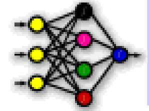
Etude d’un exemple

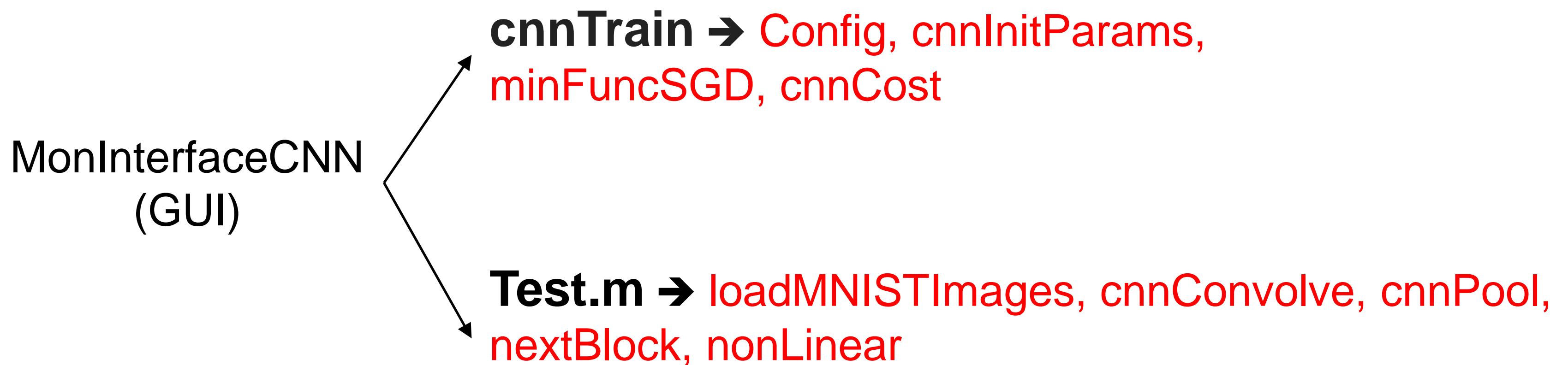


Cours 7

Cnn-master

© A. Belaïd

Mon schéma de cnn



© A. Belaïd

cnnTrain

```
% Etape 1: Configurer la structure du réseau
cnnConfig = config();
%  Initialiser les paramètres sur le réseau ainsi configuré
[theta meta] = cnnInitParams(cnnConfig);
% Etape 2: Charger les données MNIST et leurs labels, mettre les images à la taille indiquée dans la structure du réseau
images = loadMNISTImages('train-images-idx3-ubyte');
d = cnnConfig.layer{1}.dimension;
images = reshape(images,d(1),d(2),d(3),[]);
labels = loadMNISTLabels('train-labels-idx1-ubyte');
labels(labels==0) = 10; % Remap 0 to 10
```

```
%Etape 3 : cnn Train : fixer les paramètres d'apprentissage
    options.epochs = 3;
    options.minibatch = 128;
    options.alpha = 1e-1;
    options.momentum = .95;

% Implementer le gradient stochastique et la fonction de coût
    opttheta = minFuncSGD(@(x,y,z) cnnCost(x,y,z,cnnConfig,meta),theta,images,labels,options);
```

cnnTrain (suite)

```
% Etape 4: Tester les performances du modèle ainsi entraîné en utilisant l'ensemble de test : MNIST. La précision doit être au-dessus de 97% après 3 epochs

testImages = loadMNISTImages('t10k-images-idx3-ubyte');
testImages = reshape(testImages,d(1),d(2),d(3),[]);
testLabels = loadMNISTLabels('t10k-labels-idx1-ubyte');
testLabels(testLabels==0) = 10; % Remap 0 to 10

% Calcul de la précision avec la fonction de coût  sur les données préparées
[cost,grad,preds]=cnnCost(opttheta,testImages,testLabels,cnnConfig,meta,true);

acc = sum(preds==testLabels)/length(preds);

fprintf('Accuracy is %f\n',acc);

%diary off;
```

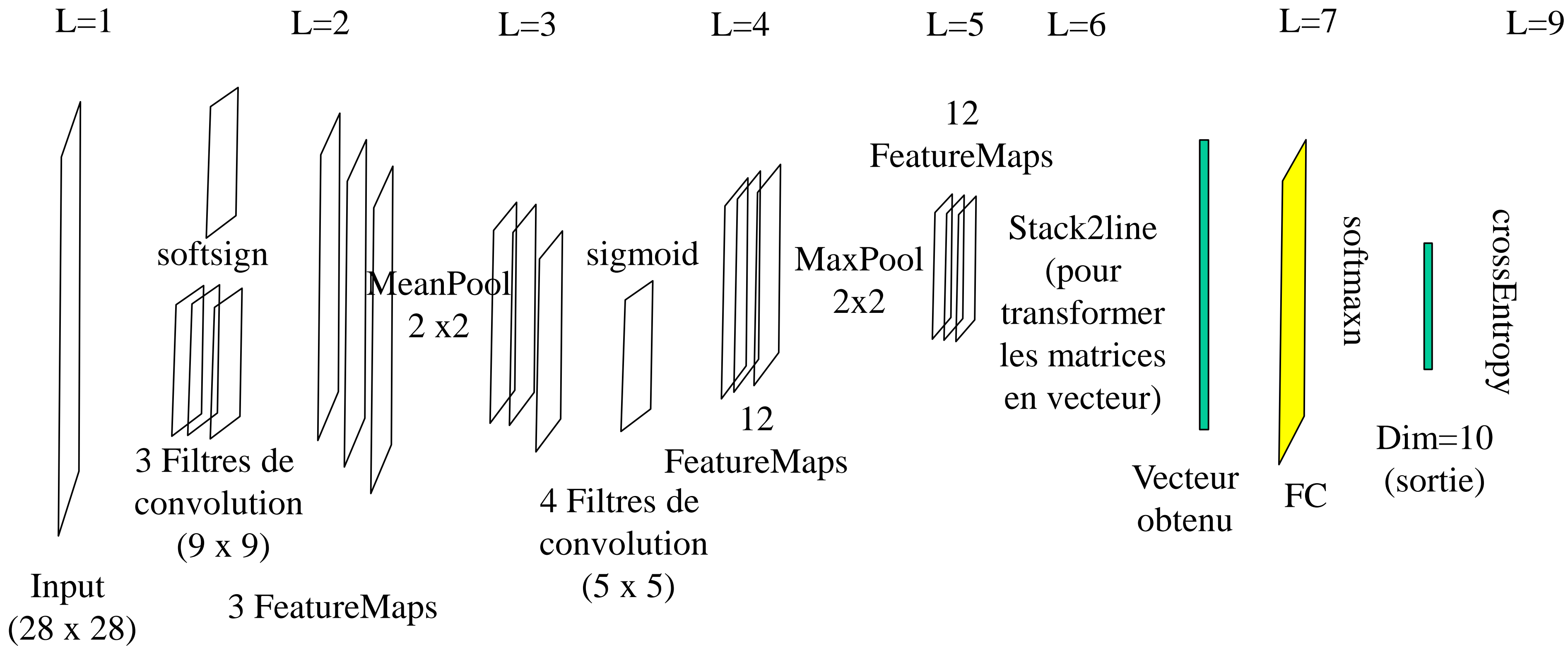
© A. Belaïd

config

```
function cnnConfig = config()
cnnConfig.layer{1}.type = 'input';
cnnConfig.layer{1}.dimension = [28 28 1];

cnnConfig.layer{2}.type = 'conv';
cnnConfig.layer{2}.filterDim = [9 9];
cnnConfig.layer{2}.numFilters = 20;
cnnConfig.layer{2}.nonLinearType =
'sigmoid';
cnnConfig.layer{2}.conMatrix =
ones(1,20);

cnnConfig.layer{3}.type = 'pool';
cnnConfig.layer{3}.poolDim = [2 2];
cnnConfig.layer{3}.poolType = 'meanpool';

cnnConfig.layer{4}.type = 'stack2line';

cnnConfig.layer{5}.type = 'sigmoid';
cnnConfig.layer{5}.dimension = 360;

cnnConfig.layer{6}.type = 'sigmoid';
cnnConfig.layer{6}.dimension = 60;

cnnConfig.layer{7}.type = 'softmax';
cnnConfig.layer{7}.dimension = 10;

cnnConfig.costFun = 'crossEntropy';
end
```

© A. Belaïd

Architecture



Config à tester

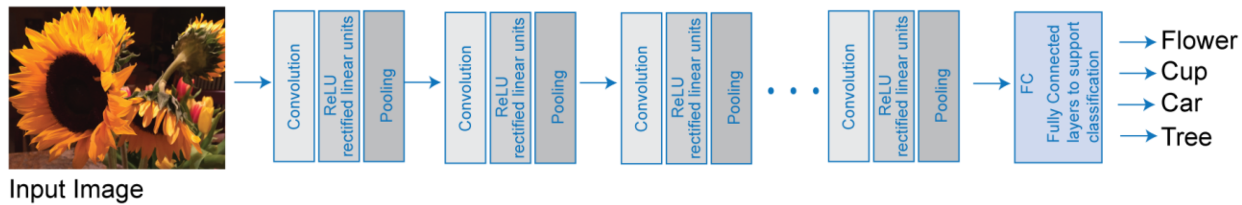
```
function cnnConfig = config()
l = 1;
cnnConfig.layer{l}.type = 'input';
cnnConfig.layer{l}.dimension = [224 224
1];
l = l + 1;
cnnConfig.layer{l}.type = 'conv';
cnnConfig.layer{l}.filterDim = [5 5];
cnnConfig.layer{l}.numFilters = 16;
cnnConfig.layer{l}.nonLinearType = 'relu';
cnnConfig.layer{l}.conMatrix = ones(1,16);
l = l + 1;
cnnConfig.layer{l}.type = 'pool';
cnnConfig.layer{l}.poolDim = [5 5];
cnnConfig.layer{l}.poolType = 'meanpool';
l = l + 1;
cnnConfig.layer{l}.type = 'conv';
cnnConfig.layer{l}.filterDim = [5 5];
cnnConfig.layer{l}.numFilters = 16;
cnnConfig.layer{l}.nonLinearType = 'relu';
cnnConfig.layer{l}.conMatrix =
ones(16,16); l = l + 1;
cnnConfig.layer{l}.type = 'pool';
cnnConfig.layer{l}.poolDim = [5 5];
cnnConfig.layer{l}.poolType = 'maxpool';
l = l + 1;
cnnConfig.layer{l}.type = 'stack2line';
l = l + 1;
cnnConfig.layer{l}.type = 'relu';
cnnConfig.layer{l}.dimension = 512;
l = l + 1;
cnnConfig.layer{l}.type = 'relu';
cnnConfig.layer{l}.dimension = 256;
l = l + 1;
cnnConfig.layer{l}.type = 'softmax';
cnnConfig.layer{l}.dimension = 106;
l = l + 1;
cnnConfig.costFun = 'crossEntropy';
end
```

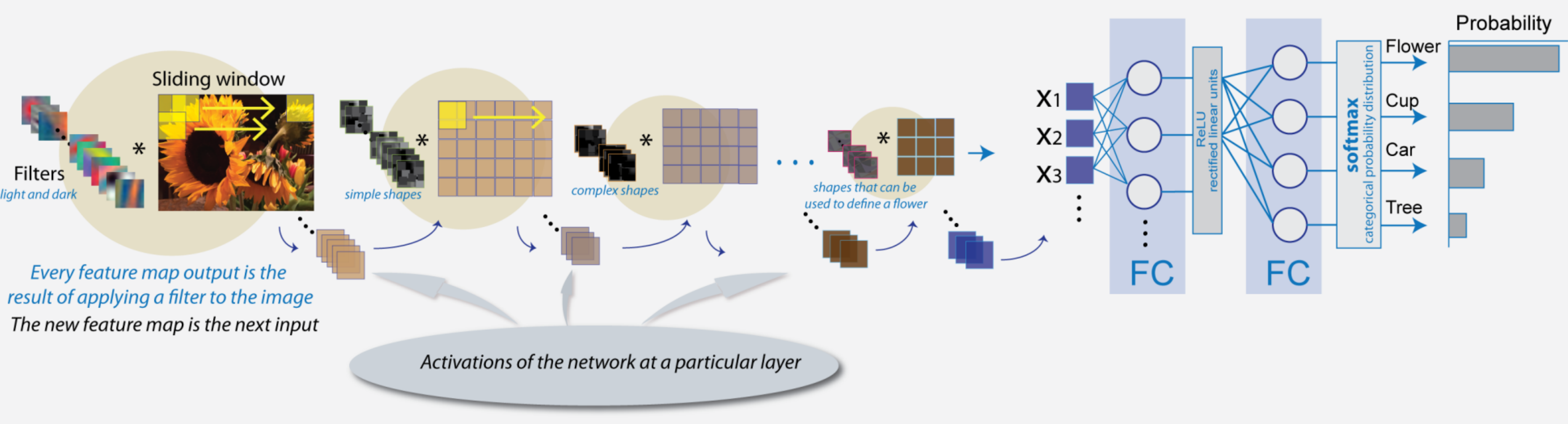
© A. Belaïd

Config à tester

The learning parameters:

- options.epochs = 3;
- options.minibatch = 128;
- options.alpha = 0.01;
- options.momentum = .9;


Input Image
Convolution
ReLU
rectified linear units
Pooling
FC
Fully Connected layers to support classification
Flower
Cup
Car
Tree

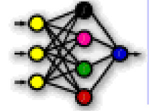

Sliding window
Filters
light and dark
simple shapes
complex shapes
shapes that can be used to define a flower
Every feature map output is the result of applying a filter to the image
The new feature map is the next input
Activations of the network at a particular layer
X1
X2
X3
FC
ReLU
rectified linear units
FC
softmax
categorical probability distribution
Probability
Flower
Cup
Car
Tree

configTestGradient

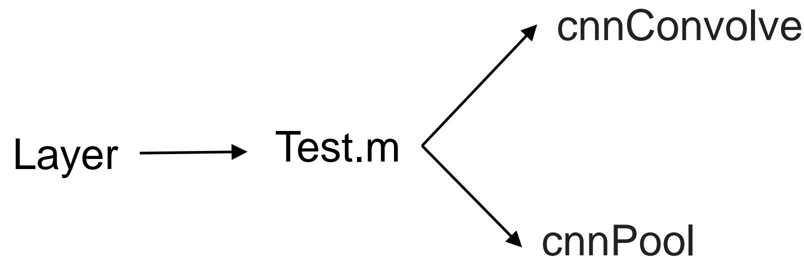
```
l = 1;
cnnConfig.layer{l}.type = 'input';
cnnConfig.layer{l}.dimension = [28 28 1];
l = l + 1;
cnnConfig.layer{l}.type = 'conv';
cnnConfig.layer{l}.filterDim = [9 9];
cnnConfig.layer{l}.numFilters = 3;
cnnConfig.layer{l}.nonLinearType = 'softsign';
cnnConfig.layer{l}.conMatrix = ones(1,3);
l = l + 1;
cnnConfig.layer{l}.type = 'pool';
cnnConfig.layer{l}.poolDim = [2 2];
cnnConfig.layer{l}.poolType = 'meanpool';
l = l + 1;
cnnConfig.layer{l}.type = 'conv';
cnnConfig.layer{l}.filterDim = [5 5];
cnnConfig.layer{l}.numFilters = 4;
cnnConfig.layer{l}.nonLinearType = 'sigmoid';
cnnConfig.layer{l}.conMatrix = ones(3,4);
l = l + 1;
cnnConfig.layer{l}.type = 'pool';
cnnConfig.layer{l}.poolDim = [2 2];
cnnConfig.layer{l}.poolType = 'maxpool';
l = l + 1;
cnnConfig.layer{l}.type = 'stack2line';
l = l + 1;
% cnnConfig.layer{l}.type = 'tanh';
% cnnConfig.layer{l}.dimension = 60;
% l = l + 1;
cnnConfig.layer{l}.type = 'softmax';
cnnConfig.layer{l}.dimension = 10;
l = l + 1;
cnnConfig.costFun = 'crossEntropy';
end
```

Les fonctions Matlab

Layer : Convolution et pooling



Ce répertoire contient du code qui vous aide à démarrer l'exercice de convolution et de mise en commun (pooling). Dans ce répertoire, vous devez uniquement utiliser Test.m, cnnConvolve.m et cnnPool.m



© A. Belaïd

Test.m

```
% STEP 0: Initialisation des paramètres utilisés ici et chargement des données

imageDim = 28;
filterDim = 8;
numFilters = 100; % nb de feature maps
numImages = 60000;
poolDim = 3;   % dimension de la region de pooling
% Chargement des images MNIST pour l'apprentissage
addpath(genpath('C:\Abelaid\1-…\Dataset'));
images = loadMNISTImages('C:\Abelaid\1-…\MNIST\train-images-idx3-ubyte');
images = reshape(images,imageDim,imageDim,1,numImages);

W = randn(filterDim,filterDim,1,numFilters);
b = rand(numFilters);
```

```
% STEP 1: on implémente la convolution et on la teste sur une petite partie des données

% STEP 1a: Implementer la convolution
% Utiliser uniquement 8 images pour tester
convImages = images(:, :, 1,1:8);
convolvedFeatures = cnnConvolve(convImages, W, b);
```

Test.m (suite 1)

```
%STEP 1b: Verifier la convolution
%  Pour s'assurer que vous avez convolué les caractéristiques correctement, nous avons fourni du code pour comparer les résultats de votre convolution avec des activations de l'auto-encodeur sparse (entrées = sorties)

% Pour 1000 points aléatoires
for i = 1:1000
    filterNum = randi([1, numFilters]);
    imageNum = randi([1, 8]);
    imageRow = randi([1, imageDim - filterDim + 1]);
    imageCol = randi([1, imageDim - filterDim + 1]);

    patch = convImages(imageRow:imageRow + filterDim - 1, imageCol:imageCol + filterDim - 1,1, imageNum);
feature = sum(sum(patch.*W(:,:,1,filterNum)))+b(filterNum);
feature = 1./(1+exp(-feature));

if abs(feature - convolvedFeatures(imageRow, imageCol,filterNum, imageNum)) > 1e-9
        fprintf('Convolved feature does not match test feature\n');
        fprintf('Filter Number : %d\n', filterNum);
        ….
    end
end
disp('Congratulations! Your convolution code passed the test.');
```

© A. Belaïd

Test.m (suite 2)

```
% STEP 2: Implementer et tester le pooling

% STEP 2a: Implementer le pooling
pooledFeatures = cnnPool([poolDim
poolDim], convolvedFeatures, 'meanpool');

% STEP 2b : verifier votre pooling
% Pour vous assurer que vous avez mis en
œuvre correctement le pooling, nous allons
utiliser votre fonction de pooling sur une
matrice de test et vérifier les résultats

testMatrix = reshape(1:64, 8, 8);
expectedMatrix = [mean(mean(testMatrix(1:4,
1:4))) mean(mean(testMatrix(1:4, 5:8))); ...
            mean(mean(testMatrix(5:8, 1:4)))
mean(mean(testMatrix(5:8, 5:8))); ];

testMatrix = reshape(testMatrix, 8, 8, 1, 1);

 pooledFeatures = squeeze(cnnPool([4 4],
testMatrix, 'meanpool'));

if ~isequal(pooledFeatures, expectedMatrix)
   disp('Pooling incorrect');
   …
else
   disp('Congratulations! Your pooling code
passed the test.');
end
```

© A. Belaïd

cnnConvolve

%cnnConvolve Retourne la convolution des entités données par W et b avec les images données

% Paramètres :

- images à convoluer avec une matrice sous la forme : images(row, col, channel, image number)
- W est de la forme : (filterDim,filterDim,channel,numFilters)
- b est de la forme : (numFilters,1)
- nonlineartype : type de la non-linearité: 'sigmoid' : default, 'relu, 'tanh‘, 'softsign‘…
- con_matrix: la connexion entre le canal d'entrée et les cartes de sortie. Si le ième canal d'entrée a une connexion avec la jème carte de sortie, alors con_matrix(i,j) = 1, sinon, 0;

% Retourne : convolvedFeatures – matrice des caractéristiques convoluées sous la forme :

convolvedFeatures(imageRow, imageCol, featureNum, imageNum)

```
[filterDimRow,filterDimCol,channel,numFilters] = size(W);
if ~exist('con_matrix','var') || isempty(con_matrix)
    con_matrix = ones(channel, numFilters);
end
if ~exist('nonlineartype','var')
    nonlineartype = 'sigmoid';
end
if ~exist('shape','var')
    shape = 'valid';
end
[imageDimRow, imageDimCol,~, numImages] = size(images);
convDimRow = imageDimRow - filterDimRow + 1;
convDimCol = imageDimCol - filterDimCol + 1;
convolvedFeatures = zeros(convDimRow, convDimCol, numFilters, numImages);
```

cnnConvolve (suite 1)

```
%   Convolue  chaque filtre avec chaque image
pour produire : :convolvedFeatures utilisant
convolvedFeatures(imageRow, imageCol,
featureNum, imageNum)

for imageNum = 1:numImages
  for filterNum = 1:numFilters
      convolvedImage = zeros(convDimRow,
convDimCol);
      for channelNum = 1:channel
          if con_matrix(channelNum,filterNum) ~= 0
            % Obtenir thla caractéristique souhaitée
(filterDim x filterDim) pendant la convolution
            filter = W(:,:,channelNum,filterNum);
            % Retourner la matrice de
caractéristiques en raison de la définition de
convolution, comme expliqué plus tard
filter = rot90(squeeze(filter),2);

% Obtenir l'image
            im = squeeze(images(:, :,
channelNum,imageNum));

            % Convolue "filter" avec "im", en
ajoutant lee resultat à convolvedImage
            % s'assurer de faire une convolution
'valide'
            convolvedImage = convolvedImage
+ conv2(im, filter, shape);
          end
            % Ajouter l'unite de biais
      end
      convolvedImage = convolvedImage +
b(filterNum);
% on obtient filterNum featureMaps
      convolvedFeatures(:, :, filterNum,
imageNum) = convolvedImage;
  end
end
```

© A. Belaid

cnnConvolve (suite 2)

```
linTrans = convolvedFeatures;
%Effectuer la non-linéarité
switch nonlineartype
    case 'sigmoid'
        convolvedFeatures = 1./(1+exp(-convolvedFeatures));
    case 'relu'
        convolvedFeatures = max(0,convolvedFeatures);
    case 'tanh'
        convolvedFeatures = tanh(convolvedFeatures);
    case 'softsign'
        convolvedFeatures = convolvedFeatures ./ (1 + abs(convolvedFeatures));
    case 'none'
        % ne pas faire de nonlinearty
    otherwise
        fprintf('error: no such nonlieartype%s',nonlineartype);
end
end
```

cnnPool

%cnnPool : regroupe (pools) les caractéristiques convoluées

% Paramètres:

- poolDim : dimension de la region de pooling, c’est un vecteur 2D : (poolDimRow poolDimCol);
- convolvedFeatures : les caractéristiques convoluées à regrouper (celles données par cnnConvolve)
- convolvedFeatures(imageRow, imageCol, featureNum, imageNum)

% Retourne :

- pooledFeatures – matrice des caractéristiques regroupées sous la forme :

  pooledFeatures(poolRow, poolCol, featureNum, imageNum)
- weights : exprimant combine l’entrée contribute à la sortie

© A. Belaid

cnnPool (suite 1)

```
if nargin < 3
    pooltypes = 'meanpool';
end

numImages = size(convolvedFeatures, 4);
numFilters = size(convolvedFeatures, 3);
convolvedDimRow = size(convolvedFeatures,
1);
convolvedDimCol = size(convolvedFeatures,
2);
pooledDimRow = floor(convolvedDimRow /
poolDim(1));
pooledDimCol = floor(convolvedDimCol /
poolDim(2));

weights = zeros(size(convolvedFeatures));
featuresTrim =
convolvedFeatures(1:pooledDimRow*poolDim(
1),1:pooledDimCol*poolDim(2),:,:);

if strcmp(pooltypes, 'meanpool')
    weights(1:pooledDimRow*poolDim(1),
1:pooledDimCol*poolDim(2),:,:) =
ones(size(featuresTrim)) / poolDim(1) /
poolDim(2);
end

pooledFeatures = zeros(pooledDimRow,
pooledDimCol, numFilters, numImages);

poolFilter = ones(poolDim) *
1/poolDim(1)/poolDim(2);
```

© A. Belaid

cnnPool (suite 2)

```
for imageNum = 1:numImages
    for filterNum = 1:numFilters
    features = featuresTrim(:,:,filterNum,
imageNum);
    switch pooltypes
        case 'meanpool'
            poolConvolvedFeatures =
conv2(features,poolFilter,'valid');
            pooledFeatures(:,:,filterNum,imageNum)
=
poolConvolvedFeatures(1:poolDim:end,1:poolDi
m:end);
pooledFeatures(:,:,filterNum,imageNum) =
blockproc(features,poolDim,@maxblock);
case 'maxpool'
      temp = im2col(features, poolDim, 'distinct');
      [m, i] = max(temp);
      temp = zeros(size(temp));
      temp(sub2ind(size(temp),i,1:size(i,2))) = 1;
weights(1:pooledDimRow*poolDim(1),1:pool
edDimCol*poolDim(2),filterNum,imageNum)
= col2im(temp,
poolDim,[pooledDimRow*poolDim(1)
pooledDimCol*poolDim(2)], 'distinct');

pooledFeatures(:,:,filterNum,imageNum) =
reshape(m, size(pooledFeatures,1),
size(pooledFeatures,2));


otherwise
            error('wrongLayertype:
%s',pooltypes);
end
end
end
end
function b= maxblock(a)
    b = max(max(a.data));
end
```

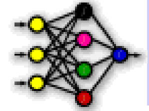
minFuncSGD

uçh

uygèygg

Les fonctions Matlab

Demo : initialisation et apprentissage



Demo → cnnTrain

- Config : configure les layers du réseau (7) en donnant les dimensions, les dim des filtres, la function à seuil utilisée, et la function de coût } Demo
- cnnInitParams : prend les paramètres dans Config et prépare l'architecture pour chaque type de layer : input, convolution, pooling, …
- cnnCost : Calcule le coût et le gradient : backbropgation } TrainingMethod
- loadMNISTImages : ouvre le fichier d'apprentissage et range les données dans le fichier images
- loadMNISTLabels : crée le vecteur label } MNIST