ARTIFICIAL NEURAL NETWORK LAB CLASS GUIDE

Task 1 – Compute a neuron output

Objective: Calculate and visualize the output of a single neuron.

1) Define neuron parameters

```
% Neuron weights
w = [4 -2];
% Neuron bias
b = -3;
% Activation function: Hyperbolic tangent sigmoid function
func = 'tansig';

% Activation function: Logistic sigmoid transfer function
% func = 'logsig'

% Activation function: Hard-limit transfer function (threshold)
% func = 'hardlim'

% Activation function: Linear transfer function
% func = 'purelin'
```

2) Define input vectors

```
p = [2 3];
```

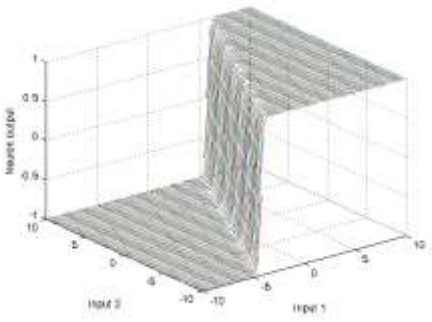
3) Calculate neuron output

```
% Aggregation function
net_input = p*w'+b;

% Activation function
neuron_output = feval(func, net_input)
```

4) Plot neuron output over the range of inputs

```
[p1,p2] = meshgrid(-10:.25:10);
z = feval(func, [p1(:) p2(:)]*w'+b );
z = reshape(z,length(p1),length(p2));
plot3(p1,p2,z);
grid on;
xlabel('Input 1');
ylabel('Input 2');
zlabel('Neuron output');
```



5) Change the activation function and plot neuron output again to see the different output surfaces.

Activation/Transfer functions:
hardlim: Positive hard limit transfer function; hardlims: Symmetric hard limit transfer function; purelin: Linear transfer function; satlin: Positive saturating linear transfer function; logsig: Logistic sigmoid transfer function; tansig: Hyperbolic tangent sigmoid symmetric transfer function

Task 2 – Analyze a single neuron

Objective: Analyze the change in the output of a single neuron when changing the weight, the bias and the transfer function.

1) Run demo nnd2n1

nnd2n1

2) Study how the different values of **weight**, **bias**, **transfer function** and **input p** modify the output of the neuron.

Task 3 – Changing the number of neurons

Objective: Analyse the change in the number of neurons in the hidden layer. How increasing of hidden layer neurons affects to function approximation? Are there any side-effects if number of hidden layer neurons is high?

1) Run demo nnd11gn

nnd11gn

Task 4 – Classification of linearly separable data with a Rosenblatt's perceptron

Objective: Two clusters of data, belonging to two classes, are defined in a 2-dimensional input space. Classes are linearly separable. The task is to construct a Perceptron for the classification of data.

Recall: The simplest kind of neural network is a *single-layer perceptron* network, which consists of a single layer of output nodes; the inputs are fed directly to the outputs via a series of weights. In this way it can be considered the simplest kind of feed-forward network. Perceptrons can be trained by the Rosenblatt's algorithm. It calculates the errors between calculated output and sample output data, and uses this to create an adjustment to the weights, thus implementing a form of gradient descent.

1) Define input and output data

```
close all, clear all, clc
% number of samples of each class
N = 20;
% define inputs and outputs
offset = 5; % offset for second class
x = [randn(2,N) randn(2,N)+offset]; % inputs
y = [zeros(1,N) ones(1,N)]; % outputs
% Plot input samples with plotpv (Plot perceptron input/target vectors)
figure(1)
plotpv(x,y);
```

2) Create and train the perceptron

```
net = perceptron;

% Take a look to the default parameters of this perceptron. Notice that the performance/cost %function used is mae (mean absolute error), the training function is trainc (trains a network %with weight and bias learning rules with incremental updates) and the transfer function %used is hardlim (Hard-limit).

net = train(net, x, y);
view(net);
```

3) Plot decision boundary

```
figure(1)
plotpc(net.IW{1},net.b{1});
% Plot a classification line on a perceptron vector plot
```

Task 5a – Classification of a 4-class problem with a perceptron

Objective: Perceptron network with 2-inputs and 2-outputs is trained to classify input vectors into 4 categories.

1) Define input and output data

```
close all, clear all, clc
% number of samples of each class
K = 30;
% define classes
q = .6; % offset of classes
A = [rand(1,K)-q; rand(1,K)+q];
B = [rand(1,K)+q; rand(1,K)+q];
C = [rand(1,K)+q; rand(1,K)-q];
D = [rand(1,K)-q; rand(1,K)-q];
% plot classes
plot(A(1,:),A(2,:),'bs')
hold on
grid on
plot(B(1,:),B(2,:),'r+')
```

```
plot(C(1,:),C(2,:),'go')
plot(D(1,:),D(2,:),'m*')

% text labels for classes
text(.5-q,.5+2*q,'Class A')
text(.5+q,.5+2*q,'Class B')
text(.5+q,.5-2*q,'Class C')
text(.5-q,.5-2*q,'Class D')

% define output coding for classes. This coding is used for visualizations purposes.
a = [0 1]';
b = [1 1]';
c = [1 0]';
d = [0 0]';
```

2) Prepare inputs and outputs for perceptron training

```
% define inputs (combine samples from all four classes)
P = [A B C D];
% define targets
T = [repmat(a,1,length(A)) repmat(b,1,length(B)) ...  % repmat: Replicate and tile an array
repmat(c,1,length(C)) repmat(d,1,length(D)) ];
plotpv(P,T);
```

3) Create a perceptron

```
net = perceptron;
```

4) Train a perceptron (step by step in order to allow the visual adjustment of the network). *Adapt* returns a new network object trained one additional step, the network output, and the error. This loop allows the network to adapt, plots the classification line and continues until the error is zero.

```
% To see the adaptation you need to look at the plot while the code is running
E = 1;
net.adaptParam.passes = 1;
linehandle = plotpc(net.IW{1},net.b{1});
n = 0;
while (sse(E) & n<1000)  % sse: Sum squared error or mae: mean absolute error
    n = n+1;
    [net,Y,E] = adapt(net,P,T);
    linehandle = plotpc(net.IW{1},net.b{1},linehandle);
    drawnow;
    pause(2); % 2 seconds pause
end

% show perceptron structure
view(net);
```

5) How to use trained perceptron (simulation of the network with new data)

```
% For example, classify an input vector of [0.7; 1.2]
p = [0.7; 1.2]
y = net(p)
```

```
% compare response with output coding (a,b,c,d)
```

Task 5b – Perceptron cannot learn not linearly separable data

Objective: When the data is not linearly separable perceptron network is not an option.

1) Create input data

```
close all, clear all, clc
% number of samples of each cluster
K = 100;
% offset of clusters
q = .6;
% define 2 groups of input data
A1 = [rand(1,K)-q; rand(1,K)+q];
B1 = [rand(1,K)+q; rand(1,K)+q];
C1 = [rand(1,K)+q; rand(1,K)-q];
D1 = [rand(1,K)-q; rand(1,K)-q];
A = [A1 C1];
B = [B1 D1];
% plot data
plot(A(1,:),A(2,:),'k+',B(1,:),B(2,:),'b*')
xlim([-2,3]);
ylim([-2,3]);
grid on
hold on
```

2) Define output coding

```
% coding (+1/0) for 2-class XOR problem
a = 0;
b = 1;
```

3) Prepare inputs and outputs for network training

```
% define inputs (combine samples from all two classes)
P = [A B];
% define targets
T = [repmat(a,1,length(A)) repmat(b,1,length(B))];
```

4) Create a perceptron

```
net = perceptron;
```

5) Train a perceptron (step by step in order to allow the visual adjustment of the network). *Adapt* returns a new network object trained one additional step, the network output, and the error. This loop allows the network to adapt, plots the classification line and continues until the error is zero.

```
% To see the adaptation you need to look at the plot while the code is running
E = 1;
net.adaptParam.passes = 1;
linehandle = plotpc(net.IW{1},net.b{1});
n = 0;
while (sse(E) & n<20)  % sse: Sum squared error or mae: mean absolute error
```

```
    n = n+1;
    [net,Y,E] = adapt(net,P,T);
    linehandle = plotpc(net.IW{1},net.b{1},linehandle);
    drawnow;
    pause(2); % 2 seconds pause
end

% show perceptron structure
view(net);
```

Task 6a – Solving XOR problem with a multilayer perceptron

Objective: 2 groups of linearly inseparable data (A,B) are defined in a 2-dimensional input space. The task is to define a neural network for solving the XOR classification problem.

In this case the Matlab feedforwardnet is used maintaining all its default parameters

1) Create input data

```
close all, clear all, clc
% number of samples of each cluster
K = 100;
% offset of clusters
q = .6;
% define 2 groups of input data
A1 = [rand(1,K)-q; rand(1,K)+q];
B1 = [rand(1,K)+q; rand(1,K)+q];
C1 = [rand(1,K)+q; rand(1,K)-q];
D1 = [rand(1,K)-q; rand(1,K)-q];
A = [A1 C1];
B = [B1 D1];
% plot data
plot(A(1,:),A(2,:),'k+',B(1,:),B(2,:),'b*')
grid on
hold on
```

2) Define output coding

```
% coding (+1/0) for 2-class XOR problem (plots will be more interpretable with codes 0/1 instead of [0 1]/[1/0])
a = 0;
b = 1;
```

3) Prepare inputs and outputs for network training

```
% define inputs (combine samples from all two classes)
P = [A B];
% define targets
T = [repmat(a,1,length(A)) repmat(b,1,length(B))];
```

4) Create and train a multilayer perceptron

```
% create a neural network
% Input, output and output layers sizes are set to 0. These sizes will automatically be
% configured to match particular data by train
```

```
net = feedforwardnet([20]);  % We create a network with one hidden layer of 20 neurons (try
                             %also with [5 3])

% look at the default parameters
% ... mse, purelin, …
%

net

% train the neural network
[net,tr,Y,E] = train(net,P,T);
% Notice that tr has the indices of the training, validation and test data sets:
% tr.trainInd
% tr.valInd
% tr.testInd

% show network
view(net)

% Accuracy
fprintf('Accuracy: %f\n',100-100*sum(abs((Y>0.5)-T))/length(T))
```

5) Plot targets and network response to see how good the network learns the data

```
figure(2)
plot(T','linewidth',2)
hold on
plot(Y','r--')
grid on
legend('Targets','Network response','location','best')
ylim([-1.25 1.25])
```

6) Plot classification result for the complete input space (separation by hyperplanes)

```
% generate a grid
span = -1:.005:2;
[P1,P2] = meshgrid(span,span);
pp = [P1(:) P2(:)]';
% simulate neural network on a grid
aa = net(pp);
% plot classification regions
figure(1)
mesh(P1,P2,reshape(aa,length(span),length(span))-5);
colormap cool
view(2)

% show network
view(net)
```

Task 6b – Solving XOR problem with a multilayer perceptron

The Matlab default parameters for the feedforwardnet, seen in the previous task (6a), are commonly used for regression.

Now, you are going to set the parameters well suited for a classification problem

```
close all, clear all, clc
```

```
% number of samples of each cluster
K = 100;
% offset of clusters
q = .6;
% define 2 groups of input data
A1 = [rand(1,K)-q; rand(1,K)+q];
B1 = [rand(1,K)+q; rand(1,K)+q];
C1 = [rand(1,K)+q; rand(1,K)-q];
D1 = [rand(1,K)-q; rand(1,K)-q];
A = [A1 C1];
B = [B1 D1];
% plot data
plot(A(1,:),A(2,:),'k+',B(1,:),B(2,:),'b*')
grid on
hold on

% Define output coding
% coding (+1/0) for 2-class XOR problem
a = 0;
b = 1;

% Prepare inputs and outputs for network training
% define inputs (combine samples from all two classes)
P = [A B];
% define targets
T = [repmat(a,1,length(A)) repmat(b,1,length(B))];

% create a neural network
net = feedforwardnet([20]);

 % adjust the network for classification

 for i =1:(length(net.layers)-1)
 %net.layers{i}.transferFcn = 'logsig';  % you can use either logsig or tansig
   net.layers{i}.transferFcn = 'tansig';
 end

 net.layers{end}.transferFcn = 'softmax';

 % Cost function: crossentropy
 net.performFcn = 'crossentropy';

 % Train function: Gradient descent with momentum
 net.trainFcn = 'traingdm'; net.trainParam.lr = 0.5;  net.trainParam.mc = 0.8;  net.trainParam.epochs = 2000;

 % Train function:  Scaled Conjugate Gradients. You can tried it instead of traingdm.
 %net.trainFcn = 'trainscg'; net.trainParam.lr = 0.1;  net.trainParam.mc = 0.8;  %net.trainParam.epochs = 1000;

 % The processFcns parameter is used when you want to perform certain preprocessing
 % steps on the network inputs and targets. Be careful, Matlab perform the most common
 % preprocessings automatically when you create a network. If you do not want any
 % preprocessing you need to disable this parameter.
 net.outputs{end}.processFcns = {};

% train the neural network
```

```
 [net,tr,Y,E] = train(net,P,T);
% Notice that tr has the indices of the training, validation and test data sets:
% tr.trainInd
% tr.valInd
% tr.testInd

% show network
view(net)

% Accuracy
fprintf('Accuracy: %f\n',100-100*sum(abs((Y>0.5)-T))/length(T))

% Plot targets and network response to see how good the network learns the data
figure(2)
plot(T','linewidth',2)
hold on
plot(Y','r--')
grid on
legend('Targets','Network response','location','best')
ylim([-1.25 1.25])

% Plot classification result for the complete input space (separation by hyperplanes)
% generate a grid
span = -1:.005:2;
[P1,P2] = meshgrid(span,span);
pp = [P1(:) P2(:)]';
% simulate neural network on a grid
aa = net(pp);
% plot classification regions
figure(1)
mesh(P1,P2,reshape(aa,length(span),length(span))-5);
colormap cool
view(2)
```

Task 7 – Custom networks

Objective: Create and view custom neural networks.

1) Define sample data (i.e. inputs and outputs) for a regression problem.

For example, you can have 6 instances of 1 input variable that have as output 1 output value.

```
close all, clear all, clc
inputs = [1:6]; % input vector (6-dimensional pattern); i.e. 1 2 3 4 5 6
outputs = [7:12]; % corresponding target output vector; i.e. 7 8 9 10 11 12
```

2) Define and custom the network

```
% create the network: 1 input, 2 layer (1 hidden layer and 1 output layer), feed-forward network
net = network( ...
1,              ... % numInputs (number of inputs)
2,              ... % numLayers (number of layers)
[1; 0],         ... % biasConnect (numLayers-by-1 Boolean vector)
[1; 0],         ... % inputConnect (numLayers-by-numInputs Boolean matrix)
[0 0; 1 0],     ... % layerConnect (numLayers-by-numLayers Boolean matrix); [a b; c d]
                ... % a: 1st-layer with itself, b: 2nd-layer with 1st-layer,
                ... % c: 1st-layer with 2nd-layer, d: 2nd-layer with itself
```

```
[0 1]               ... % outputConnect (1-by-numLayers Boolean vector)
);
% View network structure
view(net);
```

We can then see the properties of sub-objects as follows:

```
net.inputs{1}
net.layers{1}, net.layers{2}
net.biases{1}
net.inputWeights{1}, net.layerWeights{2}
net.outputs{2}
```

3) Define topology and transfer function

```
% number of hidden layer neurons
net.layers{1}.size = 5;
% hidden layer transfer function
net.layers{1}.transferFcn = 'logsig';
view(net);
```

4) Configure the network

```
net = configure(net,inputs,outputs);
view(net);
```

5) Train net and calculate neuron output

```
% initial network response without training
initial_output = net(inputs)
```

We can get the weight matrices and bias vector as follows:

```
net.IW{1}
net.LW{2}
net.b{1}

% network training
net.trainFcn = 'trainlm';   % trainlm: Levenberg-Marquardt backpropagation
% trainlm is often the fastest backpropagation algorithm in the toolbox, and is highly %recommended as a first
choice supervised algorithm for regression, although it does %require more memory than other algorithms, as for
example traingdm (Gradient descent %with momentum backpropagation) or traingdx (Gradient descent with
momentum and %adaptive learning rate backpropagation).

net.performFcn = 'mse';
net = train(net,inputs,outputs);
```

% final weight matrices and bias vector:

```
net.IW{1}
net.LW{2}
net.b{1}
```

6) simulate the network on training data

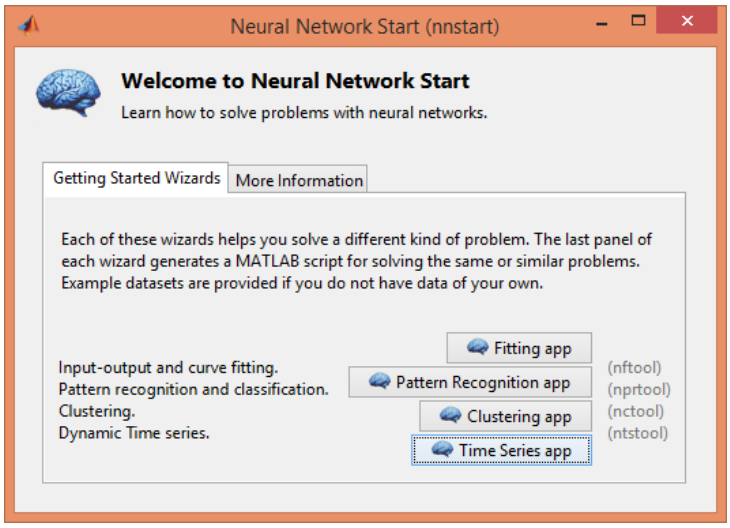
```
net(1) % For 1 as input the outputs should be close to 7
```

```
net(2) % For 1 as input the outputs should be close to 8
net(3) % For 1 as input the outputs should be close to 9
net(4) % For 1 as input the outputs should be close to 10
net(5) % For 1 as input the outputs should be close to 11
net(6) % For 1 as input the outputs should be close to 12
```

Now, simulate the network on other input data (not used for training), for example: 7, 8, 0, -1, -2.

Task 8 – NN Apps Matlab

Take a look to the NN Apps (NN Fitting, NN Time Series, NN Pattern Recognition) or directly type: nnstart



Fitting app: Regression
Pattern Recognition app: Classification
Clustering app: Group data by similarity. Unsupervised classification.
Time Series app: Dynamic prediction. Past values of time series are used to predict future values.

CONVOLUTIONAL NEURAL NETWORKS

Task 9 – Playing with convolutions

1) Load an image

```
im_Orig = imread('lena_gray_256.tif');
figure;
imshow(im_Orig,[]);
```

2) Right edge detector kernel

```
% This is like a (horizontal) FORWARD difference (https://en.wikipedia.org/wiki/Finite_difference)
% Given f(x) on a grid of values with equidistant points at a distance h: {x0, x1, x2, ...} so that
% f_i=f(xi), we can compute the forward difference as an approximation of the forward derivative
%    fd_i(h) = (f_{i+h} - f{i}) / h
% with h=1 (h must be positive)

% kernel1 and kernel2 are different variations of the same idea, you can test both
kernel1 = [-1   1; -1   1; -1   1];
kernel2 = [-1 0 1; -1 0 1; -1 0 1];
kernel  = kernel1;

% Now apply the kernel and plot the feature map
feature_map = conv2(im_Orig,kernel,'same');
figure; subplot(1,2,1); imshow(im_Orig,[]); subplot(1,2,2); imshow(feature_map,[]);
```

3) Left edge detector kernel

```
% This is like a (vertical) BACKWARD difference
% Same as before, but in the opposite direction, so that the signs change

% kernel1 and kernel2 are different variations of the same idea, you can test both
kernel1 = [1   -1; 1   -1; 1   -1];
kernel2 = [1 0 -1; 1 0 -1; 1 0 -1];
kernel  = kernel1;

% Now apply the kernel and plot the feature map
feature_map = conv2(im_Orig,kernel,'same');
figure; subplot(1,2,1); imshow(im_Orig,[]); subplot(1,2,2); imshow(feature_map,[]);
```

4) Edge detector kernel

```
% This detects contours, by applying horizontal and vertical derivatives, obtained
% as a combination of directional edge detectors with second-order finite differences
kernel = [0 -1 0; -1 4 -1; 0 -1 0];

% Now apply the kernel and plot the feature map
feature_map = conv2(im_Orig,kernel,'same');
figure; subplot(1,2,1); imshow(im_Orig,[]); subplot(1,2,2); imshow(feature_map,[]);
```

5) Blurring kernel

```
% The simplest possible blur is done by averaging neighboring pixels

% You can test all these kernels to see different blurring levels
kernel1 = ones([3,3])/9;
kernel2 = ones([4,4])/16;
kernel3= ones([5,5])/25;
kernel4 = ones([6,6])/36;
kernel = kernel1;

% Now apply the kernel and plot the feature map
feature_map = conv2(im_Orig,kernel,'same');
figure; subplot(1,2,1); imshow(im_Orig,[]); subplot(1,2,2); imshow(feature_map,[]);
```

Task 10 – Construct and train a Convolutional Neural Network

1) Load a Small synthetic handwritten digit dataset in
fullfile(matlabroot,'toolbox','nnet','nndemos', 'nndatasets','digitSmallCellArrayData.m')

```
[inputs, targets] = digitSmallCellArrayData;

% 'inputs' is a 1-by-500 cell array, with each cell containing a 28-by-28 matrix
% representing a synthetic image of a handwritten digit
class(inputs)
size(inputs)
inputs(1)
inputs{1}

% 'targets' is a 10-by-500 matrix containing the labels for the images in a 1ofC format
class(targets)
size(targets)
targets(:,1)
targets(:,100)
```

2) Plot several input images randomly selected

```
perm = randperm(numel(inputs),20);
figure;
for i = 1:20
    subplot(4,5,i);
    imshow(inputs{perm(i)});
end
```

THINGS THAT YOU CAN TRY:
- Change the dataset (for example, load images with 3 rbg channels) and repeat the process

3) Kernels from a part of the image and beyond

```
im_Input = inputs{1};
```

```
% This is a part of the image we want to identify
kernel1 = im_Input(14:18,20:24);
figure; subplot(1,2,1); imshow(im_Input,[]); subplot(1,2,2); imshow(kernel1,[]);

% Now apply the kernel and plot the feature map
feature_map = conv2(im_Input,kernel1,'same');
figure; subplot(1,2,1); imshow(im_Input,[]); subplot(1,2,2); imshow(feature_map,[]);

% Now we can consider the mirror kernel
kernel2 = zeros(size(kernel1));
for i=1:size(kernel1,1)
  kernel2(i,:) = kernel1(i,end:-1:1);
end
figure; subplot(1,2,1); imshow(im_Input,[]); subplot(1,2,2); imshow(kernel2,[]);

% Now apply the kernel and plot the feature map
feature_map = conv2(im_Input,kernel2,'same');
figure; subplot(1,2,1); imshow(im_Input,[]); subplot(1,2,2); imshow(feature_map,[]);

% Now a nonsense kernel
kernel3 = rand(15,15);
figure; subplot(1,2,1); imshow(im_Input,[]); subplot(1,2,2); imshow(kernel3,[]);

% Now apply the kernel and plot the feature map
feature_map = conv2(im_Input,kernel3,'same');
figure; subplot(1,2,1); imshow(im_Input,[]); subplot(1,2,2); imshow(feature_map,[]);
```

4) Prepare data for training and testing

```
%% Inputs
% Store the total number of images and the sizes of the images (in this case all images have the same % size)
NImages    = numel(inputs);
SizesImage = size(inputs{1});
% Converts the cell array of inputs to a 3D matrix
inputs_matrix = zeros(NImages,SizesImage(1),SizesImage(2));
for i=1:NImages
  inputs_matrix(i,:,:) = inputs{i};
end
% Reshapes the 3D matrix to a 4D matrix by adding the channel dimension
if length(SizesImage) == 2
  NChannels = 1;
else
  NChannels = SizesImage(3);
end
Inputs_Net  = reshape(inputs_matrix,[SizesImage(1), SizesImage(2), NChannels, NImages]);

%% Targets
Labels_1ofC = targets;              % 1ofC coding scheme
Labels_Num  = vec2ind(targets);     % Numeric coding scheme
NLabels     = numel(unique(Labels_Num));
```

```
%% Number of examples of every class
[Nc,dummy]=hist(Labels_Num,unique(Labels_Num));
fprintf('Number of examples in every of the %d classes (%d):',numel(Nc),sum(Nc));
for n=Nc, fprintf(' %d',n); end; fprintf('\n');
```

5) Training/Validation partition 75%/25% (holdout)

```
% For k-fold cross validation:  cv = cvpartition(Labels_Num,'KFold',k,'Stratify',true);
PCTValidation = 0.25;
cv = cvpartition(Labels_Num,'Holdout',PCTValidation,'Stratify',true);

% For k-fold cross validation: idxTrain = training(cv,fold_number);
idxTrain      = training(cv);

% For k-fold cross validation: idxValid = test(cv,fold_number);
idxValid      = test(cv);

% Obtain the inputs for training and validation from the indexes of cvpartition
Inputs_Train = Inputs_Net(:,:,:,idxTrain);
Inputs_Valid = Inputs_Net(:,:,:,idxValid);
Labels_Train = categorical(Labels_Num(idxTrain));  % Labels for the network must be categorical
Labels_Valid = categorical(Labels_Num(idxValid));  % Labels for the network must be categorical

% Statistics of the partition
fprintf('Number of examples in the training set: %d\n',size(Inputs_Train,4));
fprintf('Number of examples in the validation set: %d\n',size(Inputs_Valid,4));
[Nc,C]=hist(Labels_Train,unique(Labels_Train));
fprintf('Number of examples in every of the %d classes in the training set (%d):  ',numel(Nc),sum(Nc));
for n=Nc, fprintf(' %3d',n); end; fprintf('\n');
[Nc,C]=hist(Labels_Valid,unique(Labels_Valid));
fprintf('Number of examples in every of the %d classes in the validation set (%d):',numel(Nc),sum(Nc));
for n=Nc, fprintf(' %3d',n); end; fprintf('\n');
```

THINGS THAT YOU CAN TRY:

- Change the percentage of validation data and check the number of examples in every data set
- Change the 'cvpartition' method to 'KFold' and see the differences in the partitions
- Modify the parameter 'Stratify' in 'cvpartition' and check the number of examples in every class

6) Define the architecture of the network (model representation)

See https://es.mathworks.com/help/deeplearning/ug/layers-of-a-convolutional-neural-network.html?lang=en for the list of specific layers for a CNN and https://es.mathworks.com/help/deeplearning/ug/list-of-deep-learning-layers.html?lang=en for the list of deep learning layers

```
architecture = [
    imageInputLayer([SizesImage(1), SizesImage(2), NChannels])
    convolution2dLayer(3,128,'Padding','same')   %KernelSize, NumberOfKernels, rest of parameters
    reluLayer
    maxPooling2dLayer(2,'Stride',2)              %PoolingSize, rest of parameters
    fullyConnectedLayer(NLabels)
    softmaxLayer
```

```
    classificationLayer
];
```

THINGS THAT YOU CAN TRY:

- Change the kernel size value of the convolution
- Change the number of kernels/filters in the convolution and the rest of parameters
- Change the parameters of the pooling layer
- Add more layers (batchNormalizationLayer, for example) to this basic architecture
- Add more blocks of layers (convolution - nonlinearity - pooling) to this basic architecture

7) Define the solver (Optimization technique) and its parameters

See https://es.mathworks.com/help/deeplearning/ref/trainingoptions.html?lang=en
for a list of the parameters of every solver

```
optimizationSolver = ...
 trainingOptions('sgdm', ...                                    %'sgdm' | 'rmsprop' | 'adam' | 'lbfgs'
                 'InitialLearnRate', 0.01, ...                  %Defalut value: 0.01
                 'Momentum', 0.9, ...                           %Defalut value: 0.9000
                 'L2Regularization', 1.0000e-04, ...            %Defalut value: 1.0000e-0
                 'MaxEpochs', 30, ...                           %Defalut value: 30
                 'MiniBatchSize', 128, ...                      %Defalut value: 128
                 'Verbose', 1, ...                              %Defalut value: 1
                 'ValidationData', {Inputs_Valid,Labels_Valid}, ...  %Defalut value: []
                 'ValidationFrequency', 50, ...                 %Defalut value: 50
                 'ValidationPatience', 5, ...                   %Defalut value: 5
                 'Shuffle', 'once', ...                         %Defalut value: 'once'
                 'Plots', 'training-progress' ...                %Defalut value: 'none'
 );
```

8) Train the network

https://es.mathworks.com/help/deeplearning/ref/trainnetwork.html

```
net = trainNetwork(Inputs_Train,Labels_Train,architecture,optimizationSolver);
```

THINGS THAT YOU CAN TRY:

- Change the solver name and its parameters, and retrain the network

9) Compute the predictions of the trained network

```
Predictions_Train = predict(net,Inputs_Train);
Predictions_Valid = predict(net,Inputs_Valid);
```

10) Compare the predictions with the real labels

```
Labels_1ofC_Train = Labels_1ofC(:,idxTrain);
Labels_1ofC_Valid = Labels_1ofC(:,idxValid);

% Accuracies
sum(vec2ind(Predictions_Train') == vec2ind(Labels_1ofC_Train)) / sum(idxTrain)
```

```
sum(vec2ind(Predictions_Valid') == vec2ind(Labels_1ofC_Valid)) / sum(idxValid)

% See also...
% https://es.mathworks.com/help/deeplearning/ug/create-simple-deep-learning-network-for-classification.html?lang=en
% https://es.mathworks.com/help/deeplearning/ug/define-custom-deep-learning-layers.html?lang=en
```