% function are called with two types

% either cnn\_cifar('coarse') or cnn\_cifar('fine')

% coarse will classify the image into 20 catagories

% fine will classify the image into 100 catagories

function cnn\_cifar(type, varargin)

type = 'fine';

if ~(strcmp(type, 'fine') || strcmp(type, 'coarse'))

error('The argument has to be either fine or coarse');

end

% record the time

tic

%% --------------------------------------------------------------------

% Set parameters

% --------------------------------------------------------------------

%

% data directory

opts.dataDir = fullfile('cifar\_data','cifar') ;

% experiment result directory

opts.expDir = fullfile('cifar\_data','cifar-baseline') ;

% image database

opts.imdbPath = fullfile(opts.expDir, 'imdb.mat');

% set up the batch size (split the data into batches)

opts.train.batchSize = 100 ;

% number of Epoch (iterations)

opts.train.numEpochs = 25 ;

% resume the train

opts.train.continue = true ;

% use the GPU to train

opts.train.useGpu = false ;

% set the learning rate

opts.train.learningRate = [0.001\*ones(1, 10) 0.0001\*ones(1,15)] ;

% set weight decay

opts.train.weightDecay = 0.0005 ;

% set momentum

opts.train.momentum = 0.9 ;

% experiment result directory

opts.train.expDir = opts.expDir ;

% parse the varargin to opts.

% If varargin is empty, opts argument will be set as above

opts = vl\_argparse(opts, varargin);

% --------------------------------------------------------------------

% Prepare data

% --------------------------------------------------------------------

imdb = load(opts.imdbPath) ;

%% Define network

% The part you have to modify

net.layers = {} ;

%=============layer1======================

% 1 conv1

net.layers{end+1} = struct('type', 'conv', ...

'weights', {{1e-4\*randn(3,3,3,32, 'single'), zeros(1,32, 'single')}},...

'filtersLearningRate', 1, ...

'biasesLearningRate', 2, ...

'stride', 1, ...

'pad', 1) ;

%1.1 bnorm

net.layers{end+1} = struct('type', 'bnorm', 'name', 'batcfd', ...

'weights', {{ones(32, 1, 'single'), zeros(32, 1, 'single'), zeros(32, 2, 'single')}}, ...

'learningRate', [2 1 0.05], ...

'weightDecay', [0 0]) ;

% 3 relu1

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

%=============layer2======================

% 1 conv2

net.layers{end+1} = struct('type', 'conv', ...

'weights', {{1e-4\*randn(3,3,32,32, 'single'), zeros(1,32, 'single')}},...

'filtersLearningRate', 1, ...

'biasesLearningRate', 2, ...

'stride', 1, ...

'pad', 1) ;

%1.1 bnorm2

net.layers{end+1} = struct('type', 'bnorm', 'name', 'batcfd', ...

'weights', {{ones(32, 1, 'single'), zeros(32, 1, 'single'), zeros(32, 2, 'single')}}, ...

'learningRate', [2 1 0.05], ...

'weightDecay', [0 0]) ;

% 3 relu1

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

% 2 pool1 (max pool)

net.layers{end+1} = struct('type', 'pool', ...

'method', 'max', ...

'pool', [2 2], ...

'stride', 2, ...

'pad', [0 1 0 1]) ; % Emulate caffe

%=============layer3======================

% 4 conv3

net.layers{end+1} = struct('type', 'conv', ...

'weights', {{0.01\*randn(3,3,32,64, 'single'), zeros(1,64, 'single')}},...

'filtersLearningRate', 1, ...

'biasesLearningRate', 2, ...

'stride', 1, ...

'pad', 1) ;

%1.1 bnorm

net.layers{end+1} = struct('type', 'bnorm', 'name', 'batcfd', ...

'weights', {{ones(64, 1, 'single'), zeros(64, 1, 'single'), zeros(64, 2, 'single')}}, ...

'learningRate', [2 1 0.05], ...

'weightDecay', [0 0]) ;

% 5 relu2

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

%=============layer4======================

% 4 conv3

net.layers{end+1} = struct('type', 'conv', ...

'weights', {{0.01\*randn(3,3,64,64, 'single'), zeros(1,64, 'single')}},...

'filtersLearningRate', 1, ...

'biasesLearningRate', 2, ...

'stride', 1, ...

'pad', 1) ;

%1.1 bnorm

net.layers{end+1} = struct('type', 'bnorm', 'name', 'batcfd', ...

'weights', {{ones(64, 1, 'single'), zeros(64, 1, 'single'), zeros(64, 2, 'single')}}, ...

'learningRate', [2 1 0.05], ...

'weightDecay', [0 0]) ;

% 5 relu2

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

% 4 conv1

net.layers{end+1} = struct('type', 'conv', ...

'weights', {{0.01\*randn(1,1,64,64, 'single'), zeros(1,64, 'single')}},...

'filtersLearningRate', 1, ...

'biasesLearningRate', 2, ...

'stride', 1, ...

'pad', 0) ;

%1.1 bnorm

net.layers{end+1} = struct('type', 'bnorm', 'name', 'batcfd', ...

'weights', {{ones(64, 1, 'single'), zeros(64, 1, 'single'), zeros(64, 2, 'single')}}, ...

'learningRate', [2 1 0.05], ...

'weightDecay', [0 0]) ;

% 5 relu2

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

% 6 pool2 (max pool)

net.layers{end+1} = struct('type', 'pool', ...

'method', 'max', ...

'pool', [2 2], ...

'stride', 2, ...

'pad', [0 1 0 1]) ; % Emulate caffe

%=============layer4======================

% 7 conv3

net.layers{end+1} = struct('type', 'conv', ...

'weights', {{0.01\*randn(3,3,64,128, 'single'), zeros(1,128, 'single')}},...

'filtersLearningRate', 1, ...

'biasesLearningRate', 2, ...

'stride', 1, ...

'pad', 1) ;

%1.1 bnorm

net.layers{end+1} = struct('type', 'bnorm', 'name', 'batcfd', ...

'weights', {{ones(128, 1, 'single'), zeros(128, 1, 'single'), zeros(128, 2, 'single')}}, ...

'learningRate', [2 1 0.05], ...

'weightDecay', [0 0]) ;

% 8 relu3

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

%=============layer5======================

% 7 conv3

net.layers{end+1} = struct('type', 'conv', ...

'weights', {{0.01\*randn(3,3,128,128, 'single'), zeros(1,128, 'single')}},...

'filtersLearningRate', 1, ...

'biasesLearningRate', 2, ...

'stride', 1, ...

'pad', 1) ;

%1.1 bnorm

net.layers{end+1} = struct('type', 'bnorm', 'name', 'batcfd', ...

'weights', {{ones(128, 1, 'single'), zeros(128, 1, 'single'), zeros(128, 2, 'single')}}, ...

'learningRate', [2 1 0.05], ...

'weightDecay', [0 0]) ;

% 8 relu3

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

% 4 conv1

net.layers{end+1} = struct('type', 'conv', ...

'weights', {{0.01\*randn(1,1,128,128, 'single'), zeros(1,128, 'single')}},...

'filtersLearningRate', 1, ...

'biasesLearningRate', 2, ...

'stride', 1, ...

'pad', 0) ;

%1.1 bnorm

net.layers{end+1} = struct('type', 'bnorm', 'name', 'batcfd', ...

'weights', {{ones(128, 1, 'single'), zeros(128, 1, 'single'), zeros(128, 2, 'single')}}, ...

'learningRate', [2 1 0.05], ...

'weightDecay', [0 0]) ;

% 8 relu3

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

% 9 pool3 (avg pool)

net.layers{end+1} = struct('type', 'pool', ...

'method', 'max', ...

'pool', [2 2], ...

'stride', 2, ...

'pad', [0 1 0 1]) ; % Emulate caffe

%=================fully connect layer 1==================

% 10 fc1

net.layers{end+1} = struct('type', 'conv', ...

'weights', {{0.1\*randn(4,4,128,128, 'single'), zeros(1,128, 'single')}}, ...

'filtersLearningRate', 1, ...

'biasesLearningRate', 2, ...

'stride', 1, ...

'pad', 0) ;

% bnorm

net.layers{end+1} = struct('type', 'bnorm', 'name', 'batcfd', ...

'weights', {{ones(128, 1, 'single'), zeros(128, 1, 'single'), zeros(128, 2, 'single')}}, ...

'learningRate', [2 1 0.05], ...

'weightDecay', [0 0]) ;

%=================fully connect layer 2==================

% 11 fc2

net.layers{end+1} = struct('type', 'conv', ...

'weights', {{0.1\*randn(1,1,128,100, 'single'), zeros(1,100, 'single')}}, ...

'filtersLearningRate', 1, ...

'biasesLearningRate', 2, ...

'stride', 1, ...

'pad', 0) ;

% bnorm

net.layers{end+1} = struct('type', 'bnorm', 'name', 'batcfd', ...

'weights', {{ones(100, 1, 'single'), zeros(100, 1, 'single'), zeros(100, 2, 'single')}}, ...

'learningRate', [2 1 0.05], ...

'weightDecay', [0 0]) ;

% 12 loss

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

% --------------------------------------------------------------------

% Train

% --------------------------------------------------------------------

% Take the mean out and make GPU if needed

imdb.images.data = bsxfun(@minus, imdb.images.data, mean(imdb.images.data,4)) ;

if opts.train.useGpu

imdb.images.data = gpuArray(imdb.images.data) ;

end

%% display the net

vl\_simplenn\_display(net);

%% start training

net = vl\_simplenn\_tidy(net);

[net,info] = cnn\_train\_cifar(net, imdb, @getBatch, ...

opts.train, ...

'val', find(imdb.images.set == 2) , 'test', find(imdb.images.set == 3)) ;

%% Record the result into csv and draw confusion matrix

load(['cifar\_data/cifar-baseline/net-epoch-' int2str(opts.train.numEpochs) '.mat']);

load(['cifar\_data/cifar-baseline/imdb' '.mat']);

fid = fopen('cifar\_prediction.csv', 'w');

strings = {'ID','Label'};

for row = 1:size(strings,1)

fprintf(fid, repmat('%s,',1,size(strings,2)-1), strings{row,1:end-1});

fprintf(fid, '%s\n', strings{row,end});

end

fclose(fid);

ID = 1:numel(info.test.prediction\_class);

dlmwrite('cifar\_prediction.csv',[ID', info.test.prediction\_class], '-append');

val\_groundtruth = images.labels(45001:end);

val\_prediction = info.val.prediction\_class;

val\_confusionMatrix = confusion\_matrix(val\_groundtruth , val\_prediction);

cmp = jet(50);

figure ;

imshow(ind2rgb(uint8(val\_confusionMatrix),cmp));

imwrite(ind2rgb(uint8(val\_confusionMatrix),cmp) , 'cifar\_confusion\_matrix.png');

toc

% --------------------------------------------------------------------

%% call back function get the part of the batch

function [im, labels] = getBatch(imdb, batch , set)

% --------------------------------------------------------------------

im = imdb.images.data(:,:,:,batch) ;

% data augmentation

if set == 1 % training

for i=1:100

% fliplr

if rand > 0.5, im(:,:,:,i) = fliplr(im(:,:,:,i)); end

% noise

if rand > 0.5, im(:,:,:,i) = im(:,:,:,i) + 10\*rand(size(im(:,:,:,i))); end

%

% if rand > 0.5

% ma = max(max(max(max(im(:,:,:,i)))));

% mi = min(min(min(min(im(:,:,:,i)))));

% im(:,:,:,i) = (im(:,:,:,i).\* ( (0.8 + (1.2 - 0.8)\*rand) \*ones(size(im(:,:,:,i))) + rand/2));

%im(:,:,:,i) = single(im(:,:,:,i) \*(ma-mi) + mi);

% end

% rotate randomly

% if rand > 0.66, im(:,:,:,i) = imrotate(im(:,:,:,i), (rand-0.5)\*100,'crop'); end

% random crop

x1 = round(rand\*8);

y1 = round(rand\*8);

for j = 1:32

for k = 1:32

if (j < x1)

im(j, k, :, i) = 0;

elseif (j > 32 - x1)

im(j, k, :, i) = 0;

elseif(k < y1)

im(j, k, :, i) = 0;

elseif(k > 32 - y1)

im(j, k, :, i) = 0;

else

continue;

end

end

end

% and other data augmentation

end

end

if set ~= 3

labels = imdb.images.labels(1,batch) ;

end

Setup.m

function setup(varargin)

run vlfeat/toolbox/vl\_setup ;

run matconvnet/matlab/vl\_setupnn ;

% addpath matconvnet/examples ;

opts.useGpu = false ;

opts.verbose = false ;

opts = vl\_argparse(opts, varargin) ;

try

vl\_nnconv(single(1),single(1),[]) ;

catch

warning('VL\_NNCONV() does not seem to be compiled. Trying to compile it now.') ;

vl\_compilenn('enableGpu', opts.useGpu, 'verbose', opts.verbose) ;

end

if opts.useGpu

try

vl\_nnconv(gpuArray(single(1)),gpuArray(single(1)),[]) ;

catch

vl\_compilenn('enableGpu', opts.useGpu, 'verbose', opts.verbose) ;

warning('GPU support does not seem to be compiled in MatConvNet. Trying to compile it now') ;

end

end

% addpath('matconvnet/matlab');