mdCNN – Multi dimensional CNN

**Table of Contents**

[1. General 2](#_Toc530997032)

[2. dataset format 2](#_Toc530997033)

[3. Network configuration 2](#_Toc530997034)

[A. Layer specification 3](#_Toc530997035)

[I. Input layer 3](#_Toc530997036)

[II. Fully connected layer 4](#_Toc530997037)

[III. Softmax layer 4](#_Toc530997038)

[IV. Convolutional layer 4](#_Toc530997039)

[V. batchnorm layer 5](#_Toc530997040)

[VI. regression layer 5](#_Toc530997041)

[B. Hyper params 7](#_Toc530997042)

[C. Run params 8](#_Toc530997043)

[4. Training a network 8](#_Toc530997044)

[5. Demo 9](#_Toc530997045)

# General

mdCNN network can handle input data with 1 2 or 3 dimensions. Every input can have several feature maps.

This document describes in short how to configure and train a network

# dataset format

Input to ‘Train’ is a struct with 4 elements. I,labels,I\_test,in the following format:

* Training data is stored in array called ‘I’ , the labels are stored in a vector named ‘labels’. I and labels are at the same length.
* Testing data is stored in ‘I\_test’ array and ‘labels\_test’ vector.

Training process:

The network state is saved after each iteration loop. The net is saved to a file called ‘net.mat’. This file contains the full state of the network, including the Training state, so you can halt the training process anytime and then load the network from file using load(‘net.mat’)

It’s possible to load the network file, and then continue the training process by calling the ‘Train’ function

# Network configuration

To configure a network you must create a config file. Config file describes the network structure, training parameters and all other possible configuration.

Example config file :

%%%%%%%%%%%%%%%%%%%%% Layers specification %%%%%%%%%%%%%%%%%%

net.layers{end+1}.properties = struct('type','input', 'sizeFm' ,[28 28],'numFm',1 ); %inputLayer

net.layers{end+1}.properties = struct('type','conv','numFm',12 ,'kernel',5,'pad',2);

net.layers{end+1}.properties = struct('type','conv','numFm',24 ,'kernel',13);

net.layers{end+1}.properties = struct('type','batchNorm');

net.layers{end+1}.properties = struct('type','fc','numFm',128);

net.layers{end+1}.properties = struct('type','fc','numFm',10);

net.layers{end+1}.properties = struct('type','softmax');

net.layers{end+1}.properties = struct('type','regression','lossFunc',@CrossEnt,'costFunc',@CrossEnt\_Cost); %regression Layer

%%%%%%%%%%%%%%%%%%%%% Hyper params - training %%%%%%%%%%%%%%%%%%

net.hyperParam.trainLoopCount = 1000; %on how many images to train before evaluating the network

net.hyperParam.testImageNum = 2000; % after each loop, on how many images to evaluate network performance

net.hyperParam.ni\_initial = 0.05; % ni to start training process

net.hyperParam.ni\_final = 0.025; % final ni to stop the training process

net.runInfoParam.verifyBP = 1;

net.runInfoParam.batchNum = 1;

net.hyperParam.randomizeTrainingSamples = 0;

Any configuration parameter that is not given a value in the config file is assigned with default value.

All possible configuration settings and default values can be found in ‘CreateNet.m’ in ‘initNetDefaults’ function and also listed below

**There are 3 groups of network configuration**

* Layer specification
* Hyper params
* Run params

## Layer specification

Describes the network structure and input/output size. Example:

net.layers{end+1}.properties = struct('type','input', 'sizeFm' ,[28 28 28],'numFm',1 ); %inputLayer

net.layers{end+1}.properties = struct('type','conv','numFm',7 , 'Activation',@Relu, 'dActivation',@dRelu,'kernel',5,'pad',2, 'stride', [2 2 4], 'pooling', [1 1 1]);

net.layers{end+1}.properties = struct('type','conv','numFm',17 , 'kernel',[5 5 3] ,'pad',[1 1 0], 'pooling', [1 1 1], 'dropOut' ,0.8);

net.layers{end+1}.properties = struct('type','batchNorm','initGamma',1,'initBeta',0,alpha',2^-5);

net.layers{end+1}.properties = struct('type','fc','numFm',128);

net.layers{end+1}.properties = struct('type','fc','numFm',10);

net.layers{end+1}.properties = struct('type','softmax');

net.layers{end+1}.properties = struct('type','regression','lossFunc',@CrossEnt,'costFunc',@CrossEnt\_Cost); %regression Layer

Every network layer is a struct with the below fields (when not specified default is used) where ‘type’ specifies the layer type:

### Input layer

The input layer must be the first layer and specifies the size of a single sample from the dataset.

For example in case of RGB image:

net.layers{end+1}.properties = struct('type','input', 'sizeFm' ,[32 32],'numFm',3 );

For gray scale image, using Relu activation (Sigmoid is the default)

net.layers{end+1}.properties = struct('type','input', 'sizeFm' ,[32 32],'numFm',1, 'Activation',@Relu, 'dActivation',@dRelu,'kernel' );

For native 3D data

net.layers{end+1}.properties = struct('type','input', 'sizeFm' ,[28 28 28],'numFm',1 );

### Fully connected layer

‘numFM’ field indicates the number of outputs. Every neuron in the FC layer is connected to all outputs of the previous layer

Example – fully connected layer with 10 outputs:

net.layers{end+1}.properties = struct('type','fc','numFm',10);

### Softmax layer

Softmax layer usually appears after a fully connected layer (but not a must) , it performs the softmax function on the previous layer outputs.

Number of feature map (outputs) is derived from the previous layer

Example:

net.layers{end+1}.properties = struct('type','softmax');

### Convolutional layer

* **‘numFM’ –** number of feature map the layer has.
* **‘padding’/’stride’/’pooling’** – this property can have a scalar value or a vector value. When providing scalar this value will be used for all dimension i.e ‘stride’ ,3

When providing a vector, different value can be used per dimension. i.e ‘stride’ , [2 , 1, 5]

* **‘kernel’ –** size of the convolutional kernel for type 2 layers**.** This property can have a scalar value or a vector value**.**
* **‘dropOut’ –** specify the dropout ratio for the layer , number between 0 and 1 , where 1 means no dropout (default)

Default for padding is 0 for all dimensions (no padding)

Default for stride is 1 for all dimensions (no stride)

Default for pooling is 1 for all dimensions (no pooling)

For pooling/stride there is no requirement that the previous layer out is a multiple of the given value. In case this happens input is expended with zeroes.

Example:

net.layers{end+1}.properties = struct('type','conv','numFm',17 , 'kernel',[5 5 3] ,'pad',[1 1 0], 'pooling', [1 1 1], 'dropOut' ,0.8);

### batchnorm layer

The batch norm layer normalizes a batch of outputs with ‘initGamma’ and ‘initBeta’ as parameters

Example:

net.layers{end+1}.properties = struct('type','batchNorm','initGamma',1,'initBeta',0,alpha',2^-5);

Gamma and beta are adjusted variables. One per input.

alpha is a number smaller than one, used to calculate the running batch mean and variance (alpha filtering)

### regression layer

The regression layer specifies the loss/cost function for doing back propagation.

The two main methods are MSE – mean square error and CrossEnt – Cross entropy

You need to provide function pointers for loss function (derivative of the cost) and cost function pointer

Example for cross entropy

net.layers{end+1}.properties = struct('type','regression','lossFunc',@CrossEnt,'costFunc',@CrossEnt\_Cost); %regression Layer

Example for MSE

net.layers{end+1}.properties = struct('type','regression','lossFunc',@MSE,'costFunc',@MSE\_Cost); %regression Layer

**Activation field**

For fully connected/convolutional layers an activation function handle can be given. Default is Sigmoid

Also the derivative of the activation needs to be given as a function handle

Example using Relu:

net.layers{end+1}.properties = struct('type','conv','numFm',7 , 'Activation',@Relu, 'dActivation',@dRelu,'kernel',5,'pad',2, 'stride', [2 2 4], 'pooling', [1 1 1]);

Example using Tanh:

net.layers{end+1}.properties = struct('type','fc','numFm',10,'Activation',@Tanh, 'dActivation',@dTanh);

When not specified, default activation is Sigmoid

Some activation/deactivation functions exist in the ‘Training’ folder, but you can use any predefined functions.

## Hyper params

Hyper parameters affects mainly the training process

**Below are the valid ones:**

net.hyperParam.trainLoopCount=1000;%on how many images to train before evaluating the network

net.hyperParam.testImageNum=2000;

net.hyperParam.batchNum = 1; %on how many samples to train before updating weights. batch>1 converges slower , but in some cases can improve accuracy

net.hyperParam.ni\_initial = 0.05;% ni to start training process

net.hyperParam.ni\_final = 0.00001;% final ni to stop the training process

net.hyperParam.noImprovementTh=50; % if after noImprovementTh there is no improvement , reduce ni

net.hyperParam.momentum=0;

net.hyperParam.constInitWeight=nan; %Use nan to set initial weight to random. Any other value to fixed

net.hyperParam.lambda=0; %L2 regularization factor, set 0 for none. Above 0.01 not recommended

net.hyperParam.errorMethod=1; % 0 for MSE , 1 for cross entropy

net.hyperParam.testOnData=0; % to perform testing after each epoc on the data inputs or test inputs

net.hyperParam.addBackround=0; % random background can be added to images before passing to net in order to improve noise resistance.

net.hyperParam.testOnNull=0;% Training on non data images without any feature to detect (I call them null images)

%%%%%%%%%%%%%% Augmentation %%%%%%%%%%%%%%

net.hyperParam.augmentImage=0; % set to 0 for no augmentation

net.hyperParam.augmentParams.noiseVar=0.02;

net.hyperParam.augmentParams.maxAngle=45/3;

net.hyperParam.augmentParams.maxScaleFactor=1.1;

net.hyperParam.augmentParams.minScaleFactor=1/1.5;

net.hyperParam.augmentParams.maxStride=4;

net.hyperParam.augmentParams.maxSigma=2;%for gauss filter smoothing

net.hyperParam.augmentParams.imageComplement=0;% will reverse black/white of the image

net.hyperParam.augmentParams.medianFilt=0; %between 0 and one - if this value is 0.75 it will zero all 75% lower points. 0 will mean no point is changed, 1 will keep the highest point only

%%%%%%%%%%%%%% Centralize image before passing to net? %%%%%%%%%%%%%%

net.hyperParam.centralizeImage=0;

net.hyperParam.cropImage=0;

net.hyperParam.flipImage=0; % fill randomly flip the input hor/vert before passing to the network. Improves learning in some instances

net.hyperParam.useRandomPatch=0;

net.hyperParam.testNumPatches=1; % on how many patches from a single image to perform testing. network is evaluated on several patches and result is averaged over all patches.

net.hyperParam.selevtivePatchVarTh=0; %in order to drop patches that their variance is less than th

net.hyperParam.testOnMiddlePatchOnly=0; %will test on the middle patch only

net.hyperParam.normalizeNetworkInput=1; %will normalize every input to net to be with var=1, mean 0

## Run params

%%%%%%%%%%%%%% Run info - parameters that change every epoc iteration %%%%%%%%%%%%%%

net.runInfoParam.storeMaxMSENet = 0; % this enables the trainer to store also the net with the highest MSE found (in addition to the latest one)

net.runInfoParam.verifyBP = 1; % can perform pre-train back-propagation verification. Useful to detect faults in the application

net.runInfoParam.displayConvNet = 0;

# Training a network

First create a network using a config file

net = CreateNet('../../Configs/mnist.conf');

Then, call ‘Train’ function with the dataset containing the train/test samples:

net = Train(MNIST,net, 15000);

Here , MNIST is the dataset , this will train for 15000 images from the test set in a cyclic manner.

In order to train longer , you can specify ‘Inf’ as the last parameter, network will train until learning rate (ni) reach below the given threshold

When the function returns the net variable is updated and also saved to disk.

You can call Train again to continue training from the last point.

In order to test samples on the trained network you need to call ‘feedForward’

Please follow ‘checkNetwork’ for details on how to run the net on new samples

# Demo

There are several examples for networks pre-configured to run MNIST, CIFAR10 , and 3dMNIST - a special enhancement of MNIST dataset to 3D volumes.

MNIST Demo reach 99.2% in several minutes, and CIFAR10 demo reaches about 80%

I have used this framework in a project for classifying Vertebra in a 3D CT images.

=~=~=~=~=~=~=~=~=~=~=~=~=~=~=~=~=~=~=~=~

To run MNIST demo: Go into the folder 'Demo/MNIST' , Run 'demoMnist.m' file. The file will download MNIST dataset and start training the network.

After 15 iterations (several minutes) it will open a GUI where you can test the network performance. In addition layer 1 filters will be shown.