CNN for intelligent control

Wen Yu

NN for image Classification

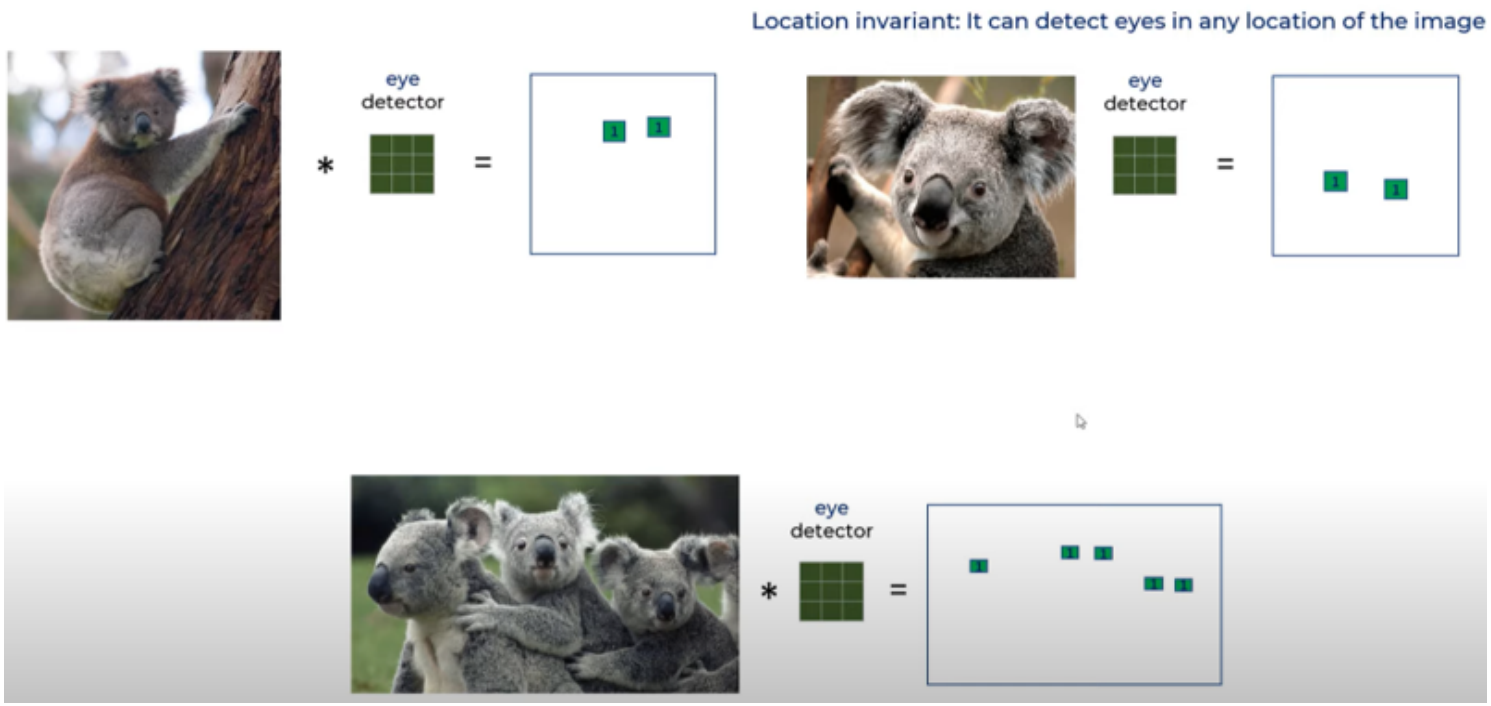
Image size = 1920 x 1080 X 3

First layer neurons = 1920 x 1080 X 3 ~ 6 million

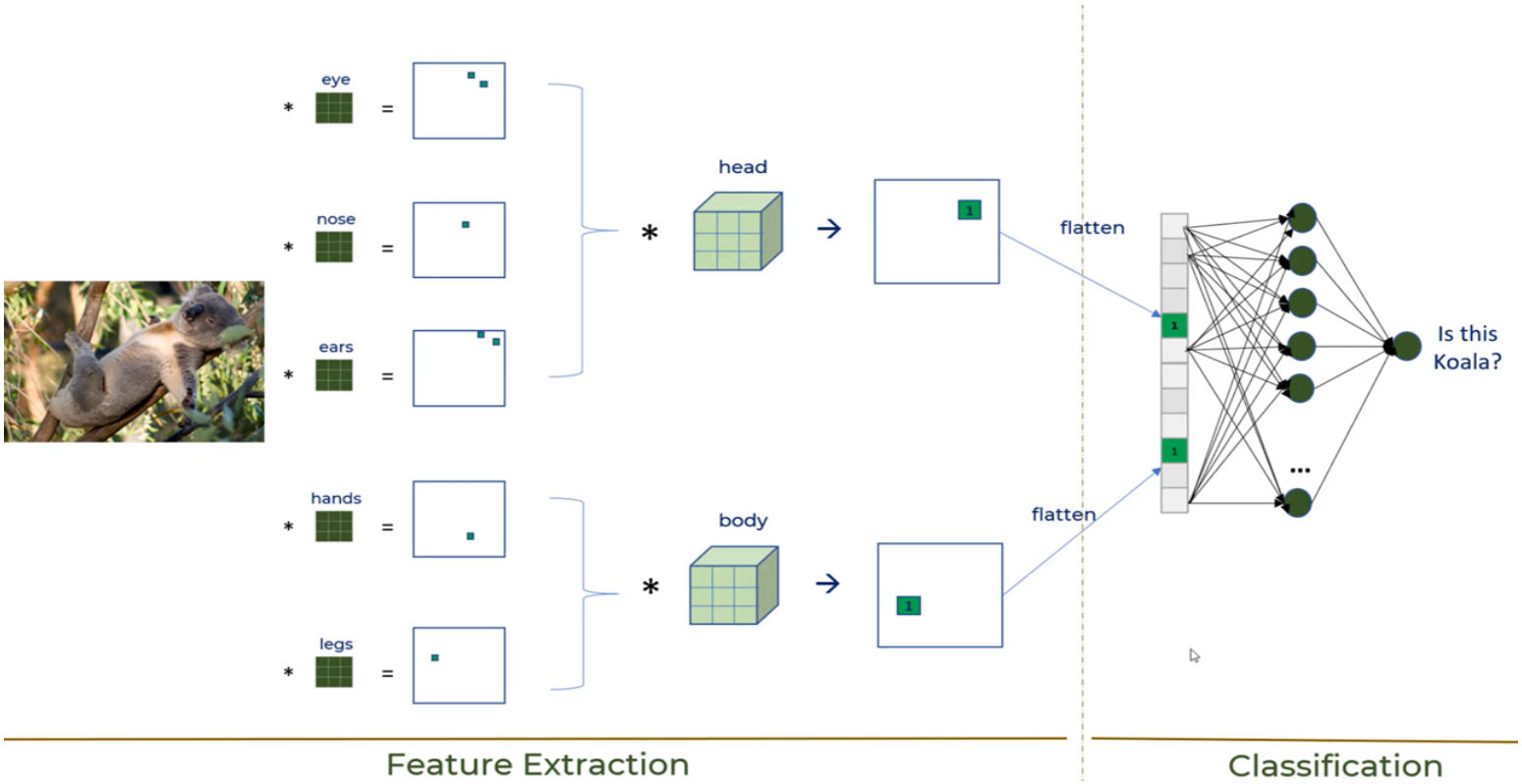
Hidden layer neurons = Let's say you keep it ~ 4 million

Weights between input and hidden layer = 6 mil * 4 mil
= 24 million

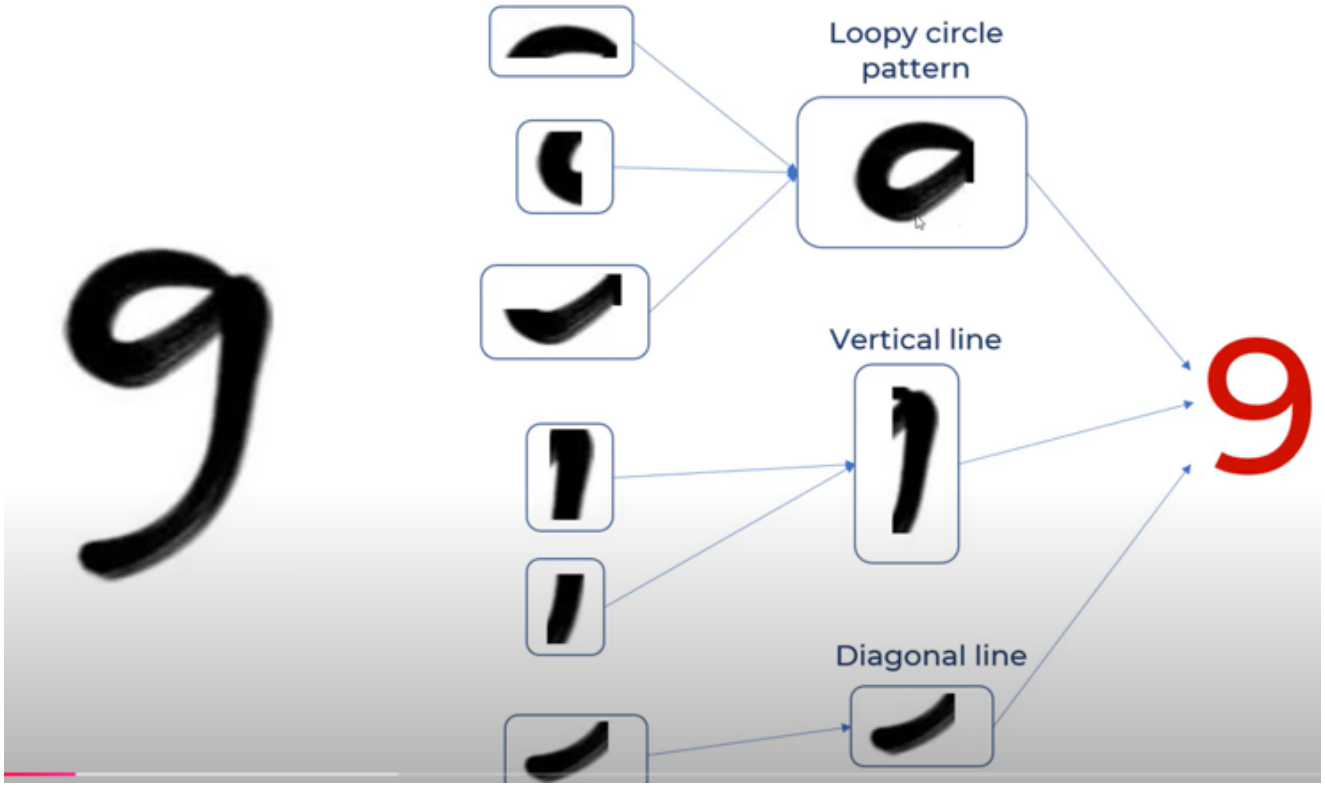
Feature extraction



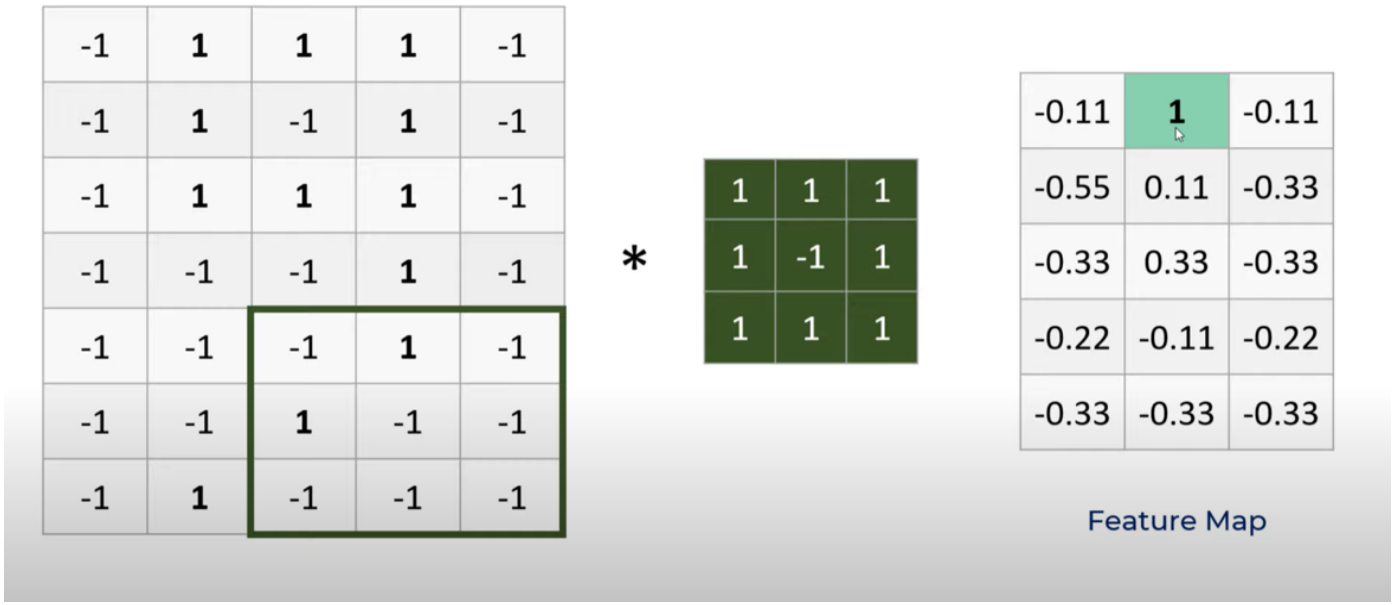
Classification



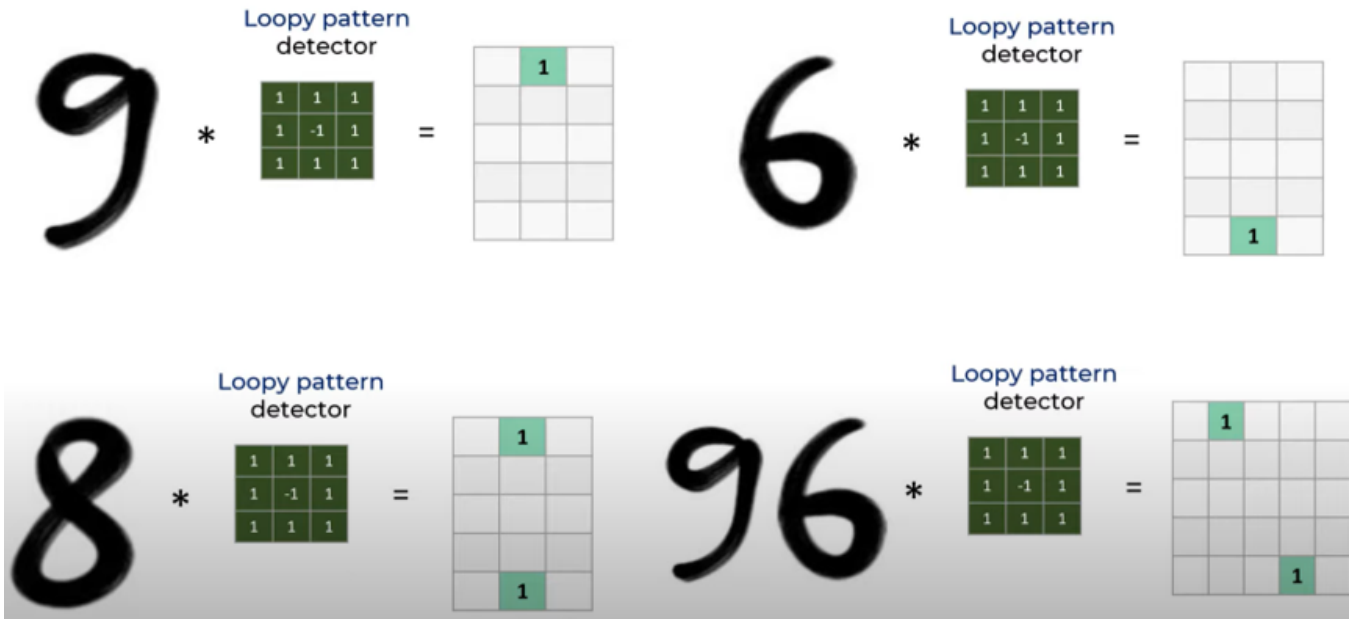
Features



Convolution operation

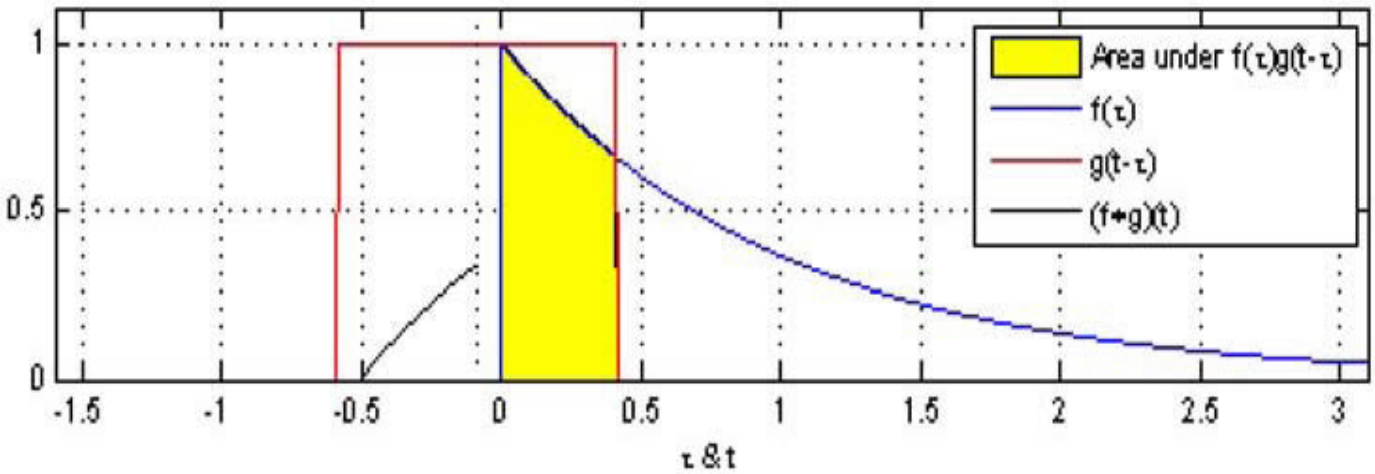


Feature extraction



Convolution in discrete-time

$$f * g = \sum_{k=-\infty}^{\infty} f(k)\, g(n-k)$$



Convolution operation

- Convolution produces the area amount of two overlap functions
- multiplication of a original function and its translation.

$$h(t) = f * g = \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau$$

- The output of a linear time-invariant system (LTI) is the convolution of the input signal and the system (the impulse response) gives
- In probability theory, the probability distribution of the **sum** of two independent random variables is the convolution of their individual distributions.

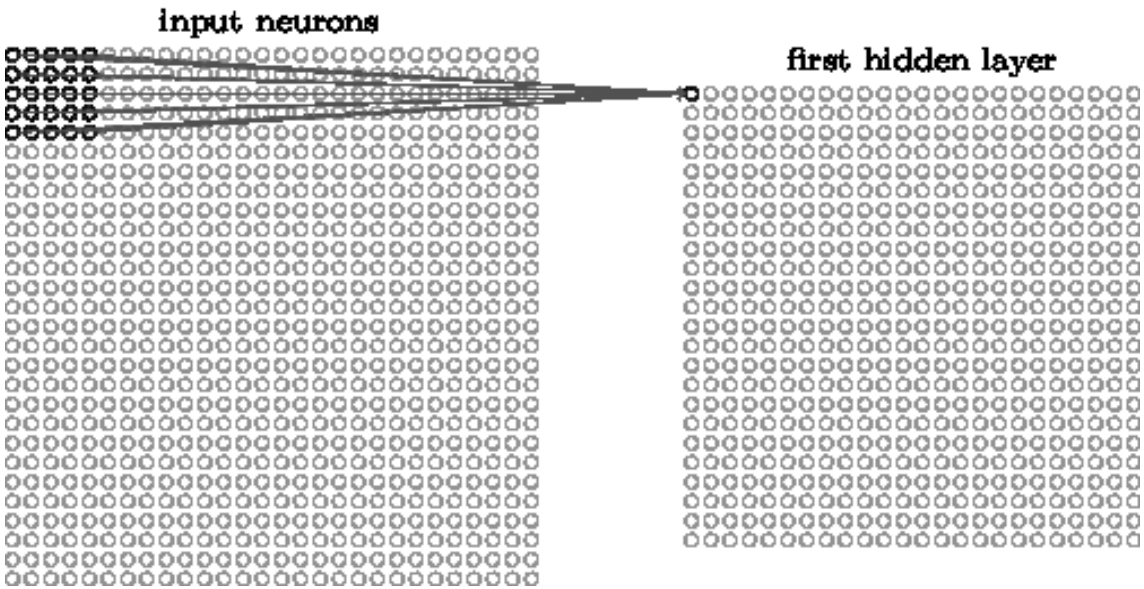
The distribution of the sum $Z = X + Y$ of two independent discrete variables is

$$P(Z = z) = \sum P(X = k)\, P(y = z - k)$$
$$h(z) = \int_{-\infty}^{\infty} f(\tau)\, g(z-\tau)\, d\tau$$

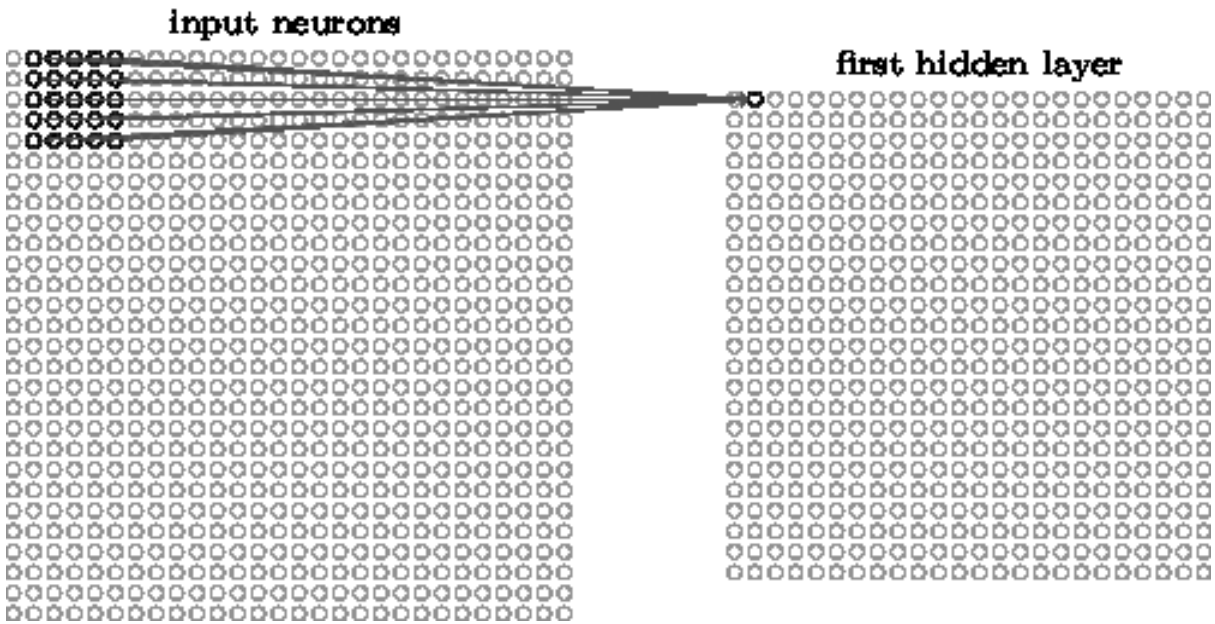
where$f(t)$ and $g(t)$ are density functions

Convolution: Local receptive fields

Every input pixel are not connected to every hidden neuron. Only localized regions of the input image is sent to hidden layers.
A image with 28×28 pixels, only 5×5 region is activated, this region is called the "local receptive field" for the hidden neuron.



Convolution



Hidden layer has $28 - 5 + 1 = 24$, 24×24 pixels, when step length is 1.
We can also move the local receptive field 2 pixels (step is 2)

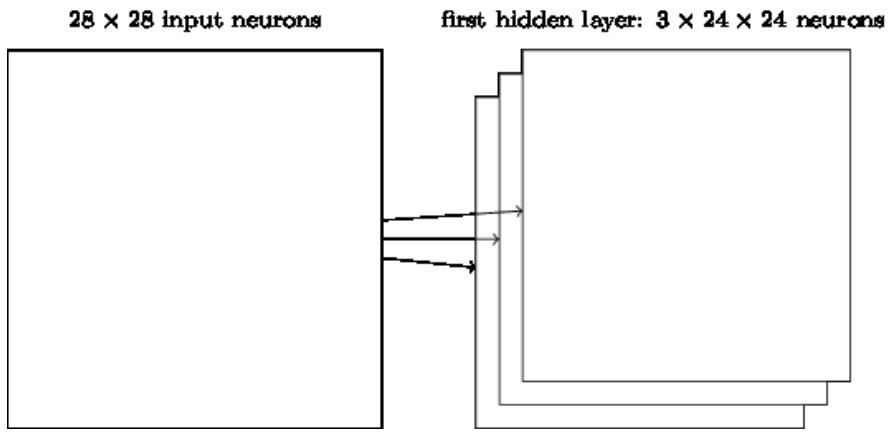
- The 5×5 region is called: "filter", or neuron, or kernel
- the region which the filter passed called "receptive field"
- The values of the elements of the filter called "weights"
- 24×24 pixels are called "activation map", or feature map

Convolution

The values of hidden neurons are

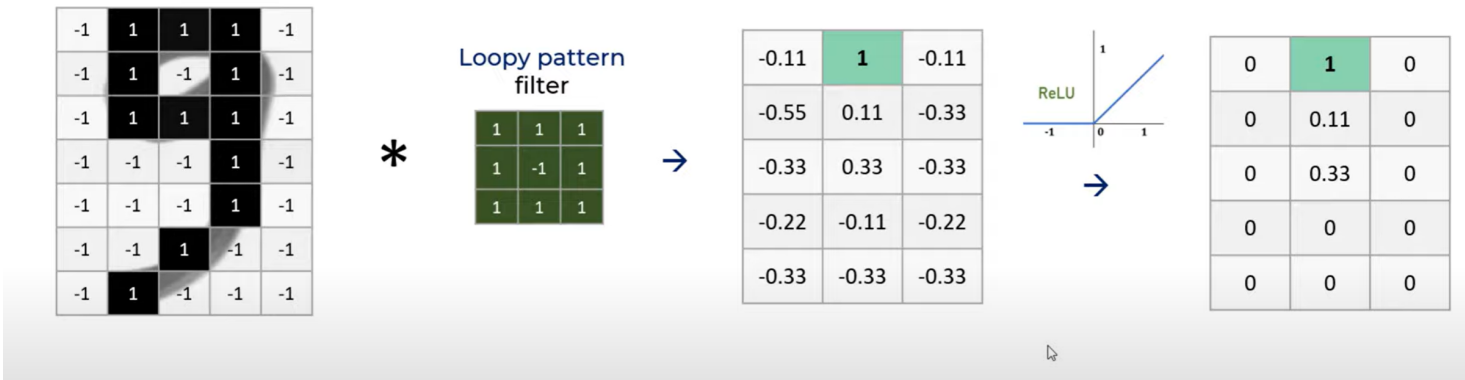
$$y = b + \sum_{i=1}^{5} \sum_{j=1}^{5} w_{i,j} x_{i,j}$$

where b is the shared value for the bias, $w_{i,j}$ is a 5×5 array of shared weights, $x_{i,j}$ denotes the input activation at position i, j
The convolution is regarded as local features extraction. If we need more than one feature map, we need several different filters to convolute the image.



Training parameter:

Activation function: ReLU



Activation function: ReLU

$$y = \phi\left(b + \sum_{i=1}^{5}\sum_{j=1}^{5} w_{i,j}x_{i,j}\right)$$

where ϕ is the neural activation function - perhaps the sigmoid function or tanh.

Rectified linear unit (ReLU)

$$y = \max\left[0, \left(b + \sum_{i=1}^{5}\sum_{j=1}^{5} w_{i,j}x_{i,j}\right)\right]$$

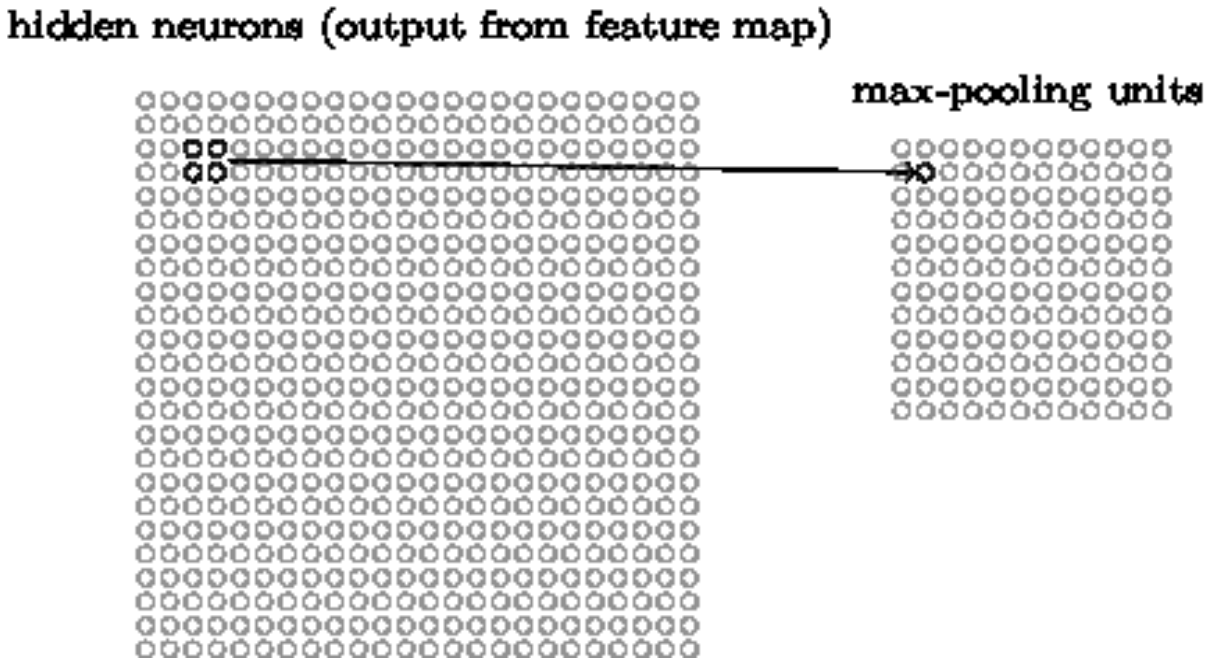
This activation function enables better training of deeper networks, the most popular activation function for deep neural networks.

Convolutional networks are well adapted to the translation invariance of images:

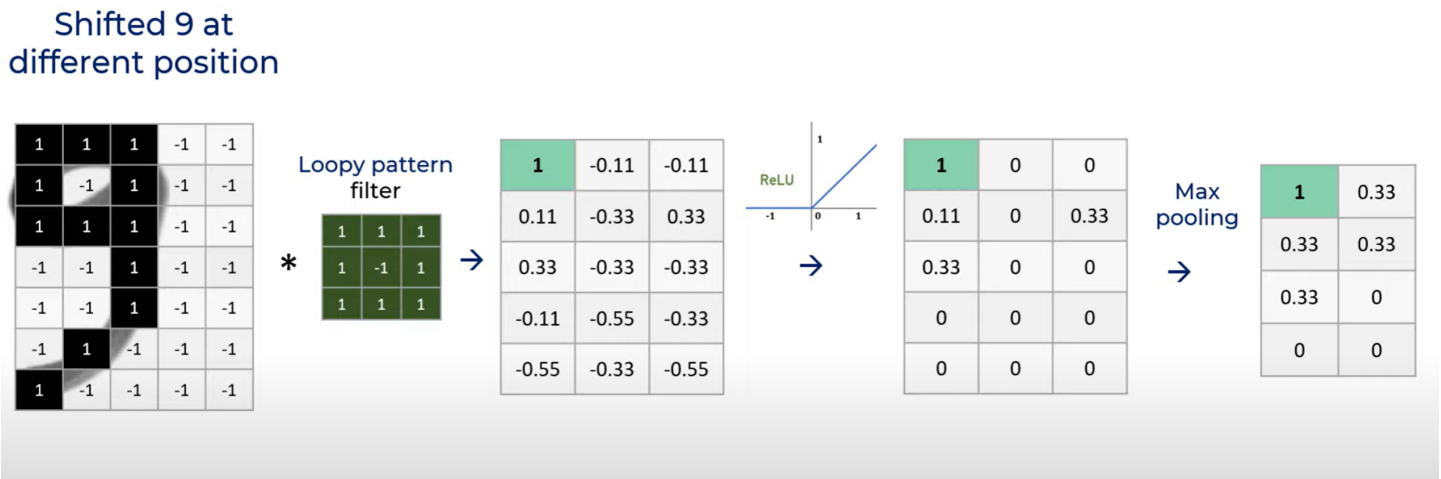
- All the neurons in the first hidden layer detect exactly the same feature
- The weights and bias correspond a particular local receptive field,

Polling

Simplify the information from the convolutional layer. It is a condensed feature map



Polling

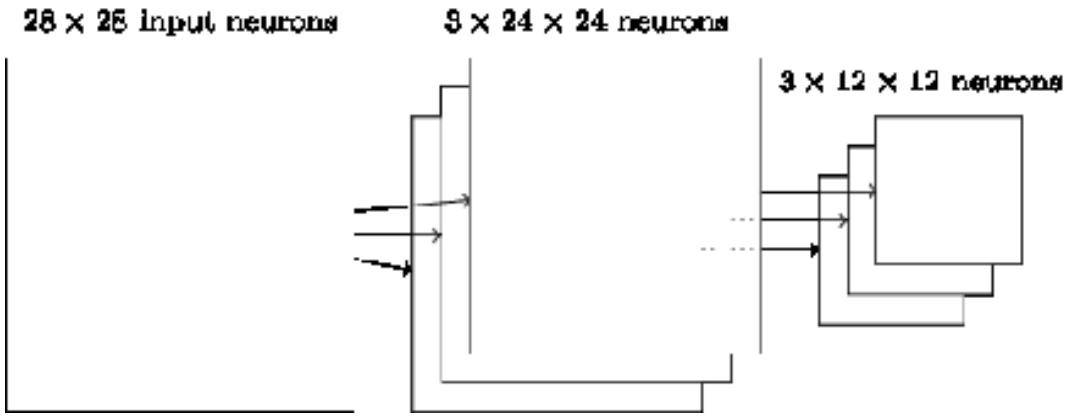

Shifted 9 at different position
Loopy pattern filter
ReLU
Max pooling

Polling

Each unit in the pooling layer summarizes a region of 2×2 in the convolutional layer. Many fewer pooled features, and so this helps reduce the number of parameters needed in later layers. $24 \times 24 \longrightarrow 12 \times 12$
Operation of the "summarization"

- max-pooling: the maximum of the 2×2 input region: where a given feature is found, then throws away the exact positional information. Once a feature has been found, its exact location isn't as important
- L_2 pooling: square root of the sum of the 2×2 input region: is a way of condensing information from the convolutional layer.
- average: $\frac{x_i}{\max(x_i)}$

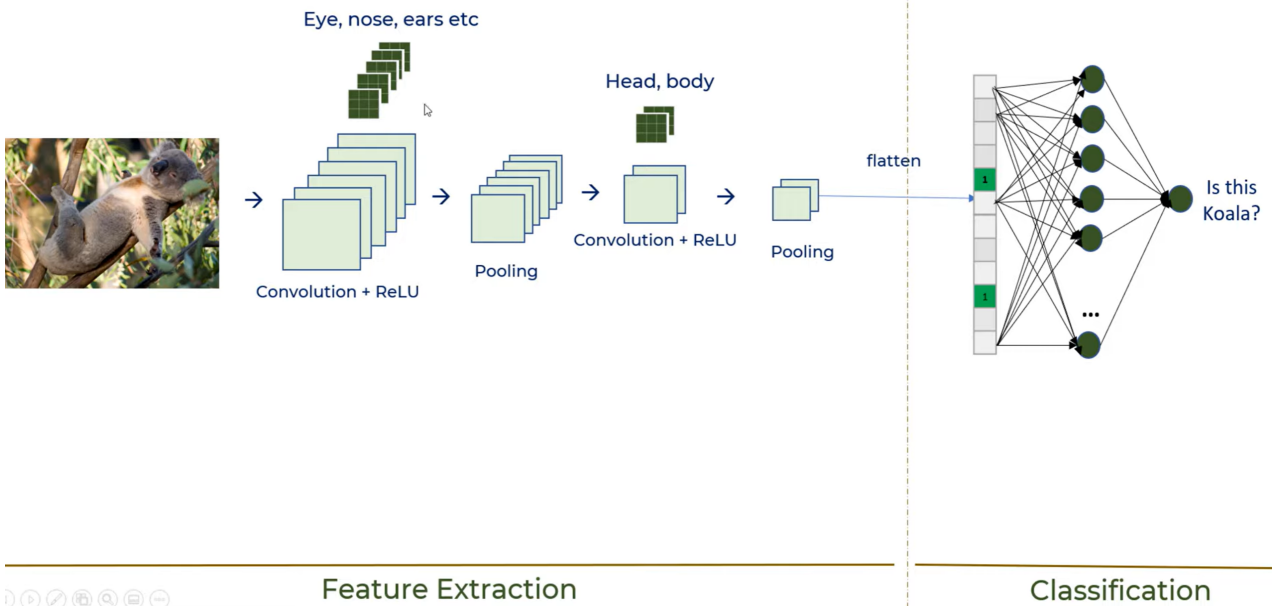
Fully-connected layer



If we want to classify 10 digits, they are 10 output neurons.
This layer connects every neuron from pooling layer to every one of the 10 output neurons.
There are $12 \times 12 \times 3 \times 10 = 4,320$ weights

CNN-structure

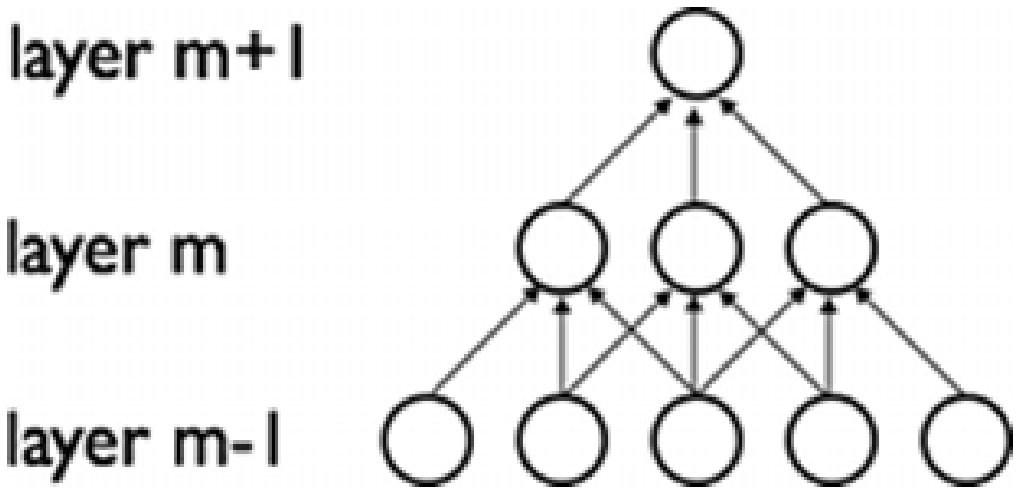
[Convolution ReLU $\longrightarrow$ Pooling (Sub-sampling) $\longrightarrow$ Convolution ReLU $\longrightarrow$ Poo
[Full connection]



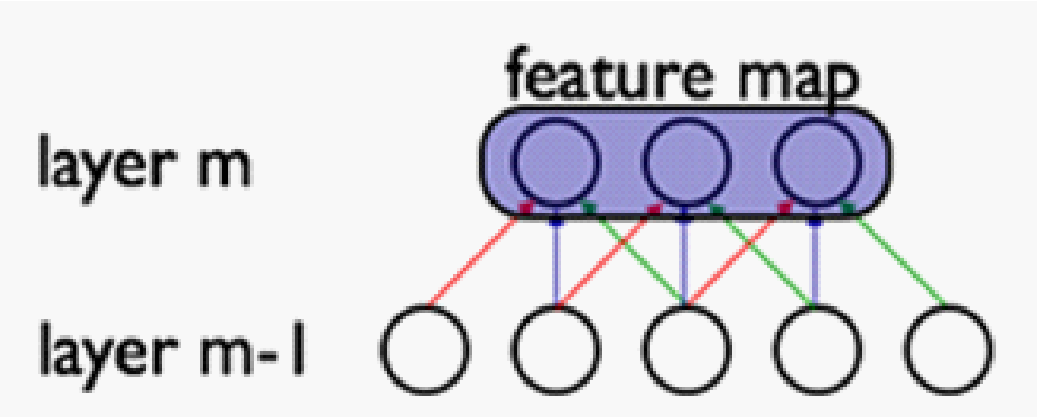
Properties of CNN

1. local receptive fields
2. shared weights
3. pooling

Properties of CNN: sparse Connectivity - local receptive fields



Properties of CNN: Shared weights



Why does the convolutional neural network have higher accuracy

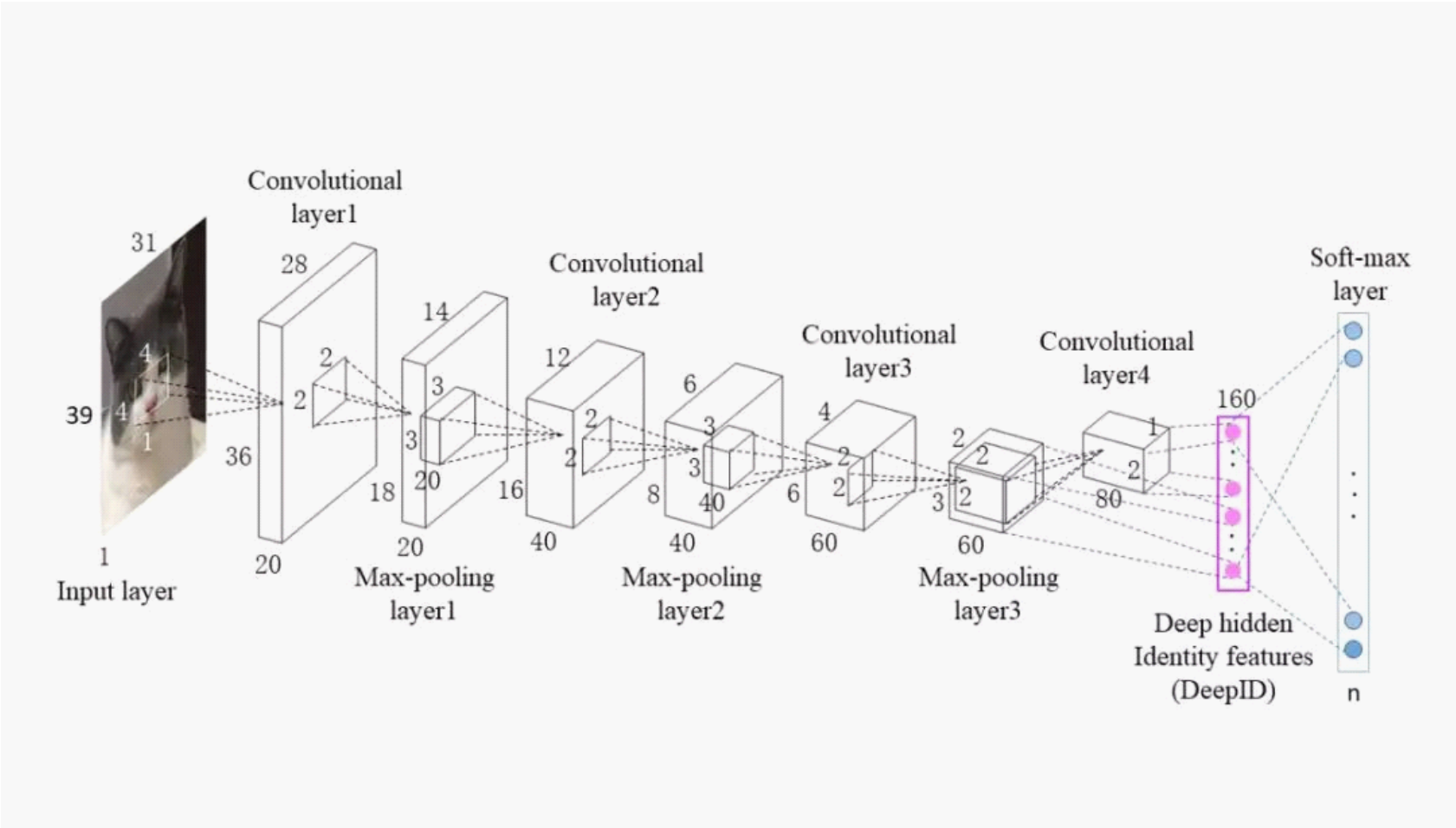
Machine learning is largely about two things

- Optimization: How you fit your model to the data
- Trade-offs: Precision/recall; Bias/variance;Powerful/interpretable;Complexity/training speed/data needed

Advantage of CNN

- CNNs utilize the spatial information, weight sharing to reduce number of parameters
- Each convolutional kernel scans the entire input feature map to produce an output map: a feature is the same no matter where in the receptive field that feature is located. This means they learn more meaningful features
- Pooling builds up tolerance to severe distortions, such as lens distortions, camera view angle. This pooling takes place at multiple levels, the low-level features are allowed to shift. This makes the CNNs very robust.
- Context: Spatial pooling also makes the receptive fields larger and hence context gets to be increasingly considered after each pooling operation. Hence at high-level layers, the receptive fields will include a huge area of the input stimuli such as an image. It is a well known fact that context improves performance in many recognition tasks.
- Hierarchical feature learning.

Training


Convolutional layer1
Convolutional layer2
Convolutional layer3
Convolutional layer4
Soft-max layer
Max-pooling layer1
Max-pooling layer2
Max-pooling layer3
Input layer
Deep hidden Identity features (DeepID)
n

Soft-max

$$y = \phi\left(Wx\right), \qquad \phi\left(z\right) = \frac{e^{z_i}}{\sum e^{z_i}}$$

$y = \phi\left(z\right)$ is probability of z_i. The loss function of soft-max (Cross Entropy) is

$$J = \frac{1}{n}\sum^{n}\left[y_i \log p_i + \left(1 - y_i\right)\log\left(1 - p_i\right)\right]$$

where p_i is the probability of y_i

Training process

1. Full connection layer
2. Convolution layer
3. Polling layer

Training error

If the pattern $x(k)$ belongs to the class i, for simple case the class is $y_i(k)$. The $i-$th element of the output $y_i(k)$, the training number is $k = 1, \cdots, N$

$$L = \sum_{i=1}^{10} \sum_{k=1}^{N} \left[y_i(k) - \hat{y}_i(k) \right]^2$$

$y(k)$ depends on the choice of the output activation function (soft-max). For the individual errors on each pattern i

$$J_i = \sum_{k=1}^{N} \left[y_i(k) - \hat{y}_i(k) \right]^2$$

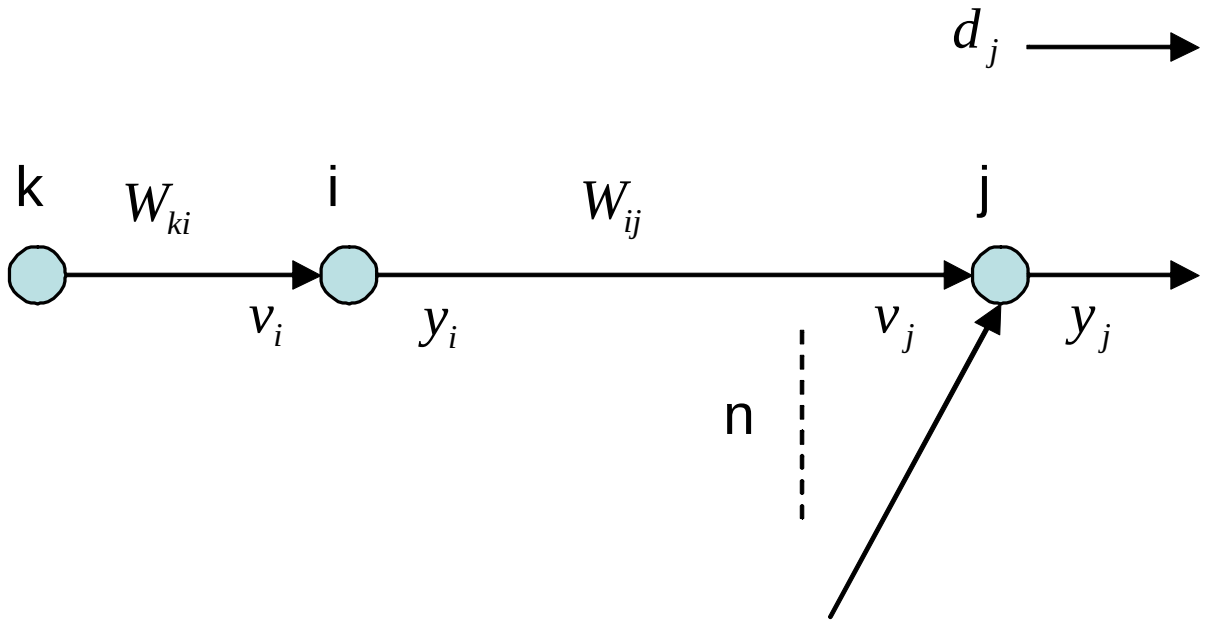
Full connection layer training

Gradient for "Full connection"

$$\hat{y}(k) = W\varphi(x(k))$$

$$W(k+1) = W(k) - \eta \frac{\partial J}{\partial W}$$

Full connection layer training

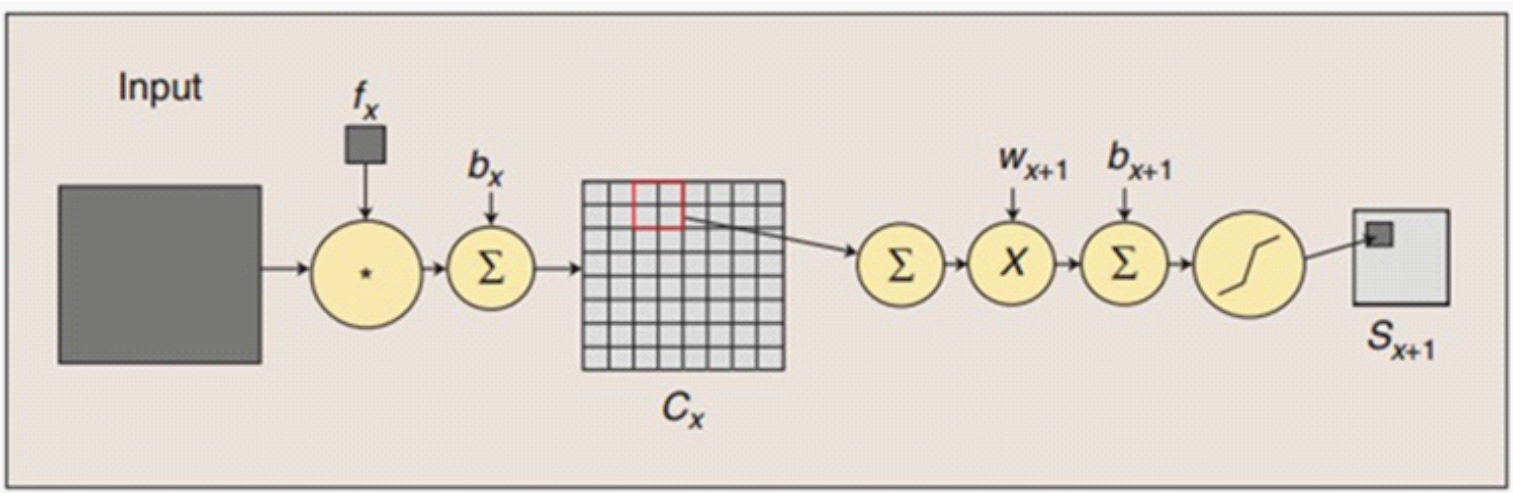


$$w_{ki}(k+1) = w_{ki}(k) - \eta e_i y_k$$

where

$$e(k) = y_j(k) - d_j(k), \qquad e_i = e_j W_{ij} \phi_i'$$

Convolution layer: feedforward

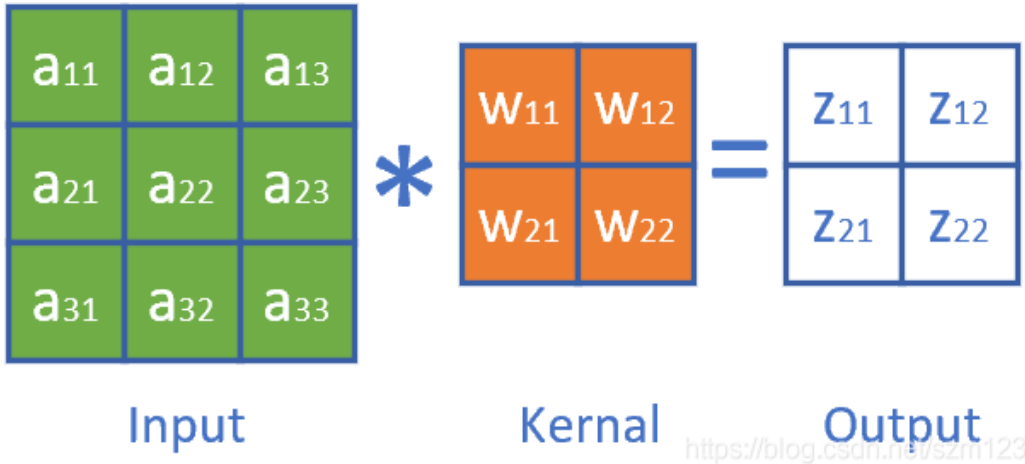


Feedforward calculation (feature map),

$$y = \phi\left(b + \sum_{i=1}^{5}\sum_{j=1}^{5} w_{i,j}x_{i,j}\right) = \phi\left(b + \sum_{i,j\in M} w_{i,j}x_{i,j}\right) = WX + b$$

where ϕ is the active function (ReLU), $w_{i,j}$ b also called kernel and bias.

Convolution layer: feedforward

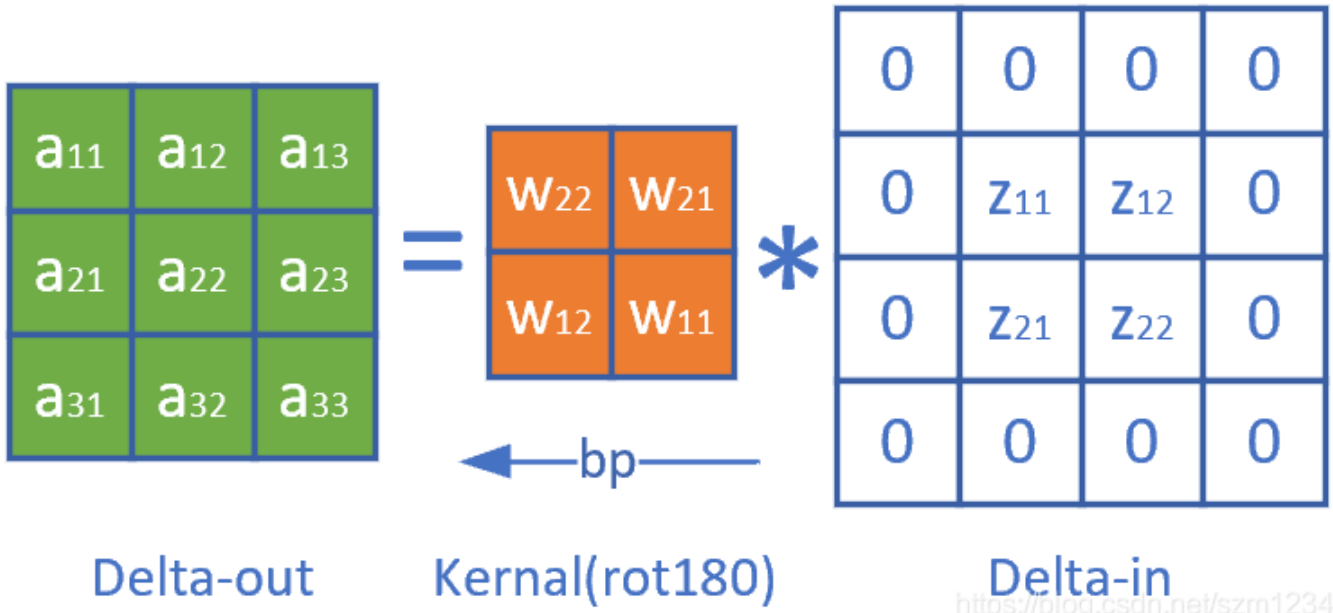


$$z_{11} = w_{11}a_{11} + w_{12}a_{12} + w_{21}a_{21} + w_{22}a_{22} + b$$
$$z_{12} = w_{11}a_{12} + w_{12}a_{13} + w_{21}a_{22} + w_{22}a_{23} + b$$

some gradients are

$$\frac{\partial J}{\partial a_{11}} = \frac{\partial J}{\partial z_{11}}\frac{\partial z_{11}}{\partial a_{11}} = e_{z_{11}}w_{11}$$
$$\frac{\partial J}{\partial a_{12}} = \frac{\partial J}{\partial z_{11}}\frac{\partial z_{11}}{\partial a_{11}} + \frac{\partial J}{\partial z_{12}}\frac{\partial z_{12}}{\partial a_{11}} = e_{z_{11}}w_{12} + e_{z_{12}}w_{11}$$
$$\frac{\partial J}{\partial a_{22}} = \frac{\partial J}{\partial z_{11}}\frac{\partial z_{11}}{\partial v} + \frac{\partial J}{\partial z_{12}}\frac{\partial z_{12}}{\partial a_{22}} + \frac{\partial J}{\partial z_{21}}\frac{\partial z_{21}}{\partial a_{22}} + \frac{\partial J}{\partial z_{22}}\frac{\partial z_{212}}{\partial a_{22}}$$

Convolution layer training: BP



$$\frac{\partial J}{\partial a_{22}} = e_{z_{11}} w_{22} + e_{z_{12}} w_{21} + e_{z_{21}} w_{12} + e_{z_{22}} w_{11}$$
$$\rightarrow z_{11} w_{22} + z_{12} w_{21} + z_{21} w_{12} + z_{22} w_{11}$$

The error is propagated as: the original kernel W is rotated by 180 degrees, and then do the convolution operation.
In the corner the padding is needed (fill zero in corners)

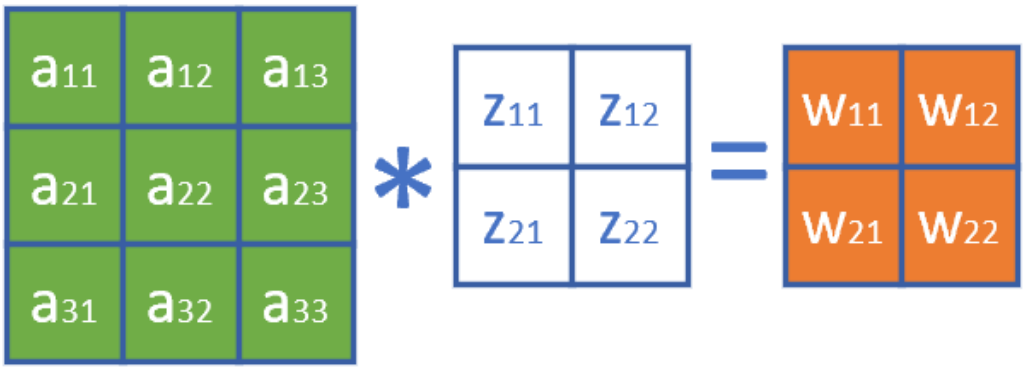
Convolution layer training: BP

The error in the convolution layer is propagated as

$$e_{out} = e_{in} * W^{rot(180)}$$

$$W(k+1) = W(k) - \eta \frac{\partial J}{\partial W} = W(k) - \eta A e_{out}$$

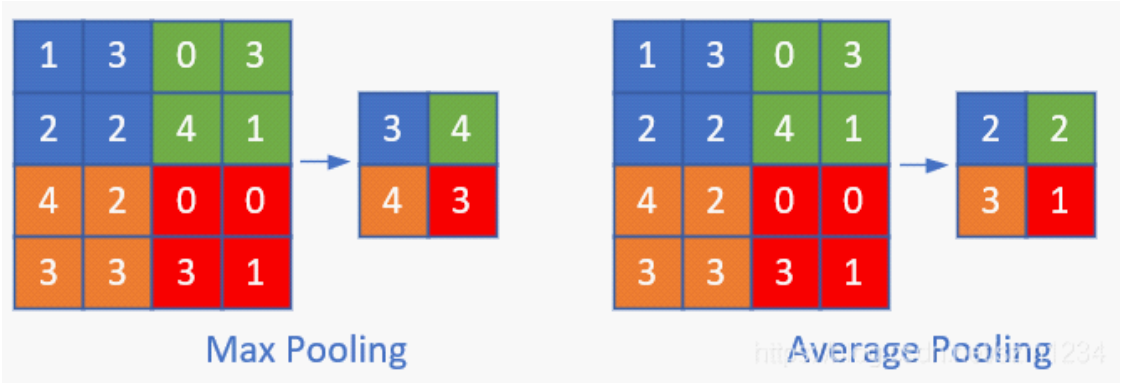
It can be regarded as



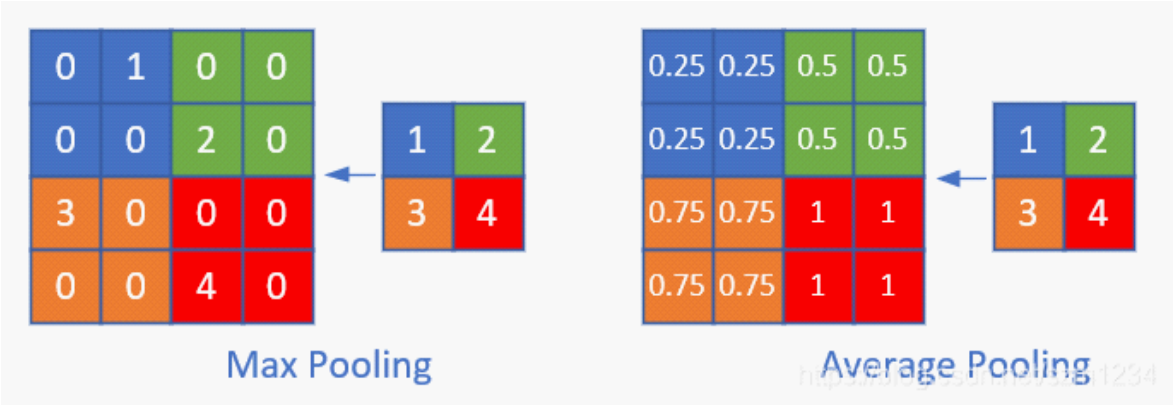
So the convolution layers updated as

$$W(k+1) = W(k) - \eta A e_{in} * W^{rot(180)}$$

Polling layer training



Maximum pooling, the error will be propagated to the position of the original maximum value, and the other positions are all 0.
Average pooling, the error are propagated to all position with the averaged values.



Polling layer training

Max pooling: if

$$y = \max(a, b, c, d) = b$$

$$e_{out,a} = 0, \qquad e_{out,b} = e_{in},, \qquad e_{out,c} = 0,, \qquad e_{out,d} = 0$$

Average pooling: if

$$y = \frac{1}{4}(a + b + c + d)$$

$$e_{out,a} = \frac{1}{4}e_{in}, \qquad e_{out,b} = \frac{1}{4}e_{in},, \qquad e_{out,c} = \frac{1}{4}e_{in},, \qquad e_{out,d} = \frac{1}{4}e_{in}$$

Up-sampling of average pooling is

$$e_{out} = up\left[e_{in}\right], \quad up(z) = z \otimes 1_{n \times m}$$

where $\otimes$ is the Kronecker product.
No parameter gradient, only error propagation

Application of CNN: nonlinear system identification

Consider the following unknown discrete-time nonlinear system

$$y(k) = \Gamma\left[x\left(k\right)\right]$$

where

$$x\left(k\right) = \left[y\left(k-1\right), \cdots y\left(k-n_y\right), u\left(k\right), \cdots u\left(k-n_u\right)\right]^T \tag{1}$$

Simulation model (parallel)

$$\hat{y}(k) = F[u\left(k\right), \cdots, u\left(k-n\right)] \tag{2}$$

also

$$\hat{y}(k) = N[\hat{y}(k-1), \cdots, \hat{y}(k-m), u\left(k\right), \cdots, u\left(k-n\right)] \tag{3}$$

Prediction model (series-parallel)

$$\hat{y}(k) = N[y\left(k-1\right), \cdots, y\left(k-m\right), u\left(k\right), \cdots, u\left(k-n\right)] \tag{4}$$

CNN structure

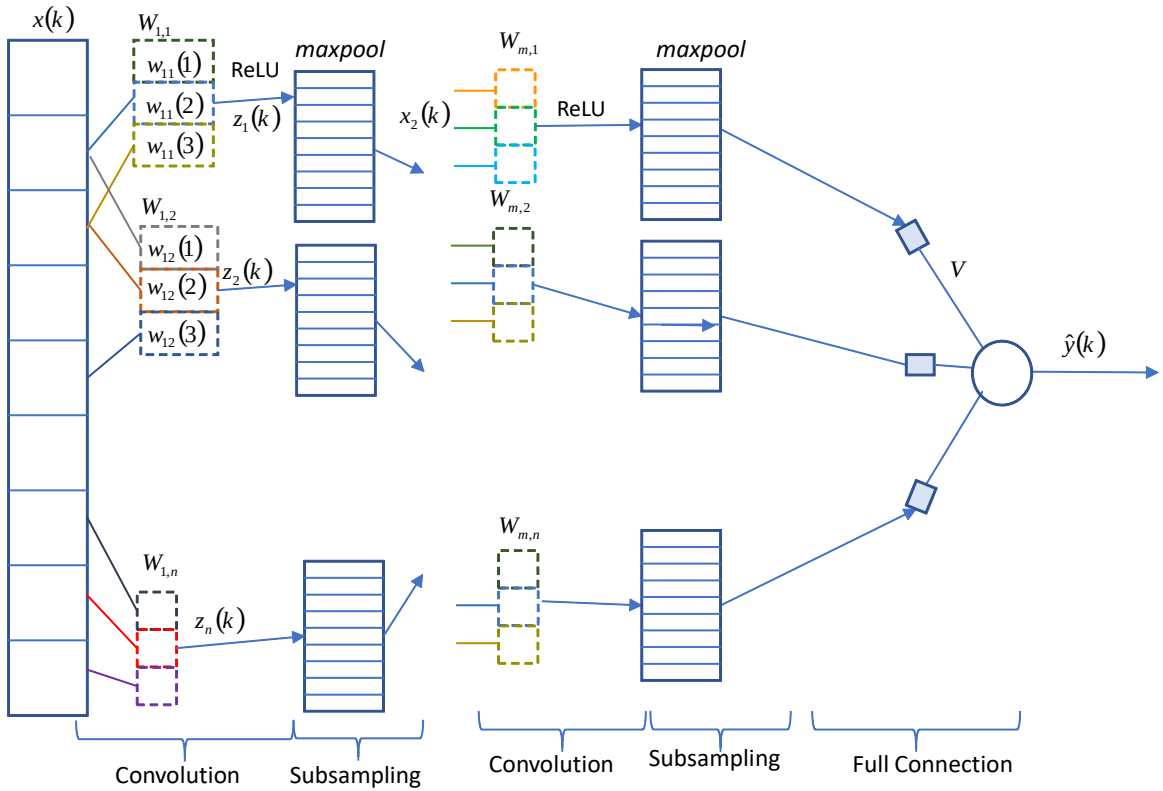


Figure: CNN model for system identification

CNN structure

$$\widehat{y}(k) = V\Phi[x(k)] \tag{5}$$

the convoluted output is

$$z = x * W_{ij} \quad zz_q = \sum_{k=1}^{p} x[p(q-1)+k]\, w_{ij}[k], \quad z = [zz_1 \cdots zz_{N-p}] \tag{6}$$

we use the rectified linear unit (ReLU)

$$Z(k) = \max[0, z(k)], \quad z = [z_1 \cdots z_n] \tag{7}$$

For system identification,

$$x_2(k) = \max_p [Z(k), s] \tag{8}$$

where s is the shrink parameter

Training

- In the pre-training, the weights of the convolutional filters W_{ij} are randomly assigned first, then we use a small dataset and the Backpropagation of CNN to update them.
- In the fine tuning, the weights of the convolutional filters are fixed as the values of the pre-training, we only update the weight V in the last layer.

Pre-training

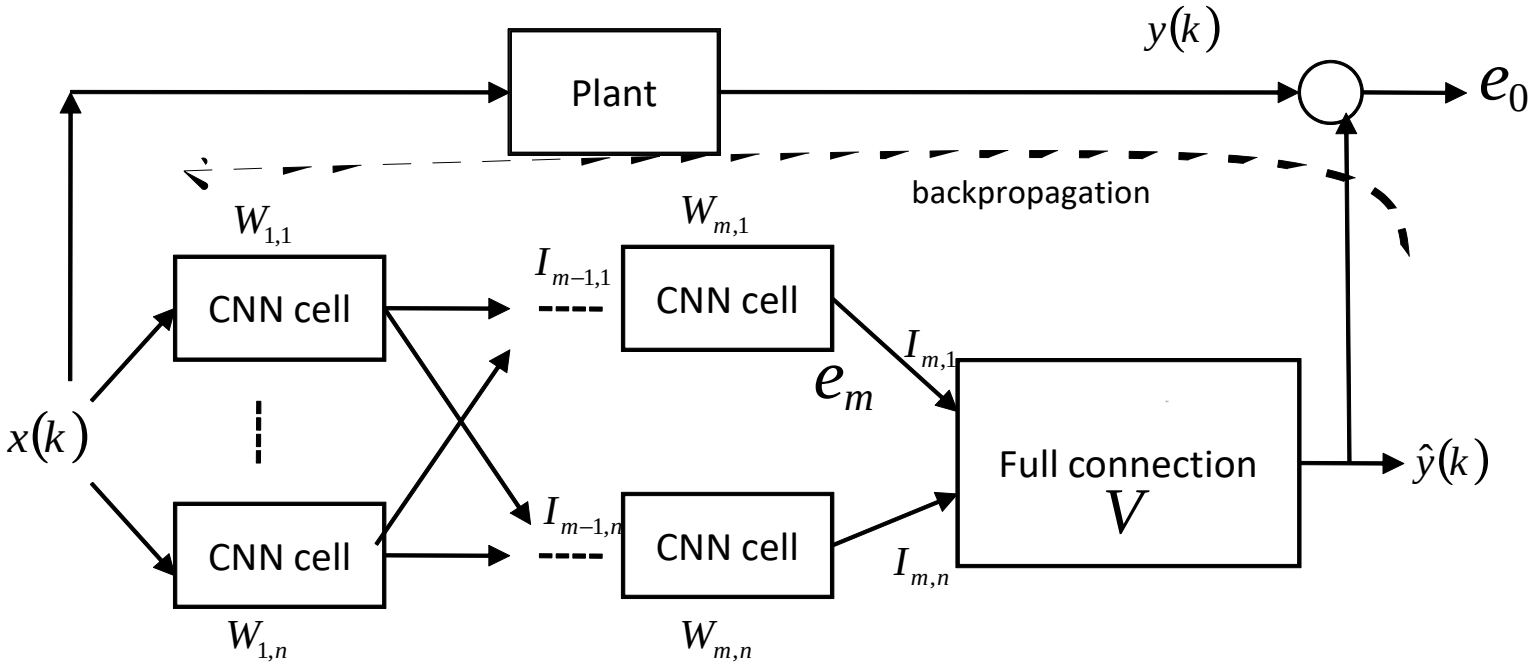


Figure: The scheme of the system identification with CNN

$$J = \frac{1}{2} e_0^2 \quad e_o = \hat{y} - y \tag{9}$$

$$V(k+1) = V(k) - \eta I_m(k) e_o(k) \tag{10}$$

Pre-training

The error is back-propagated to the layer m as

$$e_m(k) = e_o(k) \frac{\partial \sigma}{\partial t} V = e_o(k) V(k) \tag{11}$$

where σ is the active function of the full connection layer. Here we use linear function, $\frac{\partial \sigma}{\partial t} = I$.
For each CNN cell, we need to update the filter weight W_{ij} in (6)

$$W_{ij}^{(k+1)} = W_{ij}^{(k)} - \eta \frac{\partial J}{\partial W_{ij}}$$

where J is defined in (9).
For $m-$th CNN cell

$$W_{mj}^{(k+1)} = W_{mj}^{(k)} - \eta I_{m-1,j}(k) e_m(k)$$

Pre-training

In the max-pooling,

$$e_{mp}(k) = up\left[e_m(k)\right]$$

When the error e_{mp} goes through the convolution operation

$$e_{mc} = e_{mp} * W_{ij}$$

we use discrete-time de-convolution

$$e_{mc} = e_{mp} * \left[rot180\left(W_{ij}\right)\right]$$

Pre-training

$$e_{m-1} = up\left[e_m\left(k\right)\right] * \left[rot180\left(W_{ij}\right)\right] \tag{12}$$

For any CNN cell, the training law of the filters is

$$\begin{aligned} W_{ij}^{(k+1)} &= W_{ij}^{(k)} - \eta I_{i-1,j}\left(k\right) e_i\left(k\right) \\ e_i\left(k\right) &= up\left[e_{i+1}\left(k\right)\right] * \left[rot180\left(W_{ij}\right)\right] \end{aligned} \tag{13}$$

Fine tuning

The object to train the weight V is

$$J = \min_{V} \sum_{k} \|y(k) - \hat{y}(k)\|^2 \tag{14}$$

where

$$\widehat{y}\,(k) = V\Phi\left[x\,(k)\right]$$

The training data are $y\,(k)$ and $\Phi\left[k\right]$.

Fine tuning

Consider all data $k = 1 \cdots N$,

$$\begin{array}{l} \hat{Y} = \left[\begin{array}{ccc} V\Phi(1) & \cdots & V\Phi(N) \end{array} \right] = V\Psi \\ Y = \left[\begin{array}{ccc} V\Phi(1) + e_o(1) & \cdots & V\Phi(N) + e_o(N) \end{array} \right] \end{array} \tag{15}$$

where $e_o(k)$ is the modeling error which defined. (15) in matrix form is

$$Y = V\Psi + E \tag{16}$$

where $E = [e_o(1), \cdots, e_o(N)]$. (16) is a linear-in-parameter system.

Fine tuning

When

$$\hat{V} = Y\Psi^T \left(\Psi\Psi^T\right)^{-1} = Y\Psi^+ \tag{17}$$

where Ψ^+ is the pseudoinverse of Ψ

$$J = \min_V \sum_k \|y(k) - \hat{y}(k)\|^2 \tag{18}$$

arrives the minimum value.

CNN training

In each convolutional layer, there are n filters, the convoluted output is

$$z = x * W_{ij}, \quad z = [zz_1 \cdots zz_{N-p}], \quad zz_q = \sum_{k=1}^{p} x\left[p(q-1)+k\right] w_{ij}\left[k\right] \tag{19}$$

where p is the filter length, N is the length of $x\left(k\right)$,
$z\left(k\right) = \left[zz_1 \cdots zz_{N-p}\right]$ is the feature map generated in this layer.
Linear unit (ReLU) as

$$Z\left(k\right) = \max\left[0, z\left(k\right)\right], \quad z = \left[z_1 \cdots z_n\right] \tag{20}$$

We use the max-pooling method in the sub-sampling layer

$$x_2\left(k\right) = \max_{p}\left[Z\left(k\right), s\right] \tag{21}$$

where s is the shrink parameter, which depends on the layer, $x_2\left(k\right)$ is a new input for the second convolutional layer

Frequency domain

If we use discrete Fourier transform (DFT), the convolution operation in (6) can be transformed into multiplication,

$$\mathcal{F}(x * W) = \mathcal{F}(x) \times \mathcal{F}(W) = \mathcal{F}(x)\,\mathcal{F}(W) \tag{22}$$

where F represents the DFT for the corresponding vector, $*$ represents the convolution operation, $F(x)$ and $F(W)$ are vectors in the frequency domain, $\times$ represents element-wise product.

Optimal input for random filters

Theorem

The input data $x(k)$, $k = 1 \cdots N$, *pass through the filter* $W_{ij} = [w_{ij}(1), \cdots w_{ij}(p)]$, $p < N$. *There is the optimal input, which maximizes the CNN cell with the ReLU and the max-pooling as*

$$x^*(k) = \frac{\sqrt{2}}{N} \cos\left(\frac{2\pi f_0}{N} k + \phi\right) \tag{23}$$

where f_0 *is the optimal frequency,* ϕ *is the phase.* .

- This theorem shows that for any inputs (random or determined), it corresponds to the optimal filter. The frequency of the optimum input is the frequency of maximum magnitude in the filter.
- Any random filter containing some frequencies of moderate magnitude can generate the maximum magnitude of the best input frequency.