

RETELELOR NEURONALE ARTIFICIALE - RETELELOR NEURONALE CONVOLUTIONALE

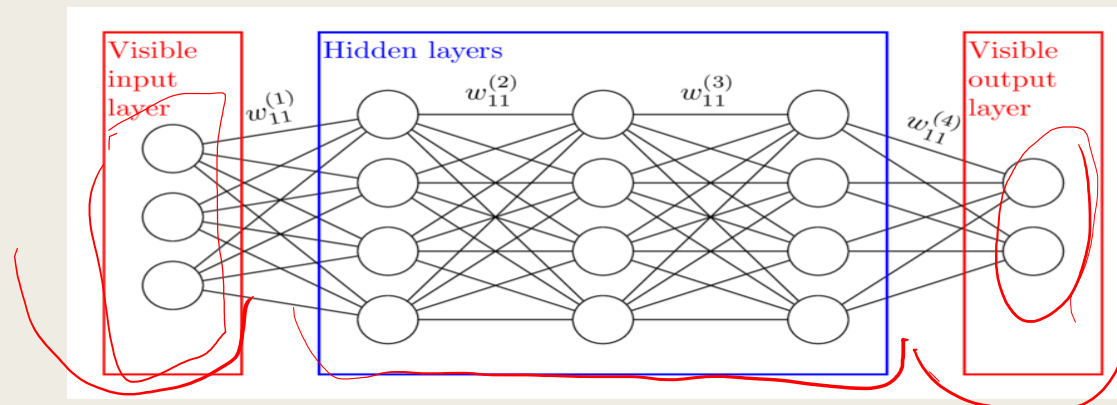
Sl. Dr. Ing. Camelia FLOREA
(Camelia.Florea@com.utcluj.ro)

Retelelor Neuronale Artificiale

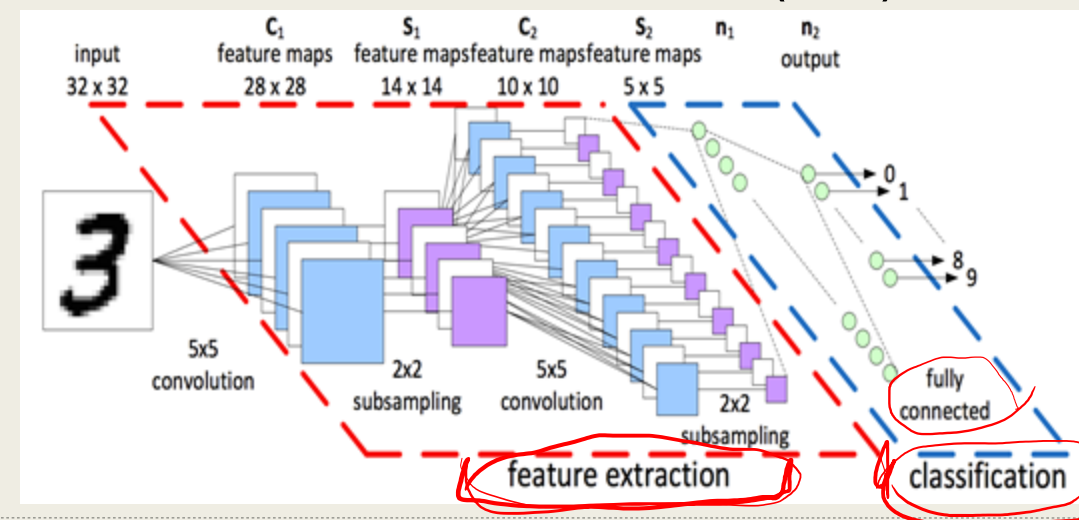
■ Caracteristici "împrumutate" de la creier:

- **capacitatea de a învăța**
 - învățare din exemple (antrenare cu seturi mari de date)
- **capacitatea de a generaliza**
 - pot da răspunsuri corecte pentru intrări ușor diferite de cele cu care au fost antrenate
- **capacitatea de a sintetiza**
 - pot da răspunsuri corecte pentru intrări afectate de zgomot/ imprecise/ parțiale

Artificial Neural Networks (ANN)



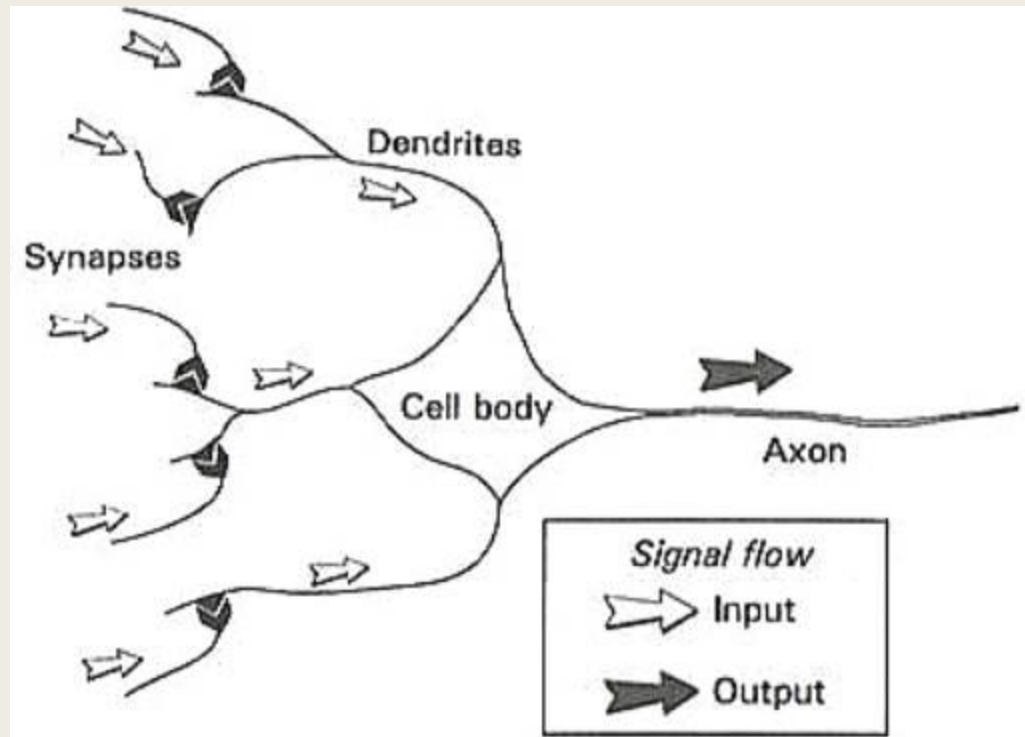
Convolutional Neural Networks (CNN)



Artificial Neuron

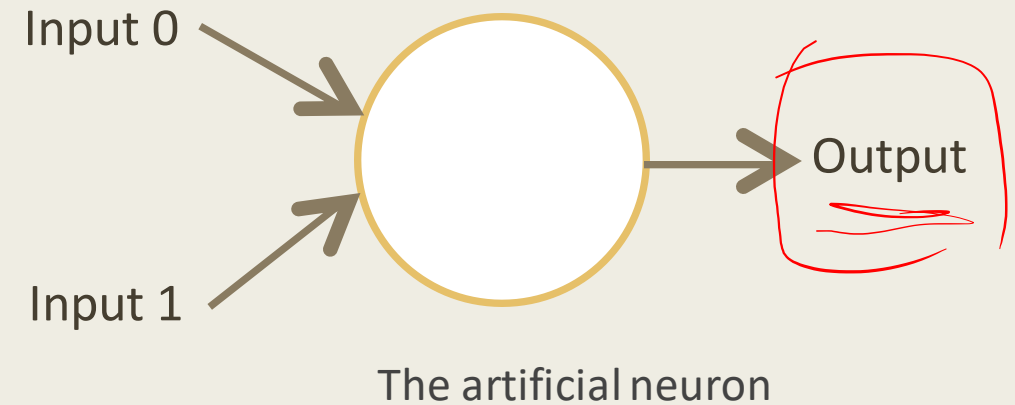
Retelele Neuronale Artificiale – au bazele in biologie

Un neuron biologic:



The biological neuron

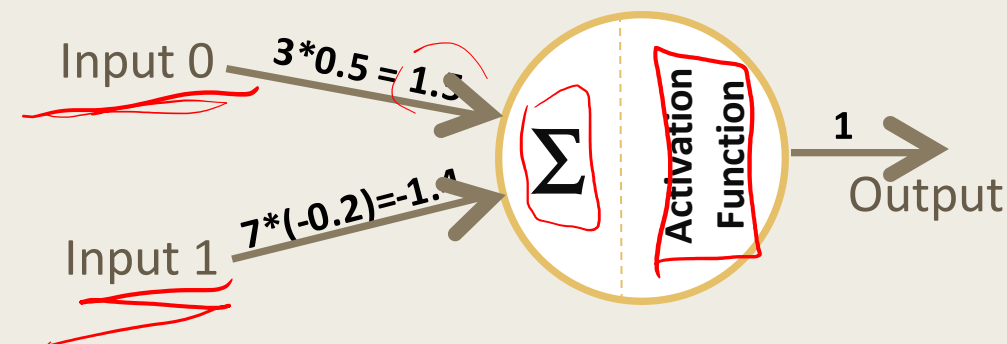
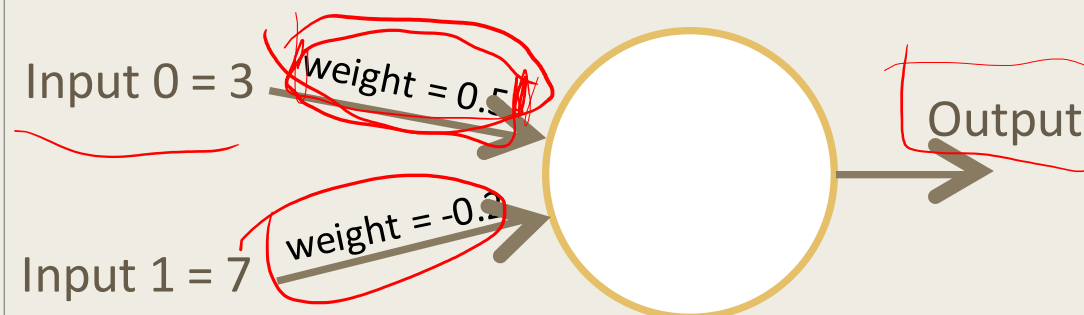
Un neuron artificial:



- Modelul simplu este cunoscut ca si perceptron.
- are intrari si iesiri similar unui neuron biologic!

Exemplu simplu de functionare

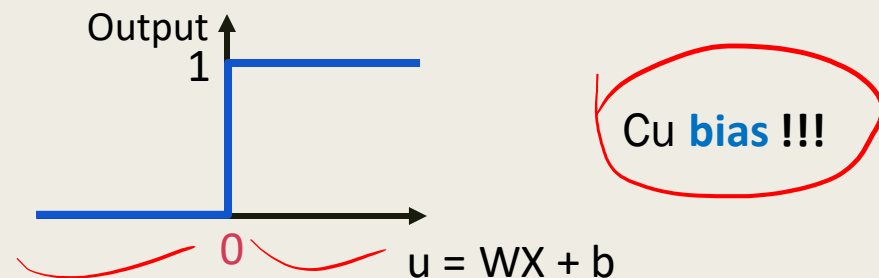
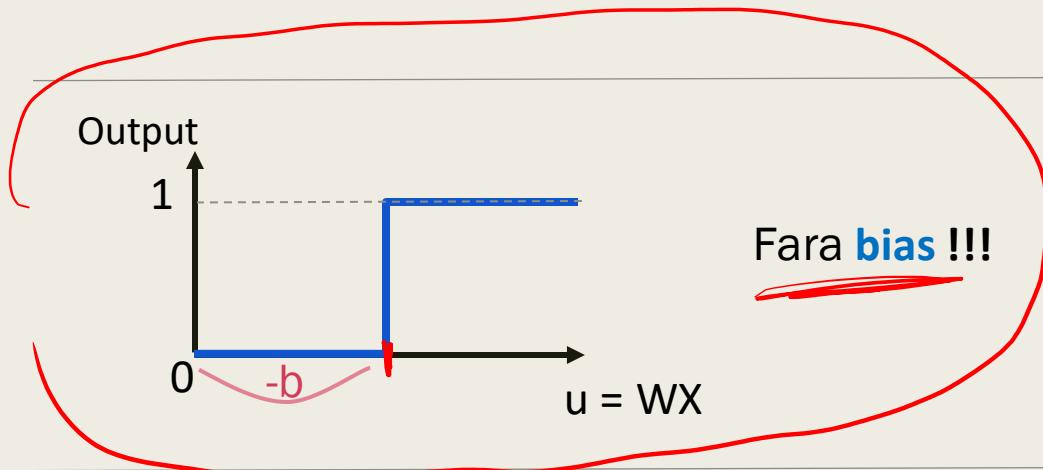
- Consideram un perceptron simplu
 - cu doua intrari si o iesire
 - fiecare intrare are o pondere
- Intrarile sunt valorile trasaturilor
 - Se vor inmulti cu ponderile aferente
 - ponderile initial vor lua valori aleatoare
- Rezultatul inmultirilor insumate vor trece printr-o functie de activare
 - Exista mai multe variante a funtiei de activare (vom dicuta mai incolo in curs)! - aici consideram o functie de activare simpla:
 - Daca suma intrarilor este pozitiva - se returneaza 1
 - Daca suma intrarilor este negativa - se returneaza 0
 - In cazul acesta $1.5 + (-1.4) = 0.1$
 - => la iesire avem 1



Termenul bias

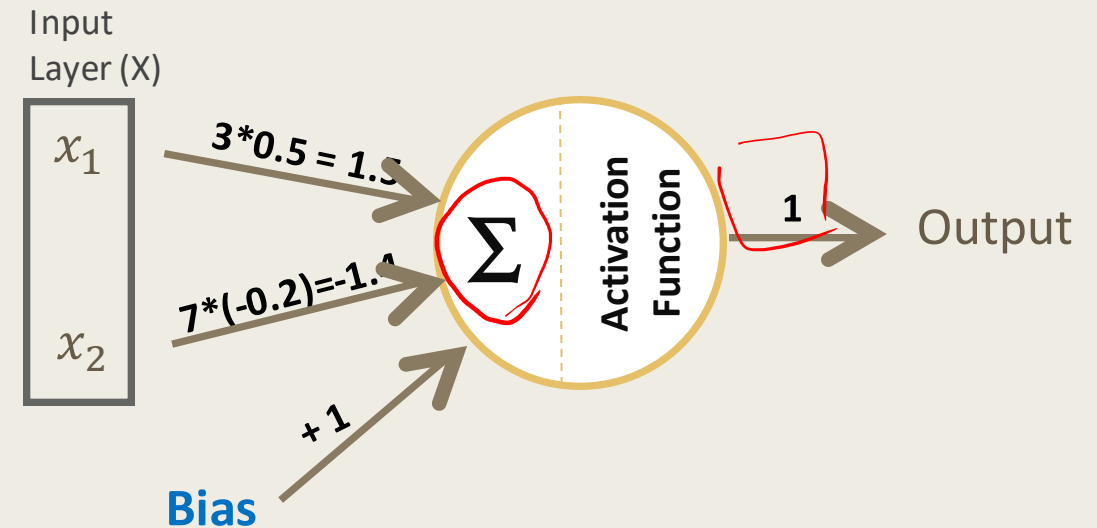
■ Termenul **bias** (în engleză) este cunoscut drept:

- *deplasare sau*
- *distanța față de origine a suprafeței de decizie.*



■ In acest caz (exemplu dat),

- *conderam bias = 1*



Formulare matematica

■ Formulare/descriere matematica perceptron

- 1 perceptron (neuron)
- M intrari (generalizare, cu mai mult de doua intrari)
- 1 iesire

$$u = \sum_{i=0}^M w_i x_i + b$$

$$y = f(u)$$

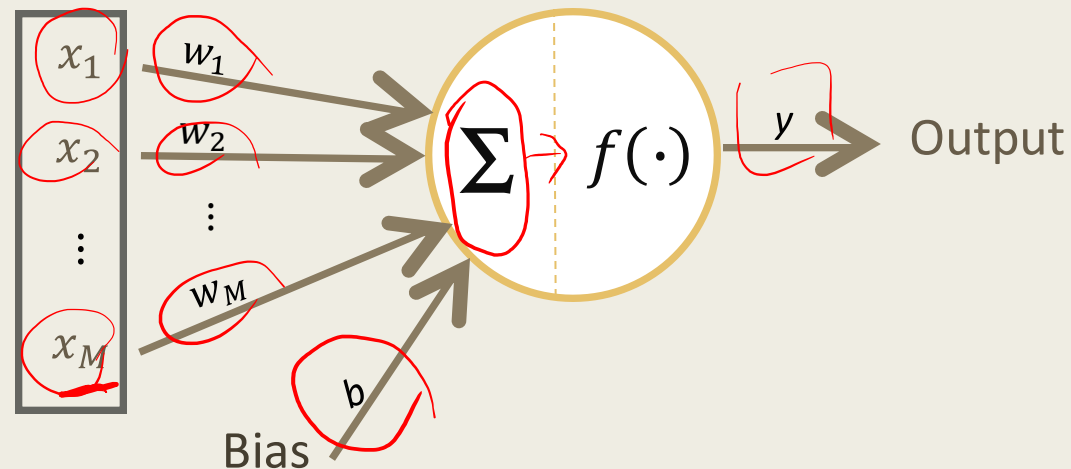
Forma matriciala:

$$y = f(WX + b)$$

$$W = [w_1 \quad w_2 \quad \dots \quad w_M]$$

$$X = [x_1 \quad x_2 \quad \dots \quad x_M]^T$$

Input Layer (X)



x_i – intrarile, (X vectorul de trasaturi,
 x_i trasatura i ,
 M – nr. trasaturi/ intrari)

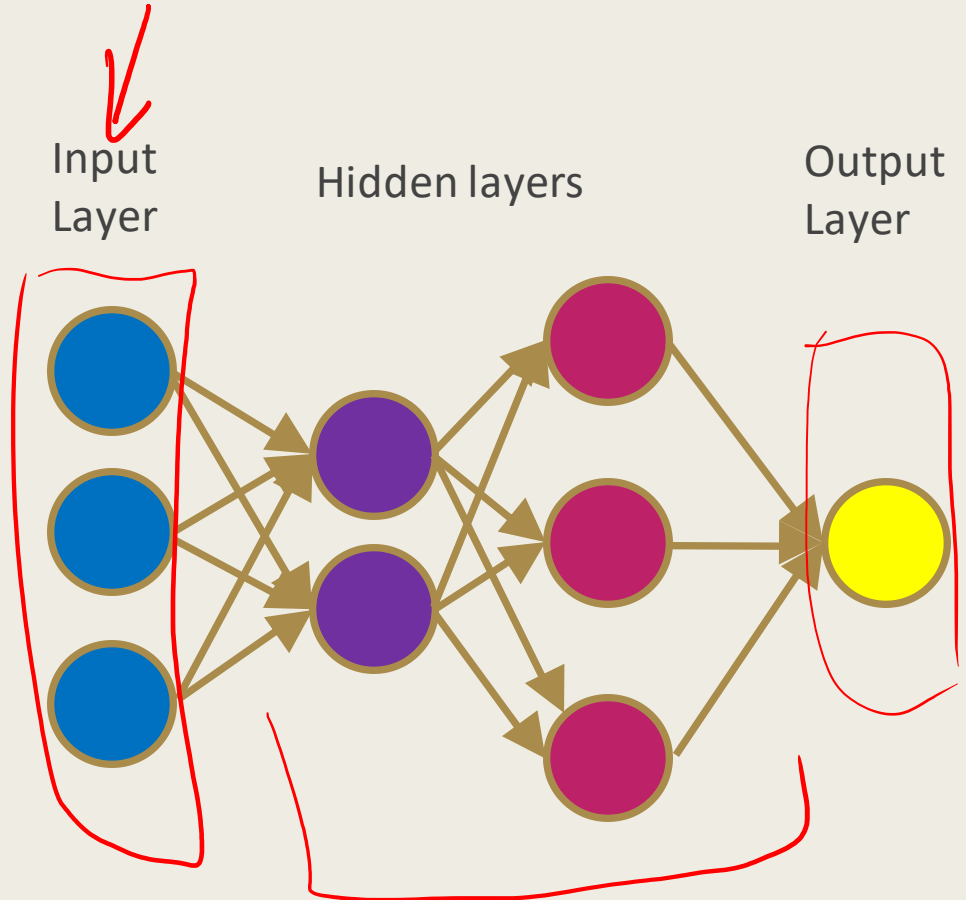
w_i – ponderile sinaptice (weights),
 W vectorul ponderilor;

b – bias; $f(\cdot)$ - functia de activare; y - iesirea

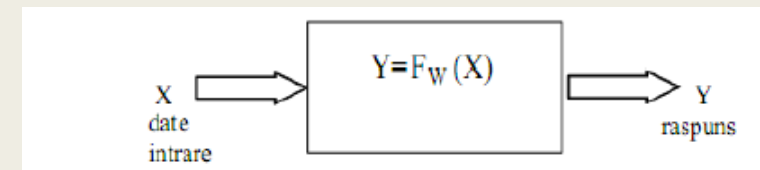
Neural Networks

Setul de date

- Multiple Perceptrons Network
 - *conectarea a mai multi perceptroni impreuna*
- Ex.: Input Layer; 2 hidden layers; Output Layer.
- Input Layers
 - *Valori reale – vectorul de trasaturi a setului de date*
- Hidden Layers
 - *Nivele/Layers intre input si output*
 - *3 sau mai multe nivele in “deep network”*
- Output Layer
 - *Estimarea finala a rezultatului*



Formulare matematica MPN (Multiple Perceptrons Network)



■ Formulare/descriere matematica perceptron

- N neuroni (multi perceptroni)
- M intrari
- C iesiri (dat de numarul de clase)

Pentru un perceptron: $y = f(WX + b)$

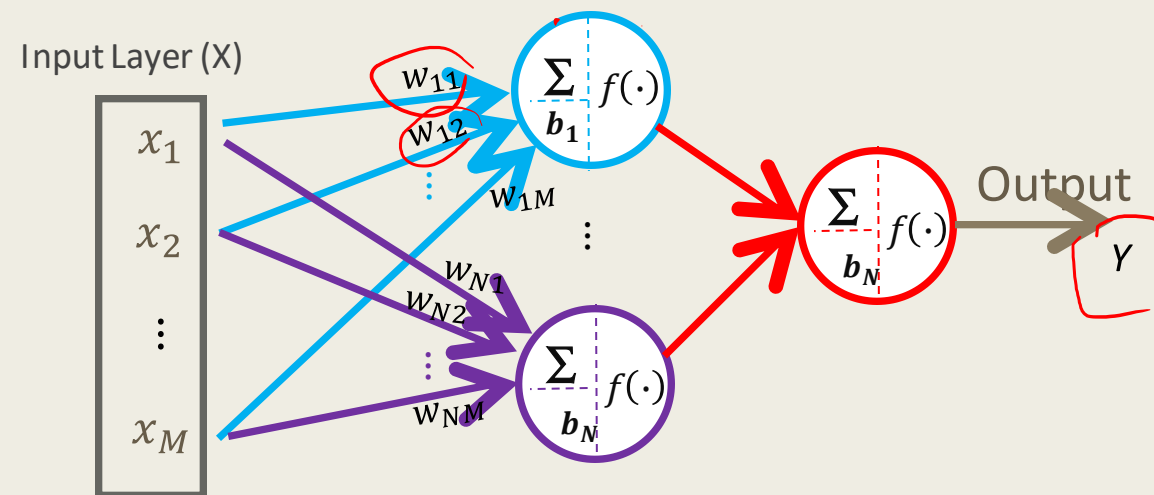
$$W = [w_1 \quad w_2 \quad \dots \quad w_M]$$

$$X = [x_1 \quad x_2 \quad \dots \quad x_M]^T$$

x_i – intrarile, (X vectorul de trasaturi,
 x_i trasatura i ,
 M – nr. trasaturi/ intrari)

w_i – ponderile sinaptice (weights),
 W vectorul ponderilor;

b – bias; $f(\cdot)$ - functia de activare; y - iesirea



$$Y = f(WX + B)$$

Mai multi perceptroni,
relatia se extinde :

- b valoare \rightarrow B vector
- W 1D \rightarrow W 2D
- y valoare \rightarrow Y vector

unde N – nr. de neuroni

$$W = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1M} \\ \dots & \dots & \dots & \dots \\ w_{N1} & w_{N2} & \dots & w_{NM} \end{bmatrix}$$

$$X = [x_1 \quad x_2 \quad \dots \quad x_M]^T$$

$$B = [b_1 \quad b_2 \quad \dots \quad b_N]$$

Funcții de activare (Activation function)

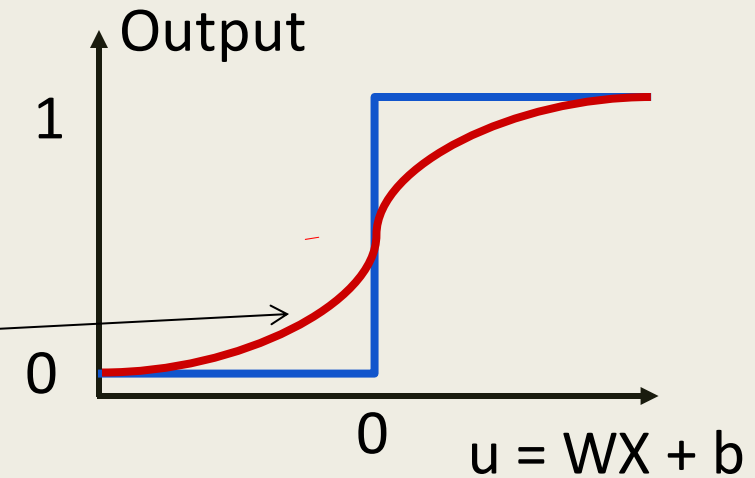
- Funcții de activare

- O funcție simplă care are la ieșire 0 sau 1
(reprezentată în albastru în graphic)

- acest tip de funcție nu reflectă schimbările ușoare

- Funcția sigmoid – este o funcție mai dinamică
(reprezentată în roșu în graphic)

$$f(u) = \frac{1}{1 + e^{-u}}$$

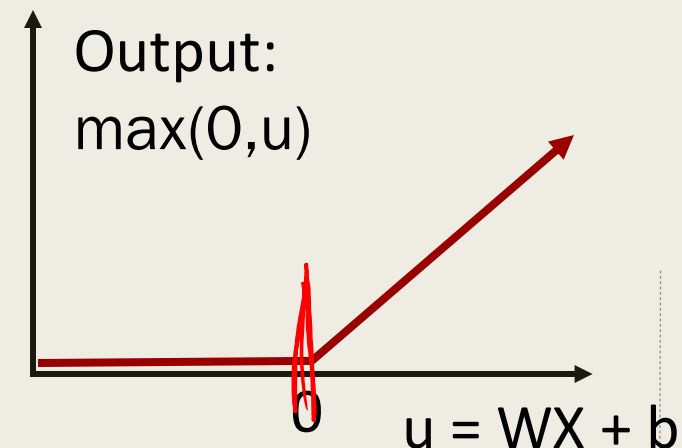


- Hyperbolic Tangent: $\tanh(u)$

$$\begin{aligned}\cosh x &= \frac{e^x + e^{-x}}{2} \\ \sinh x &= \frac{e^x - e^{-x}}{2} \\ \tanh x &= \frac{\sinh x}{\cosh x}\end{aligned}$$

- Rectified Linear Unit (ReLU)

- este o funcție relativ simplă: $\max(0, u)$



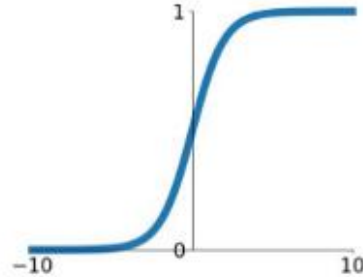
- Modificarea funcției de activare poate fi benefică (depinde mult de task)

- ReLU și Tanh – au cele mai bune performanțe

Functii de activare

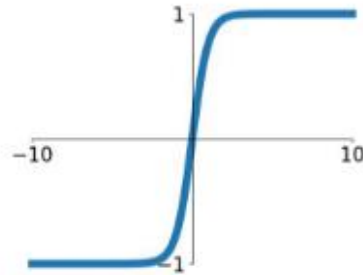
Sigmoid

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



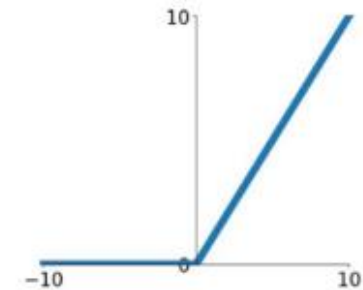
tanh

$$\tanh(x)$$



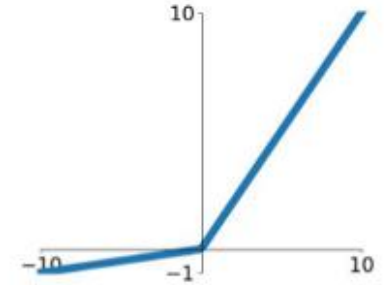
ReLU

$$\max(0, x)$$



Leaky ReLU

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

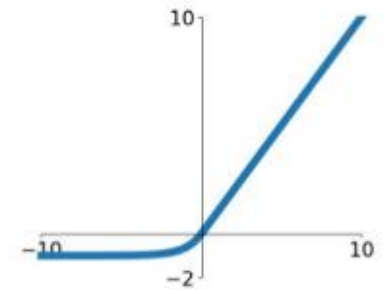


Maxout

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

ELU

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



Instruirea Retea Neuronala Artificiala

- Instruirea/antrenarea are loc prin
 - *ajustarea succesiva a ponderilor pentru a reduce diferenta dintre iesirile reale (dorite) si cele prezise pentru toate datele de antrenare.*
- Cum putem evalua performanta unui neuron?
 - *Se utilizeaza functia Cost*
 - pentru a masura cat de departe suntem de valoarea estimată/preconizată
- Se vor utiliza urmatoarele variabile:
 - *d pentru a reprezenta valoarea reală/dorită*
 - *y pentru a reprezenta predictia neuronului*
- In termini cu ponderi si bias: $w \cdot x + b = u$
u este trecut prin functia de activare $f(u) = y$

- Scopul este de a minimiza *functia de cost*
 - *prin modificarea ponderilor și a bias-ului rețelei.*
- **Quadratic Cost:** $C = \sum (d-y)^2 / N$
 - *se poate observa ca erorile mai mari sunt mai evidente datorita ridicarii la patrat*
 - *dar, din pacare acest calcul poate duce la o incetinire destul de mare a procesului de invatare*
- **Cross Entropy:**
 $C = (-1/N) \sum (d \cdot \ln(y) + (1-d) \cdot \ln(1-y))$
 - *aceasta functie Cost permite invatare mai rapida*
 - *cu cat este mai mare diferenta, cu atat mai repede poate invata neuronal*

Instruirea Retei Neuronale Artificiale

- Algoritm de instruire = modul în care se modifică ponderile, proces iterativ,

- la momentul $t+1$, actualizare pondere:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$$

$$\Delta w_{ij}(t) = -\mu \nabla C(t) = -\mu \frac{\partial C(t)}{\partial w_{ij}(t)} = -(d - y)f'(u)x_i$$

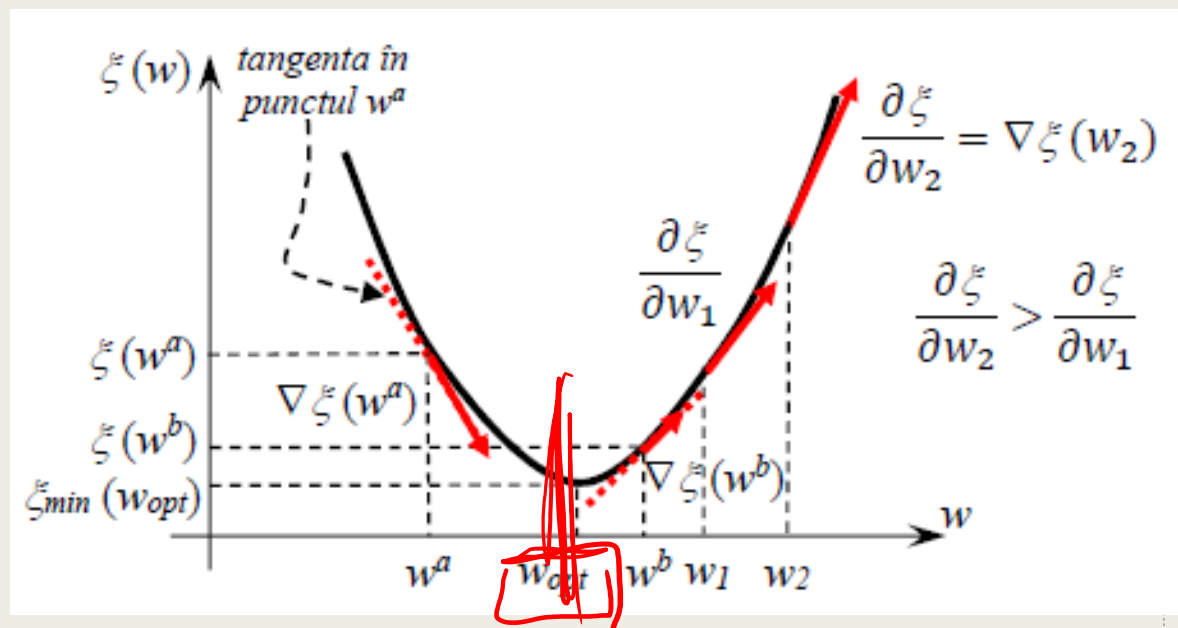
- unde:

- $\Delta w_{ij}(t)$ - algoritm de instruire
 - i - neuron de la care "pleacă" ponderea,
 - j - neuronul spre care "vine" ponderea
 - t - momentul de timp
 - μ - pas de intruire (subunitar)

- *dacă $\nabla C(t) > 0$, w_{ij} trebuie să scadă*
 - *dacă $\nabla C(t) < 0$, w_{ij} trebuie să crească*

- Optimizare prin găsirea minimului

- cea mai mică eroare posibilă – la vectorul optim de ponderi
 - metoda: gradientului descendent (derivata erorii în raport cu ponderea)



Backpropagation

- Este utilizat pentru a calcula contributia fiecarui neuron la eroare - dupa ce un lot (batch) de date este procesat
- **Backpropagation** – ajustarea parametrilor/ ponderilor in intreaga retea
 - eroarea se propaga inapoi (de aici si denumirea algoritmului), de la ultimul strat inspre primul strat.
- Algoritmul Backpropagation implica 2 faze:
 - faza **Forward**, in care parametrii retelei sunt fixati si semnalul de intrare este propagat prin retea , strat cu strat.
 - la sfarsitul acestei faze se calculeaza eroarea e
 - faza **Backward**, in care eroarea e se propaga prin retea inapoi.
 - in aceasta faza, se modifica parametrii pondere si bias, in scopul minimizarii erorii e

Pasi Antrenare / Instruire

Setul de date
→ antrenare
→ testare
→ validare

■ Mod de antrenare/ optimizare :

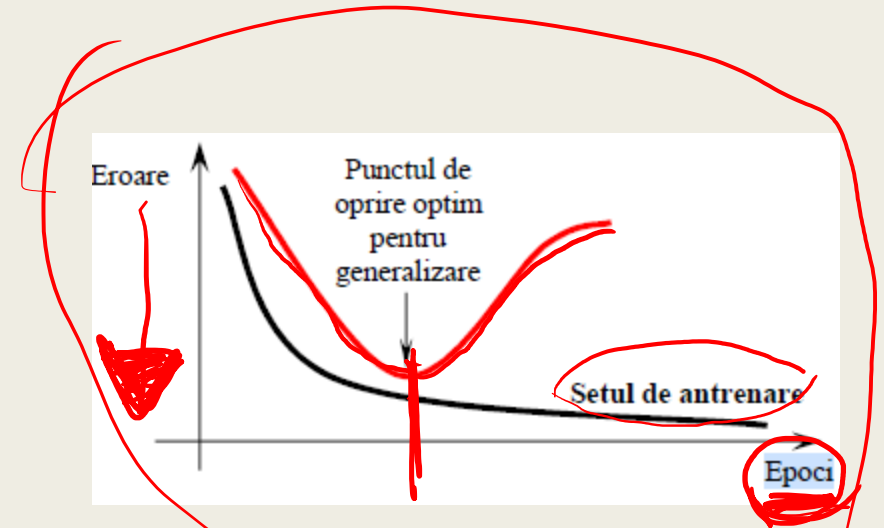
- Se generează un set de ponderi si valori bias, în mod aleator; ←
- 1) Se prezintă structurii neuronale intrările X și ieșirile dorite Y;
- 2) Se calculează ieșirile fiecărui neuron plasat pe fiecare strat neuronal;
- 3) Se calculează eroarea la nivelul de ieșire
- 4) Se propagă eroarea înapoi în rețea și se reține factorul delta corespunzător fiecărei ponderi
- 5) Se aplică ajustarea corespunzătoare fiecărei ponderi
- 6) Dacă s-a îndeplinit condiția de oprire a algoritmul de antrenare acesta se oprește; dacă nu se reia algoritmul de la pasul 1.

■ Condiția de oprire depinde

- de valoarea erorii (a scăzut sub un anumit prag acceptabil)
- și/sau numărul de epoci de antrenare

■ De ce poate diferi rezultatul la fiecare rulare?

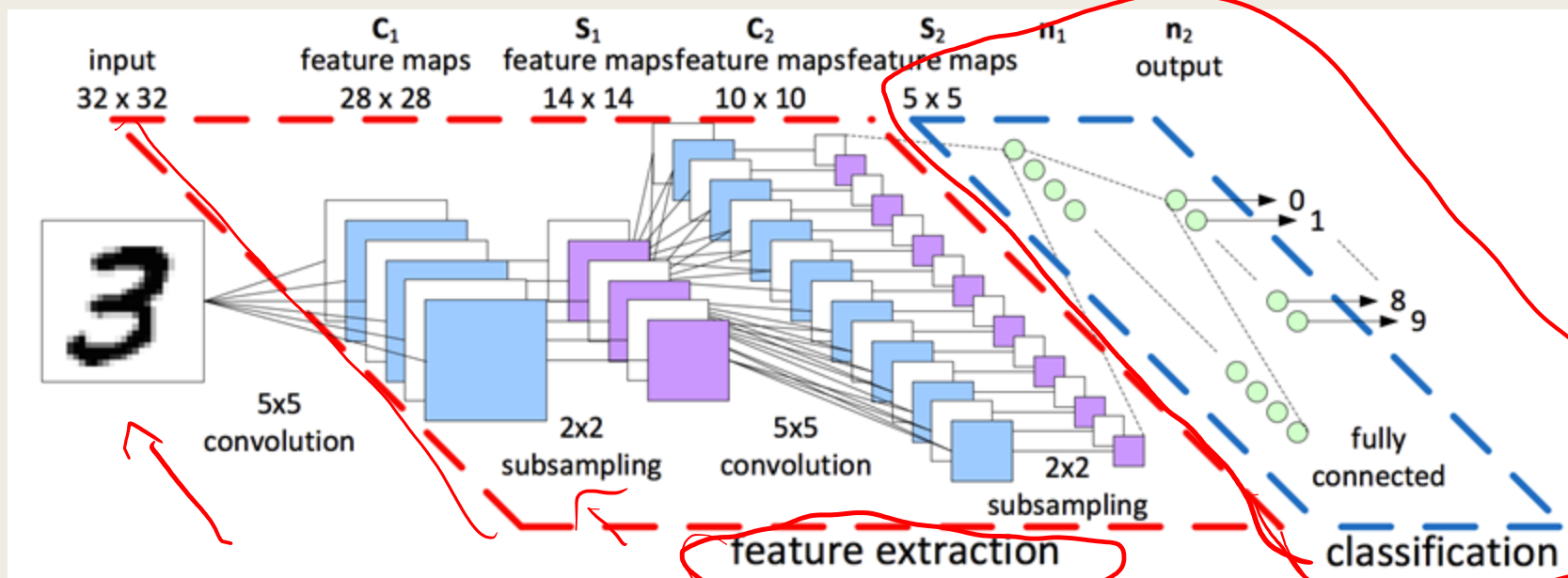
- initial avem o asociere aleatoare a ponderilor
→ rezultatul poate sa difere (in general, usor)



Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNN)

- **Neural Network** - optim daca s-au **extras/definit trasaturile** pentru descrierea datelor
- Direct pe imagini – se recomanda **Convolutional Neural Networks (CNN)**
 - *pentru a rezolva in mod eficient problemele care datele de tip **imaginile** le pot avea*
- In general, cand dicutam de CNN, vom intalni diagrame de genul:



Tensors

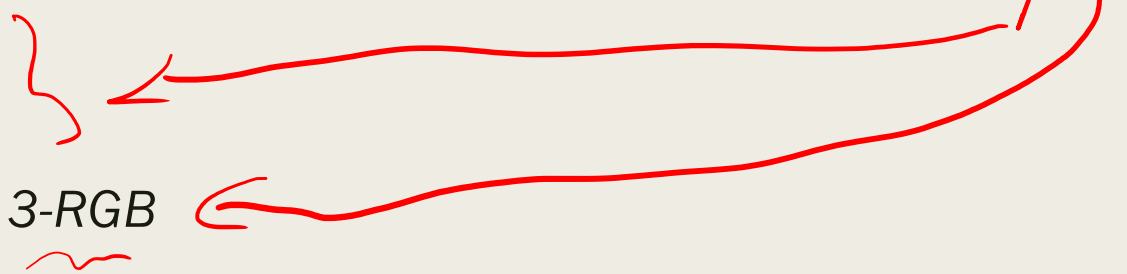
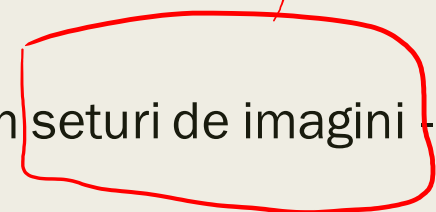
■ Tensori - N-Dimensional Arrays:

- Scalar - 3
- Vector - [3,4,5]
- Matrix - [[3,4], [5,6], [7,8]]
- Tensor - [[[1, 2], [3, 4]],
[[5, 6], [7, 8]]]

■ Tensori – convenint pentru ca la intrarea modelului sa avem seturi de imagini - (I,H,W,C)

- *I* : Images (numarul de imagini din setul de date)
- *H* : Height of Image in Pixels
- *W* : Width of Image in Pixels
- *C* : Color Channels: 1-Grayscale, 3-RGB

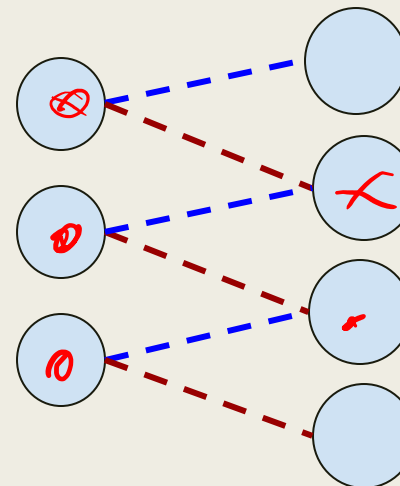
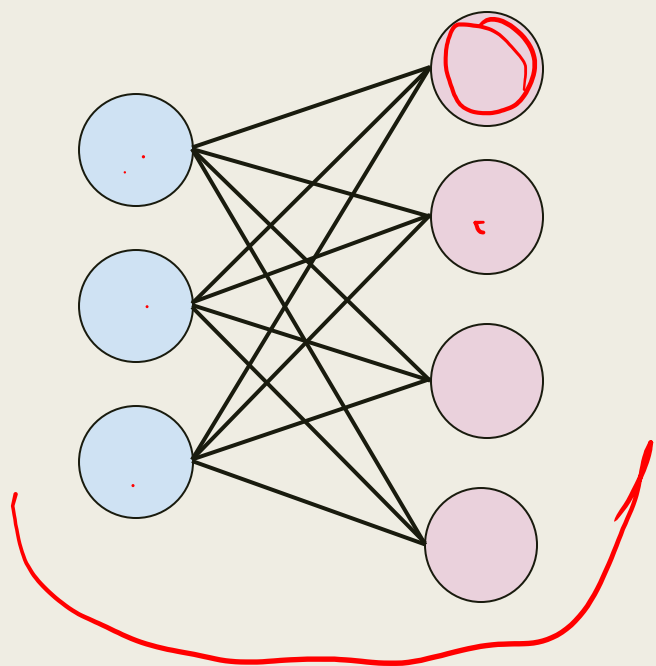
4D
contine
setul de date



Densely Connected layer/ Convolutional Layer

■ Densely Connected layer/ Convolutional Layer

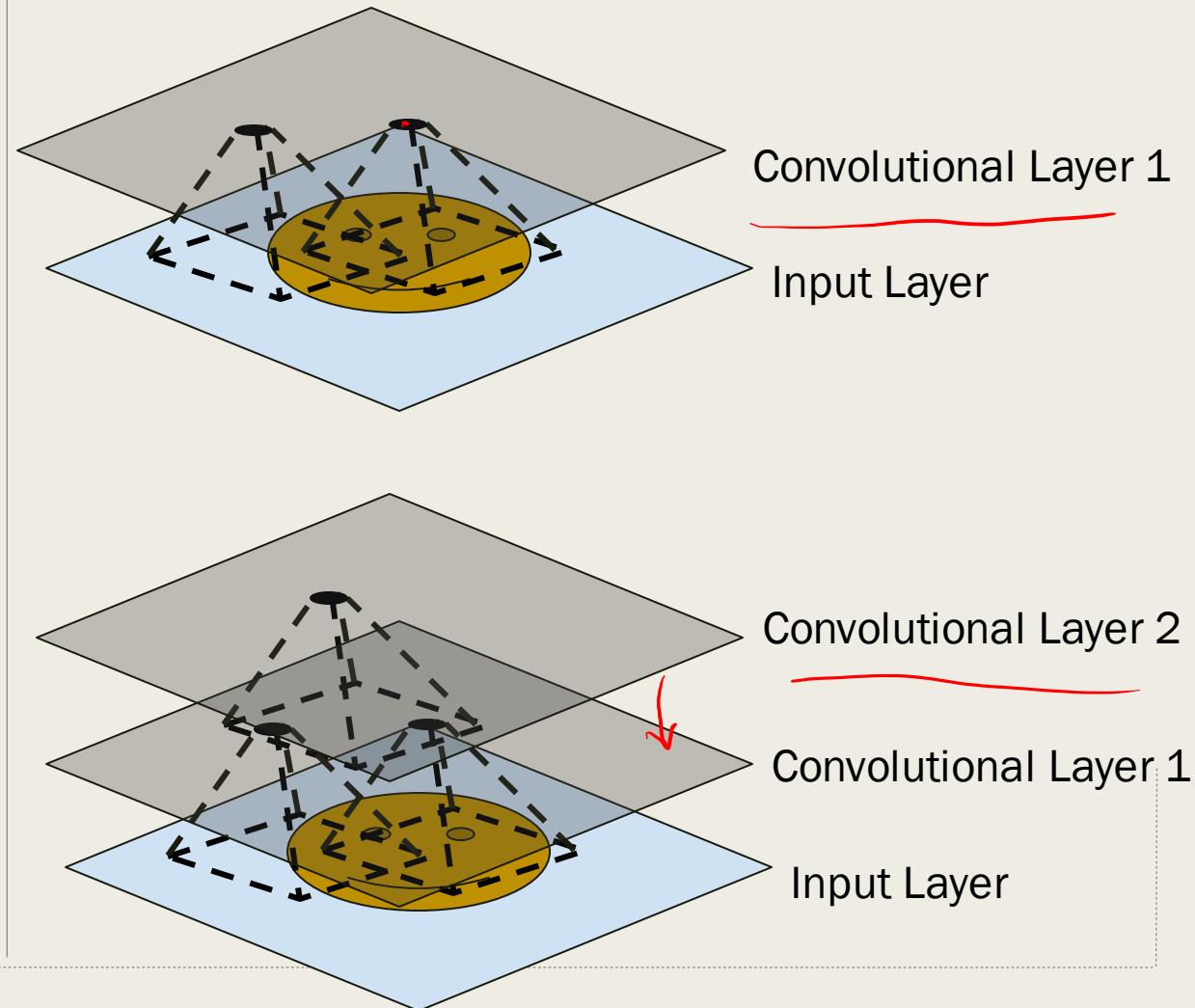
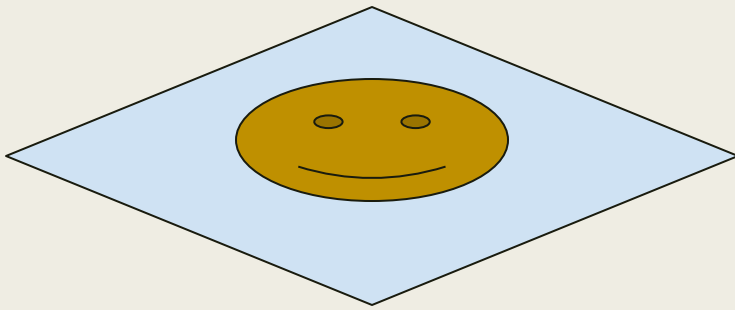
Fiecare neuron este conectat cu un numar mai mic de neuroni vecini.



Care este avantajul CNN versus DNN? Majoritatea imaginilor au o rezolutie de cel putin 256×256
→ prea multi parametri/ prea multe calcule implicate

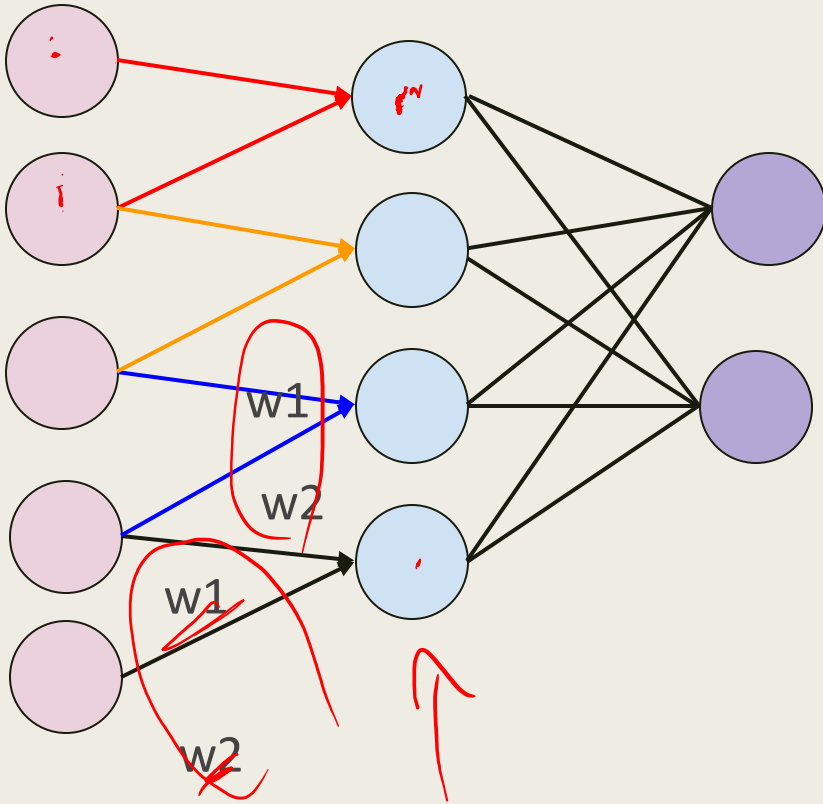
CNN

- CNN – aplicat pe imagini pentru clasificare
- Input layer – imaginea in sine
- Nivelele convolutive (Convolutional layers)
 - Sunt conectate doar catre valorile din campul lor/respectiv

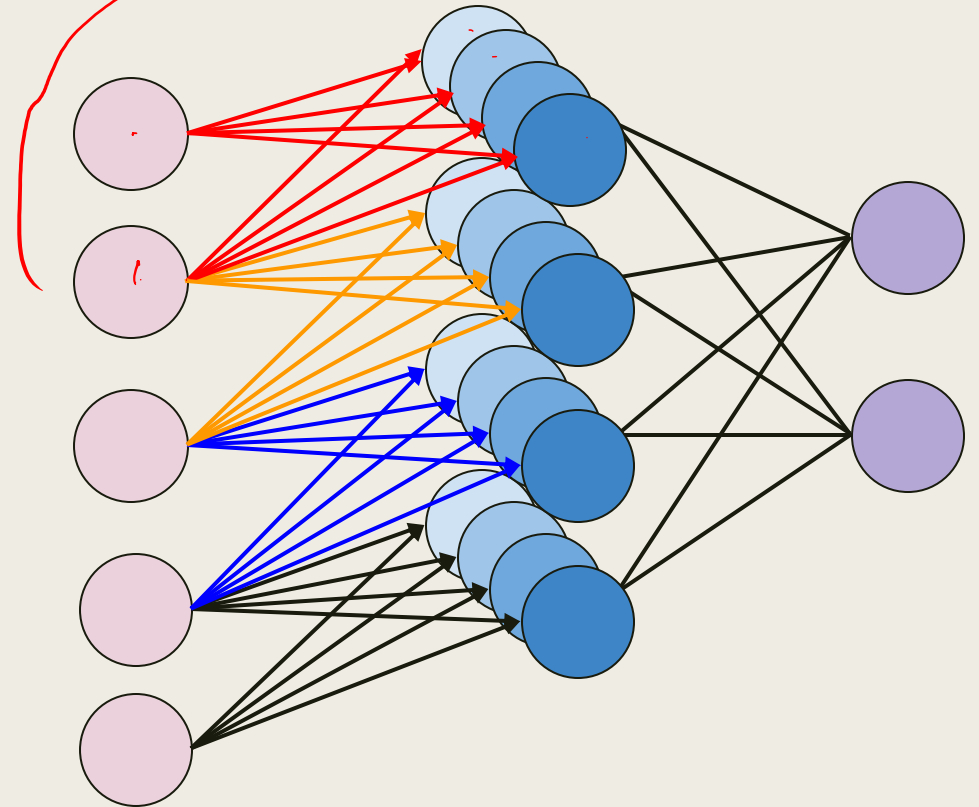


CNN, 1-D Convolution

Filters: 1/Filter Size: 2/ Stride: 1

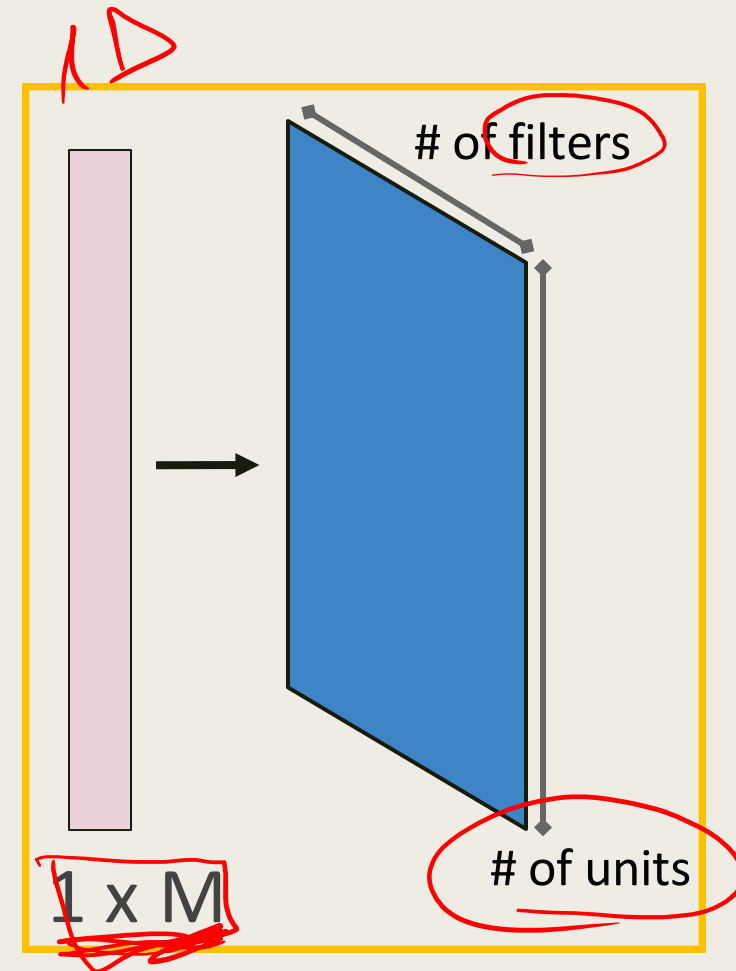
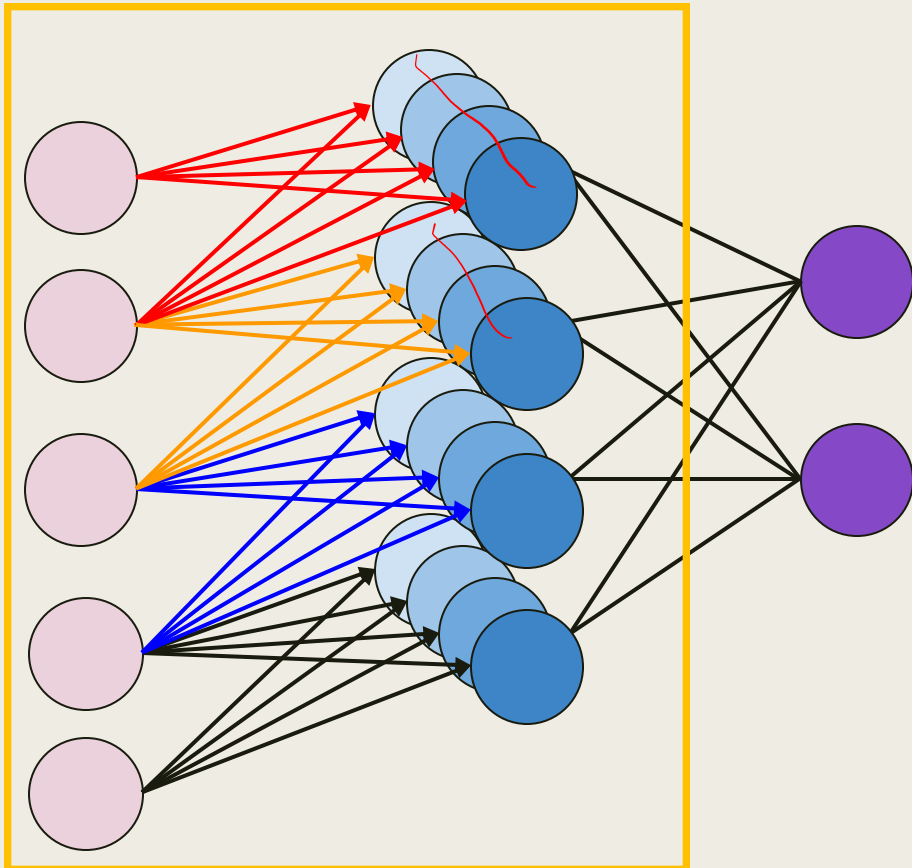


Filters: 4/Filter Size: 2/Stride: 1 (1 Unit at a time)



CNN, 1D Convolution

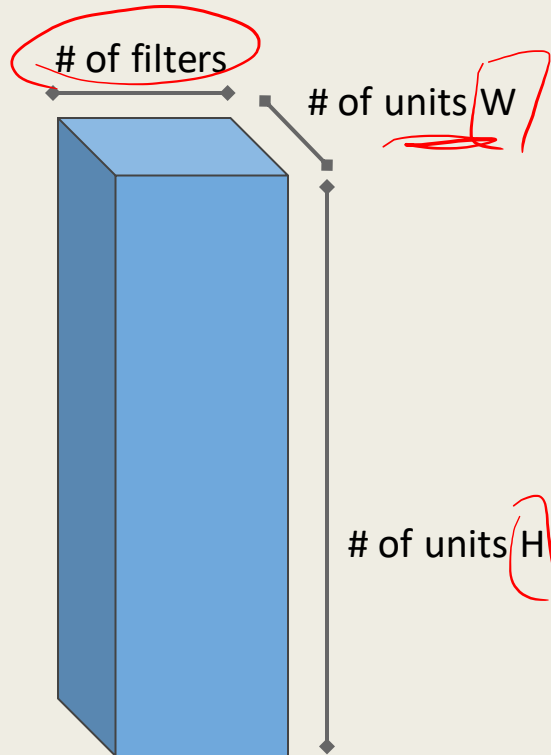
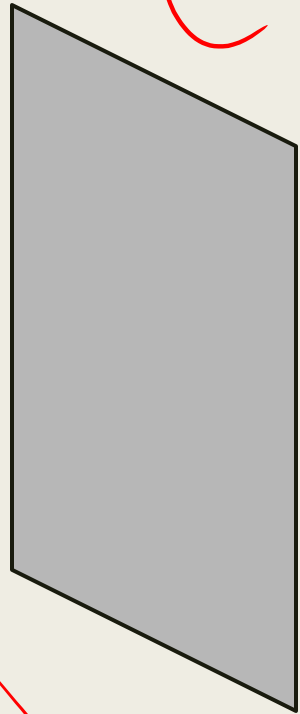
- Pentru simplificare – vizualizare ca blocuri



CNN, 2D Convolution

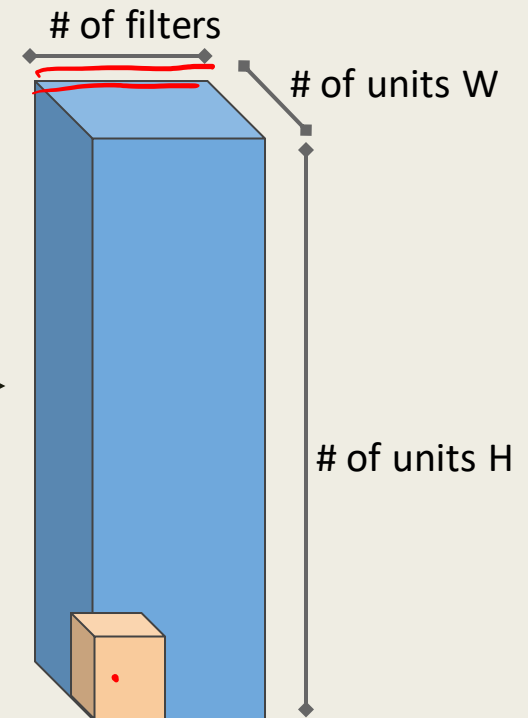
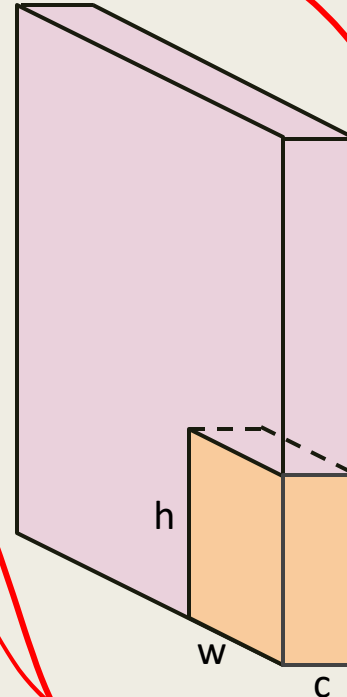
Imagini pe nivele de gri:

Input Image: (H,W)



Imagini color:

Input Image: (H,W,C)



3D

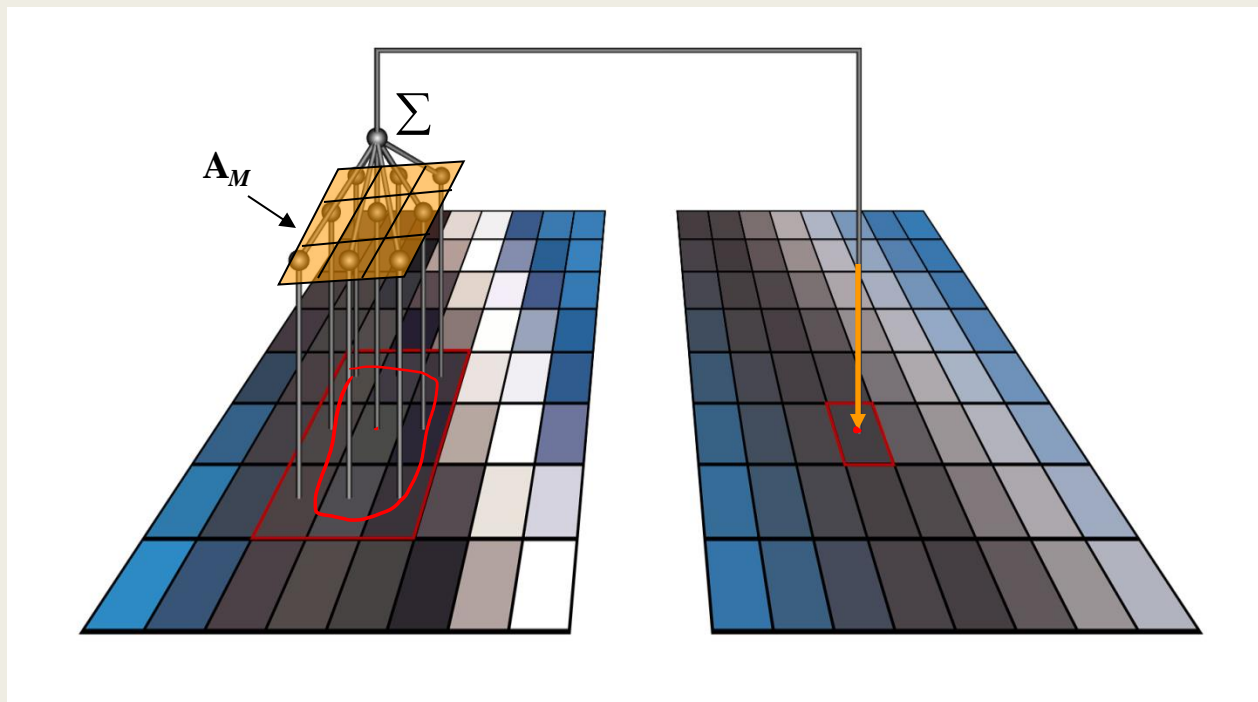
- Convolutie – are un mare avantaj pentru procesarea/analiza imaginilor
 - *pixelii vecini sunt mai corelati intre ei – efficient in descrierea locala a informatiei/ a variatiei locale*
- Fiecare nivel CNN – se poate ‘uita’ la o alta rezolutie a imaginii.
- Avand neuroni legati doar in ‘vecinatate’
 - *este limitat numarul de ponderi (weights) la dimensiunea ferestrei de convolutie*

Operatia de convolutie

- Formula de convolutie:

$$v(m,n) = \sum_{(k,l) \in W} a(k,l)u(m-k, n-l)$$

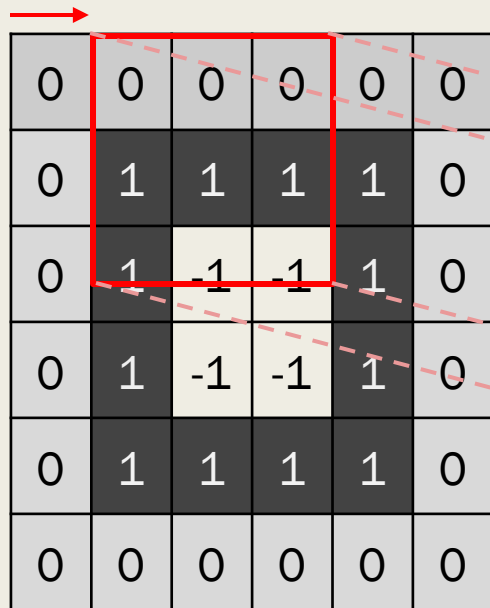
$$\mathbf{A}[K \times L] = \{a(k,l)\} \text{ - Masca de convolutie}$$



$$\mathbf{A} = \begin{bmatrix} a(-1,-1) & a(-1,0) & a(-1,1) \\ a(0,-1) & a(0,0) & a(0,1) \\ a(1,-1) & a(1,0) & a(1,1) \end{bmatrix}$$

Stride Distance

Exemplu, stride = 1

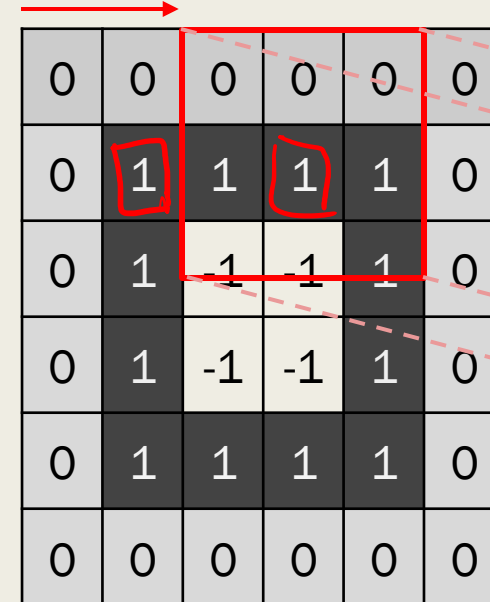


0	0	0	0	0	0
0	1	1	1	1	0
0	1	-1	-1	1	0
0	1	-1	-1	1	0
0	1	1	1	1	0
0	0	0	0	0	0

0	0	0
0	-1	1
0	1	-1

3 by 3 Filter

Exemplu, stride = 2

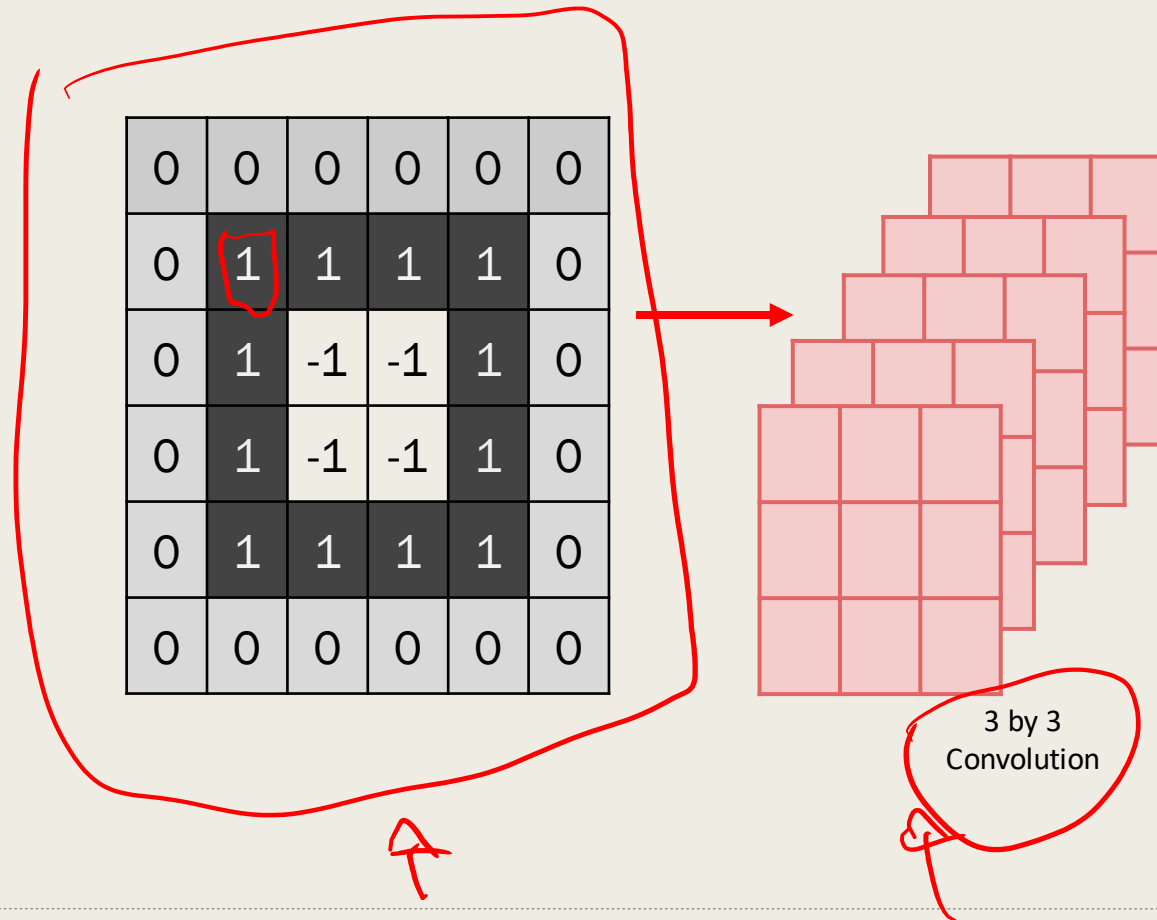


0	0	0	0	0	0
0	1	1	1	1	0
0	1	-1	-1	1	0
0	1	-1	-1	1	0
0	1	1	1	1	0
0	0	0	0	0	0

0	0	0
0	-1	1
0	1	-1

3 by 3 Filter

■ Reprezentare – filtre multiple (Multiple Filters)





blur

blur

0,0625	0,125	0,0625
0,125	0,25	0,125
0,0625	0,125	0,0625



outline

outline

-1	-1	-1
-1	8	-1
-1	-1	-1



top sobel

top sobel

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

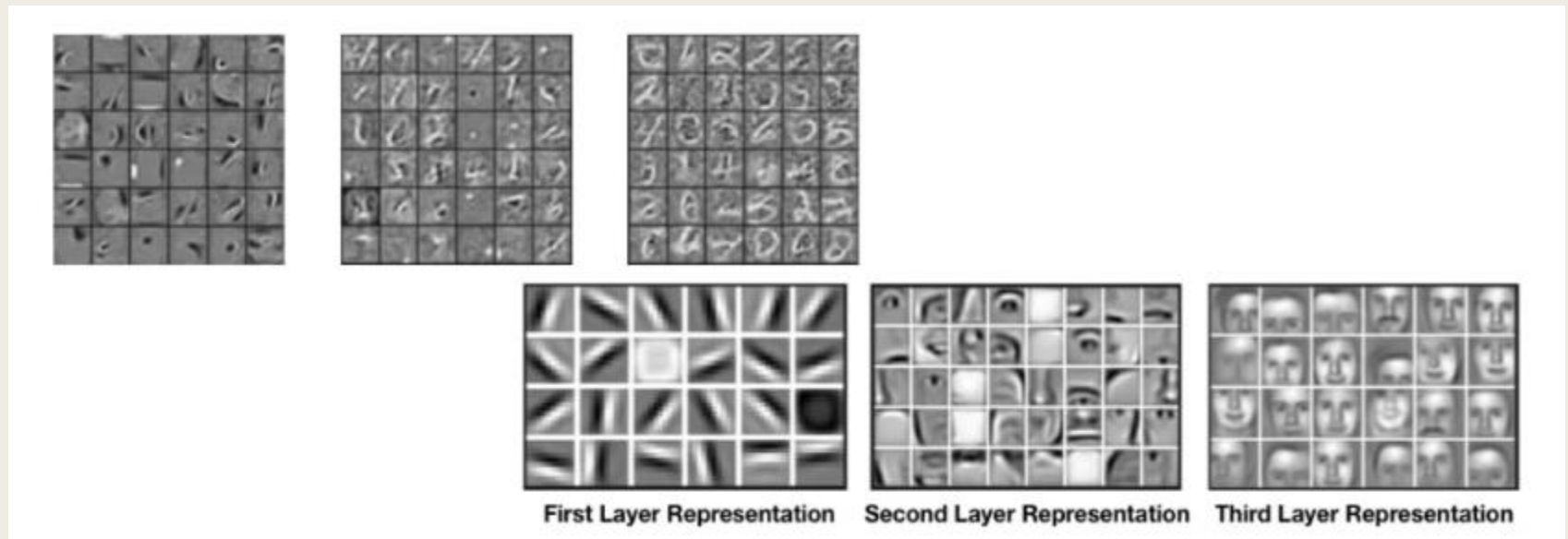


emboss

emboss

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

- Filtrele de convoluție odată „învățate” (prin ajustarea ponderilor) asigură extragerea automată a caracteristicilor specifice, începând cu unele elementare, extrem de simple în primul strat convoluțional și ajungând, prin agregarea acestora strat după strat, la caracteristici complexe reprezentând părți din, sau chiar obiecte întregi.
- <http://setosa.io/ev/image-kernels/>
- <http://cs231n.stanford.edu/>



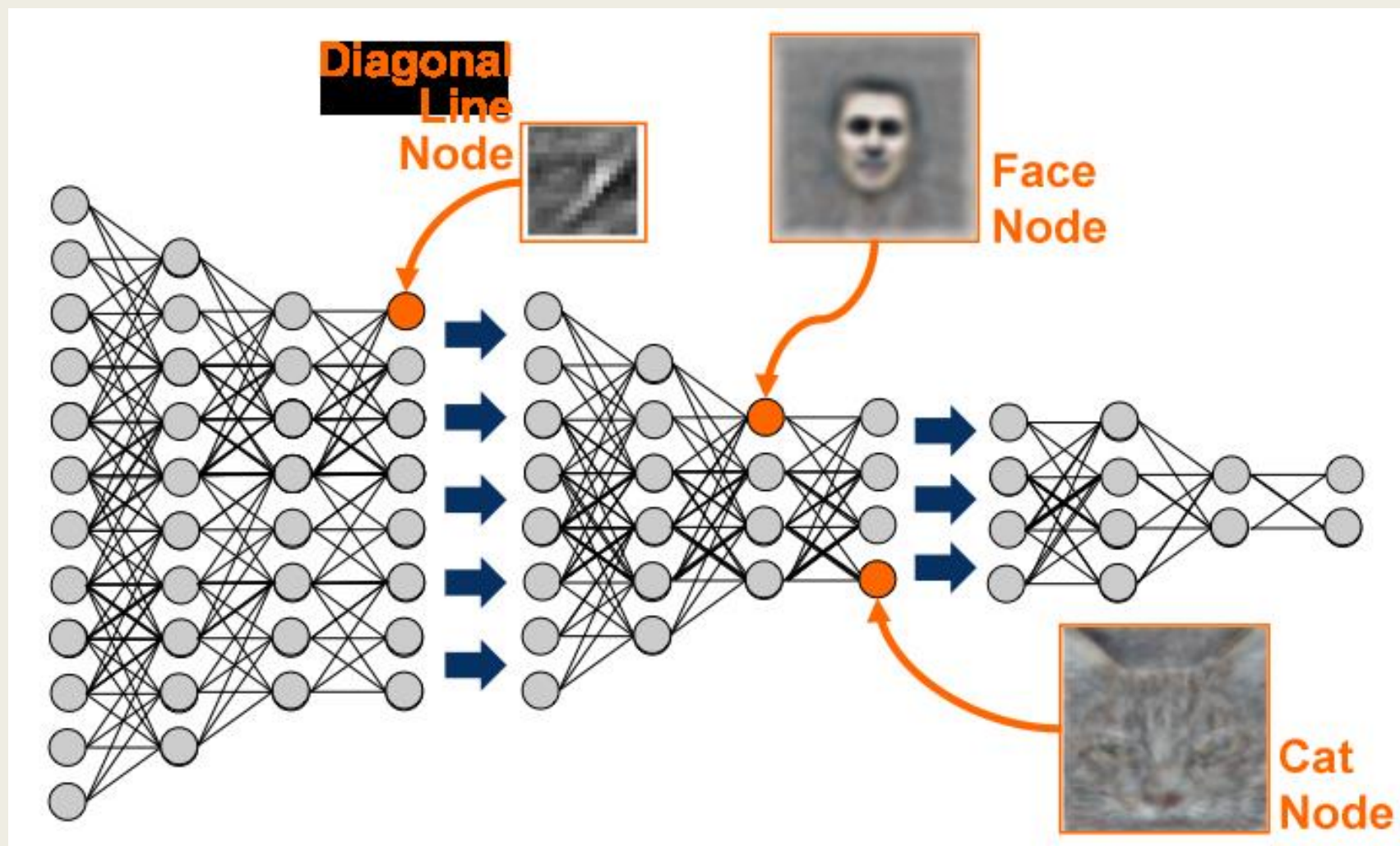
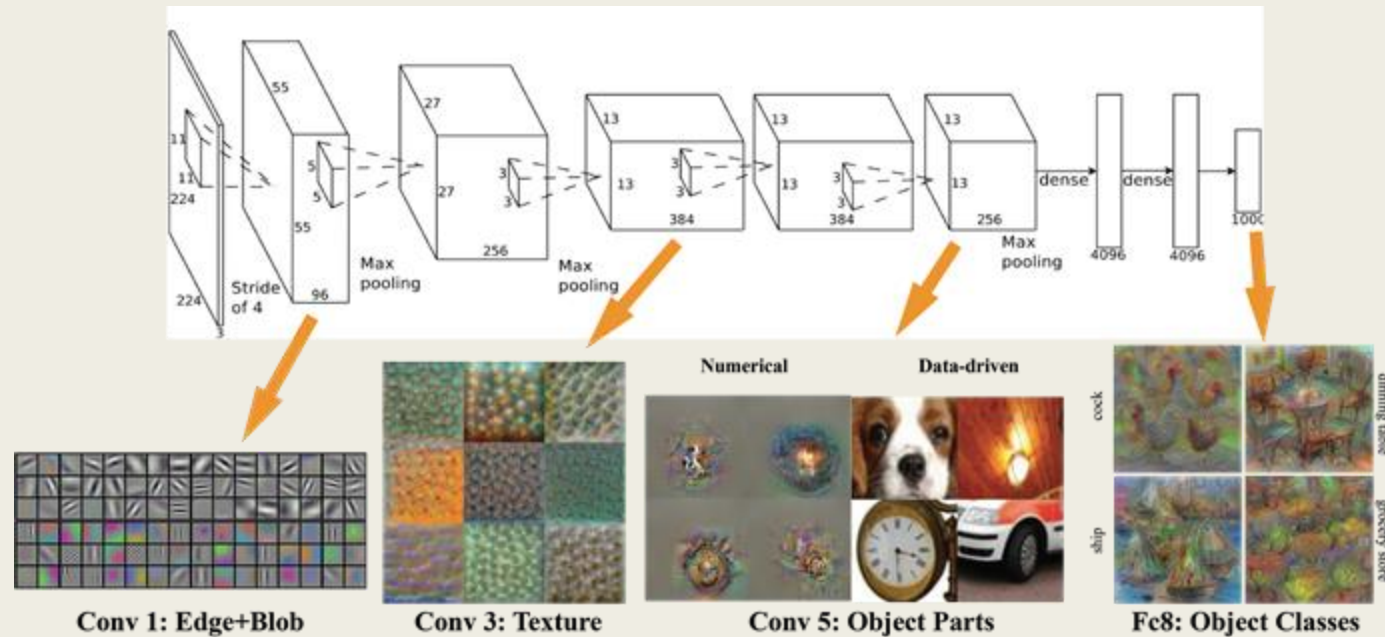


Diagrama CNN - AlexNet

- Diagrama CNN - AlexNet



Pooling Layers

■ Pooling layers

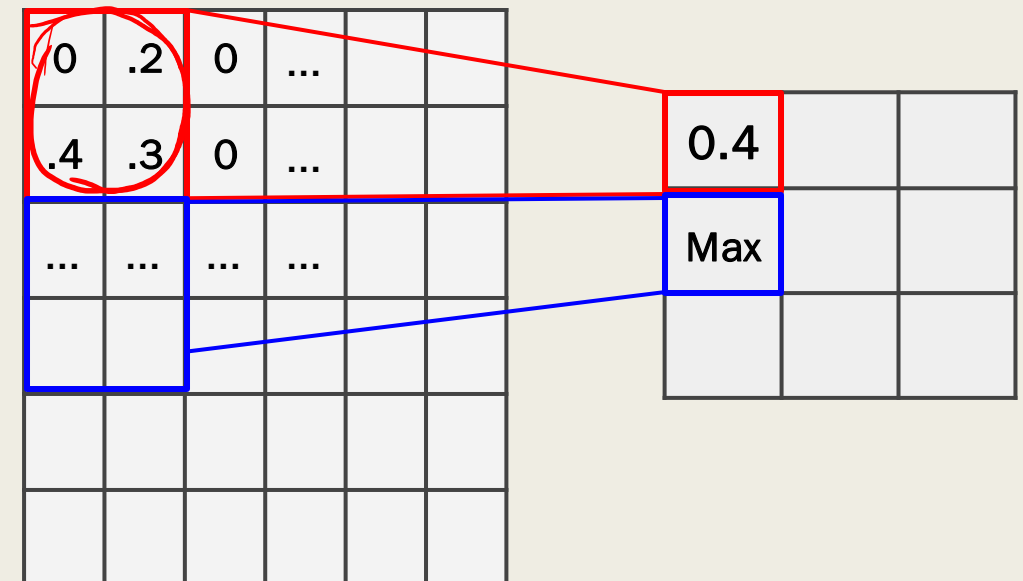
- *aceasta operatie inlocuieste valoarea unei zone din imagine/harta de trasaturi cu o statistica a acelei zone.*

■ Functia de max-pooling este cea mai folosita in aplicatii si inlocuieste valoarea dintr-o zona bine definita cu maximul acelei zone, rezultand intr-o harta de trasaturi mai mica, **dar care pastreaza cea mai relevanta statistica.**

- *In acest mod, pe langa reducerea dimensionalitatii, se obtine si o invarianta la translatii mici.*

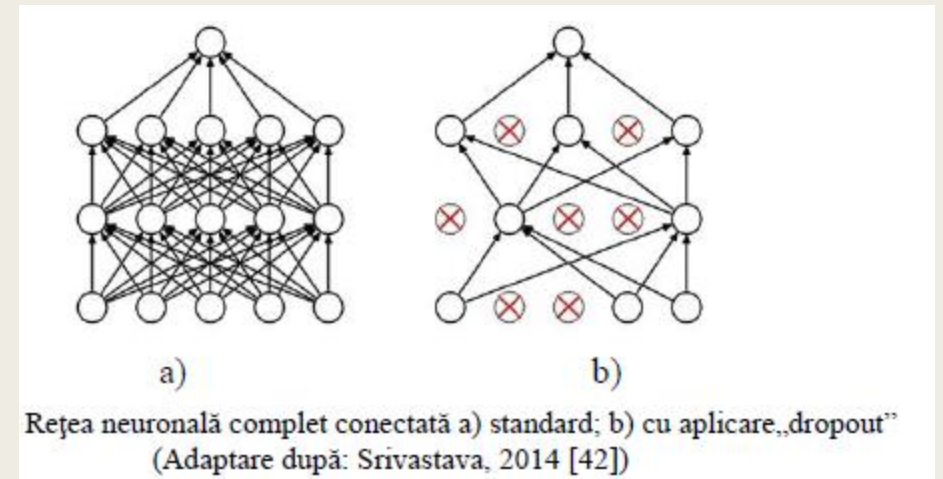
■ Ex. 2x2 max-pooling

- *pastram valoare maxima dintr-o regiune de 2x2*
- *si ne mutam cu pasul dat de 'stride' (in acest caz 2)*



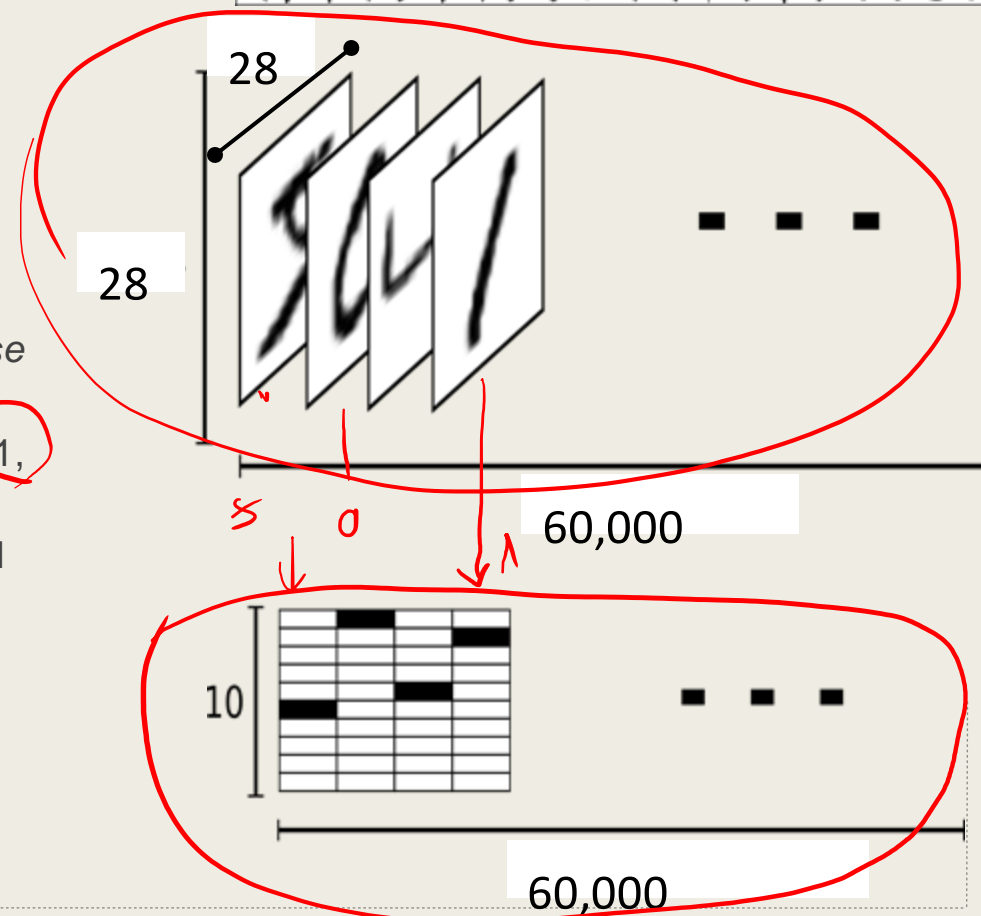
Dropout

- Tehnica “*dropout*” - constă în setarea (aleatoare) la zero a ieșirilor unor neuroni de pe straturile ascunse cu o probabilitate de 0,5 (respectiv jumătate dintre aceștia).
- Neuronii care sunt astfel “decuplați” nu contribuie la propagarea înainte a semnalului și nici nu participă la ajustarea celorlalte ponderi la antrenarea cu *backpropagation*.
- Această tehnică reduce co-adaptările complexe ale neuronilor (ajuta la prevenire overfitting),
 - *din moment ce un neuron nu se poate baza pe prezența altor neuroni particulari, fiind astfel forțat să învețe caracteristici mai robuste care sunt utile în conjuncție cu multe subseturi aleatoare de alți neuroni.*

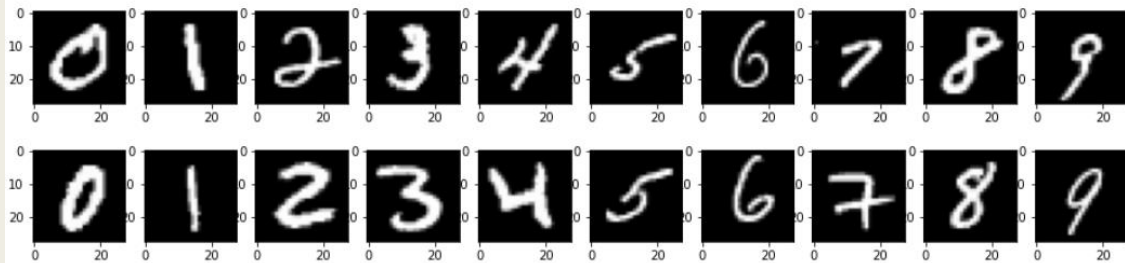


MNIST DataSet

- Contine cei 10 digiti, de la 0 la 9, scrisi de mână
 - Contine 60,000 imagini, de o rezolutie 28x28, pe nivele de gri (1 canal de culoare)
 - Reprezentat ca un tensor - 4 dimensiuni (Samples, H, W, channels)
 - $(60000, 28, 28, 1)$
 - pentru imagini color, ultima dimensiune = 3 (RGB)
 - Etichetarea se realizează sub forma One-Hot Encoding.
 - Label-urile - in loc de o valoare pe imagine din $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$ - se vor reprezenta sub forma unui vector pentru fiecare imagine
 - Vector de 10 valori (numarul claselor), unde doar un index este 1, restul zero
 - Valoarea label-ului este data de indexul pozitiei pe care avem 1
- Ex. pentru 4 vom avea label-ul $[0, 0, 0, 0, 1, 0, 0, 0, 0, 0]$ ← Y
- => Etichetele pentru setul de date devine 2D, de dimensiune $(60000, 10)$



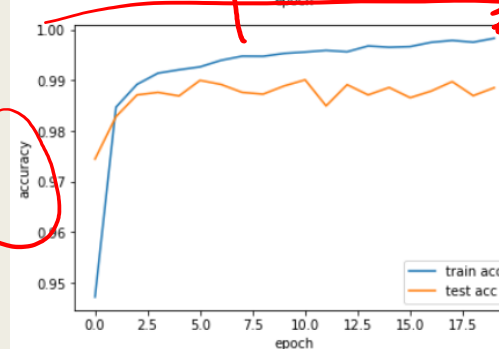
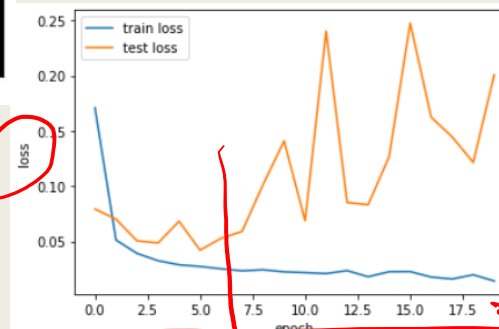
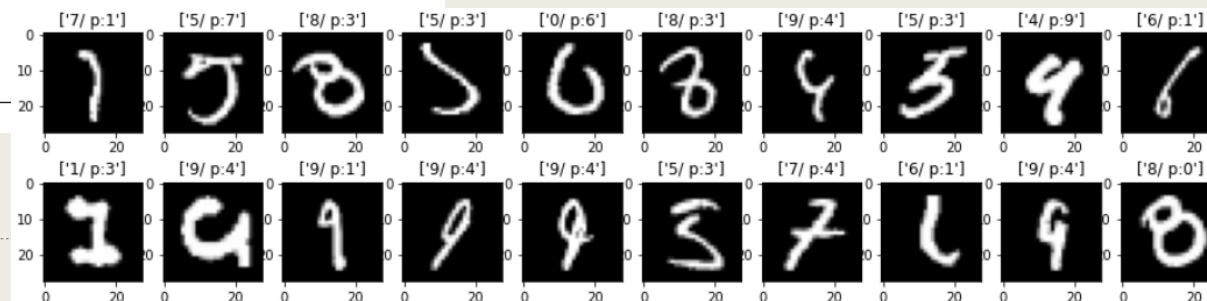
MNIST DataSet – Model Simplu



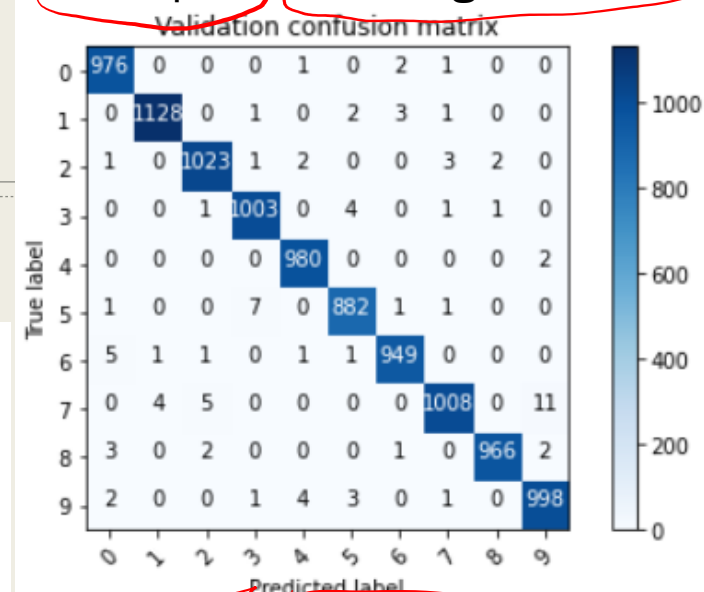
Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 25, 25, 32)	544
conv2d_3 (Conv2D)	(None, 22, 22, 32)	16416
max_pooling2d_2 (MaxPooling2)	(None, 11, 11, 32)	0
conv2d_4 (Conv2D)	(None, 8, 8, 64)	32832
conv2d_5 (Conv2D)	(None, 5, 5, 64)	65600
max_pooling2d_3 (MaxPooling2)	(None, 2, 2, 64)	0
flatten_2 (Flatten)	(None, 256)	0
dense_3 (Dense)	(None, 512)	131584
dense_4 (Dense)	(None, 10)	5130

Total params: 252,106
Trainable params: 252,106
Non-trainable params: 0

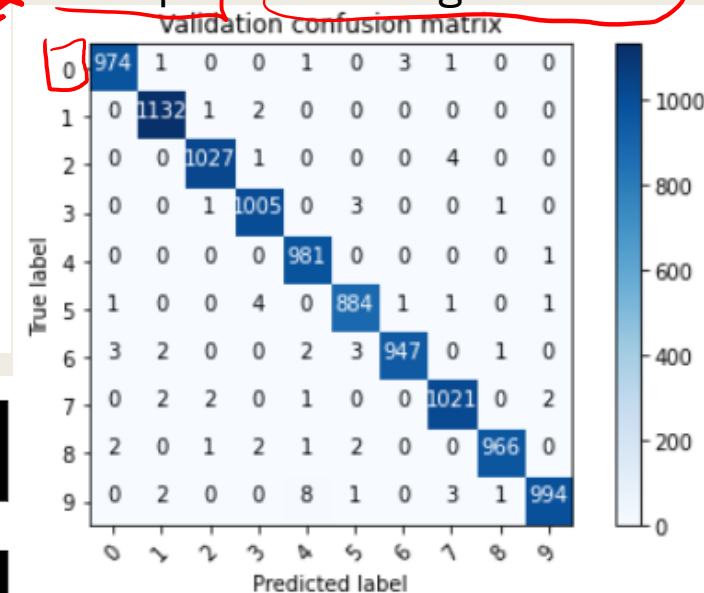
None



20 epoci - 87 imagini eronate



5 epoci - 69 imagini eronate



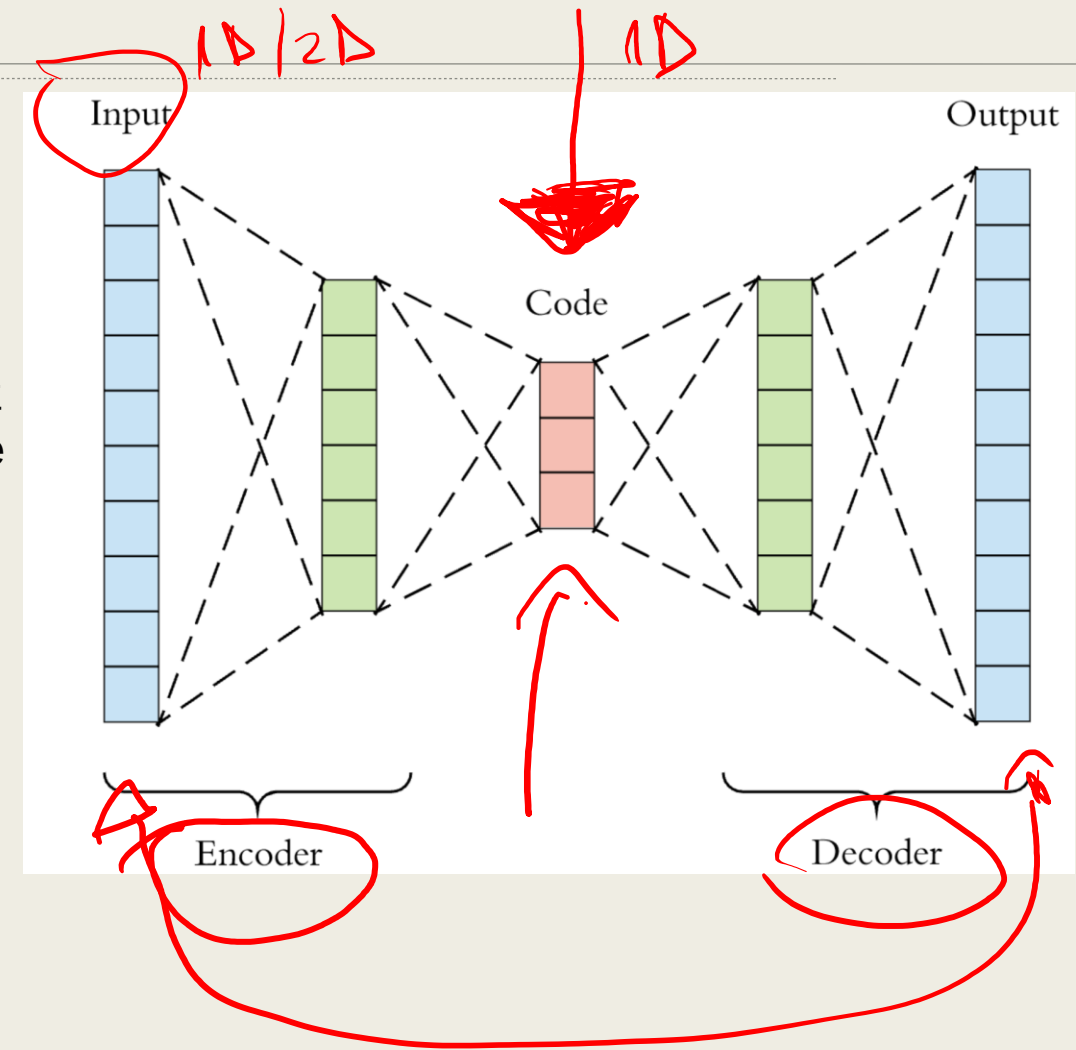
- <https://www.youtube.com/watch?v=vT1JzLTH4G4&list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv>
- <https://www.pieriandata.com/>
- <https://www.pyimagesearch.com/2018/12/10/keras-save-and-load-your-deep-learning-models/>
- <https://pylessons.com/CNN-tutorial-introduction/>
- <https://towardsdatascience.com/architecture-comparison-of-alexnet-vggnet-resnet-inception-densenet-beb8b116866d>

Autoencoders

Autoencoders are a deep learning model for transforming data from a high-dimensional space to a lower-dimensional space. They work by encoding the data, whatever its size, to a 1-D vector. This vector can then be decoded to reconstruct the original data (in this case, an image). The more accurate the autoencoder, the closer the generated data is to the original.

In contrast with **discriminative models**, there is another group called **generative models** which can create new images. For a given input image, the output of a discriminative model is a class label; the output of a generative model is an image of the same size and similar appearance as the input image.

One of the simplest generative models is the **autoencoder** (AE for short), which is the focus of this tutorial.



Autoencoders

Autoencoders are a deep neural network model that can take in data, propagate it through a number of layers to condense and understand its structure, and finally generate that data again.

To accomplish this task an autoencoder uses two different types of networks. The first is called an **encoder**, and the other is the **decoder**. The decoder is just a reflection of the layers inside the encoder. Let's clarify how this works.

The job of the encoder is to accept the original data (e.g. an image) that could have two or more dimensions and generate a single 1-D vector that represents the entire image. The number of elements in the 1-D vector varies based on the task being solved. It could have 1 or more elements. The fewer elements in the vector, the more complexity in reproducing the original image accurately.

Encoder Network



Conv

ReLU

Conv

Conv

Dense

Dense

Decoder Network

Dense

Dense

Conv

Conv

ReLU

Conv



The loss is calculated by comparing the original and reconstructed images, i.e. by calculating the difference between the pixels in the 2 images.