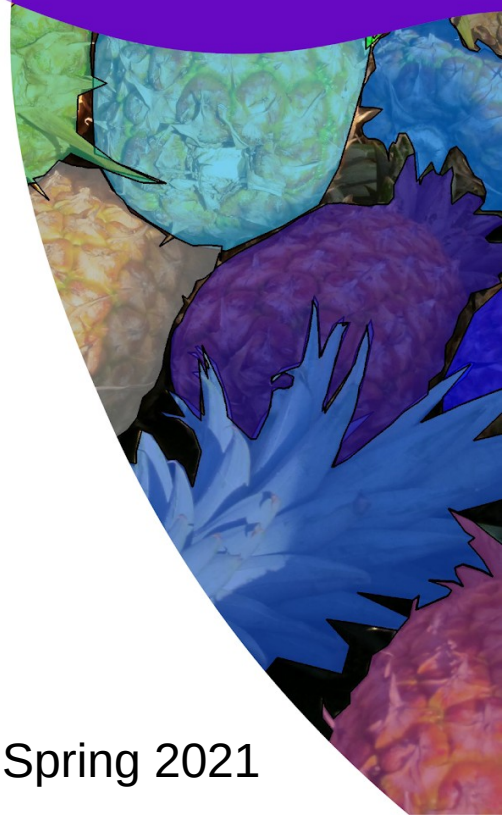
NEW YORK UNIVERSITY

Recurrent Nets, Convolutional Nets

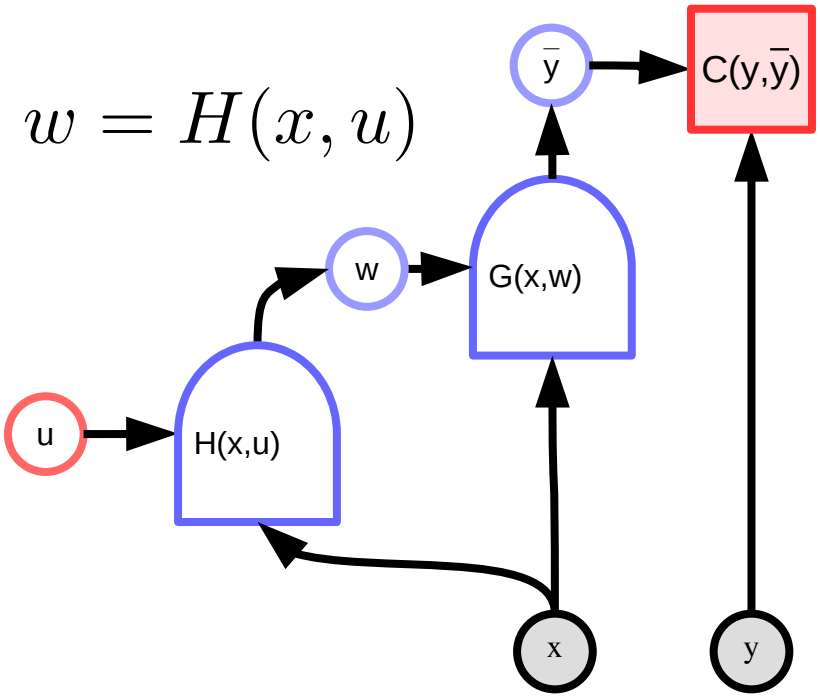


Yann LeCun
NYU - Courant Institute & Center for Data Science
Facebook AI Research

Deep Learning, NYU Spring 2021

“Hypernetwork”

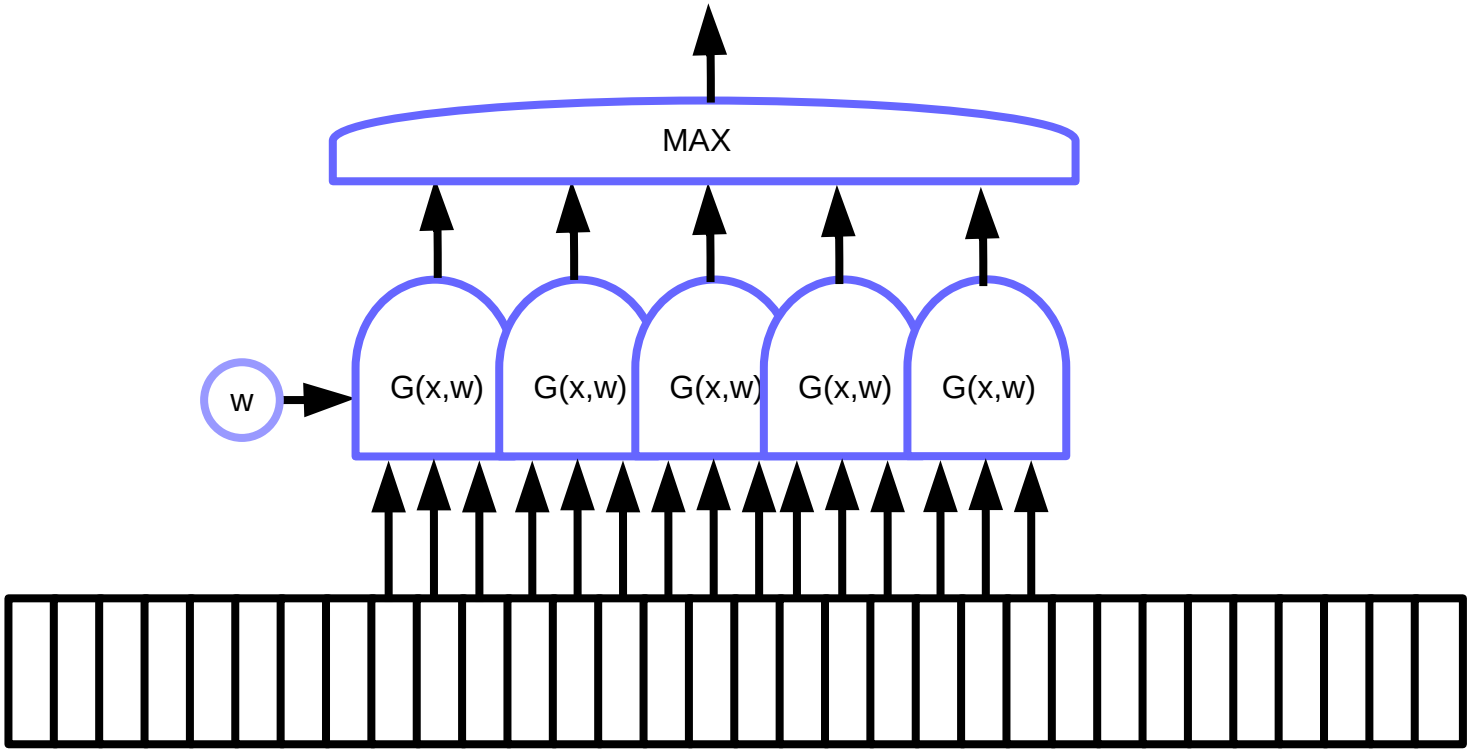
$$w = H(x, u)$$



- **When the parameter vector is the output of another network H(x,u)**
- **The weights of network G(x,w) are dynamically configured by network H(x,u)**
- **The concept is very powerful**
  - The idea is very old

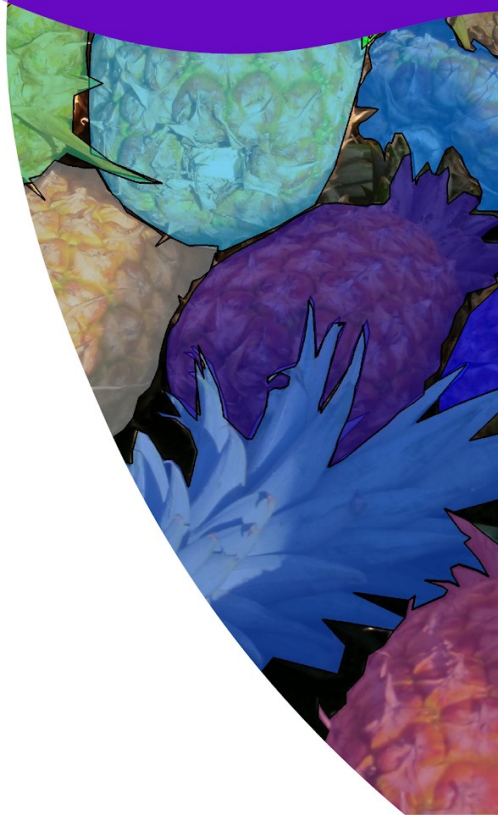
Shared Weights for Motif Detection

- **Detecting motifs anywhere on an input**



NEW YORK UNIVERSITY

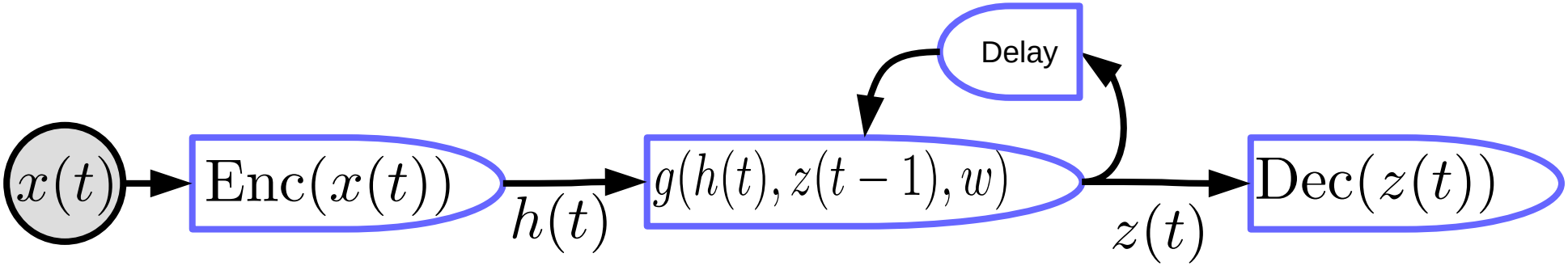
Recurrent Nets



Yann LeCun
NYU - Courant Institute & Center for Data Science
Facebook AI Research

Recurrent Networks

- **Networks with loops. For backprop, unroll the loop.**



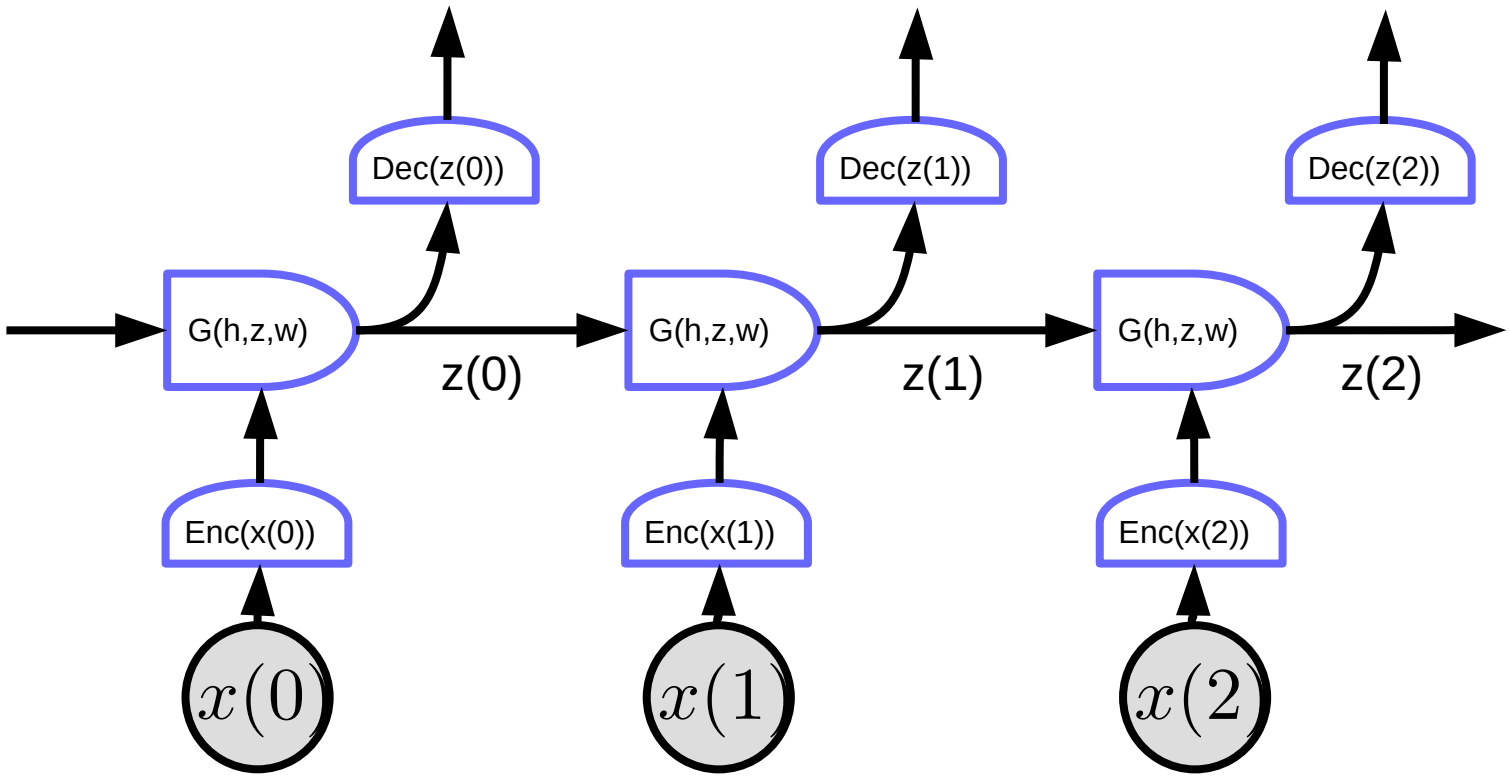
$$h(t) = \mathrm{Enc}(x(t))$$

$$z(t) = g(h(t), z(t-1), w)$$

$$y(t) = \mathrm{Dec}(z(t))$$

Recurrent Networks

- **Networks with loops. For backprop, unroll the loop.**



RNN tricks

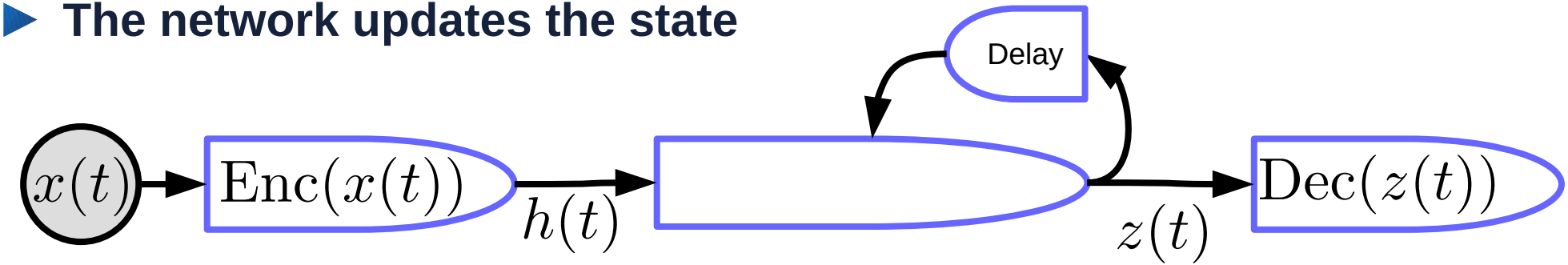
- [Pascanu, Mikolov, Bengio, ICML 2013; Bengio, Boulanger & Pascanu, ICASSP 2013]

- Clipping gradients (avoid exploding gradients)
- Leaky integration (propagate long-term dependencies)
- Momentum (cheap 2nd order)
- Initialization (start in right ballpark avoids exploding/vanishing)
- Sparse Gradients (symmetry breaking)
- Gradient propagation regularizer (avoid vanishing gradient)
- LSTM self-loops (avoid vanishing gradient)

Recurrent Networks for differential equations

- **Differential equation**
- **Time discretization**
- **The network updates the state**

$$\frac{dz(t)}{dt} = g(h(t), z(t), w)$$



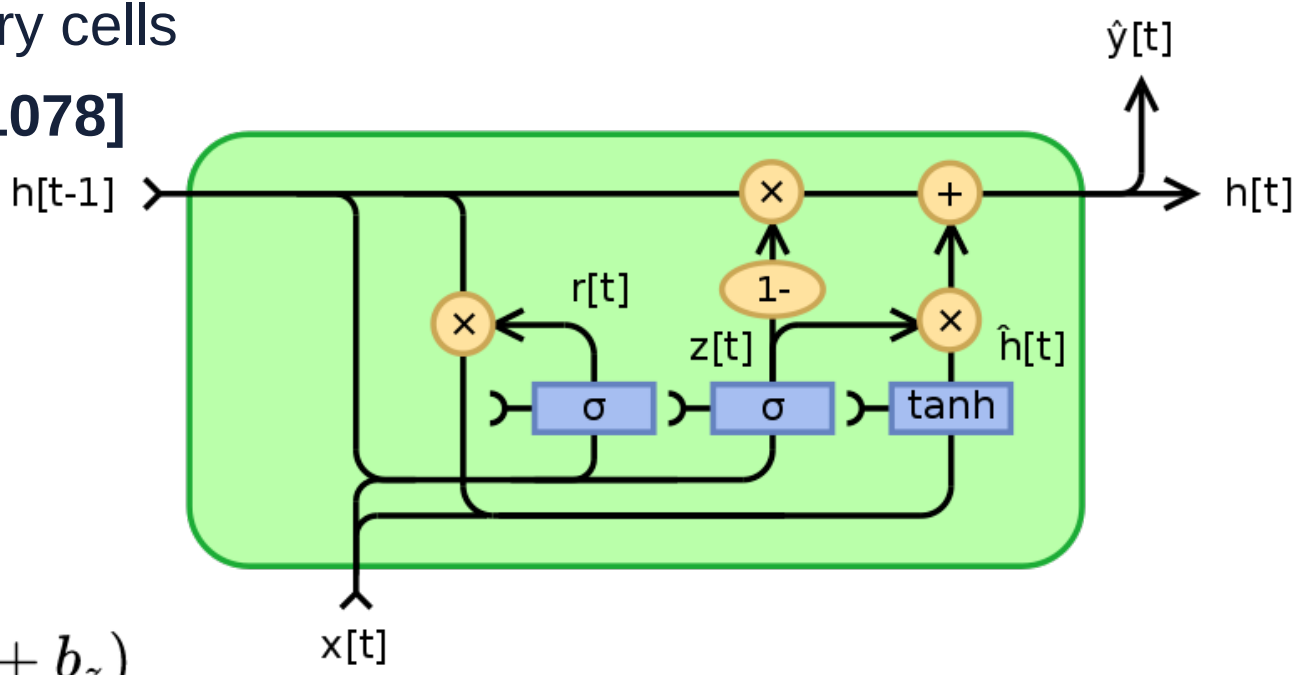
$$h(t) = \mathrm{Enc}(x(t))$$

$$z(t) = z(t - \delta t) + \delta t \; g(h(t), z(t - \delta t), w)$$

$$y(t) = \mathrm{Dec}(z(t))$$

GRU (Gated Recurrent Units)

- **Recurrent nets quickly "forget" their state**
  - Solution: explicit memory cells
- **GRU [Cho arXiv:1406.1078]**



x_t: input vector

h_t: output vector

z_t: update gate vector

r_t: reset gate vector

W, U and b: parameter ma

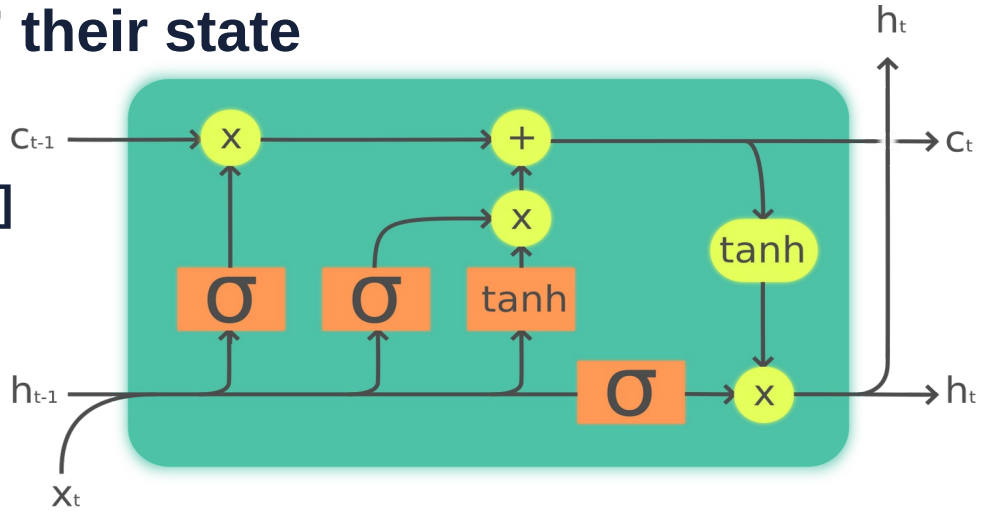
$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$

$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \phi_h(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h)$$

LSTM (Long Short-Term Memory)

- **Recurrent nets quickly "forget" their state**
  - Solution: explicit memory cells
- **LSTM [Hochreiter & Schmidhuber 97]**



$x_t \in \mathbb{R}^d$: input vector to the LSTM unit

$f_t \in \mathbb{R}^h$: forget gate's activation vector

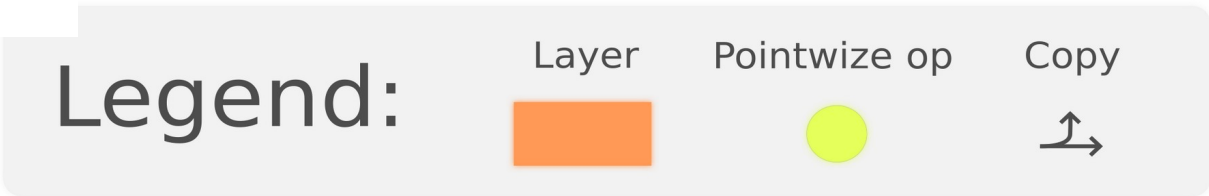
$i_t \in \mathbb{R}^h$: input/update gate's activation vector

$o_t \in \mathbb{R}^h$: output gate's activation vector

$h_t \in \mathbb{R}^h$: hidden state vector also known as output

$c_t \in \mathbb{R}^h$: cell state vector

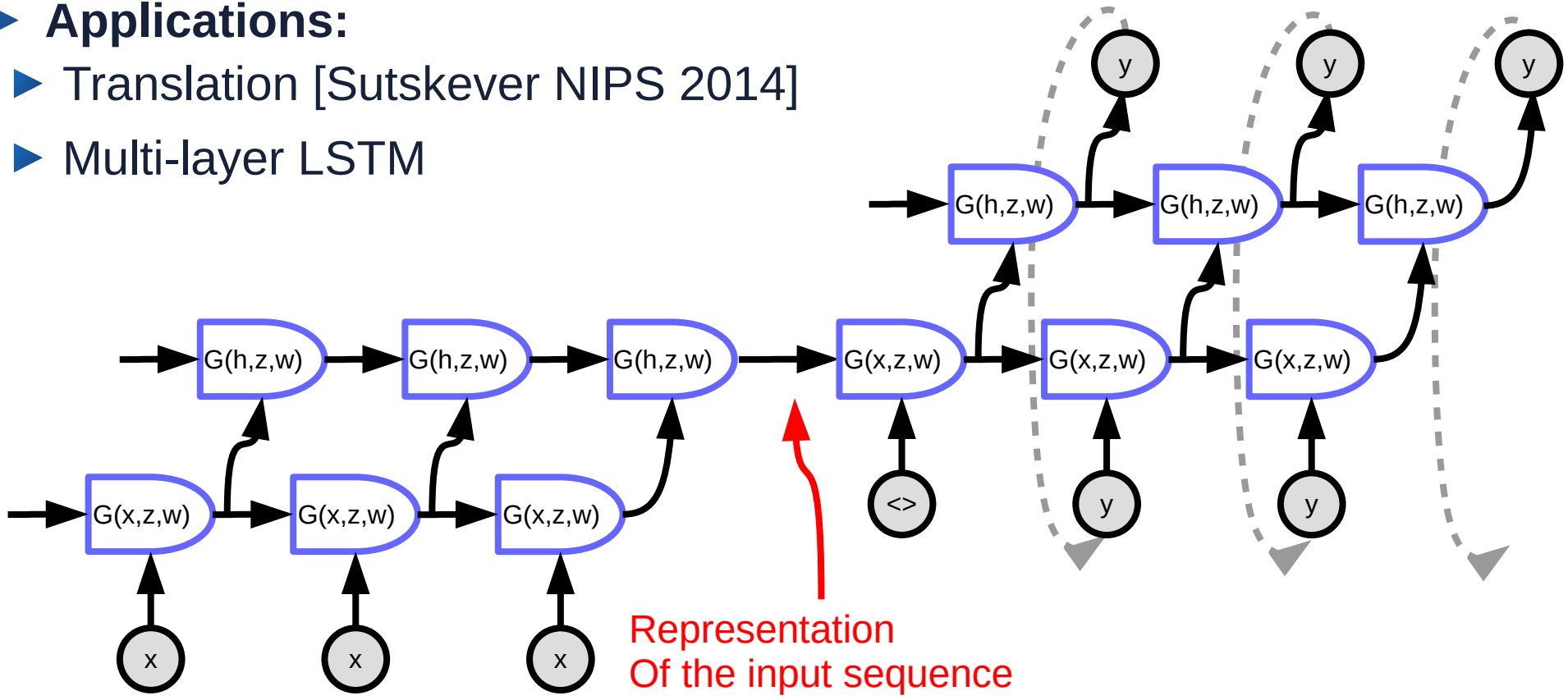
$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$$
$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$
$$h_t = o_t \circ \sigma_h(c_t)$$



Guillaume Chevalier https://commons.wikimedia.org/w/index.php?curid=71836793

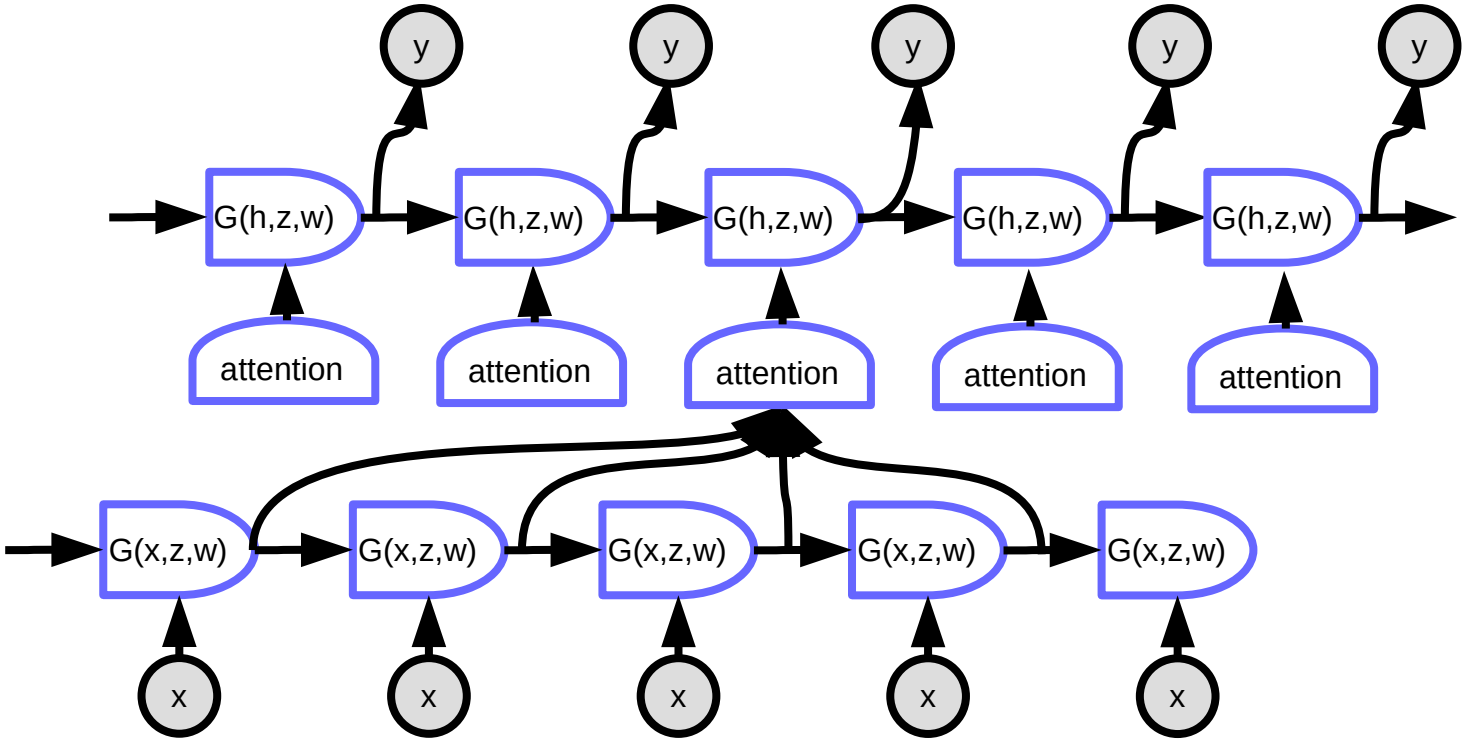
Sequence to Sequence

- **Sequence encoder → sequence decoder**
- **Applications:**
  - Translation [Sutskever NIPS 2014]
  - Multi-layer LSTM



Sequence to Sequence with Attention

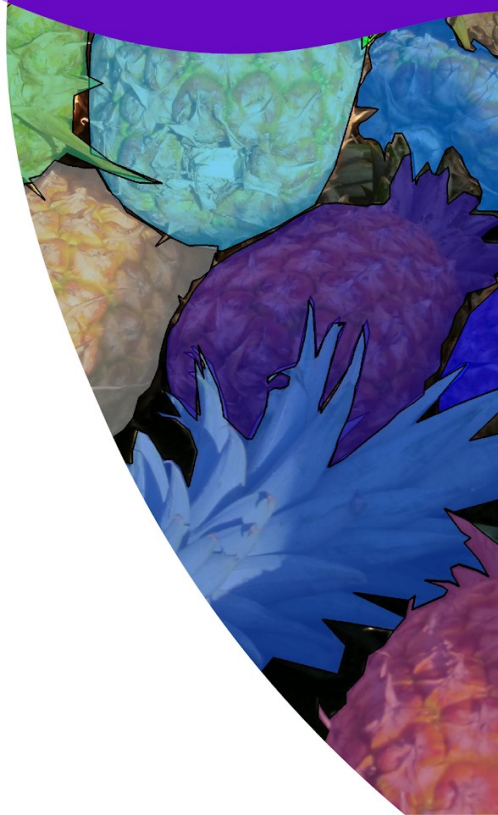
- **Sequence encoder → sequence decoder with attention**
- **Applications:**
  - Translation [Bahdanau, Cho, Bengio ArXiv:1409.0473]



NEW YORK UNIVERSITY

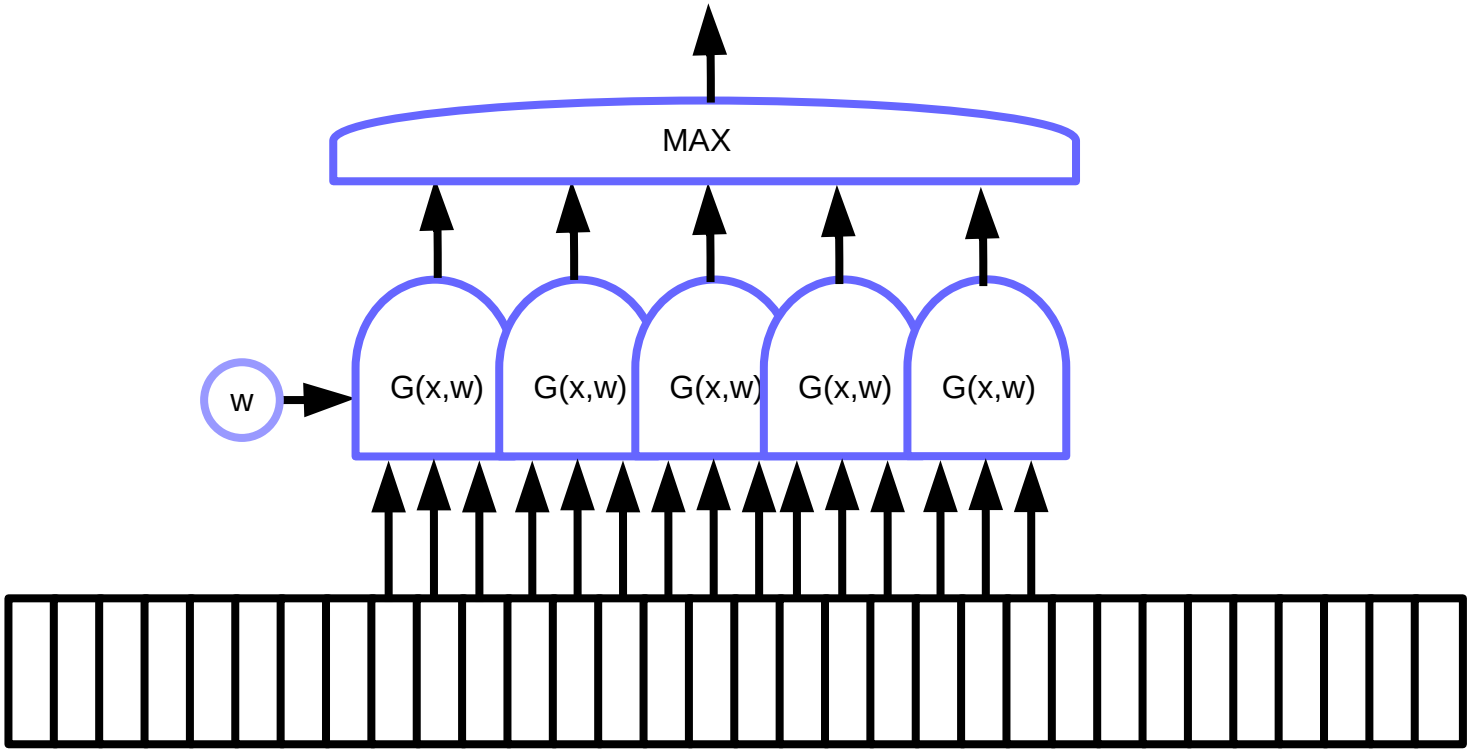
Convolutional Networks

For sequences, audio, speech, images, volumetric images, video, and other natural signals.



Shared Weights for Motif Detection

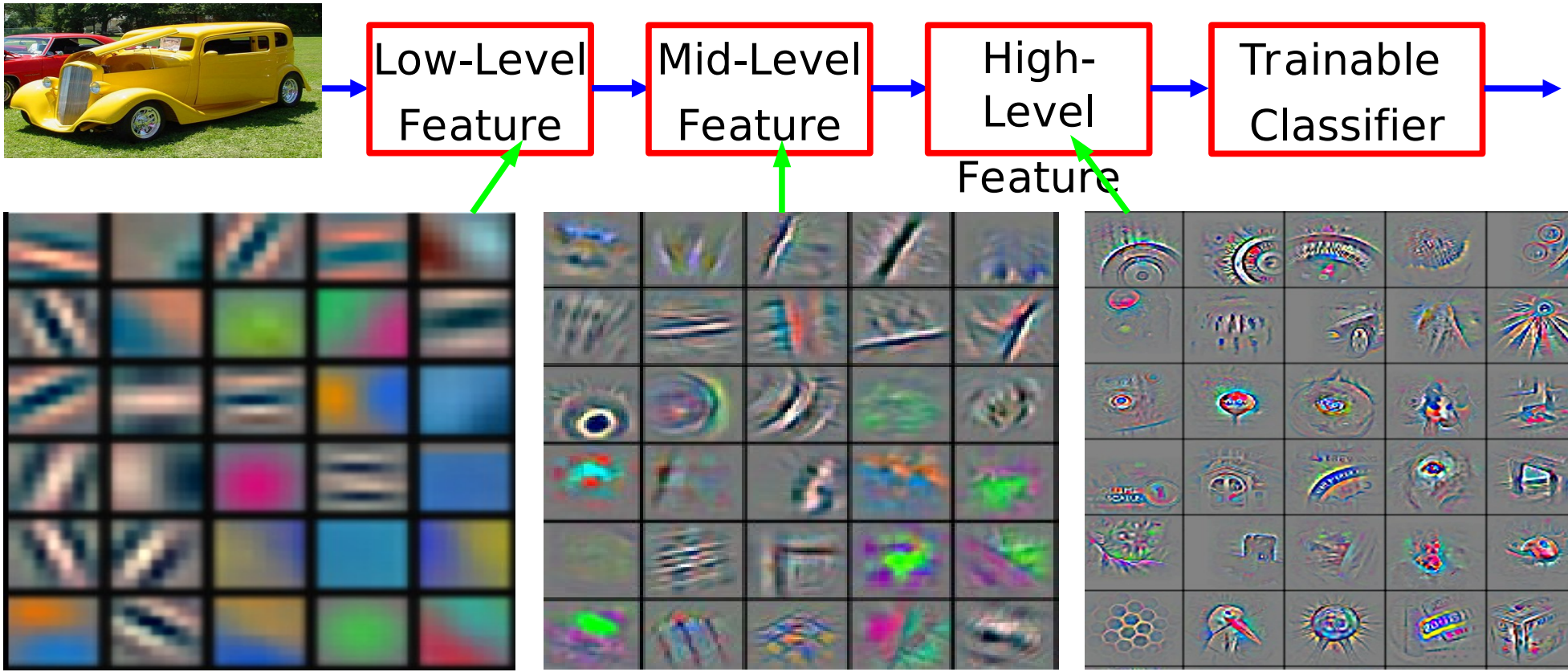
- **Detecting motifs anywhere on an input**



Deep Learning = Learning Hierarchical Representations

Y. LeCun

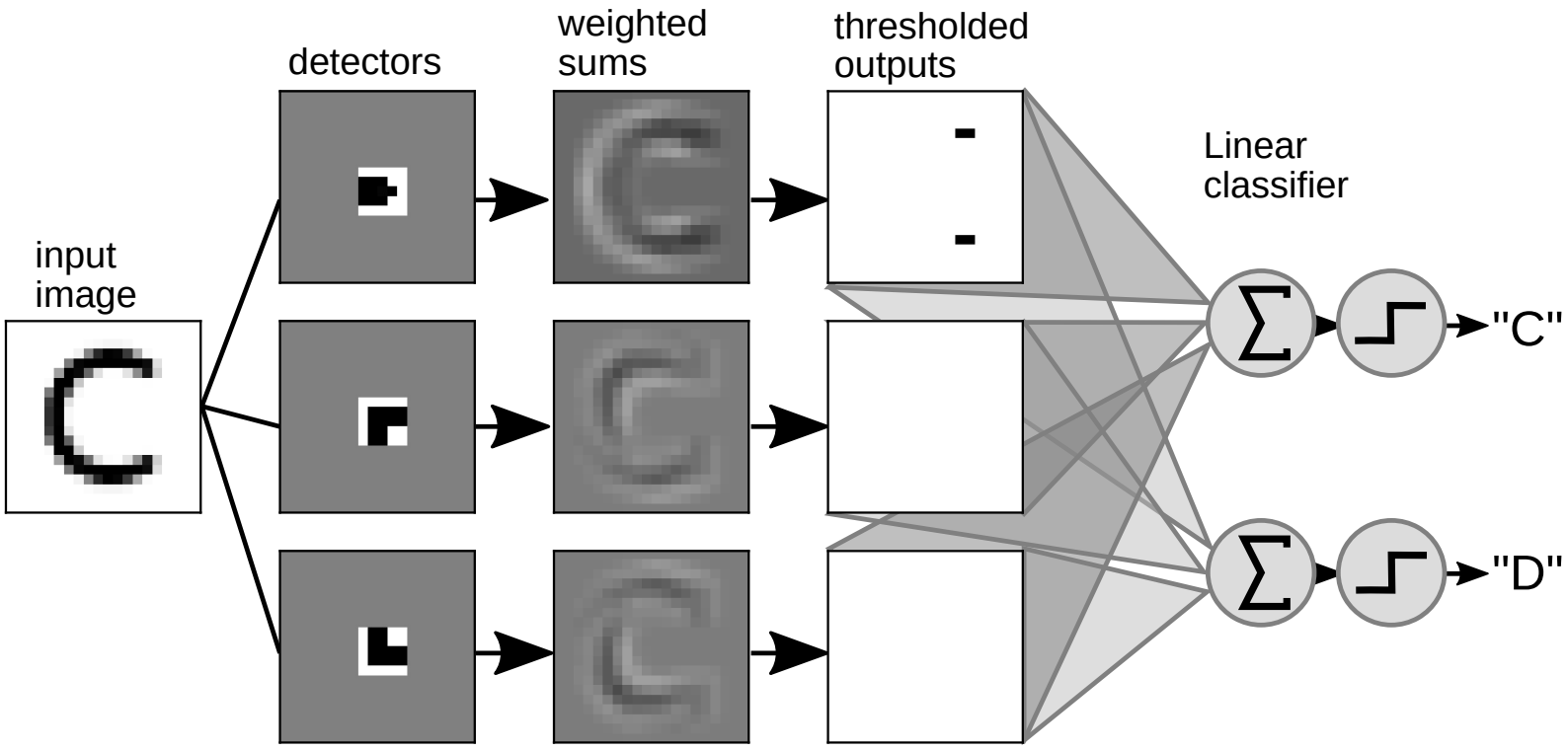
- It's **deep** if it has **more than one stage** of non-linear feature transformation



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

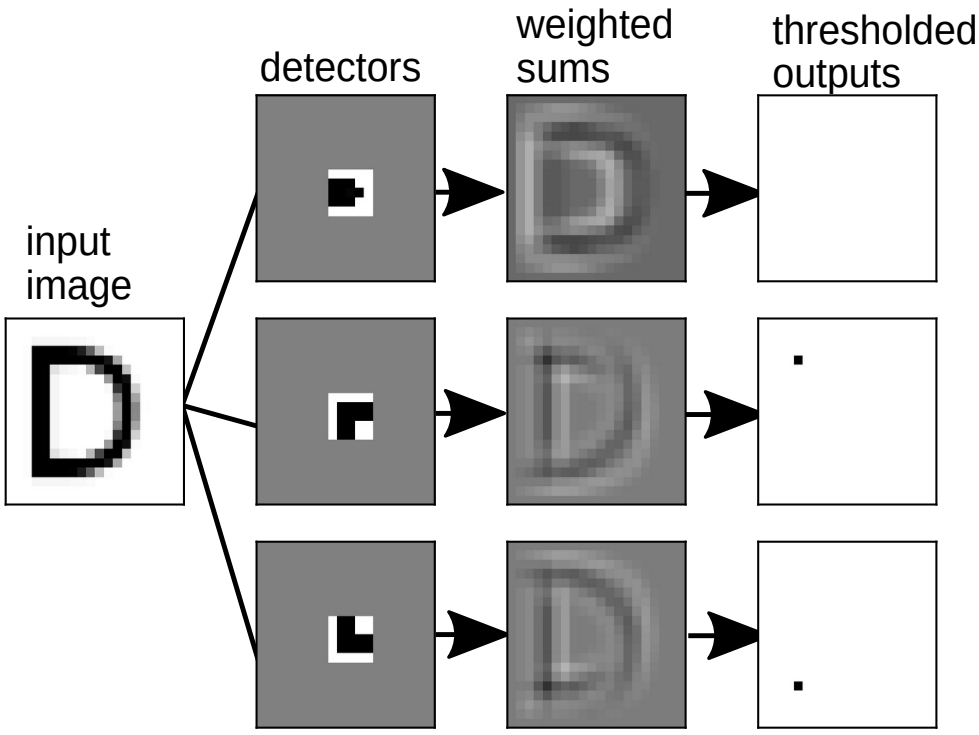
Detecting Motifs in Images

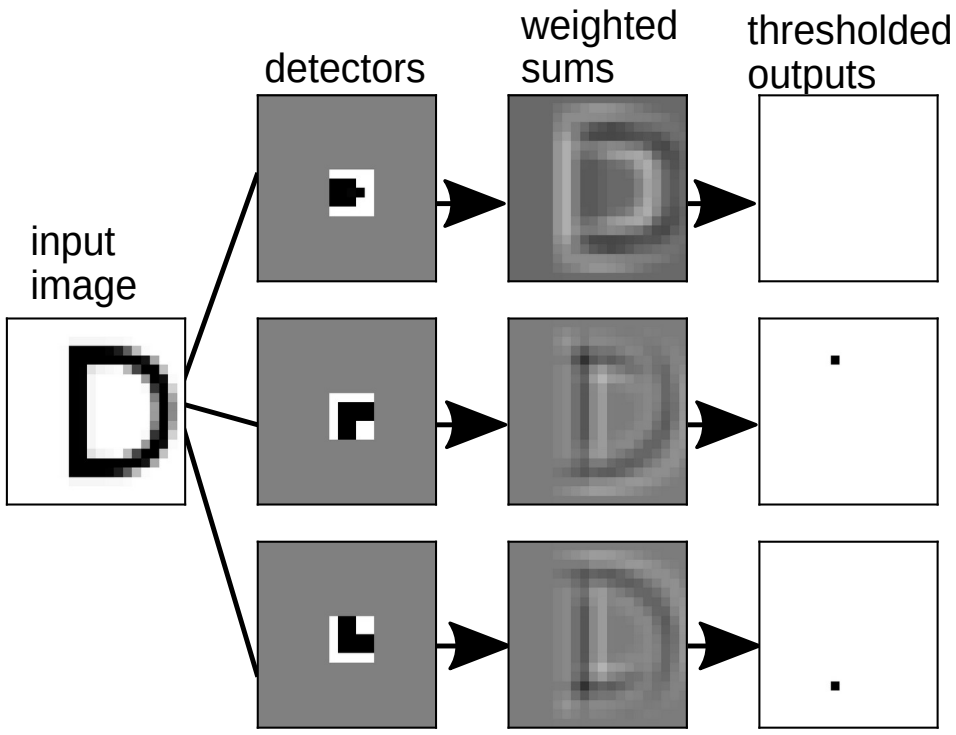
- **Swipe “templates” over the image to detect motifs**



Detecting Motifs in Images

- **Shift invariance**





Discrete Convolution (or cross-correlation)

- **Definition**
  - convolution

$$y_i = \sum_j w_j x_{i-j}$$

- **In practice**
  - Cross-correlation

$$y_i = \sum_j w_j x_{i+j}$$

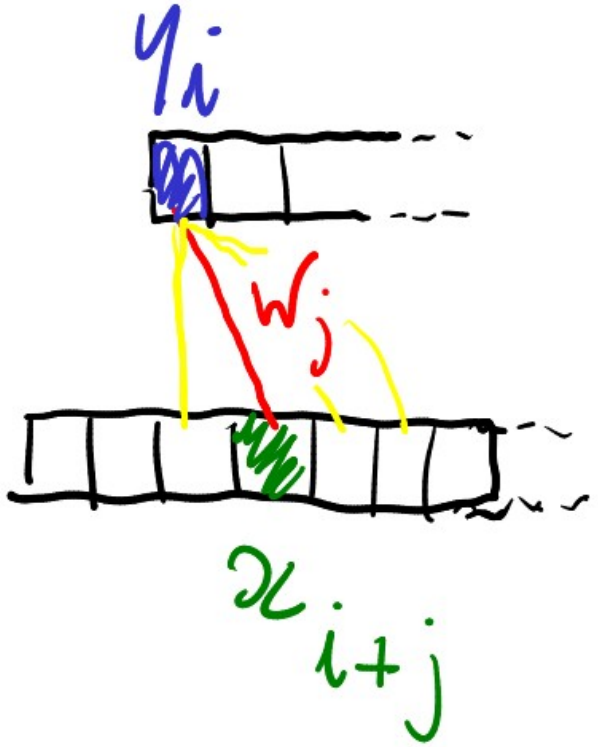
- **In 2D**

$$y_{ij} = \sum_{kl} w_{kl} x_{i+k,j+l}$$

Backpropagating through convolutions

- **Convolution**
  - (really: cross-correlation)

$$y_i = \sum_j w_j x_{i+j}$$



- **Backprop to input**
  - Sometimes called “back-convolution”

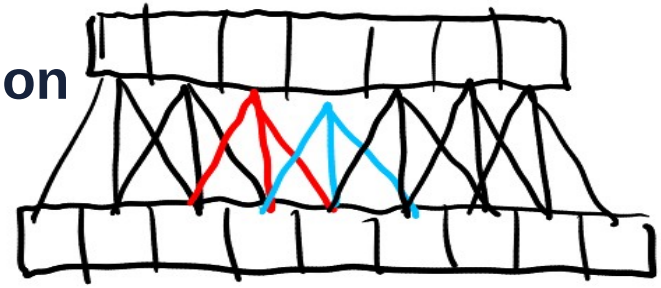
$$\frac{\partial C}{\partial x_j} = \sum_k w_k \frac{\partial C}{\partial y_{j-k}}$$

- **Backprop to weights**

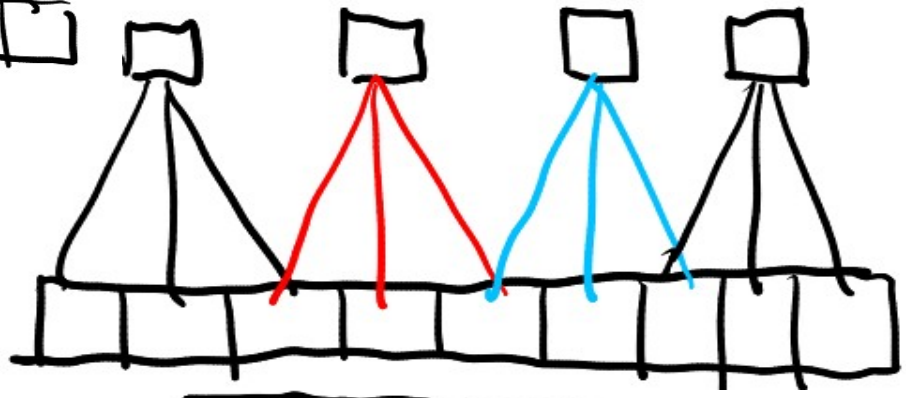
$$\frac{\partial C}{\partial w_j} = \sum_i \frac{\partial C}{\partial y_i} x_{i+j}$$

Stride and Skip: subsampling and convolution “à trous”

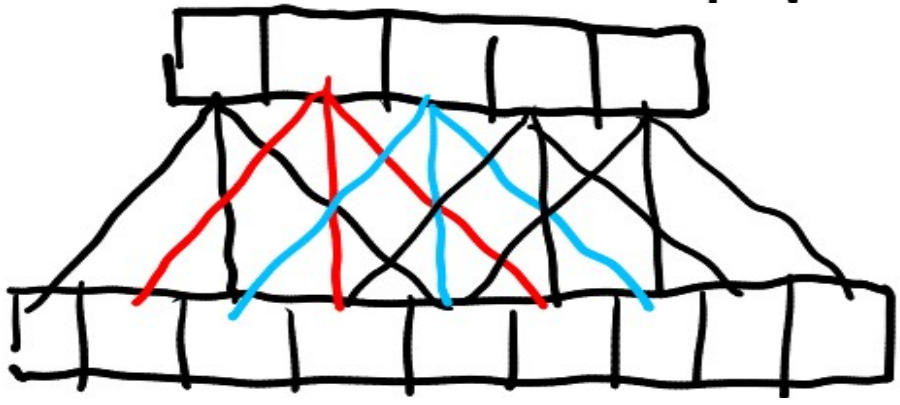
- **Regular convolution**
  - “dense”



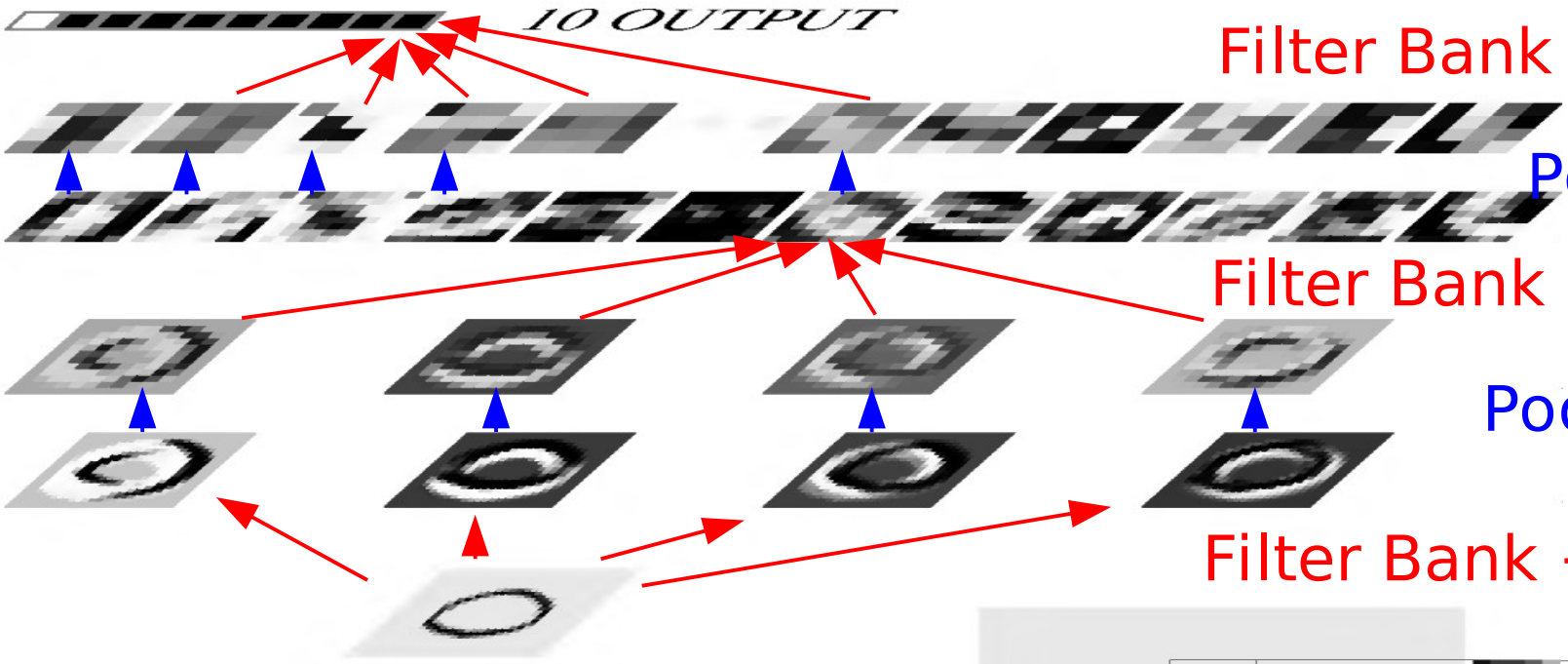
- **Stride**
  - subsampling convolution
  - Reduces spatial resolution



- **Skip, convolution “à trous”**
  - pronounced “ah troo” (means “with holes”)
  - Dimensionality reduction without loss of resolution



Convolutional Network Architecture



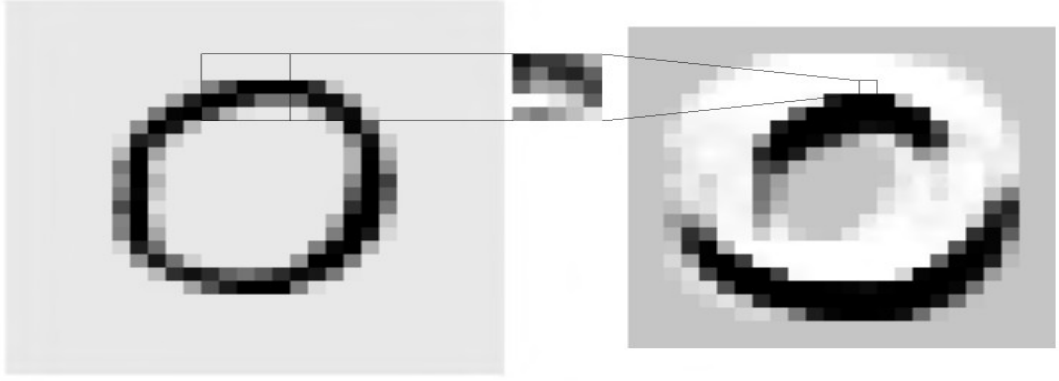
Filter Bank +non-linearity

Pooling

Filter Bank +non-linearity

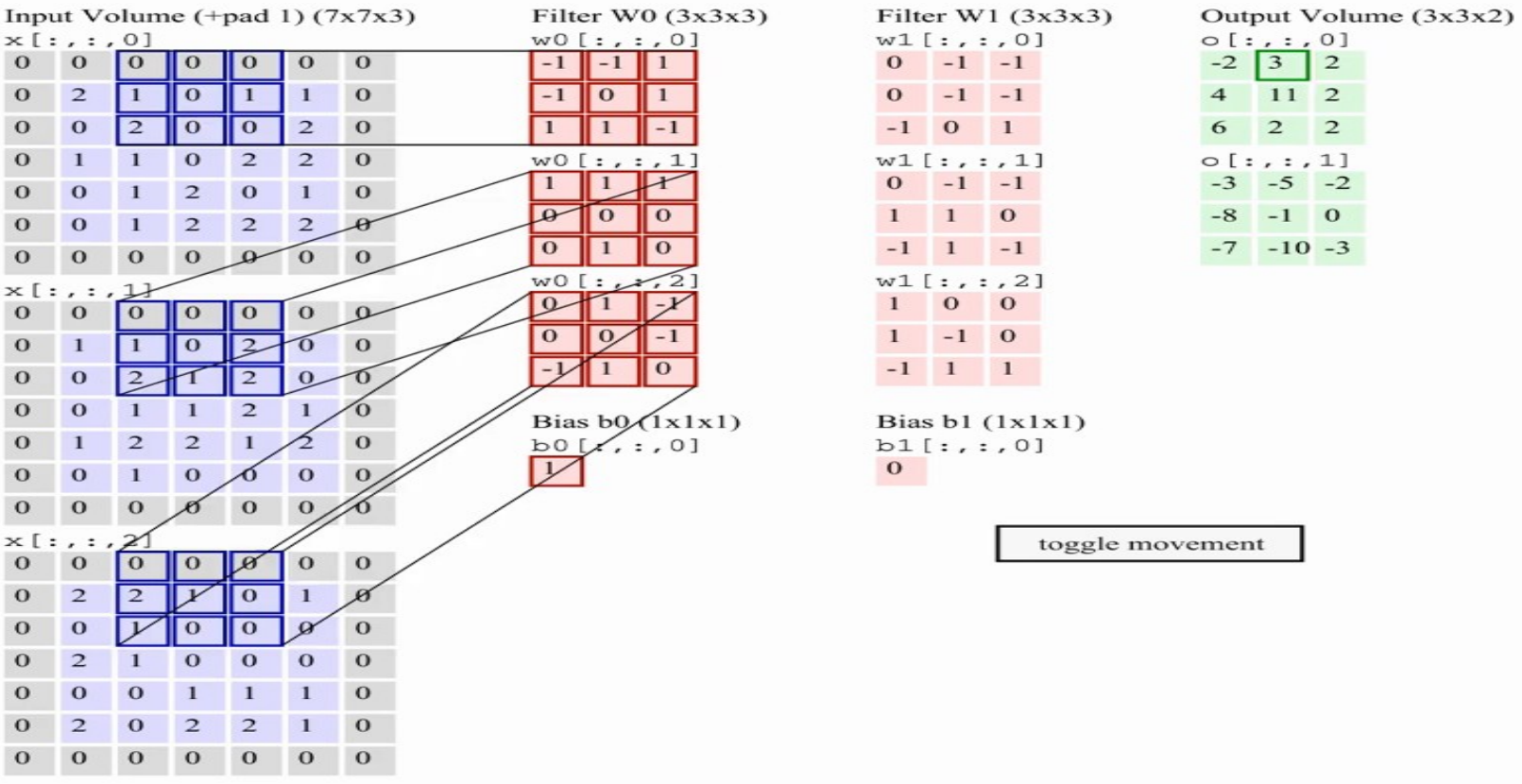
Pooling

Filter Bank +non-linearity



- [LeCun et al. NIPS 1989]

Multiple Convolutions

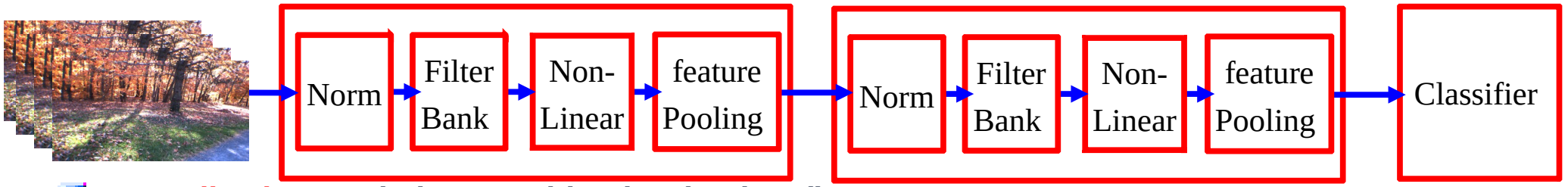


Animation: Andrej Karpathy http://cs231n.github.io/convolutional-networks/

Convolutional Network (vintage 1990)

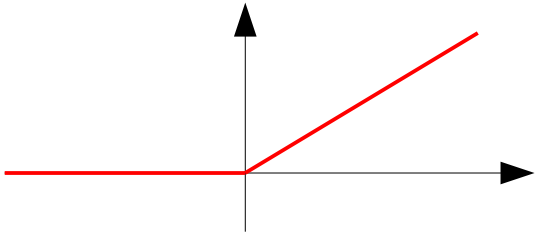
- **Filters-tanh → pooling → filters-tanh → pooling → filters-tanh**

Overall Architecture: multiple stages of
Normalization → Filter Bank → Non-Linearity → Pooling



- **Normalization: variation on whitening (optional)**
  - Subtractive: average removal, high pass filtering
  - Divisive: local contrast normalization, variance normalization
- **Filter Bank: dimension expansion, projection on overcomplete basis**
- **Non-Linearity: sparsification, saturation, lateral inhibition....**
  - Rectification (ReLU), Component-wise shrinkage, tanh,..

$$ReLU(x) = max(x, 0)$$



- **Pooling: aggregation over space or feature type**
  - Max, Lp norm, log prob.

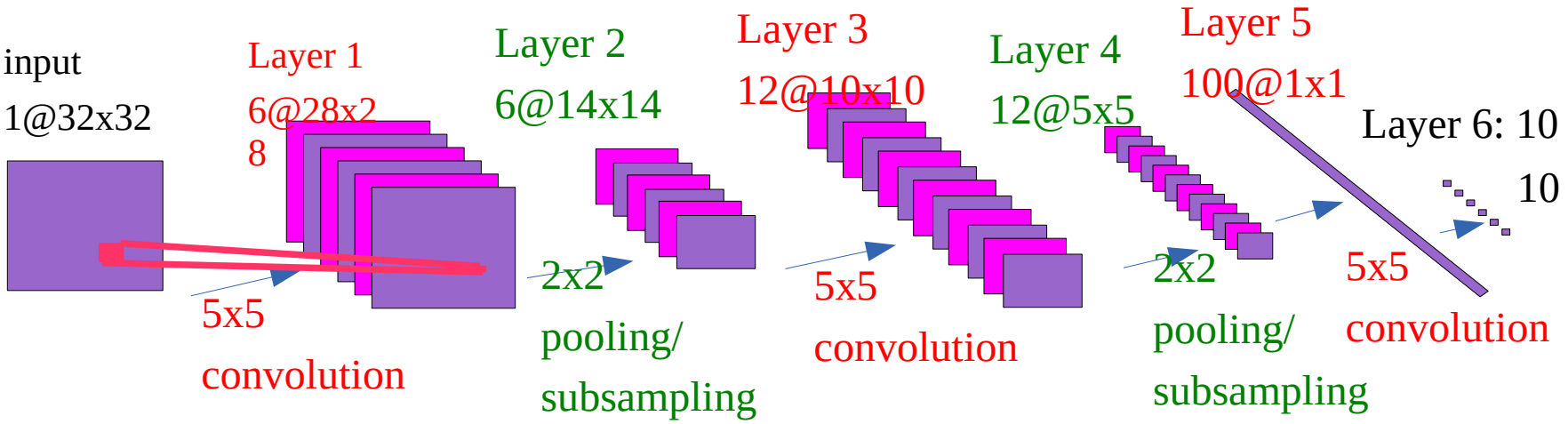
$$MAX: Max_i(X_i); \quad L_p: \sqrt[p]{X_i^p}; \quad PROB: \frac{1}{b}\log\left(\sum_i e^{bX_i}\right)$$

LeNet5

- **Simple ConvNet**
- **for MNIST**
- **[LeCun 1998]**

- **PyTorch code ------→**
  - (slightly different net)

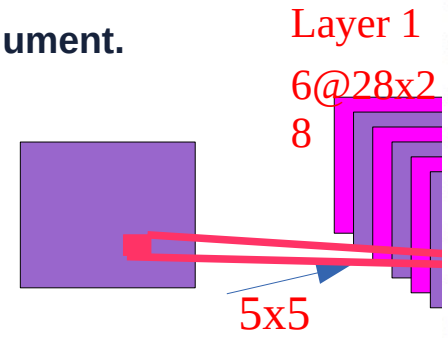
```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5, 1)
        self.conv2 = nn.Conv2d(20, 50, 5, 1)
        self.fc1 = nn.Linear(4*4*50, 500)
        self.fc2 = nn.Linear(500, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2, 2)
        x = x.view(-1, 4*4*50)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```



LeNet5

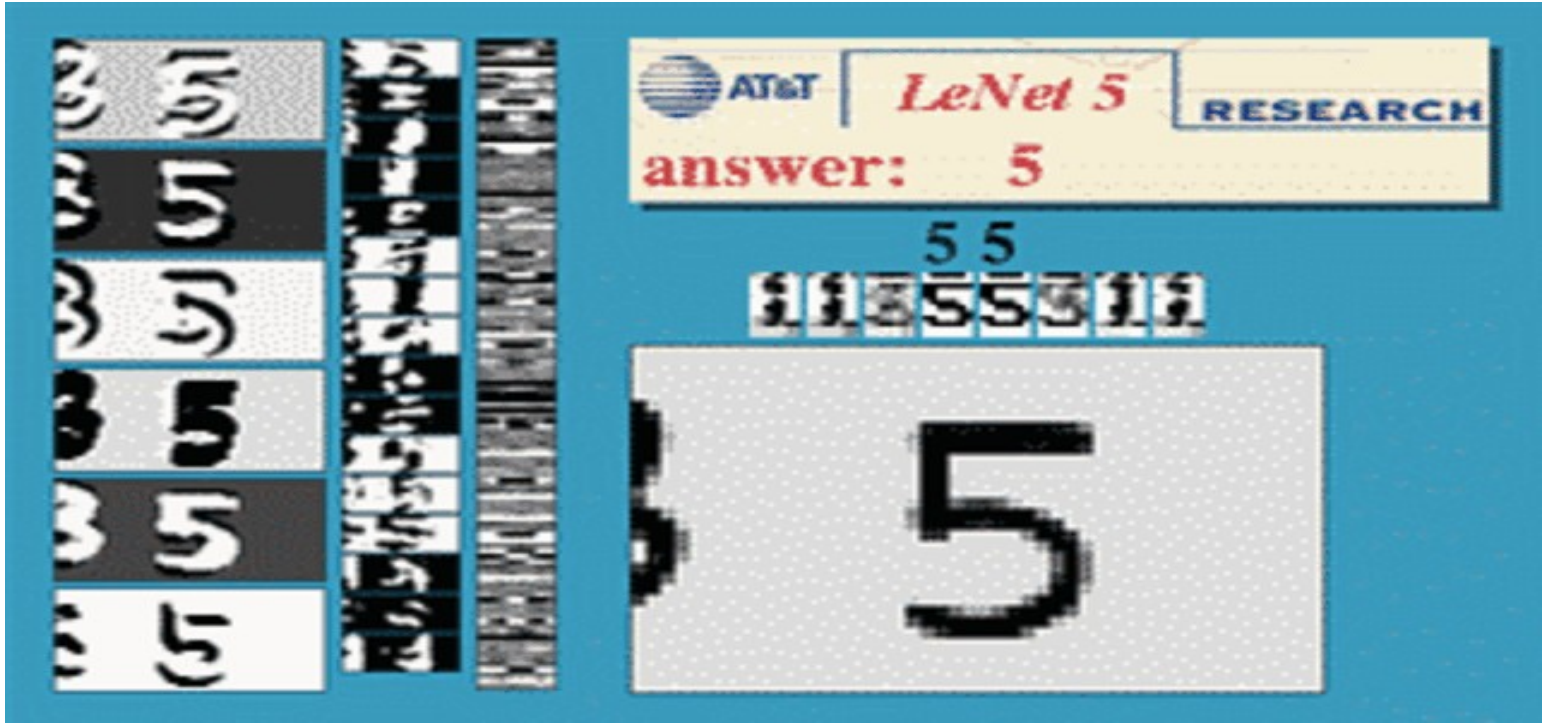
- Simple ConvNet
- for MNIST
- [LeCun 1998]

- PyTorch code --→ github.com/activatedgeek/LeNet-5
- nn.sequential() with ordered dictionary argument.




```
19      def __init__(self):
20          super(LeNet5, self).__init__()
21
22          self.convnet = nn.Sequential(OrderedDict([
23              ('c1', nn.Conv2d(1, 6, kernel_size=(5, 5))),
24              ('relu1', nn.ReLU()),
25              ('s2', nn.MaxPool2d(kernel_size=(2, 2), stride=2)),
26              ('c3', nn.Conv2d(6, 16, kernel_size=(5, 5))),
27              ('relu3', nn.ReLU()),
28              ('s4', nn.MaxPool2d(kernel_size=(2, 2), stride=2)),
29              ('c5', nn.Conv2d(16, 120, kernel_size=(5, 5))),
30              ('relu5', nn.ReLU())
31          ]))
32
33          self.fc = nn.Sequential(OrderedDict([
34              ('f6', nn.Linear(120, 84)),
35              ('relu6', nn.ReLU()),
36              ('f7', nn.Linear(84, 10)),
37              ('sig7', nn.LogSoftmax(dim=-1))
38          ]))
39
40      def forward(self, img):
41          output = self.convnet(img)
42          output = output.view(img.size(0), -1)
43          output = self.fc(output)
44          return output
```

Sliding Window ConvNet + Weighted Finite-State Machine

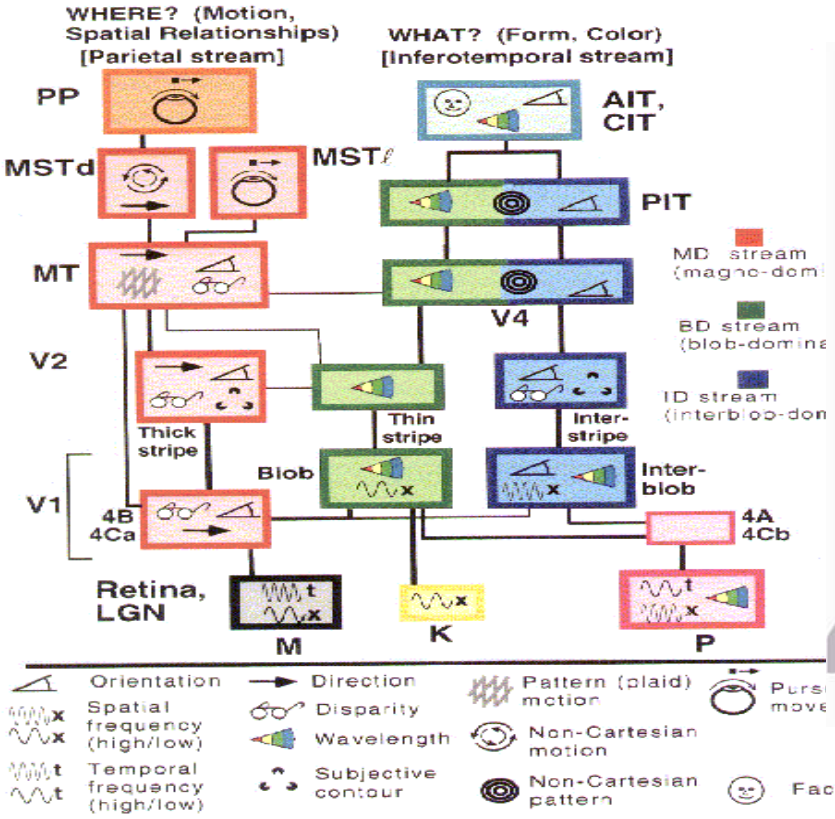


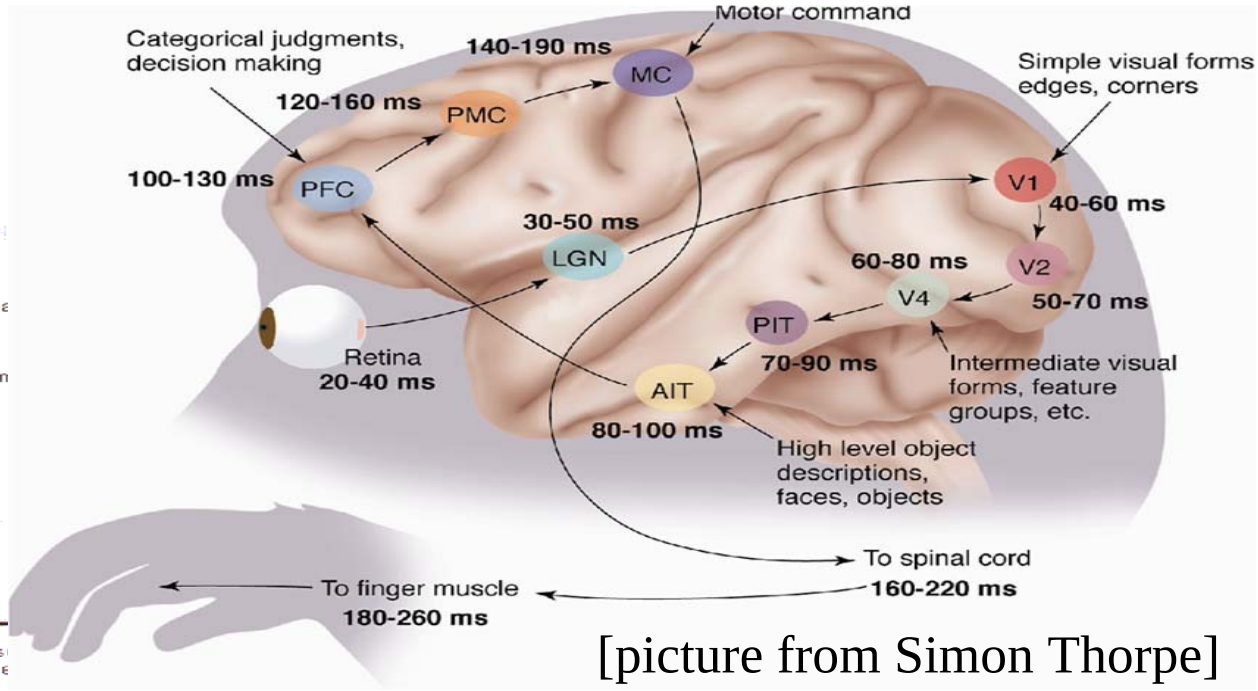
LeNet character recognition demo 1992

- **Running on an AT&T DSP32C (floating-point DSP, 20 MFLOPS)**

How does the brain interprets images?

- The ventral (recognition) pathway in the visual cortex has multiple stages
- Retina - LGN - V1 - V2 - V4 - PIT - AIT ....





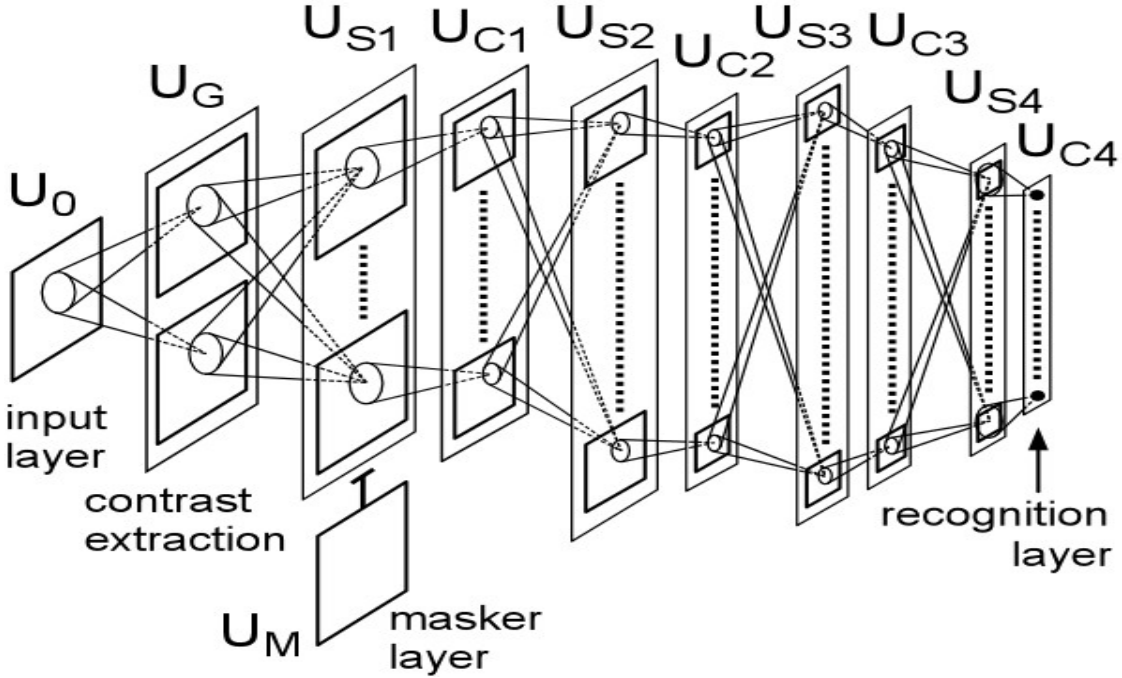
[picture from Simon Thorpe]

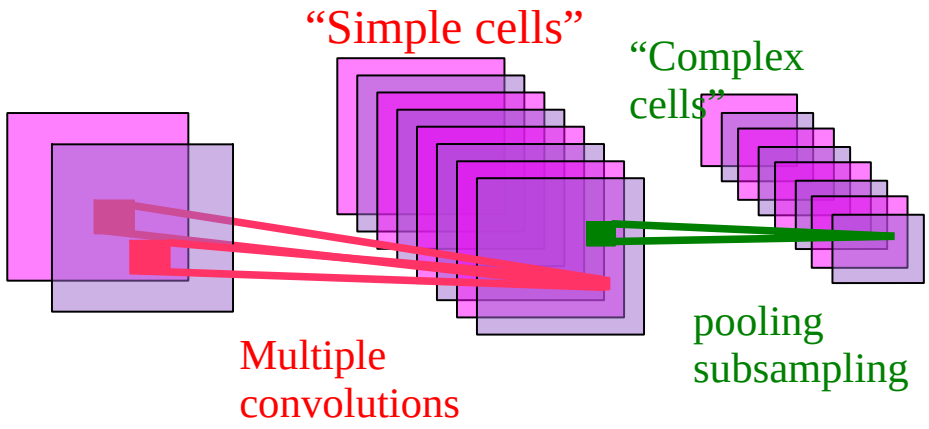
[Gallant & Van Essen]

Hubel & Wiesel's Model of the Architecture of the Visual Cortex

- **[Hubel & Wiesel 1962]:**
  - simple cells detect local features
  - complex cells "pool" the outputs of simple

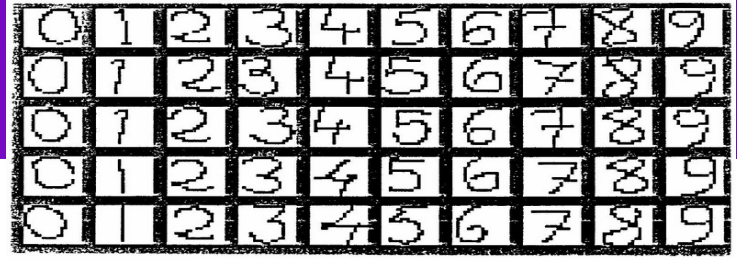
d.



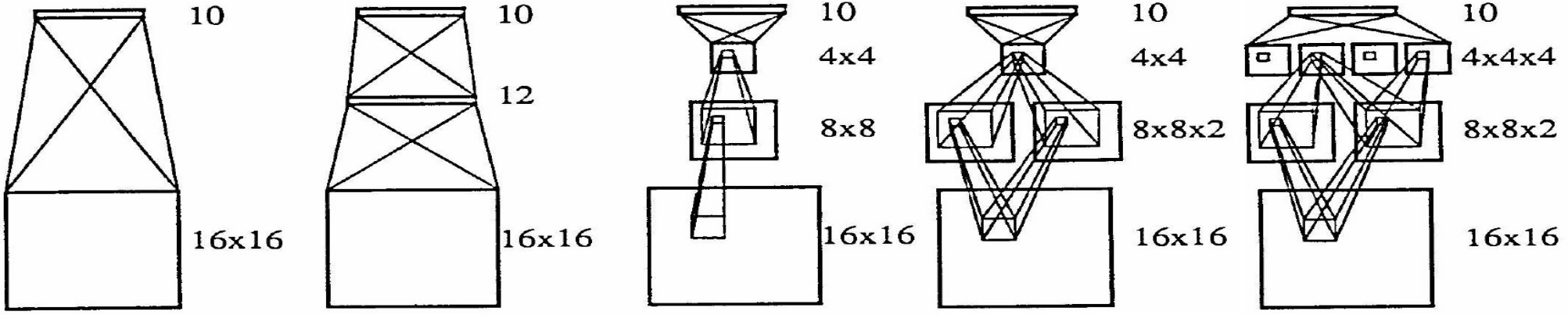


[Fukushima 1982][LeCun 1989, 1998],[Riesenhuber 1999].....

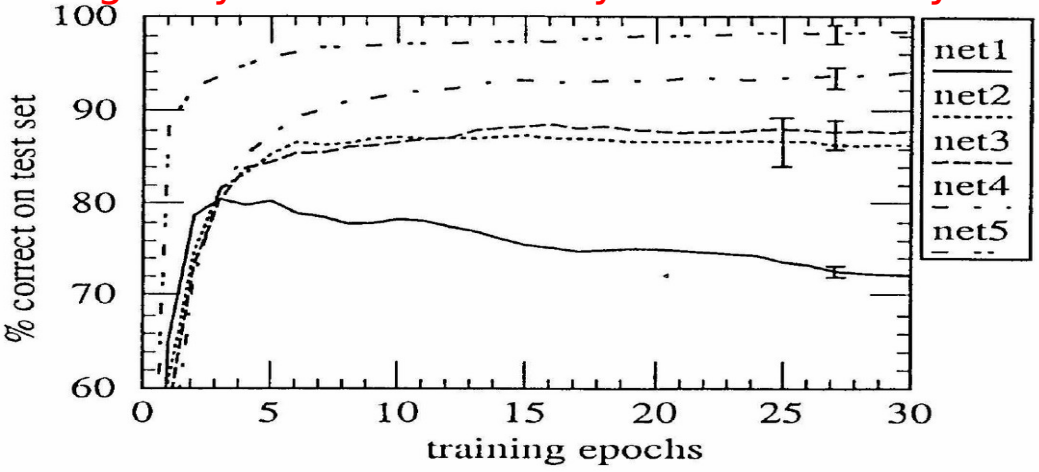
First ConvNets (U Toronto)[LeCun 88, 89]



- **Trained with Backprop. 320 examples.**



Single layer | Two layers FC | locally connected | Shared weights | Shared weights



- Convolutions with stride (subsampling)
- No separate pooling layers

network architecture	links	weights	performance
single layer network	2570	2570	80 %
two layer network	3240	3240	87 %
locally connected	1226	1226	88.5 %
constrained network	2266	1132	94 %
constrained network 2	5194	1060	98.4 %

First “Real” ConvNets at Bell Labs [LeCun et al 89]

- **Trained with Backprop.**
- **USPS Zipcode digits: 7300 training, 2000 test**
- **Convolution with stride. No separate pooling.**