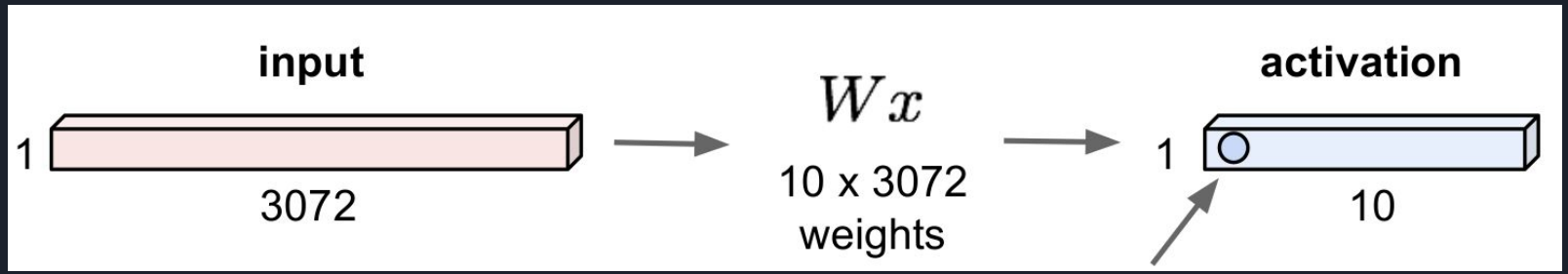
Retele Neurale Convolutionale

Curs 4

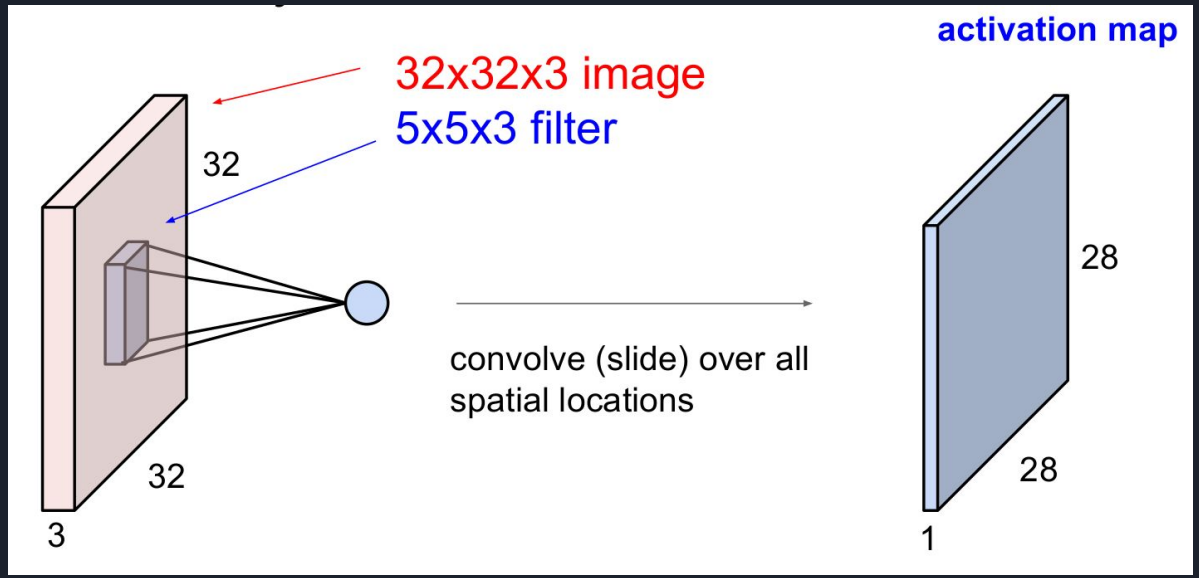
Reminder. Fully Connected

Imagine 3x32x32. Liniarizata in 3072x1 (vector)

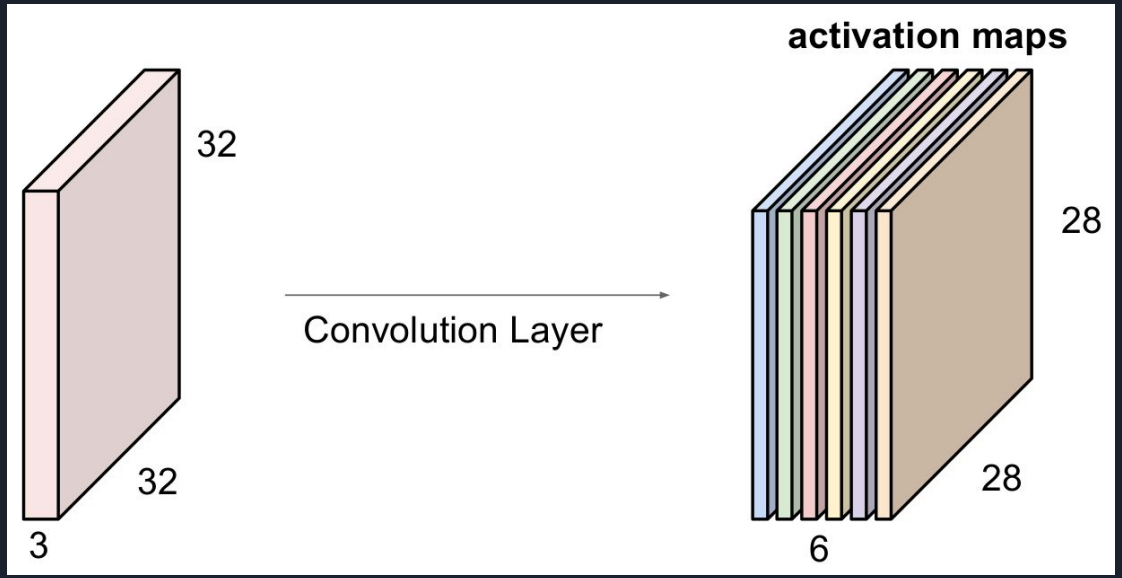


Rezultatul produsului scalar intre o linie W si input

Reminder. Convolutie



Reminder. Convolutie



- Un filtru produce un singur activation map
- Mai multe filtre produc mai multe activation maps - un volum similar cu imaginea de input
- Fiecare filtru invata un aspect (feature) diferit

Proprietati ale Convolutiei

Masoara gradul de asemenare intre 2 semnale

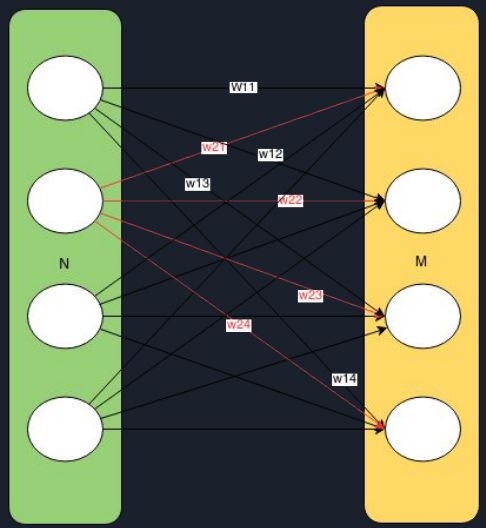
Obiectiv

- Exploatam distributia spatiala 2D a datelor
- Data Parallelism (detectam pisici si in centru in laterala pozei)

Caracterstici:

1. Sparse Connectivity
2. Parameter sharing
3. Invariance to translation
4. Receptive Field
    a. Se prefera Deep vs Wide pentru a obtine un receptive field cat toata poza

Convolutii. Sparse Connectivity



- Fully connected - structura de graf bipartit in care fiecare output interactioneaza cu fiecare input
- Convolutional: “kernel” - ul este mai mic decat input-ul
- Kernel-ul convolutional este conectat pe rand in diferite offset-uri ale input-ului
- Se pastreaza putini parametri
- Detectie locala si nu globala a caracteristicilor din input

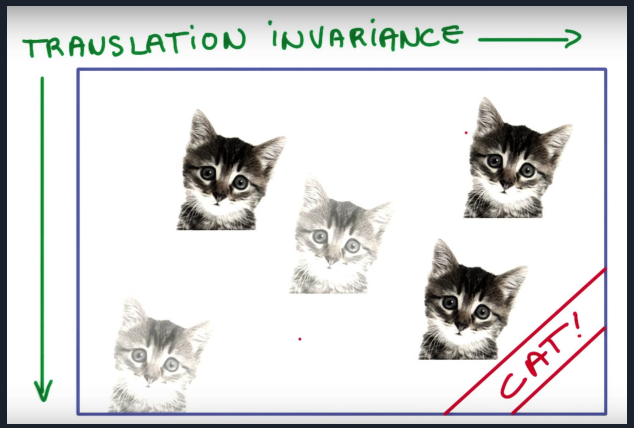
Convolutia

Parameter Sharing

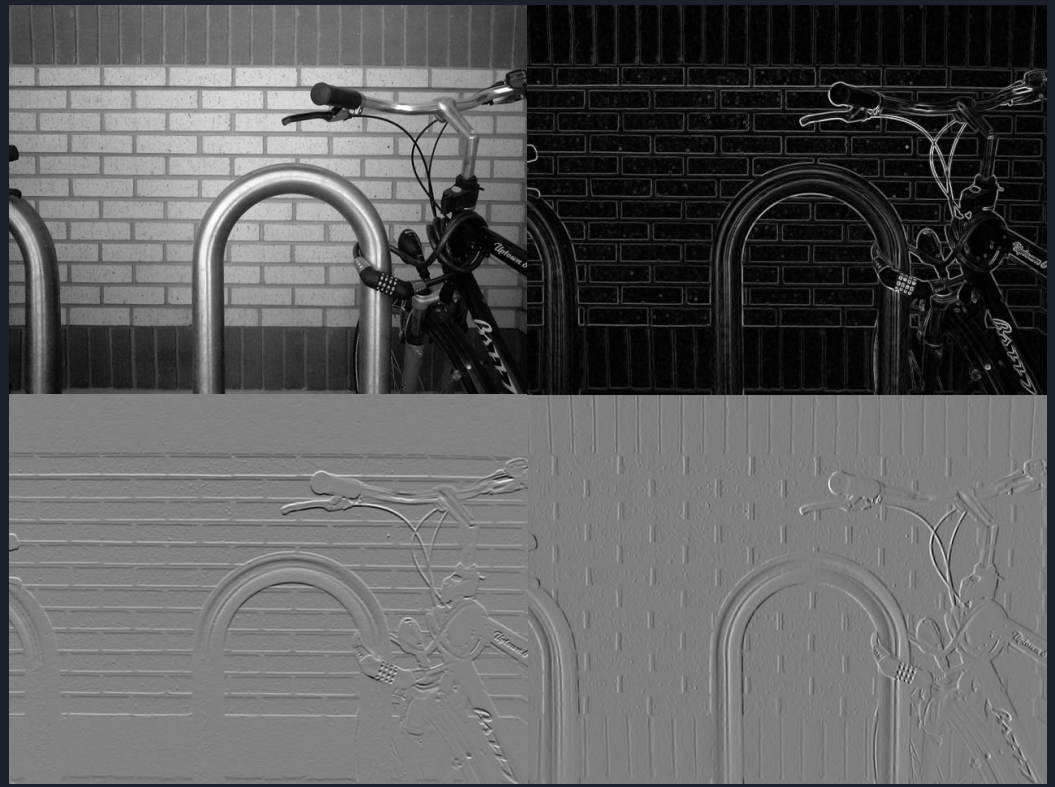
- Parametrii in fully connected sunt folositi o singura data in model pentru calcularea unui output
- In convolutie sunt refolositi cu fiecare aplicare
  - Refolosirea transforma output-ul dintr-un singur numar intr-un feature map

Invariance to translation

- Sliding window
- Modul de aplicare a parameter sharing face posibila detectia carecteristilor *oriunde* in poza
- Input-ul este in general mult mai mare dacat *feature-urile*
- Daca shift-am input-ul convolutia va produce in output shift-at



Convolutia. Invariance to translation



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

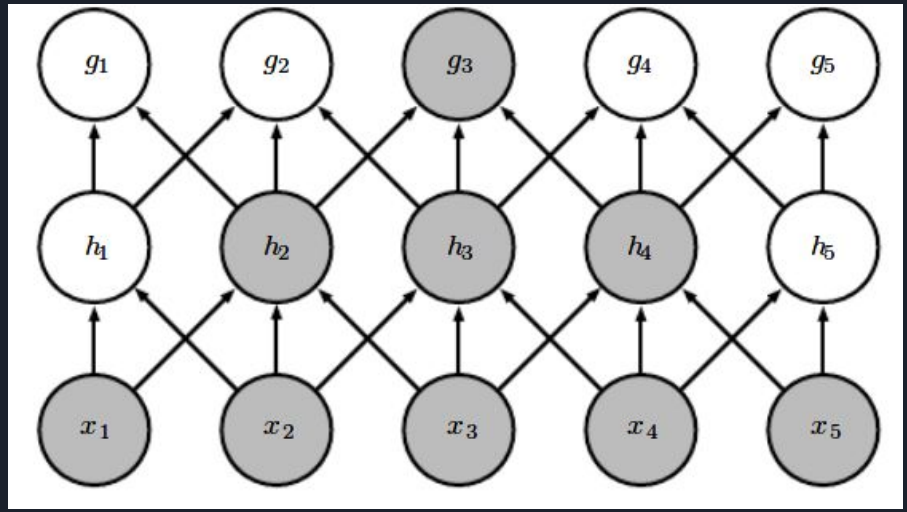
Horizontal

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

Vertical

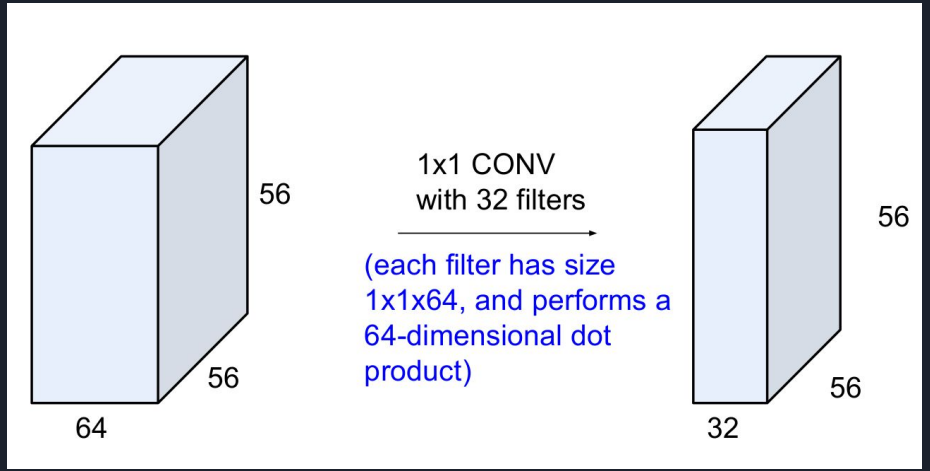
Convolutii. Receptive Field

- Inseamna ce vede un neuron la un pas de convolutie
- Mai multe layere -> deep
- Fiacare convolutie are receptive field mic:
  - 3x3, 5x5, 7x7
- Layer-ele deep vad mult mai mult
- MaxPooling / Convolutia cu stride accelereaza condensarea informatiei spatiale in features deep
- Layer-ele deep sunt conectate *indirect* la o buna parte din poza



Convolutia 1x1

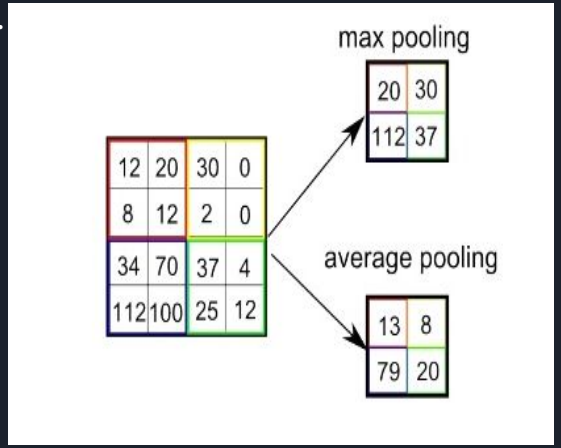
- 32 de filtre 1x1x64 aplicate pe 56x66x64
- Produc output 56x56x32
- Nu altereza dimensiunile spatiale
- Opereaza deptwhise
- Rol de reducere a dimensionalitatii
- Compresie a datelor
- Similar cu PCA
- Util in
  - task-uri de segmentare
  - Inainte de aplicarea unei operatii heavy



Pooling

Max-Pooling :

- Layer care reduce dimensionalitatea datelor de input, prin extragerea maximului din fereastra de input. Intuitiv, activarea maxima dintr-o ferastra corespunde unui feature important, care trebuie pastrat. Restul informatiei este eliminata.
- Are efect similar cu o convolutie cu stride [width,height] != [1,1]



Avg-Pooling :

- Reduce dimensionalitatea datelor de input
- Propaga valoarea medie a inputului din fiecare fereastra.
- Propaga features care sunt mai smooth. Nu elimina informatie
- Va functiona mai prost decat max-pooling pentru cazuri in care edgeuri / texturi sunt importante. Poate functiona mai bine pentru cazuri in care features subtile sunt importante.
  Exemplu : “image constrast learning”

Alternativa: convolutii cu stride

Convolutii Recap

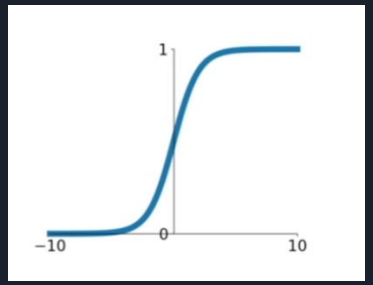
- pastreaza structura spatiala a inputului - fata de FC
- Straturile piramidale au rolul de a distila informatia (1280x720x3 -> Lena Class) si in acelasi timp de a putea determina caracteristici ale imaginii la diferite scale.
- Feature Maps Reminder : fiecare feature map se va specializa (asemenea unui filtru) pe diferite caracteristici ale volumului de input.
- Receptive field

Exemplu : pentru 6 filtre 5*5, obtinem 6 feature maps.
Observatie : daca spatiul parametric al W si b este prea mare, unele feature maps vor fi redundante sau pur si simplu nu se vor activa => nevoia de a constrange reteaua

Comparatie Functii de Activare (1)

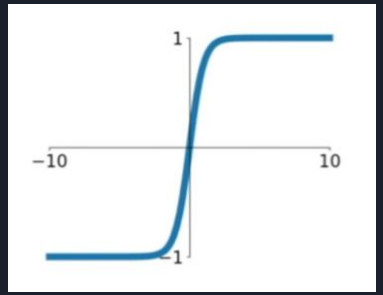
Sigmoid :

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



- Constrange inputul in range-ul [0,1]
- Gradientii din zonele plate sunt 0
- Outputul nu este “centrat la zero”
- “exp()” este o functie computational scumpa

Tanh :

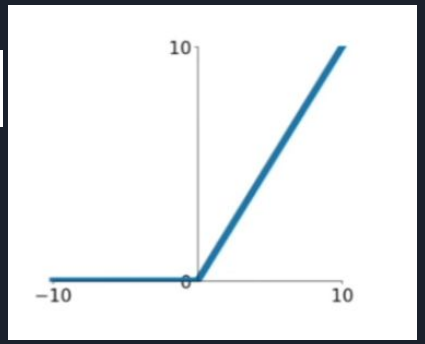


- Constrange inputul in range-ul [-1,1]
- Outputul este “centrat la zero”
- Gradientii din zonele plate sunt 0

Comparatie Functii de Activare (2)

ReLU : (Rectified Liniar Unit)

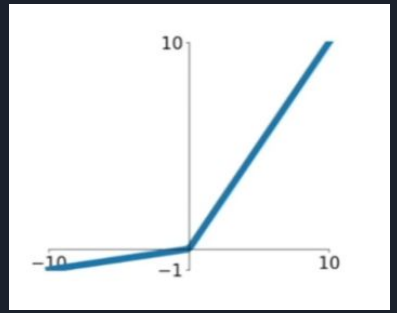
$$f(x) = \max(0,x)$$



- Gradientii sunt non-zero in regiunea “+” (input >0)
- Computational eficienta
- Converge de 6 ori mai rapid decat Sigmoid / Tanh
- Plauzibil biologic
- Outputul nu este “centrat la zero”

Leaky ReLU :

$$f(x) = \max(0.01x, x)$$



- Gradientii sunt non-zero pe tot spectrul inputului!

Parametric ReLU :

$$f(x) = \max(\alpha x, x)$$

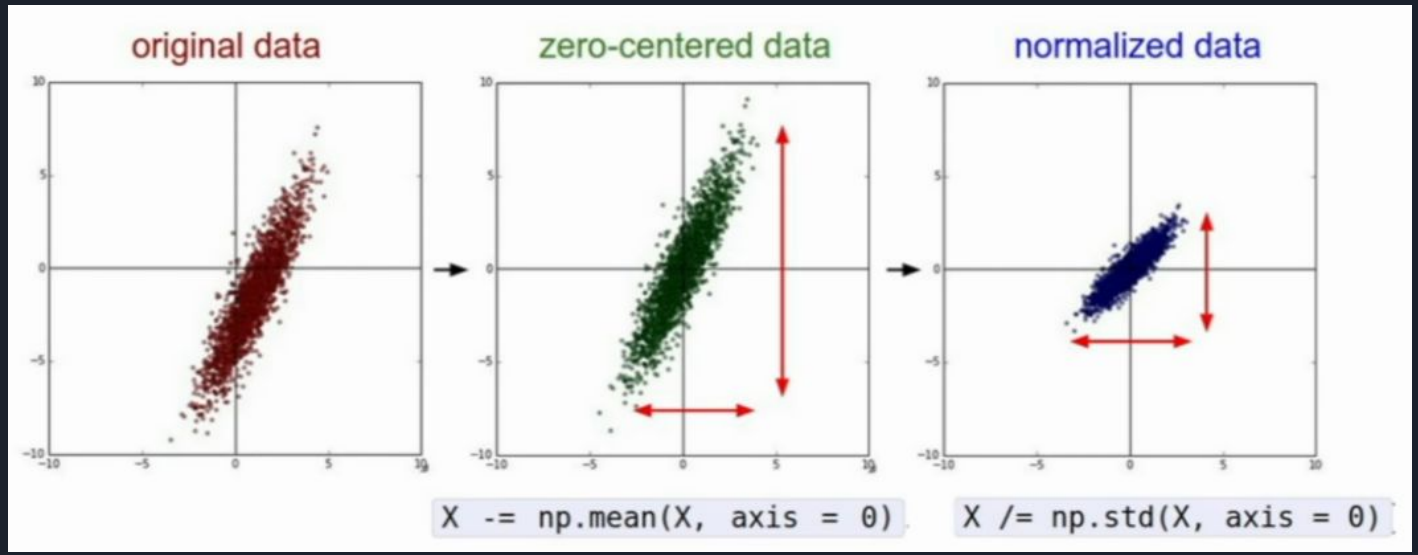
- Alpha determina panta functiei pentru input <0
- Alpha este un parametru prin care facem backpropagation (se invata, prin antrenare)

Preprocesarea Datelor (1)

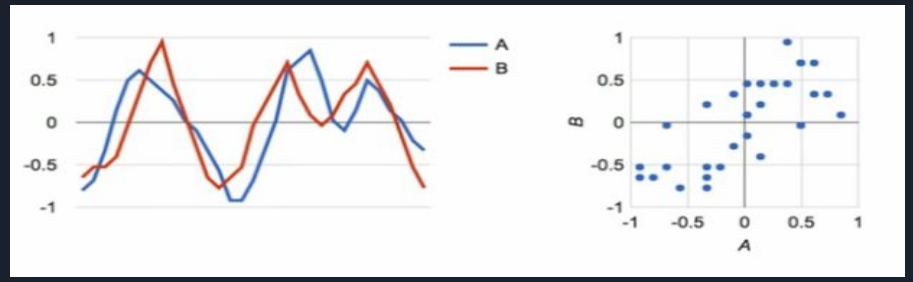
- Centrarea datelor elimina posibilele “biasuri” pe care le poate avea pozitia datelor in spatiu.
- De asemenea, dupa centrare, se pot observa, vizual, relatii surprinzatoare intre doua distributii diferite de date.
- Normalizarea volumului de input se face pentru ca toate caracteristicile acestuia sa contribuie in mod egal. (Daca unele “features” sunt foarte puternic “observabile”, atunci contribuie prin valori mai mari decat restul spatiului de date, care pot sa fie la fel de importante)

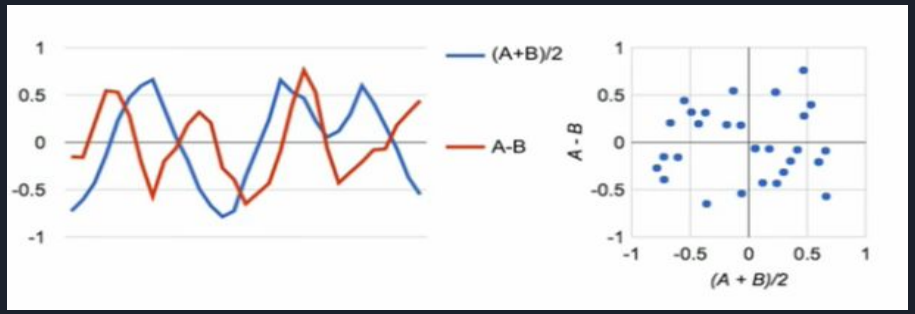
Obs :

* Pentru “vision”, sau image-processing, normalizarea nu este atat de importanta, deoarece formatul valorilor de pixeli este acelasi in toate cazurile [0,256] => imaginile sunt deja normalizate intre ele.



Preprocesarea Datelor (2) / “Data Whitening”





Caz ipotetic : Data points (A,B)

- Initial : A se regaseste in [0 , 20] ; B in [0 , 2]

Aplicam “Data whitening” : scadem media, impartim la standard deviation => zero-centered, rescaled

Observatie : A si B sunt foarte puternic corelate (evolueaza in tandem, reflecta acelasi fenomen)

- In cazul initial, aceasta corelatie era greu de observat, iar activarile provocate de A erau mult mai puternice decat cele provocate de B.

Decorelare : folosim un semnal de average (A+B)/2 si un semnal de diferenta (A-B) => 2 data-seturi suficient de “diferite” incat sa fie folositoare, cu aceeasi pondere, unei retele neurale.

Batch Normalisation (1)

Exemplu : “Albirea Datelor” se poate exprima matematic astfel :

new A	new B	=	A	B	x	0.05 0.61	0.12 -1.23	+	-1.45 0.12

[A’ B’] = [A B] * (matrice de scala / rotatie) + (translatie / shift)

[A’ B’] = [A B] * W + b

Batch Normalization :

- Un layer in retea care determina automat care este “albirea” necesara a datelor.
- Re-normalizeaza outputul unui layer, inainte de activare

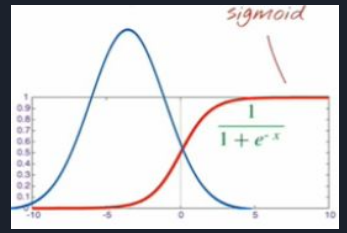
Observatii in practica :

- Biasurile din layere nu mai sunt folositoare! (translatia din BN va prelua acest rol)
- O metoda foarte eficienta de regularizare => se poate scoate DropOut-ul complet
- Activari curate => learning rate mai mare => training mai rapid.

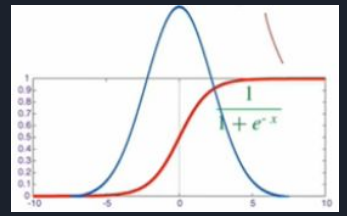
Batch Normalisation (2)

$$\hat{x} = \frac{x - avg_{batch}(x)}{stdev_{batch}(x) + \epsilon}$$

- Calculeaza media si varianta pe fiecare mini-batch
- Centreaza si re-scaleaza outputul unui layer (pregatirea inputului pentru urmatorul layer)
- Are parametrii de scala si translatie “antrenabili” (se invata prin procesul de training in asa fel incat poate sa pastreze expresivitatea unor caracteristici, daca este cazul)

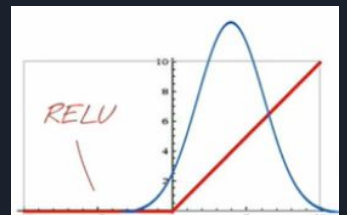


$$BN(x) = \alpha\hat{x} + \beta$$

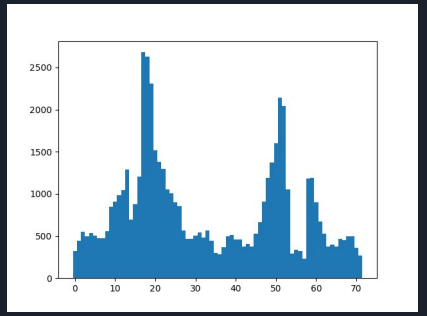


OBS :

- Daca inlocuim alpha cu stdev(x) si beta cu avg(x) obtinem : BN(x) = x
- Oferim retelei neurale sansa de a lasa setul de date neafectat, *daca aceasta este cea mai buna solutie pentru reprezentarea lor*. (ne dorim sa nu distrugem informatia prin “albire”)
- BN se aplica inainte de functia de activare!
- BN adauga 2 grade de libertate suplimentare per neuron => computation cost.



Augmentarea Datelor



Pre-procesare a datasetului care are ca scop balansarea distributiei de date, si cresterea numarului de exemple valide pe care putem sa facem trainng. More Data => More Brains

Tipuri de augmentare de date :

- Flip orizontal / vertical (necesita flip si peste informatia din Ground-Truth anno.)
- Brightness / Contrast / Blur / Hue
- Scale / Shift / Random Crop (necesita modificarea informatiei din GT anno.)
- Synthetic Data
- Distillation peste un dataset nou

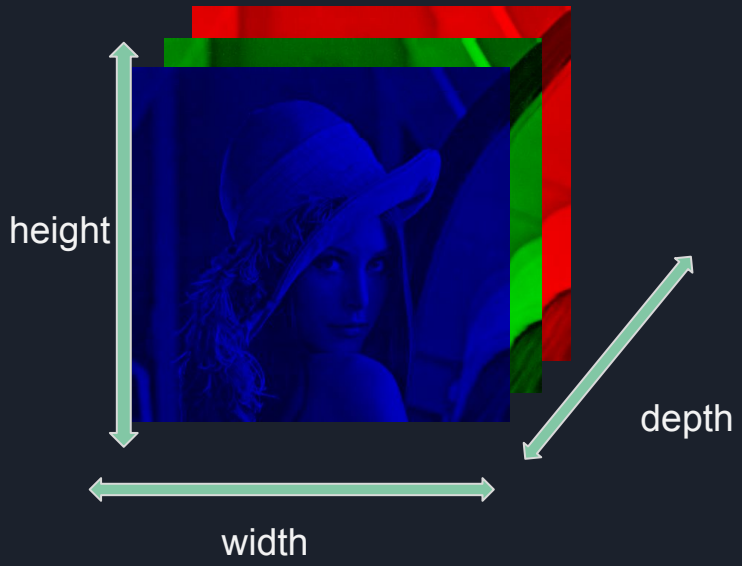
Tips :

- Cu un dataset mai bogat, bine distribuit, putem avea rezultate mai bune decat cu o arhitectura de retea superioara.
- Augmentarea se face in functie de distributia datelor
- In urma augmentarii, trebuie modificati hyper-parametrii legati de numarul de epoci de antrenare.
- TensorFlow are pipeline de augmentare de date (+ optiuni in config files), dar cel mai bine se tine sub control aceasta augmentare inainte de training, cu ploturi si analiza a datasetului.

Stocarea datelor in retele convolutionale

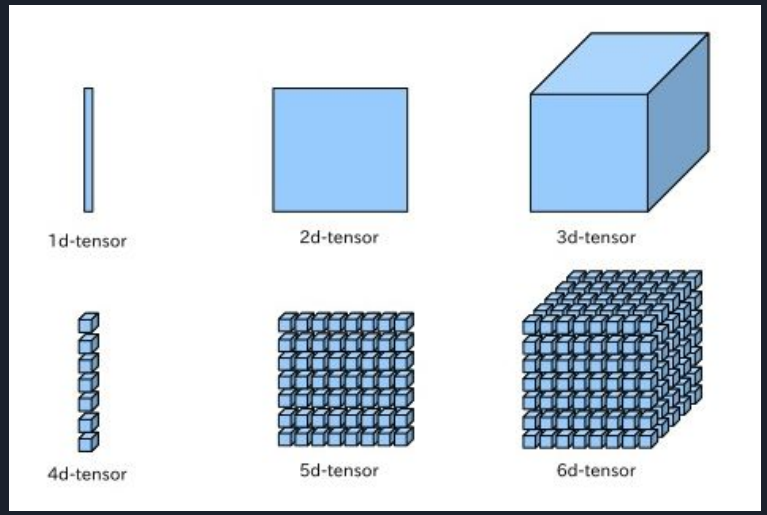
Canale (depth - RGB/BGR)

Stocarea datelor in retele convolutionale



Notiunea de Tensor

- O matrice 3D, 4D, N-D
- O matrice 3D - > o poza, activarile pentru o singura poza - un cub
- Un voum 4D :
  - Un vector de cuburi
  - (batch, height, widht, channels)



TensorFlow Slim

O librarie lightweight peste TF clasic care permite:

- Definiera de modele (include arhitecturi cunoscute ca exemple)
- Antrenarea si evaluarea modelelor complexe
- *argument_scope* - ajuta utilizatorul sa nu se repete - seteaza argumente default pentru operatii
- Queues (async IO)
- Training management:
  - anumite operatii de update (update_ops) nu sunt rulate de prin backprop
    - Batchnorm moving mean and variance update

TensorFlow Slim. Layers

Folosind tensorflow vanilla pentru definirea unei opertii de convolutie 2d este necesar:

- Crearea variabilelor pentru ponderi si bias-uri
- Operatia de convolutie explicita
- Adunarea bias-ului cu rezultatul convolutiei
- Aplicarea functiei de activate

```
import tensorflow as tf

slim = tf.contrib.slim
```

TensorFlow Slim. Layers

TensorFlow Clasic

```
input = ...
with tf.name_scope('conv1_1') as scope:
  kernel = tf.Variable(tf.truncated_normal([3, 3, 64, 128], dtype=tf.float32,
                                           stddev=1e-1), name='weights')
  conv = tf.nn.conv2d(input, kernel, [1, 1, 1, 1], padding='SAME')
  biases = tf.Variable(tf.constant(0.0, shape=[128], dtype=tf.float32),
                       trainable=True, name='biases')
  bias = tf.nn.bias_add(conv, biases)
  conv1 = tf.nn.relu(bias, name=scope)
```

Slim

```
input = ...
net = slim.conv2d(input, 128, [3, 3], scope='conv1_1')
```

TensorFlow Slim. Layers

TensorFlow Clasic

```
input = ...
with tf.name_scope('conv1_1') as scope:
  kernel = tf.Variable(tf.truncated_normal([3, 3, 64, 128], dtype=tf.float32,
                                           stddev=1e-1), name='weights')
  conv = tf.nn.conv2d(input, kernel, [1, 1, 1, 1], padding='SAME')
  biases = tf.Variable(tf.constant(0.0, shape=[128], dtype=tf.float32),
                       trainable=True, name='biases')
  bias = tf.nn.bias_add(conv, biases)
  conv1 = tf.nn.relu(bias, name=scope)
```

Slim

```
input = ...
net = slim.conv2d(input, 128, [3, 3], scope='conv1_1')
```

TensorFlow Slim. Argument Scope

- Furnizeaza activare, regularizare, initializare *default* pentru operatiile specificate

```
def arg_scope([op_list, **default_arguments):
```

- Convolutiile au multi hiperaparametri comuni, care duc la cod duplicat
- Scop: extragerea codului care este la fel de cel care difera
- Asigura o modularizare mai buna
- Clean si usor de urmarit

TensorFlow Slim. Argument Scope

Slim normal

```
padding = 'SAME'
initializer = tf.truncated_normal_initializer(stddev=0.01)
regularizer = slim.l2_regularizer(0.0005)
net = slim.conv2d(inputs, 64, [11, 11], 4,
                  padding=padding,
                  weights_initializer=initializer,
                  weights_regularizer=regularizer,
                  scope='conv1')
net = slim.conv2d(net, 128, [11, 11],
                  padding='VALID',
                  weights_initializer=initializer,
                  weights_regularizer=regularizer,
                  scope='conv2')
net = slim.conv2d(net, 256, [11, 11],
                  padding=padding,
```

Argument Scope

```
with slim.arg_scope([slim.conv2d], padding='SAME',
                    weights_initializer=tf.truncated_normal_initializer(stddev=0.01)
                    weights_regularizer=slim.l2_regularizer(0.0005)):
  net = slim.conv2d(inputs, 64, [11, 11], scope='conv1')
  net = slim.conv2d(net, 128, [11, 11], padding='VALID', scope='conv2')
  net = slim.conv2d(net, 256, [11, 11], scope='conv3')
```

TensorFlow Slim. Stacking

Code duplication. Sa consideram reteaua

```
net = ...
net = slim.conv2d(net, 256, [3, 3], scope='conv3_1')
net = slim.conv2d(net, 256, [3, 3], scope='conv3_2')
net = slim.conv2d(net, 256, [3, 3], scope='conv3_3')
net = slim.max_pool2d(net, [2, 2], scope='pool2')
```

Slim

```
net = ...
for i in range(3):
  net = slim.conv2d(net, 256, [3, 3], scope='conv3_%d' % (i+1))
net = slim.max_pool2d(net, [2, 2], scope='pool2')
```