a vector of K real numbers, and normalizes it into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers.
- After applying softmax, each component will be in the interval (0,1), and the components will add up to 1, so that they can be interpreted as probabilities

$$L^{(t)}(y, \hat{y}) = -y_{(t)} log \hat{y}_t$$

$$L(y, \hat{y}) = \sum_t L^{(t)}(y, \hat{y}) = -\sum_t y_{(t)} log \hat{y}_t$$

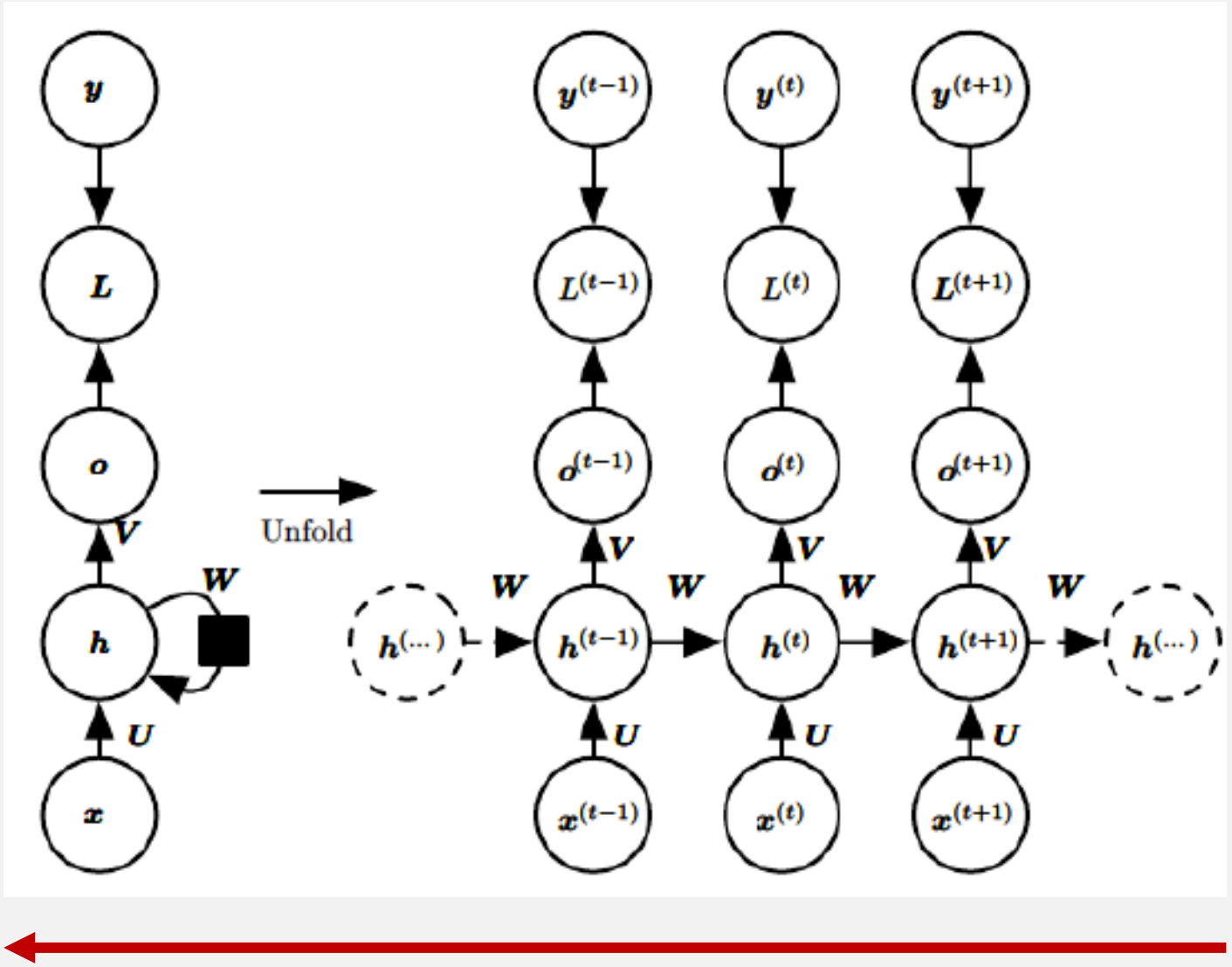
RNN – TRAINING BACKPROPAGATION

- Back propagation in Layer and in Time

$$\frac{\delta L}{\delta U}\ \frac{\delta L}{\delta V}\ \frac{\delta L}{\delta W}\ \frac{\delta L}{\delta b}\ \frac{\delta L}{\delta c}$$

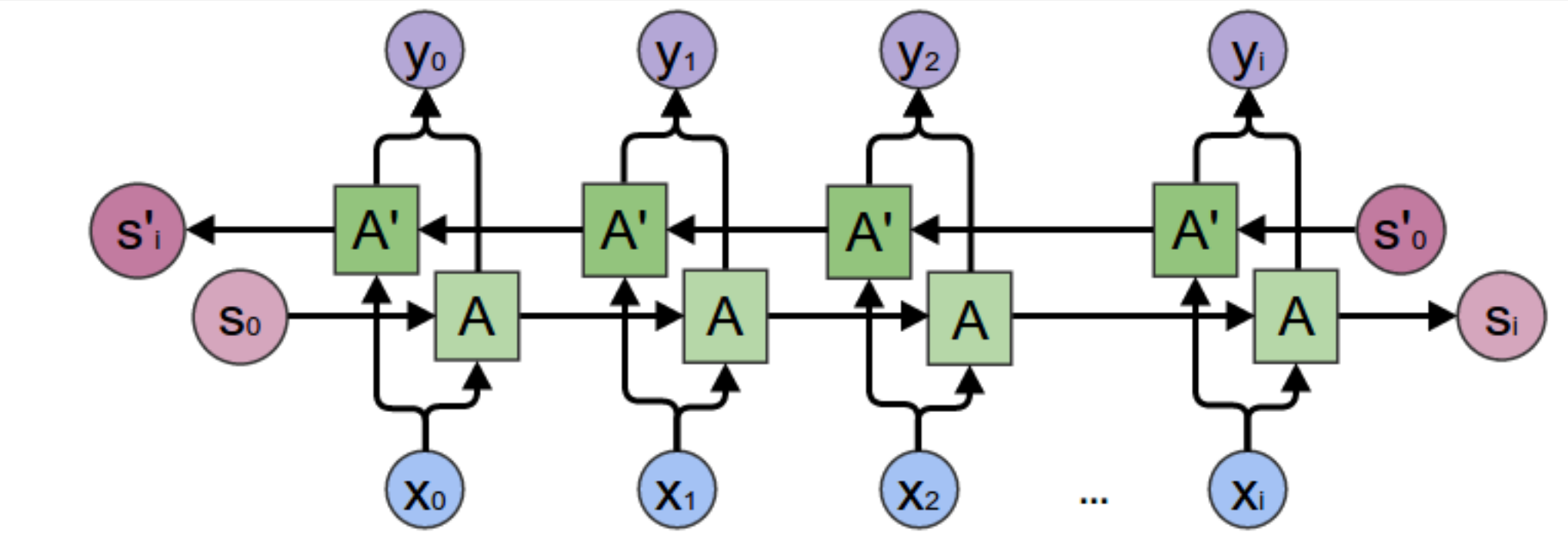
$$\frac{\delta L}{\delta W} = \sum_t \frac{\delta L^{(t)}}{\delta W}$$

- W is dependent by all $h^{(t)}$ in the sequence
- Back propagation up to the beginning of the sequence



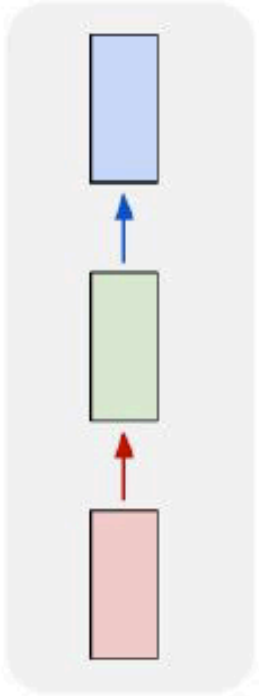
BIDIRECTIONAL RNN

- In sequence modelling it often useful to model both the past and future -> **bidirectional RNN**
- Bidirectional Recurrent Neural Networks (BRNN) connect two hidden layers of opposite directions to the same output



RNN – TYPES

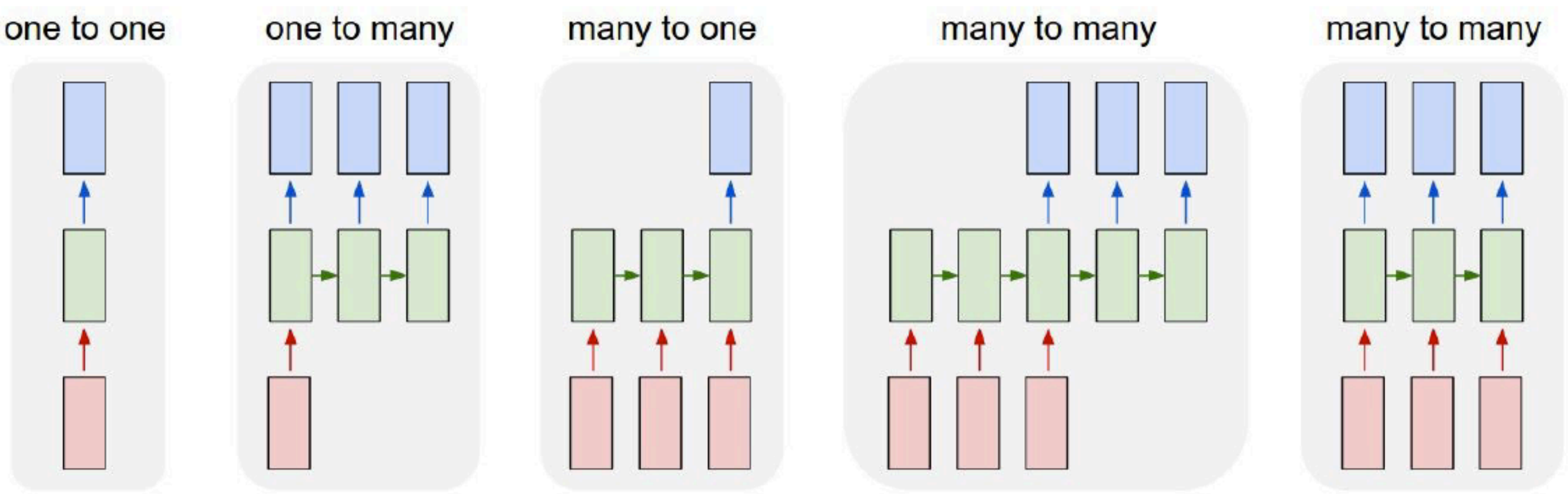
one to one



- MLP and CNN are Vanilla Nets

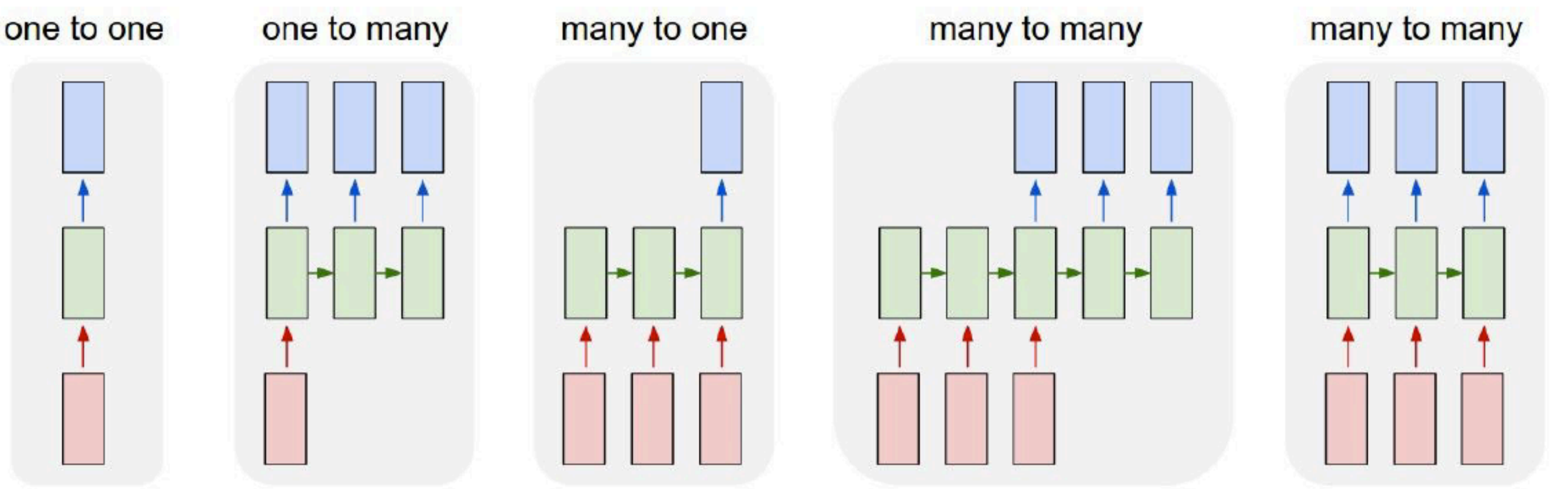
Vanilla Neural Networks

RNN – TYPES



e.g. **Image Captioning**
image -> sequence of words

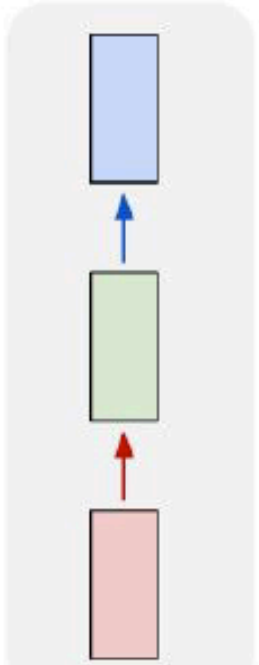
RNN – TYPES



e.g. **Sentiment Classification**
sequence of words -> sentiment

RNN – TYPES

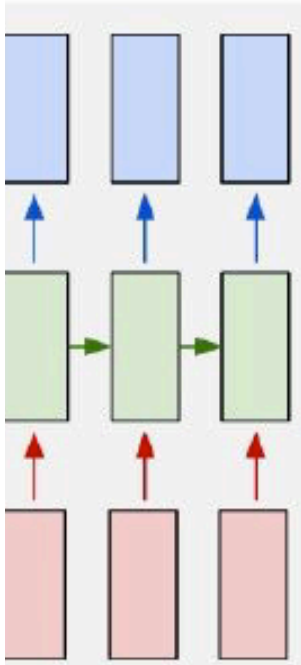
one to one



SENTIMENT ANALYSIS

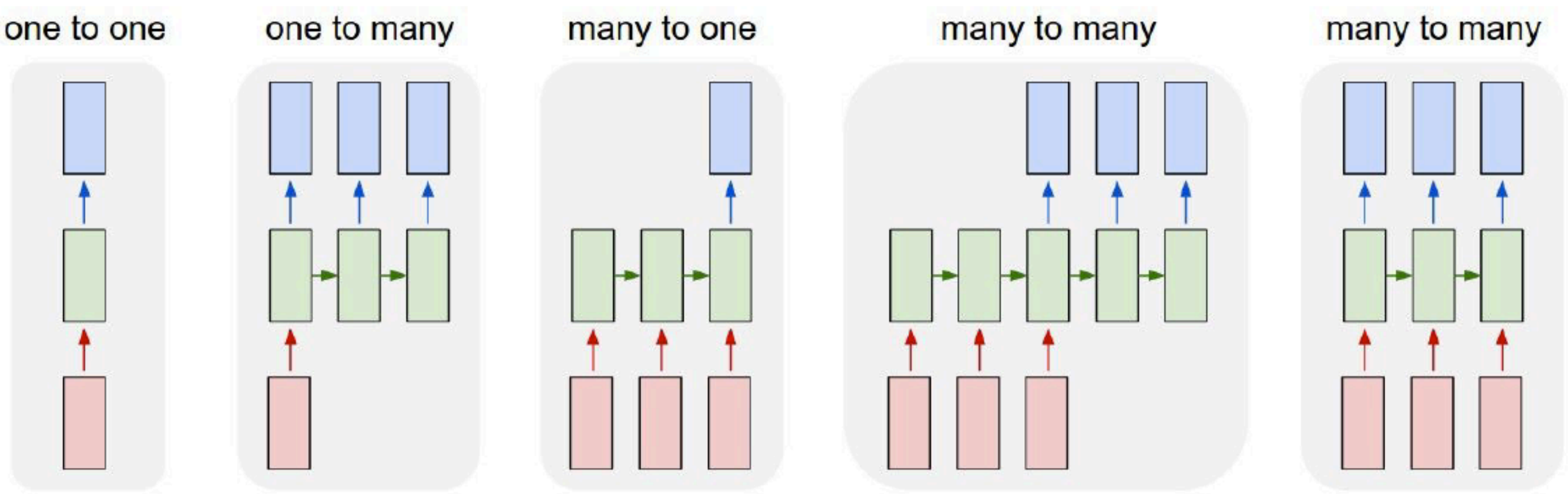
Discovering people opinions, emotions and feelings about a product or service

many to many



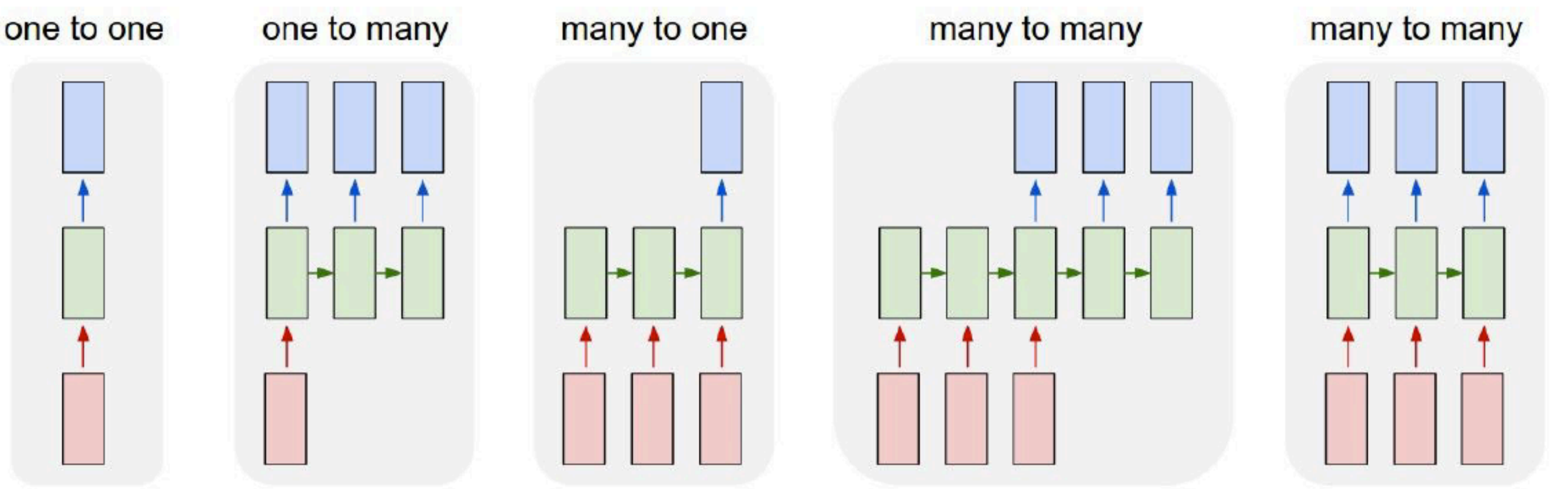
e.g. **Sentiment Classification**
sequence of words -> sentiment

RNN – TYPES



e.g. **Machine Translation**
seq of words -> seq of words

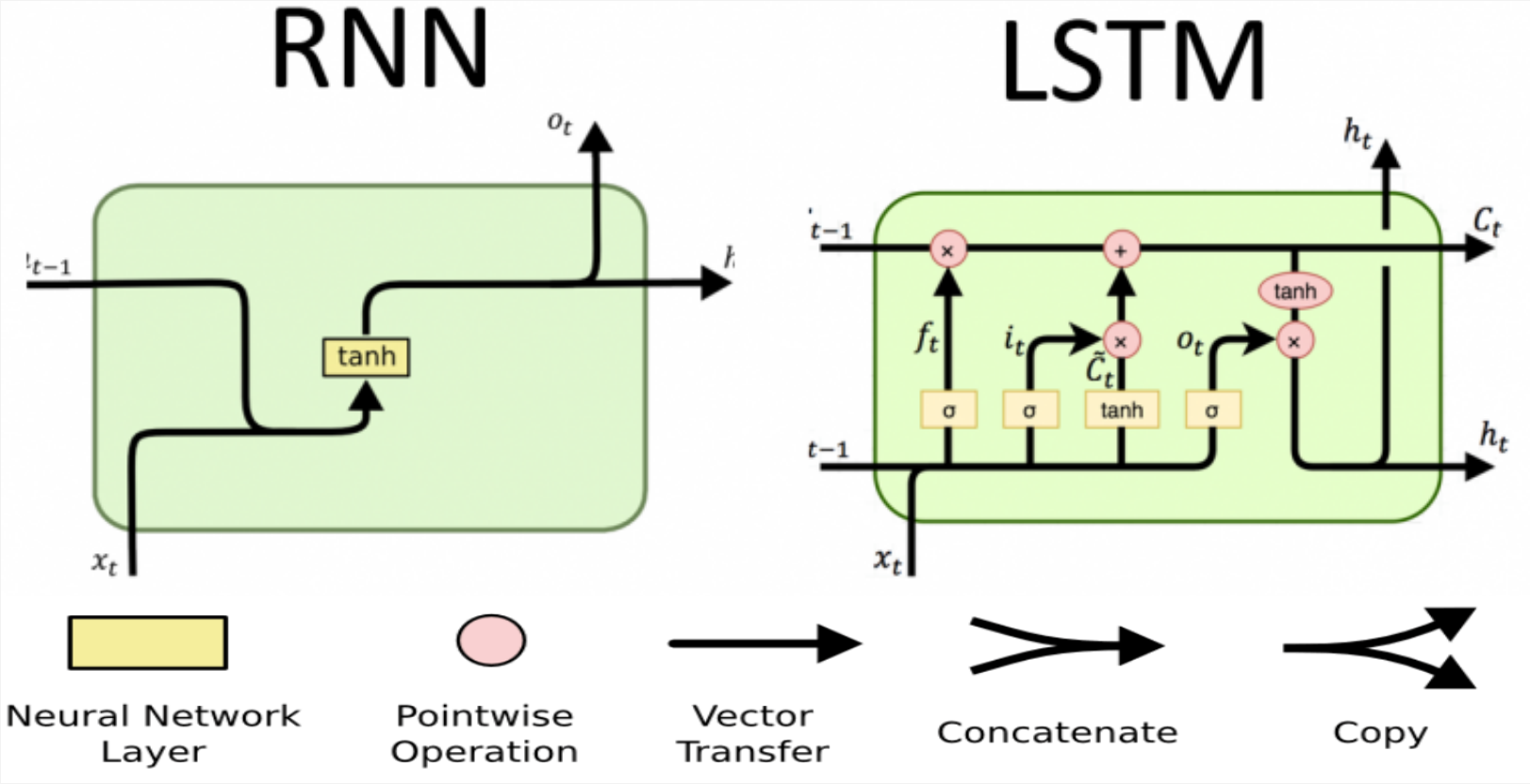
RNN – TYPES



e.g. **Video classification on frame level**

PERFORMANCE-RNN

- **Architecture**
  - **LSTM**-based RNN model
  - 3-layer with 512 cells each



- **LSTM**
  - Special kind of RNN able to learn long-term dependencies
  - Does have the ability to remove or add information to the cell state, carefully regulated by structures called gates
  - Gates are a way to optionally let information through.

RNN EXAMPLE: MUSIC PERFORMANCE

PERFORMANCE-RNN

- **Generating Music with Expressive Timing and Dynamics**
  - Learn to create piano performances

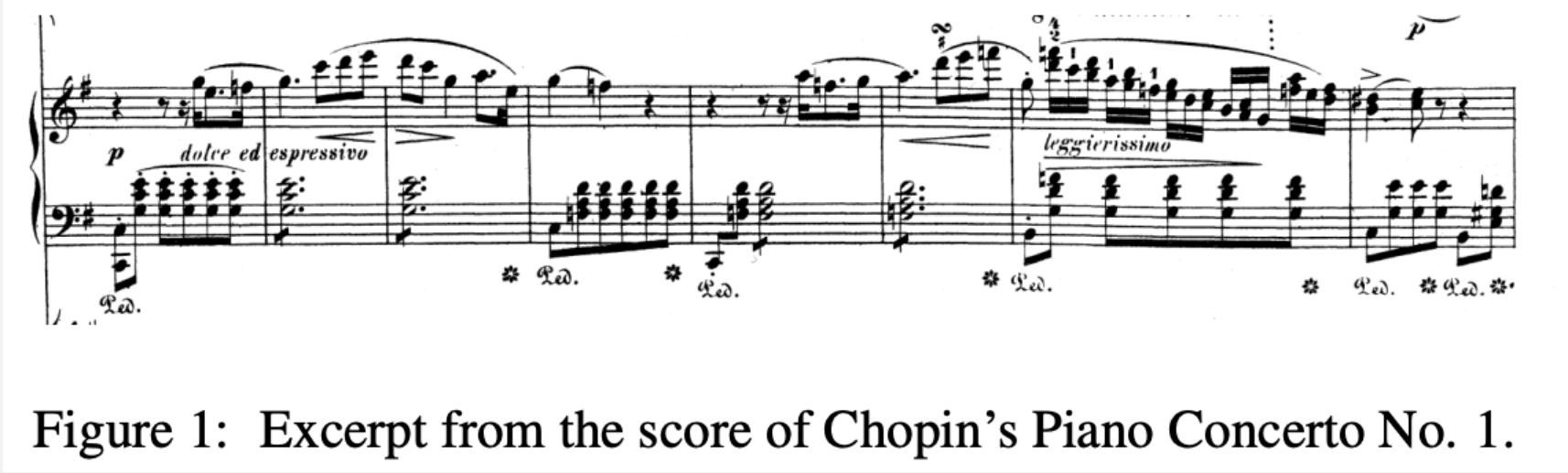


Figure 1: Excerpt from the score of Chopin's Piano Concerto No. 1.

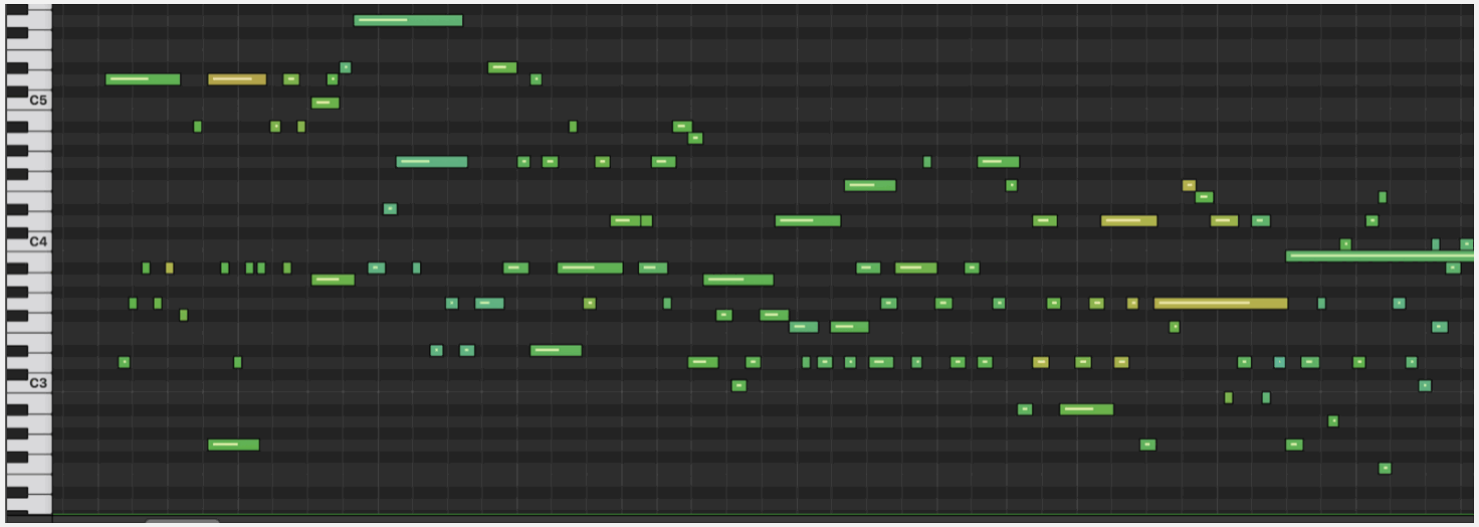
- Direct rendering

- Expressive Performance

PERFORMANCE-RNN

- Performance RNN generates expressive timing and dynamics via a stream of MIDI events.

- MIDI data as a sequence of events drawn from a vocabulary (encoded as **one-hot vector**):
  - following types of events
    - *Note-ON*
    - *Note-OFF*
    - *Set-Velocity*
    - *Time-Shift*

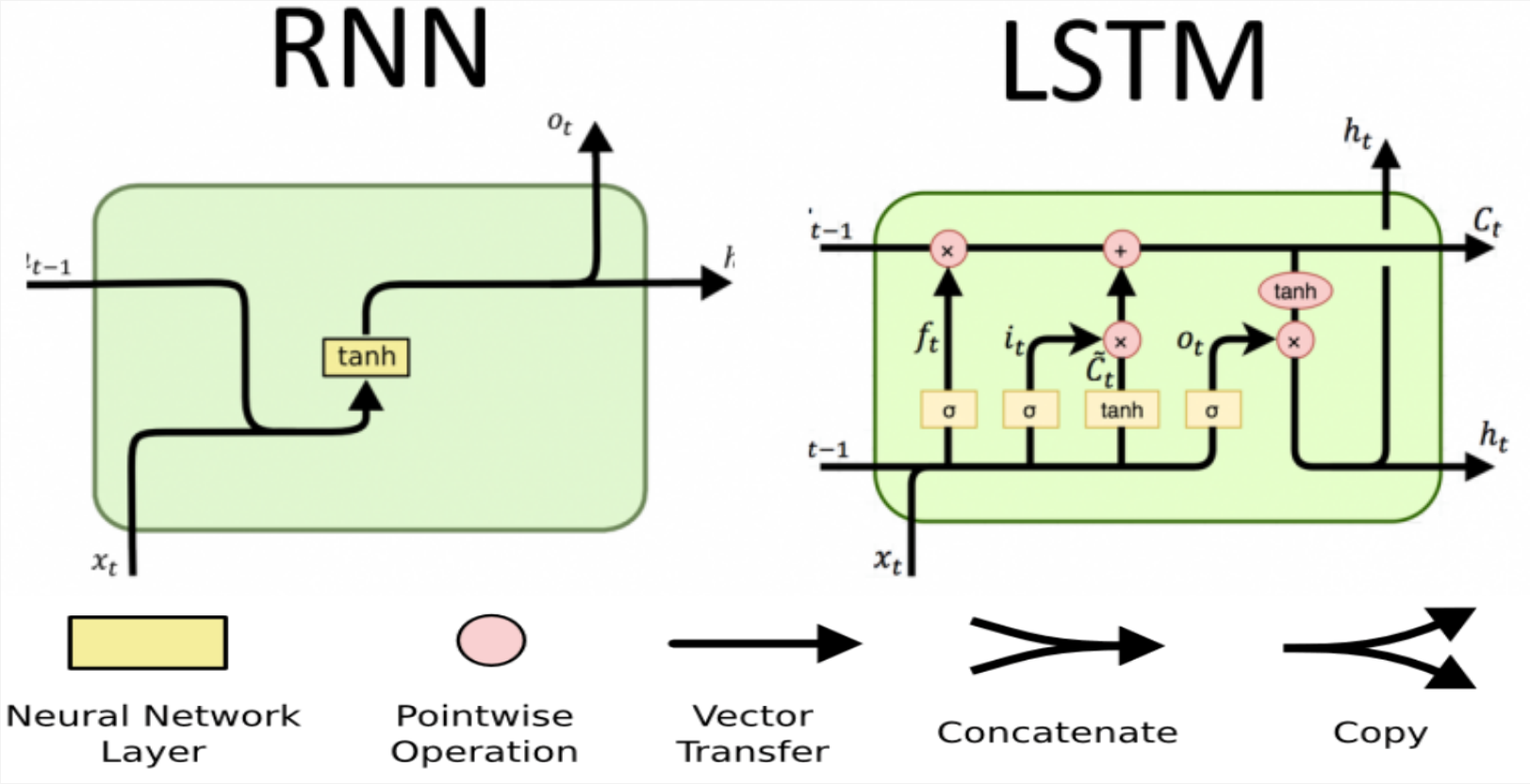


- Example Sequence

```
Set-Velocity (80), Note-On (C2), Note-On (C3), Time-Shift 304ms, Note-Off
(C2), Note-Off (C3), Time-Shift 200ms, ...
```

PERFORMANCE-RNN

- **Architecture**
  - **LSTM**-based RNN model
  - 3-layer with 512 cells each



- **LSTM**
  - Special kind of RNN able to learn long-term dependencies
  - Does have the ability to remove or add information to the cell state, carefully regulated by structures called gates
  - Gates are a way to optionally let information through.

PERFORMANCE-RNN

- **Dataset:**
  - Yamaha e-competition Dataset
  - MIDI captures of ~1400 performances by skilled pianists.

- **Temperature**
  - Can we control the output of the model?
  - Parameter that allows to change randomness of samples
  - Hyperparameter of LSTMs
  - Basically:
    - Lower temperatures -> decrease randomness
    - Higher temperatures -> increase randomness

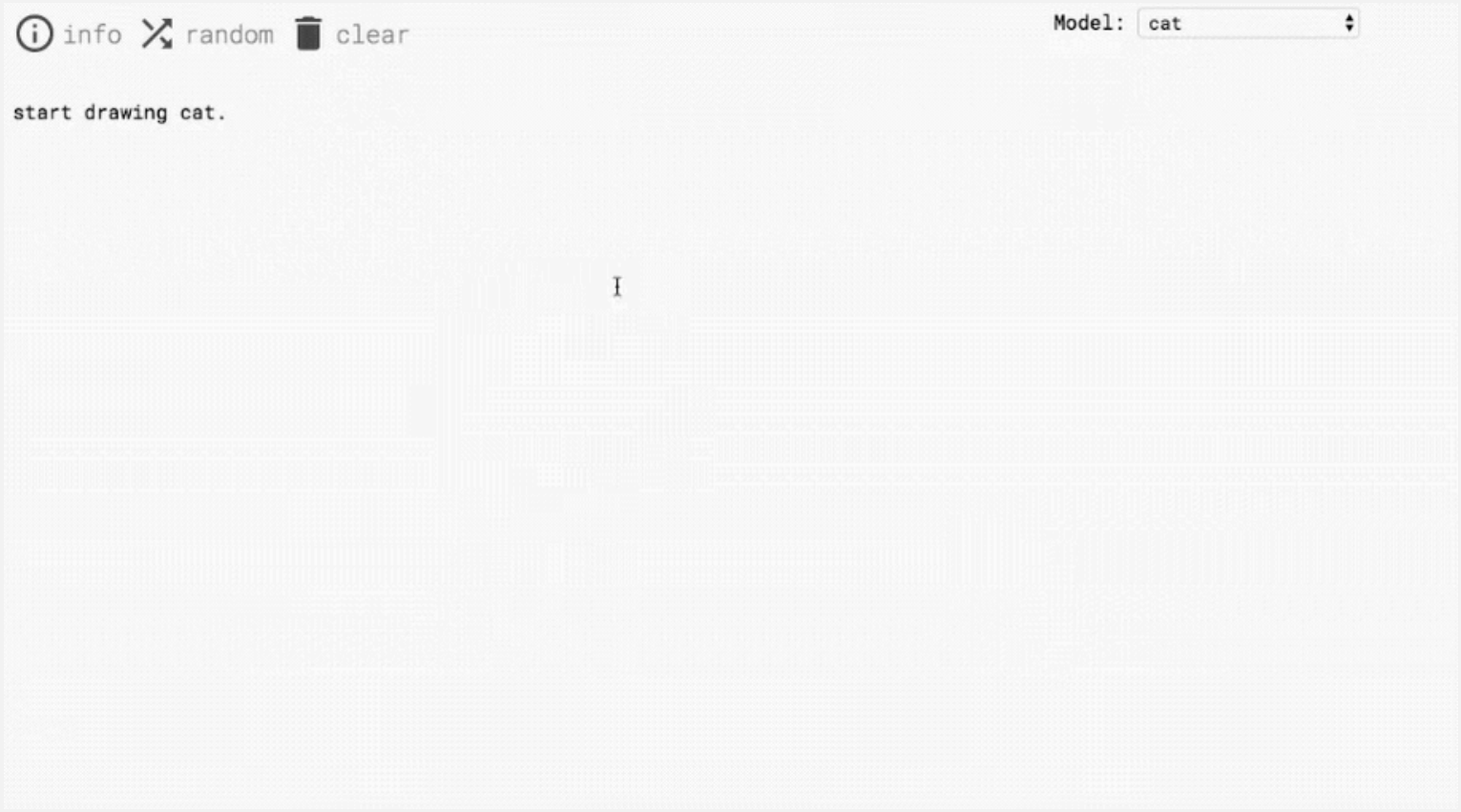
- T = 0.8

- T = 2

RNN EXAMPLE

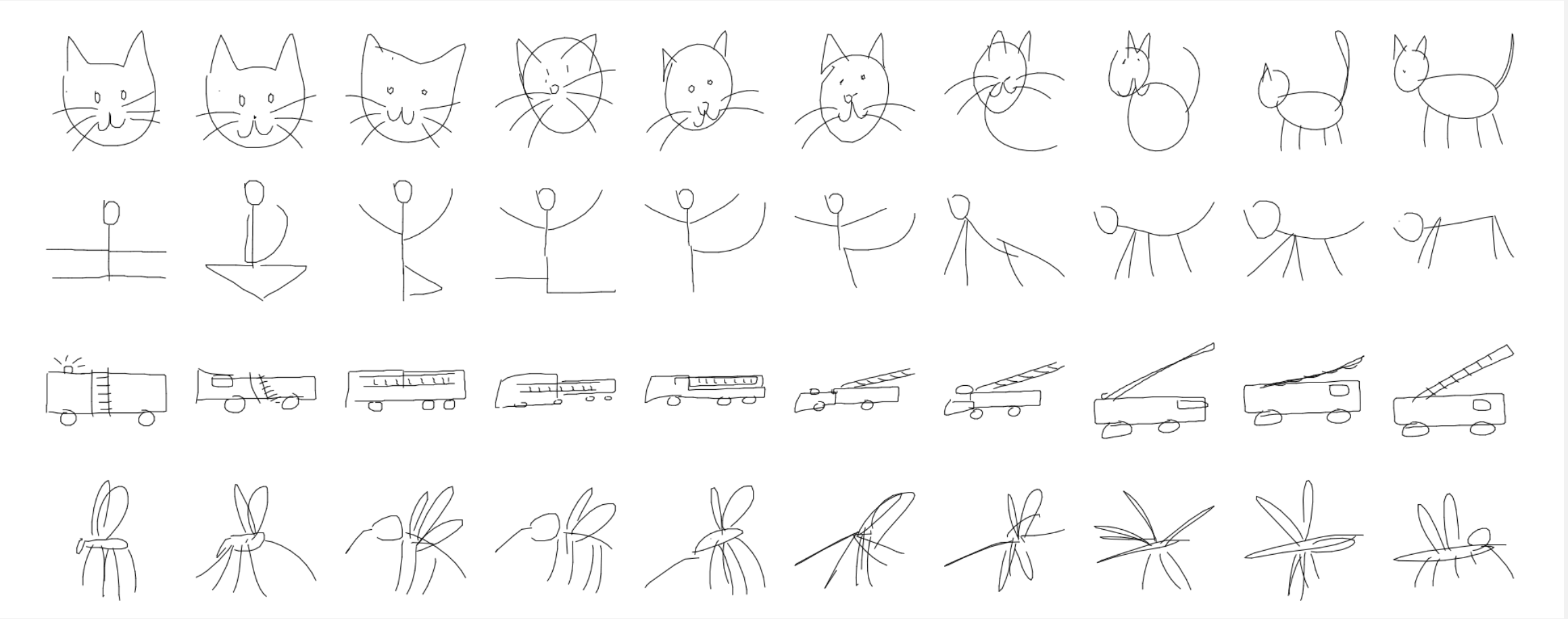
SKETCH-RNN

- **Teaching Machines to Draw**
  - Recurrent Neural Network (RNN) able to construct stroke-based drawings of common objects



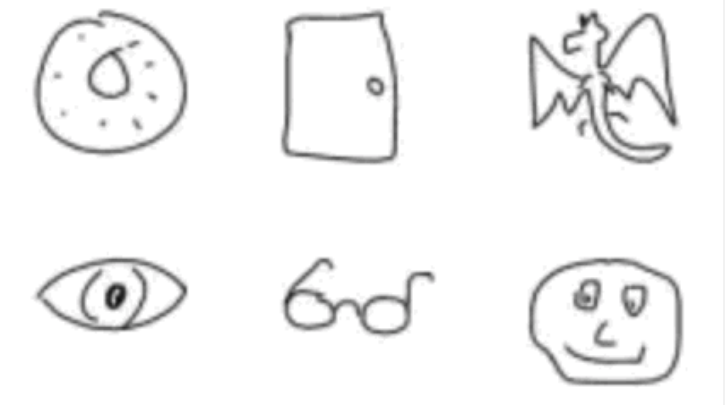
SKETCH-RNN

- **Objective**: train machines to draw and generalize abstract concepts in a manner similar to humans



SKETCH-RNN

- model trained on **QuickDraw** dataset of hand-drawn sketches, each represented as a sequence of motor actions controlling a pen
- a **Sketch** is a list of points



- Each point is a 5 element vector $[\Delta x, \Delta y, p_1, p_2, p_3]$
  - Δx $->$ offset in the x direction
  - Δy $->$ offset in the y direction
  - p_1 $->$ indicates that the pen is currently touching the paper, and that a line will be drawn connecting the next point with the current point
  - p_2 $->$ indicates that the pen will be lifted from the paper after the current point, and that no line will be drawn next.
  - p_3 $->$ indicates that the drawing has ended, and subsequent points, including the current point, will not be rendered

SKETCH-RNN

- **Architecture**:
  - **Sequence-to-Sequence Variational Autoencoder** (VAE)

  - **Encoder**: Bidirectional RNN
    - **Input**: sketch sequence S and reversed sketch sequence $S_{reverse}$
    - **Output**: hidden state $h = [\, h_{\rightarrow} ; h_{\leftarrow} \,]$, $h_{\rightarrow} = \text{encode}_{\rightarrow}(S)$, $h_{\leftarrow} = \text{encode}_{\leftarrow}(\text{Sreverse})$,
    - Fully Connected Layer: from h, we get two vectors μ and σ, each of size N_z
    - We then construct the following random vector:

$$\mu = W_\mu h + b_m u,\ \hat{\sigma} = W_\sigma h + b_\sigma,\ \sigma = \exp\frac{\hat{\sigma}}{2},\ z = \mu + \sigma \odot \mathcal{N}(0, I)$$

  - *N.B. latent vector z is not a deterministic output for a given input sketch, but a random vector conditioned on the input sketch*

SKETCH-RNN

- **Architecture**:
  - Sequence-to-Sequence Variational Autoencoder (VAE)

  - **Decoder**: Autoregressive RNN
    - **Input**: initial hidden state $[\, h0\, ;\, c0\,] \,=\, \tanh(W_{zz} \,+\, b_z)$

    - **Output**: probability distribution of the next data point:

$$\left[\left(\hat{\prod}_1\ \mu_x\ \mu_y\ \hat{\sigma_x}\ \hat{\sigma_y}\ \hat{\rho_{xy}}\right)_1 \cdots \left(\hat{\prod}_1\ \mu_x\ \mu_y\ \hat{\sigma_x}\ \hat{\sigma_y}\ \hat{\rho_{xy}}\right)_M (\hat{q}_1\ \hat{q}_2\ \hat{q}_3\ )\right] = y_i$$

    - (q1, q2, q3) as a categorical distribution to model the ground truth data (p1, p2, p3)

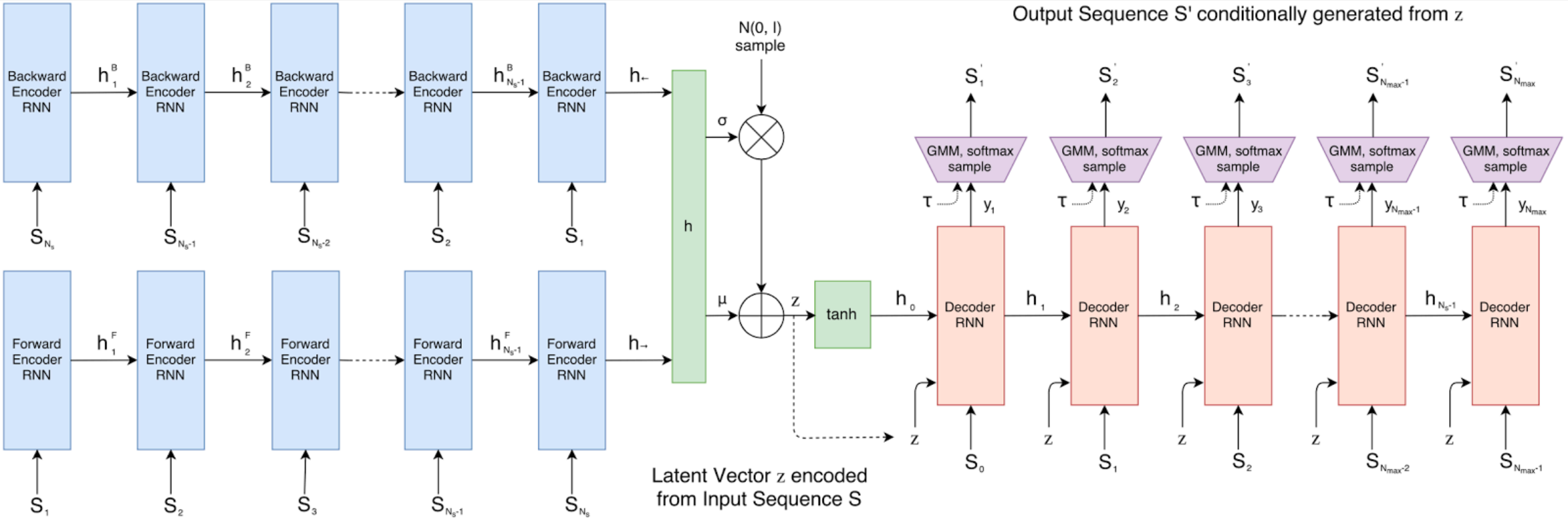
    - N.B. *We can control the level of randomness we would like our samples to have during the sampling process by introducing a temperature parameter* τ

    -

$$\hat{q}_k \to \frac{\hat{q}_k}{\tau},\ \hat{\prod}_k \to \frac{\hat{\prod}_k}{\tau},\ \sigma_x^2 \to \sigma_x^2\tau,\ \sigma_y^2 \to \sigma_y^2\tau$$

SKETCH-RNN

- **Architecture**:



SKETCH-RNN

- **Training**:
  - Model has two losses:
    - **Reconstruction** Loss maximizes the log-likehood of the generated probability distribution to explain the training data S

$$L_R = L_s + L_p$$

      - Offset terms

$$L_s = -\frac{1}{N_{max}} \sum_{i=1}^{N_s} \log \left(\sum_{j=1}^{M} \prod_{j,i} \mathcal{N}(\delta x_i, \delta y_i | \mu_{x,j,i}, \mu_{y,j,i}, \sigma_{x,j,i}, \sigma_{y,j,i}, \rho_{xy,j,i}) \right)$$

      - Pen state terms

$$L_s = -\frac{1}{N_{max}} \sum_{i=1}^{N_{max}} \sum_{k=1}^{3} p_{k,i} \log(q_{k,i}),$$

    - **Kullback-Leibler Divergence Loss**,

$$L_{KL} = -\frac{1}{2N_z}(1 + \hat{\sigma} - \mu^2 - \exp(\hat{\sigma}))$$

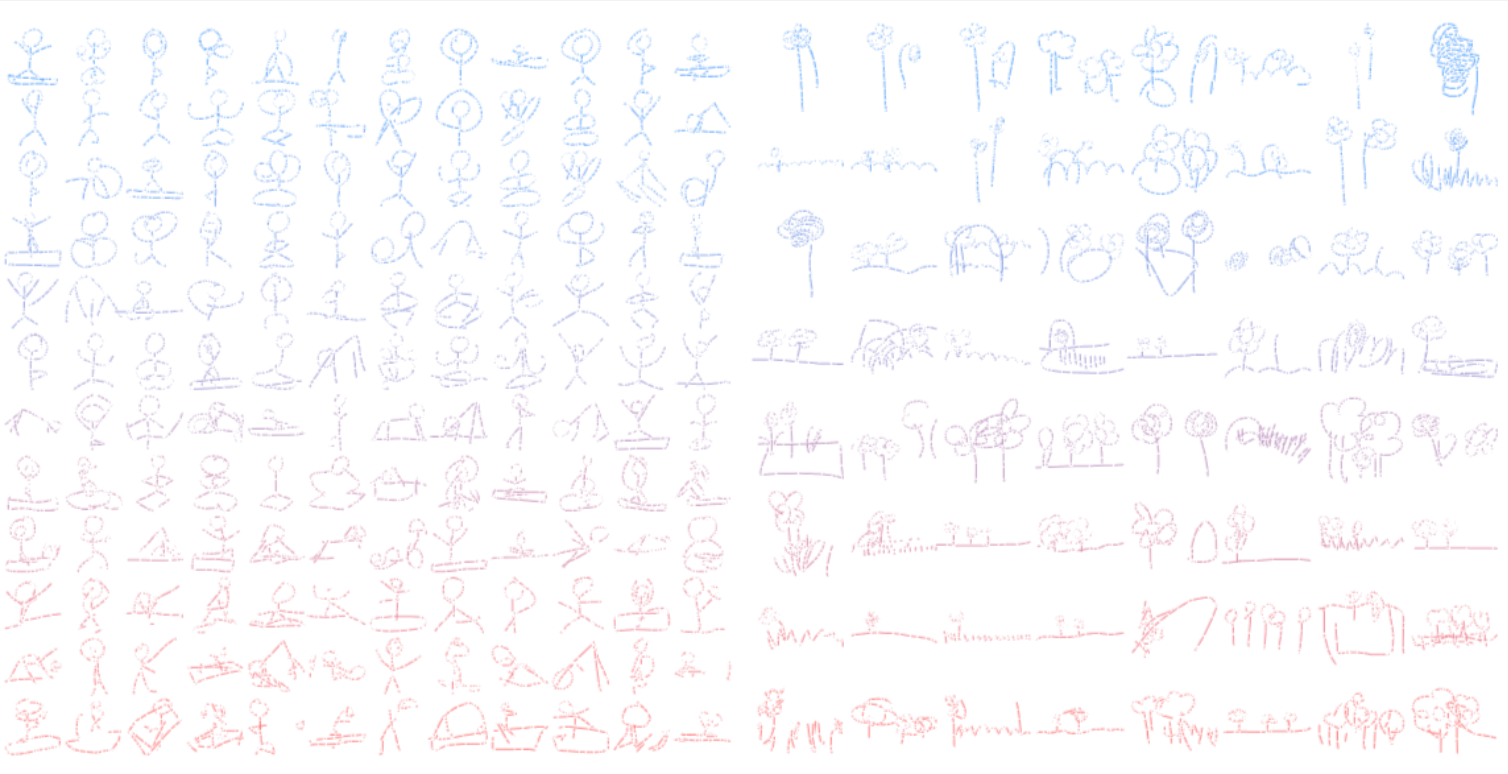
  - Trade-off between the two losses

$$Loss = L_R + w_{KL} L_{KL}$$

  - As W_{KL} –> 0, model is more similar to pure autoencoder

SKETCH-RNN

- **Unconditional Generation**:
  - Model can also be trained to generate sketches unconditionally
  - This means training only the decoder RNN module, without any input or latent vectors
  - autoregressive model without latent variables



- ì

SKETCH-RNN

- **Latent Space Interpolation**
  - By interpolating between latent vectors, we can visualize how one image morphs into another image by visualizing the reconstructions of the interpolations .