بسم الله الرحمن الرحیم

تمرین سوم شبکه عصبی 15/10/1394

خوب در ابتدا میخواهیم یک toolbox متنی برای شبکه های کانولوشن روی متلب نصب کنیم:

باید اول ببینیم که متلب کامپایر C رو میشناسه یا خیر، با دستور زیر در command window متلب این را می فهمیم:

mex –setup

برای من نشناخت به همین خاطر اخرین نسخه متلب رو یعنی 2015b رو نصب کردم تا visual studio 2015 رو بشناسه.

نصب **MatconveNet:**

MatConvNet: CNNs for MATLAB

این برنامه در واقع به متلب اضافه میشه و میتوانیم از شبکه های کانولوشنال در متلب بهره ببریم.

برای نصب اول آخرین ورژن رو از لینک زیر دانلود میکنیم:

<http://www.vlfeat.org/matconvnet/>

# ﻣﻌﺮﻓﻲ ﻧﺮم اﻓﺰارMatconveNet

MatConvNet ﻳﻜــﻲ از ﻧــﺮم اﻓﺰارﻫــﺎي ﻣﻌــﺮوف ﺟﻬــﺖ ﻛــﺎر ﺑــﺎ Convolutional Neural Networks و Deep Learning اﺳﺖ. وﺟﻪ ﺗﻤﺎﻳﺰ اﻳﻦ ﻧﺮم اﻓﺰار در ﻣﻘﺎﻳﺴﻪ ﺑﺎ ﻧﺮم اﻓﺰارﻫﺎي ﻣﻌـﺮوف دﻳﮕﺮ در اﻳﻦ ﺣﻮزه، ﻣﺎﻧﻨﺪ Theano ، Caffe و Torch راﺣﺘﻲ ﻧﺼﺐ و اﺳـﺘﻔﺎده وﻫﻤﭽﻨـﻴﻦ اﻳﺘﻨـﺮﻓﻴﺲ MATLAB اﺳﺖ ﻛﻪ آﻧﺮا ﺑﺮاي اﺳﺘﻔﺎده در ﻛﺎرﻫﺎي ﭘﮋوﻫﺸﻲ ﺑﺴﻴﺎر ﻣﻨﺎﺳﺐ ﻧﻤﻮده اﺳﺖ. ﺳﻪ ﻧـﺮم اﻓـﺰار دﻳﮕﺮ، ﻣﻌﻤﻮﻻ در ﻟﻴﻨﻮﻛﺲ ﻧﺼﺐ و ﭘﺸﺘﻴﺒﺎﻧﻲ ﻣﻲ ﺷﻮﻧﺪ، اﻣـﺎMatConveNet ﺑـﻪ راﺣﺘـﻲ در وﻳﻨـﺪوز و دﻳﮕﺮ ﺳﻴﺴﺘﻢ ﻋﺎﻣﻞ ﻫﺎ ﻧﺼﺐ ﻣﻲ ﺷﻮد. ﺑﺎ ﺗﻮﺟﻪ ﺑﻪ ﺣﺠﻢ ﻣﺤﺎﺳﺒﺎت ﺑﺎﻻ ﺑﺮاي آﻣﻮزش ﺷﺒﻜﻪ ﻫﺎي ﻋﻤﻴﻖ، ﻫﺮ ﭼﻬﺎر ﻧﺮم اﻓﺰار ﻓﻮق GPU را ﭘﺸﺘﻴﺒﺎﻧﻲ ﻣﻲ ﻛﻨﻨﺪ. ﻣﺎ در اﻳﻦ ﻓﺎﻳﻞ آﻣﻮزﺷﻴﺒﺮ روش ﻧﺼﺐ و ﻛﺎر ﻛـﺎردن ﺑـﺎ ﻧﺮم اﻓﺰار MatConvNet ﻣﺘﻤﺮﻛﺰ ﻣﻲ ﺷﻮﻳﻢ. آﺧـﺮﻳﻦ ﻧﺴـﺨﻪ ﻧـﺮم اﻓـﺰار و ﻣﺴـﺘﻨﺪات ﻣﺮﺑـﻮط ﺑـﻪ آن از ﻟﻴﻨﻚ زﻳﺮ ﻗﺎﺑﻞ ﻣﺸﺎﻫﺪه و Downlod ﻣﻲ ﺑﺎﺷﺪ.

<http://www.vlfeat.org/matconvnet/>

ﻣﺮاﺣﻞ ﻧﺼﺐ:

ﭘﺲ از Download ﻧﺮم اﻓﺰار، ﻣﺮاﺣﻞ زﻳﺮ را ﺟﻬﺖ ﻧﺼﺐ ﻧﺮم اﻓﺰار در ﻣﻄﻠﺐ اﻧﺠﺎم دﻫﻴﺪ. ﻣﺎ ﻧﺮم اﻓﺰار را در ﻣﺤﻴﻂ MATALB2015b ﻧﺼﺐ ﻧﻤﻮده اﻳﻢ. ﺟﻬﺖ ﻧﺼﺐ ﻧﺮم اﻓﺰار ﻻزم اﺳﺖ ﻛﺎﻣﭙﺎﻳﻠﺮ Studio Visual ﺑﺮ روي ﻛﺎﻣﭙﻴﻮﺗﺮ ﺷﻤﺎ ﻧﺼﺐ ﺑﺎﺷﺪ و ﺑﻪ ﻧﻈﺮ ﻣﻲ رﺳﺪ ﻛﻪ ورژن Visual Studio ﺑﺎﻳﺪ ﻗﺪﻳﻤﻲ ﺗﺮ از MATLAB ﺑﺎﺷﺪ. ﻣﺎ از Visual Studio2015 اﺳﺘﻔﺎده ﻛﺮده اﻳﻢ .

1. اﺑﺘﺪا در ﺧﻂ ﻓﺮﻣﺎن ﻣﻄﻠﺐ دﺳﺘﻮ زﻳﺮ را ﺗﺎﻳﭗ ﻛﻨﻴﺪ :

mex –setup

در ﺻﻮرت ﻧﺼﺐ ﺑﻮدن Visual studio ﺗﻮﺿﻴﺤﺎت ﻣﺮﺑﻮط ﺑﻪ ﻇﺎﻫﺮ ﻣﻲ ﮔﺮدد.

1. ﺑﺎ اﺳﺘﻔﺎده از دﺳﺘﻮر cd در ﺧﻂ ﻓﺮﻣﺎن ﻣﻄﻠﺐ ﺑﻪ ﭘﻮشه MatConNet که دانلود کرده اید بروید و این پوشه را با دستور زیر به path متلب اضافه نمایید.

addpath matlab

1. در ﺧﻂ ﻓﺮﻣﺎن ﻣﻄﻠﺐ دﺳﺘﻮر vl\_compilenn را وارد ﻛﻨﻴﺪ. و ﺻﺒﺮ ﻛﻨﻴﺪ ﺗﺎ ﻣﺮﺣﻠﻪ ﻛﻤﭙﺎﻳﻞ ﻛﺎﻣﻞ ﺷﻮد.
2. در ﺧﻂ ﻓﺮﻣﺎن ﻣﻄﻠﺐ دﺳﺘﻮر vl\_setupnn را وارد ﻛﻨﻴﺪ.
3. ﺟﻬﺖ ﺗﺴﺖ ﻧﺮم اﻓﺰار و اﻃﻤﻴﻨﺎن از ﺻﺤﺖ ﻧﺼﺐ، دﺳﺘﻮرvl\_testnn را در ﺧﻂ ﻓﺮﻣﺎن متلب وارد ﻧﻤﺎﻳﻴﺪ.

ﺧﻼﺻﻪ ﻣﺮاﺣﻞ ﻓﻮق ﺑﻪ ﺻﻮرت زﻳﺮ اﺳﺖ:

1. mex -setup

2. addpath matlab

3. vl\_compilenn

4. vl\_setupnn

5. vl\_testnn

ﻛﺎر ﺑﺎ ﻧﺮم اﻓﺰار

در اداﻣﻪ ﻃﻲ ﺳﻪ ﻣﺜﺎل ﻛﺎرﺑﺮدي ﺑﺎ ﺑﺮﺧﻲ از ﻗﺎﺑﻠﻴﺖ ﻫﺎي ﻧﺮم اﻓﺰار آﺷﻨﺎ ﻣﻲ ﺷﻮﻳﻢ.

ﻣﺜﺎل 1) ﺗﺴﺖ ﺷﺒﻜﻪ آﻣﻮزش داده ﺷﺪه GoogleNet :

در اﻳﻦ ﻣﺜﺎل ﺑﺎ ﻳﻚ ﺑﺮﻧﺎﻣﻪ ﺳﺎده، ﺑﺎ ﻗﺎﺑﻠﻴﺖ ﻫﺎ و ﻗﺪرت ﺷﺒﻜﻪ GoogleNet آﺷﻨﺎ ﻣﻲ ﺷﻮﻳﻢ. ﺑﺮاي اﻳﻨﻜﺎر ﻻزم اﺳﺖ در اﺑﺘﺪا، ﺷﺒﻜﻪ ﻓﻮق را داﻧﻠﻮد ﻛﻨﻴﻢ. ﻟﻴﻨﻚ داﻧﻠﻮد ﺷﺒﻜﻪ ﺑﻪ ﺻﻮرت زﻳﺮ اﺳﺖ:

http://www.vlfeat.org/matconvnet/models/imagenet-vgg-f.mat

ﺣﺠﻢ اﻳﻦ ﻓﺎﻳﻞ ﺣﺪود 240ﻣﮕﺎﺑﺎﻳﺖ اﺳﺖ و ﺳﺎﺧﺘﺎر ﺷﺒﻜﻪ و وزن ﻫﺎ را در ﺧﻮد ذﺧﻴﺮه ﻛﺮده اﺳـﺖ ﺑﻬﺘـﺮ اﺳﺖ ﺑﻌﺪ از داﻧﻠﻮد ﻓﺎﻳﻞ، آﻧﺮا در ﭘﻮﺷﻪ ﻛﻪ در ﻣﺮﺣﻠﻪ 2 ﻧﺼـﺐ ﻣﺴـﻴﺮ آﻧـﺮا ﺑـﻪ ﻣﻄﻠـﺐ اﺿـﺎﻓﻪ ﻛﺮده اﻳﺪ ﻛﭙﻲ ﻧﻤﺎﻳﻴﺪ ﺗﺎ ﺑﺮاي اﺳﺘﻔﺎده ﻫﺎي ﺑﻌﺪي در دﺳﺘﺮس ﺑﺎﺷﺪ. ﺷﺒﻜﻪ GoogleNet ﻳﻚ ﺷـﺒﻜﻪ 22 ﻻﻳﻪ اﺳﺖ ﻛﻪ ﺑﺎ 1/2 ﻣﻠﻴﻮن ﻋﻜﺲ رﻧﮕﻲ 224 در 224 آﻣﻮزش داده ﺷﺪه اﺳﺖ. ﺧﺮوﺟﻲ ﺷـﺒﻜﻪ ﺷـﺎﻣﻞ1000 دﺳﺘﻪ ﻣﻲ ﺑﺎﺷﺪ. ﺟﻬﺖ آﺷﻨﺎﻳﻲ ﺑﻴﺸﺘﺮ ﺑﺎ اﻳﻦ ﺷﺒﻜﻪ ﻣﻲ ﺗﻮاﻧﻴﺪ ﻣﻘﺎﻟﻪ زﻳﺮ را ﻣﻄﺎﻟﻌﻪ ﻛﻨﻴﺪ.

going deeper with convolutions [CVPR2015]

ﭘﺲ از داﻧﻠﻮد ﺷﺒﻜﻪ ﺑﺎ اﺳﺘﻔﺎده از ﺑﺮﻧﺎﻣﻪ زﻳﺮ ﺷﺒﻜﻪ را ﻟﻮد و ﺗﺴﺖ ﻧﻤﺎﻳﻴﺪ.(قبلش باید MatConNet رو نصب(کامپایل) کرده باشیم)

clc

clear

close all;

% add MatConvNet installed directory

با 2 خط کد زیر دیگر لازم نیست در هر بار اجرا دایرکتوری که MatConvNet در آن نصب شده است به صورت دستی وارد کنیم:(این2 خط کد این کار را قبل از اجرای سایر قسمت ها انجام میدهد)

cd F:\Documents\MATLAB\MatConvNet\matconvnet-1.0-beta17;

addpath matlab;

%add pre MatConvNet setup

vl\_setupnn;

% load the pre-trained CNN

net = load('imagenet-vgg-f.mat');

% load and preprocess an image

im = imread('13.jpg') ;

im\_ = single(im) ; % note: 0-255 range

im\_ = imresize(im\_, net.meta.normalization.imageSize(1:2)) ;

im\_ = im\_ - net.meta.normalization.averageImage ;

% run the CNN

res = vl\_simplenn(net, im\_) ;

% show the classification result

scores = squeeze(gather(res(end).x)) ;

[bestScore, best] = max(scores);

figure(1) ; clf ; imagesc(im) ;

title(sprintf('%s (%d), score %.3f',...

net.meta.classes.description{best}, best, bestScore)) ;

ﻣﻬﻤﺘﺮﻳﻦ دﺳﺘﻮر از ﺑﺮﻧﺎﻣﻪ ﻓﻮق، دﺳﺘﻮر vl\_simplenn اﺳﺖ ﻛﻪ ﺷﺒﻜﻪ و ﻳﻚ ﻧﻤﻮﻧﻪ ﻋﻜﺲ ﻣﻲ ﮔﻴﺮد و ﺧﺮوﺟﻲ را ﺗﻮﻟﻴﺪ ﻣﻲ ﻛﻨﺪ. اﻳﻦ دﺳﺘﻮر ﻣﻌﺎدل دﺳﺘﻮر sim از toolbox ﺷﺒﻜﻪ ﻋﺼﺒﻲ ﻣﻄﻠﺐ اﺳﺖ.

ﺗﻤﺮﻳﻦ: 10 ﻋﻜﺲ از ﻣﺤﻴﻂ اﻃـﺮاف ﺧـﻮد ﺑﮕﻴﺮﻳـﺪ و ﺑـﻪ ﺟـﺎي ﻋﻜـﺲpeooers.png ﺑـﻪ ﺷـﺒﻜﻪ ﺑﺪﻫﻴﺪ. ﻣﺸﺎﻫﺪات ﺧﻮد را ﮔﺰارش دﻫﻴﺪ.

ﻣﺜﺎل 2 - Transfer Learning

ﭘﺲ از آﺷﻨﺎ ﺷﺪن ﺑﺎ ﻗﺪرت ﺷـﺒﻜﻪgoogleNet ، ﻗﺼـﺪ دارﻳـﻢ از اﻳـﻦ ﺷـﺒﻜﻪ ﺑـﺮاي ﻛﻼﺳـﻪ ﺑﻨـﺪي ﺗﺼﺎوﻳﺮ ﺑﺮ اﺳﺎس ﻧﻴﺎز ﺧﻮد اﺳﺘﻔﺎده ﻛﻨﻴﻢ. اﺳﺘﻔﺎده از ﺷﺒﻜﻪ ﻫﺎي از ﭘﻴﺶ آﻣـﻮ زش داده ﺷـﺪه ﺑـﺮ روي داده ﻫﺎي دﻳﮕﺮ) ﻏﻴﺮ از داده ﻫﺎي آﻣﻮزﺷﻲ( ﺗﺤﺖ ﻋﻨﻮان Transfer Learning ﻣﻄﺮح ﻣﻲ ﺷﻮد. ﻣﺜﻼ ﻣﻤﻜﻦ اﺳﺖ ﺷﺨﺼﻲ، ﺷﺒﻜﻪ اي ﺑﺮاي ﺗﺸﺨﻴﺺ اﺳﺐ از زراﻓﻪ آﻣﻮزش داده ﺑﺎﺷﺪ و ﺷﻤﺎ ﻗﺼـﺪ داﺷـﺘﻪ ﺑﺎﺷﻴﺪ از اﻳﻦ ﺷﺒﻜﻪ ﺑﺮاي ﺗﺸﺨﻴﺺ ﺳﮓ از ﮔﺮﺑﻪ اﺳﺘﻔﺎده ﻛﻨﻴﺪ.

ﻣﺰﻳﺖ اﻳﻦ روش اﻳﻦ اﺳﺖ ﻛﻪ ﻣﻌﻤﻮﻻ ﺑﺎ داده ﻫﺎي آﻣﻮزﺷﻲ ﺑﻪ ﻣﺮاﺗﺐ ﻛﻤﺘﺮ از داده ﻫﺎي آﻣﻮزﺷﻲ ﺷـﺒﻜﻪ آﻣﻮزش داده ﺷﺪه اوﻟﻴﻪ، ﻣﻲ ﺗﻮاﻧﻴﻢ ﺷﺒﻜﻪ ﻗﺪرﺗﻤﻨﺪي ﺟﻬﺖ ﻣﺴﺎﻟﻪ ﺧﻮد آﻣﻮزش دﻫﻴﻢ. روش ﻛــﺎر ﺑــﻪ اﻳــﻦ ﺻــﻮرت اﺳــﺖ ﻛــﻪ از ﺷــﺒﻜﻪ آﻣــﻮزش داده ﺷــﺪه ﺑــﻪ ﻋﻨــﻮان Feature Extractor اﺳﺘﻔﺎده ﻣﻲ ﻛﻨﻴﻢ. ﻳﻌﻨﻲ داده ﻫﺎي ﺧﻮد را ﺑﻪ ﺷﺒﻜﻪ وارد ﻣﻲ ﻛﻨﻴﻢ و ﺧﺮوﺟـﻲ ﻻﻳـﻪ ﻳﻜﻲ ﻣﺎﻧﺪه ﺑﻪ آﺧﺮ را ﺑﻪ ﻋﻨﻮان ﺑﺮدار وﻳﮋﮔﻲ اﺳﺘﻔﺎده ﻣﻲ ﻛﻨﻴﻢ. ﺑﺎ در اﺧﺘﻴﺎر داﺷﺘﻦ وﻳﮋﮔﻲ ﻫـﺎي ﻣﻨﺎﺳـﺐ، ﻳﻚ ﺷﺒﻜﻪ ﺳﺎده ﺑﺮاي ﻣﺴﺎﻟﻪ ﺧﻮد ﻃﺮاﺣﻲ ﻣﻲ ﻛﻨﻴﻢ و آن را آﻣﻮزش ﻣﻲ دﻫﻴﻢ. در اداﻣﻪ، ﺑﻪ ﻋﻨﻮان ﻣﺜﺎل ﺷﺒﻜﻪ GoogleNet را ﺑﻪ ﻋﻨﻮان Feature Extractorﺟﻬﺖ ﺗﺸﺨﻴﺺ ﻫﻮاﭘﻴﻤﺎي ﻣﺴﺎﻓﺮي از ﺟﻨﮕﻨﺪه ﺑﻜﺎر ﺧﻮاﻫﻴﻢ ﮔﺮﻓﺖ. ﻣﺎ ﺣـﺪود 70 ﻋﻜـﺲ از ﻫ ﺮﻳـﻚ از ﻛـﻼس ﻫﺎي ﻫﻮاﭘﻴﻤﺎي ﺟﻨﮕﻨﺪه و ﻣﺴﺎﻓﺮﺑﺮي را ﺟﻤﻊ آوري ﻛﺮدﻳﻢ. ﻫﺮﻳﻚ از ﺗﺼﺎوﻳﺮ را ﺑﻪ ﺷﺒﻜﻪ وارد ﻛـﺮدﻳﻢ و ﺧﺮوﺟﻲ ﻻﻳﻪ ﻣﺎﻗﺒﻞ آﺧﺮ را ﺑﻪ ﻋﻨﻮان ﺑﺮدار وﻳﮋﮔﻲ ﺑﺮاي ﻫـﺮ ﻋﻜـﺲ ذﺧﻴـﺮه ﻛـﺮدﻳﻢ ﺳـﭙﺲ ﻳـﻚ SVM ﺧﻄﻲ(ﺑﺪون ﻛﺮﻧﻞ) ﺑﺮاي ﻛﻼﺳﻪ ﺑﻨﺪي آﻣﻮزش دادﻳﻢ.

clc

clear

close all;

% load the pre-trained CNN

cnnModel.net = load('imagenet-vgg-f.mat');

%% Load images from folder

% Use imageSet to load images stored in pet\_images folder imset = imageSet('Aircraft\_images','recursive');

% Preallocate arrays with fixed size for prediction

imageSize = cnnModel.net.normalization.imageSize; trainingImages = zeros([imageSize

sum([imset(:).Count])],'single');

% Load and resize images for prediction

counter = 0;

for ii = 1:numel(imset)

for jj = 1:imset(ii).Count

counter = counter +1;

trainingImages(:,:,:, counter) =

imresize(single(read(imset(ii),jj)),imageSize(1:2));

end

end

% Get the image labels

trainingLabels = getImageLabels(imset);

summary(trainingLabels) % Display class label distribution

cnnModel.info.opts.batchSize = 30;

cnnFeatures = cnnPredict(cnnModel,trainingImages);

%% Train a classifier using extracted features

% Here I train a linear support vector machine (SVM) classifier.

svmmdl = fitcsvm(cnnFeatures,trainingLabels);

% Perform crossvalidation and check accuracy

cvmdl = crossval(svmmdl,'KFold',10);

fprintf('kFold CV accuracy: %2.2f\n',1-cvmdl.kfoldLoss)

ﻓﺎﻳﻞ ﻫﺎي ﻣﺮﺑﻮﻃﻪ ﺿﻤﻴﻤﻪ اﻳﻦ ﮔﺰارش ﻫﺴﺘﻨﺪ. ﺑﺎ اﻳﻦ روش ﻣﺎ ﺑﺎ دﻗﺖ 93 درﺻﺪ رﺳﻴﺪﻳﻢ.

ﺑﻪ دﻟﺨﻮاه دو ﻛﻼس اﻧﺘﺨﺎب ﻛﻨﻴﺪ، ﺗﺼـﺎوﻳﺮي از ﻛﻼﺳـﻬﺎ ﺟﻤـﻊ آوري ﻛﻨﻴـﺪ و ﭘﺮوﺳـﻪ ﻓـﻮق را ﺑﺮاي داده ﻫﺎي ﺧﻮدﺗﺎن ﺗﻜﺮار ﻛﻨﻴﺪ. ﻧﺘﺎﻳﺞ ﺧﻮد را ﮔﺰارش دﻫﻴﺪ.

ﺗﻤﺮﻳﻦ2 ﺑﻪ ﺟﺎي SVM از ﻳﻚ ﻧﺮون ﺳﺎده ﻫﻤﺮا ﺑﺎ رﮔـﻮﻻرﻳﺰه ﻛـﺮ دن اﺳـﺘﻔﺎده ﻛﻨﻴـﺪ ﻧﺘـﺎﻳﺞ ﺧـﻮد را ﺑـﺎ ﻣﺮﺣﻠﻪ ﻗﺒﻞ ﻣﻘﺎﻳﺴﻪ ﻛﻨﻴﺪ.

ﻣﺜﺎل3 - روش ﺳﺎﺧﺖ ﻳﻚ ﺷﺒﻜﻪ ﺑﺎ اﺳﺘﻔﺎده از ﻧﺮم اﻓﺰار MatConvNet

در ﻣﺮاﺣﻞ ﻗﺒﻞ روش اﺳﺘﻔﺎده از ﺷﺒﻜﻪ ﻫﺎي از ﭘﻴﺶ آﻣﻮزش داده ﺷﺪه را ﺑﺮرﺳﻲ ﻛﺮدﻳﻢ. در اﻳﻦ ﻣﺮﺣﻠﻪ روش ﻃﺮاﺣﻲ ﻳﻚ ﺷﺒﻜﻪ از ﭘﺎﻳﻪ و آﻣﻮزش آن را ﺑﺮرس ﻣﻲ ﻛﻨﻴﻢ. ﺷﺒﻜﻪ ﺧﻮد را ﺑﺮ روي داده ﻫﺎي MNIST ﺗﺴﺖ ﻣﻲ ﻛﻨﻴﻢ. ﻣﺮاﺣﻞ اﻧﺠﺎم ﻛﺎر در ﺑﺮﻧﺎﻣﻪ زﻳﺮ آﻣﺪه اﺳﺖ:

clear;

close all;

clc

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

%

options

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

opts.dataDir = fullfile('data','mnist') ;

opts.train.batchSize = 100;

opts.train.numEpochs = 3 ;

opts.train.learningRate = 0.001 ;

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

%

Prepare data

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

imdb = getMnistImdb(opts);

f=1/100 ;

net.layers = {} ;

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

'weights', {{f\*randn(5,5,1,20, 'single'), zeros(1, 20,

'single')}}, ...

'stride', 1, ...

'pad', 0) ;

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

'method', 'max', ...

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

'stride', 2, ...

'pad', 0) ;

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

'weights', {{f\*randn(5,5,20,50,

'single'),zeros(1,50,'single')}}, ...

'stride', 1, ...

'pad', 0) ;

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

'method', 'max', ...

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

'stride', 2, ...

'pad', 0) ;

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

'weights', {{f\*randn(4,4,50,500, 'single'),

zeros(1,500,'single')}}, ...

'stride', 1, ...

'pad', 0) ;

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

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

'weights', {{f\*randn(1,1,500,10, 'single'),

zeros(1,10,'single')}}, ...

'stride', 1, ...

'pad', 0) ;

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

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

%

Train

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

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

opts.train, ...

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

ﺗﻤﺮﻳﻦ: ﺑﺮﻧﺎﻣﻪ ﻓﻮق را اﺟﺮا ﻛﻨﻴﺪ و ﻧﺘﺎﻳﺞ ﺑﺪﺳﺖ آﻣﺪه را ﺑﺎ ﻧﺘﺎﻳﺞ ﺗﻜﻠﻴﻒ ﭘﻴﺶ ﻣﻘﺎﻳﺴﻪ ﻛﻨﻴﺪ.

# کار با تصاویر در متلب

**Working with Images in MATLAB**

**Teacher's Day Workshop**

**School of Computing and Communications**

**December 2013**

1. **Work with Images in MATLAB**

Digital image is composed of a two or three dimensional matrix of pixels. Individual pixels contain a number or numbers representing what grayscale or color value is assigned to it. Color pictures generally contain three times as much data as grayscale pictures, depending on what color representation scheme is used. Therefore, color pictures take three times as much computational power to process.

MATLAB can import/export several image formats:

* BMP (Microsoft Windows Bitmap)
* GIF (Graphics Interchange Files)
* HDF (Hierarchical Data Format)
* JPEG (Joint Photographic Experts Group)
* PCX (Paintbrush)
* PNG (Portable Network Graphics)
* TIFF (Tagged Image File Format)
* XWD (X Window Dump)
* raw-data and other types of image data.
  1. ***Read and Display an Image***

You can read standard image files by using the *imread* function. The type of data returned by *imread* depends on the type of image you are reading. For example, read image1.jpg by typing (the image can be downloaded using the following link. [http://crin.eng.uts.edu.au/~rob/image1.jpg,](http://crin.eng.uts.edu.au/%7Erob/image1.jpg) and then can be copied into the current folder):

A = imread('image1.jpg');

which will stores *image1.jpg* in a matrix named A.

Now display the image using the *imshow* function. For example, type:

imshow(A);

* 1. ***Grayscale Images***

A grayscale image is a data matrix whose values represent intensities within some range. MATLAB stores a grayscale image as an individual matrix, with each element of the matrix corresponding to one image pixel.

B = rgb2gray(A);

Now display the image by typing:

imshow(B);

* 1. ***Write the Image to a Disk File***

To write the newly adjusted image B to a disk file, use the *imwrite* function. If you include the filename extension '.png', the *imwrite* function writes the image to a file in Portable Network Graphics (PNG) format, but you can specify other formats. For example, type:

imwrite (B, 'image2.png');

* 1. ***Check the Contents of the Newly Written File***

To see what *imwrite* wrote to the disk file, use the *imfinfo* function.

imfinfo('image2.png')

The *imfinfo* function returns information about the image in the file, such as its format, size, width, and height.

ans =

Filename: 'image2.png' FileModDate: '12-Nov-2013 10:43:31'

FileSize: 52936 Format: 'png'

FormatVersion: []

Width: 350

Height: 350

BitDepth: 8 ColorType: 'grayscale'

FormatSignature: [137 80 78 71 13 10 26 10]

Colormap: []

Histogram: []

InterlaceType: 'none'

Transparency: 'none'

SimpleTransparencyData: []

BackgroundColor: []

RenderingIntent: []

Chromaticities: []

Gamma: []

XResolution: []

YResolution: []

ResolutionUnit: []

XOffset: []

YOffset: []

OffsetUnit: []

SignificantBits: []

ImageModTime: '11 Nov 2013 23:43:31 +0000'

Title: []

Author: []

Description: []

Copyright: []

CreationTime: []

Software: []

Disclaimer: []

Warning: []

Source: []

Comment: []

OtherText: []

* 1. ***Resize an Image***

To resize an image, use the *imresize* function. When you resize an image, you specify the image to be resized and the magnification factor. To enlarge an image, specify a magnification factor greater than 1. To reduce an image, specify a magnification factor between 0 and 1.

imshow(B);

C = imresize(B,1.5);

figure imshow(C);

C = imresize(B,0.5); figure

imshow(C);

You can specify the size of the output image by passing a vector that contains the number of rows and columns in the output image. If the specified size does not produce the same aspect ratio as the input image, the output image will be distorted.

C = imresize(B,[300,150]); figure

imshow(C);

This example creates an output image with 300 rows and 150 columns.

* 1. ***Rotate an Image***

To rotate an image, use the *imrotate* function. When you rotate an image, you specify the image to be rotated and the rotation angle, in degrees. If you specify a positive rotation angle, *imrotate* rotates the image counterclockwise; if you specify a negative rotation angle, *imrotate* rotates the image clockwise.

C = imrotate(B,35); figure

imshow(C);

C = imrotate(B,-20); figure

imshow(C);

* 1. ***Crop an Image***

Cropping an image means creating a new image from a part of an original image. To crop an image using the Image Viewer, use the ***Crop Image*** tool or use the *imcrop* function.

**Using the Crop Image Tool:**

By default, if you close the Image Viewer, it does not save the modified image data. To save the cropped image, you can use the Save As option from the Image Viewer File menu to store

the modified data in a file or use the Export to Workspace option to save the modified data in the workspace variable. To use the Crop Image tool, follow this procedure:

1. View an image in the Image Viewer.

imtool(A);

1. Start the Crop Image tool by clicking ***Crop Image *** in the Image Viewer toolbar or selecting ***Crop Image*** from the Image Viewer Tools menu. (Another option is to open a figure window with imshow and call imcrop from the command line.) When you move the pointer over the image, the pointer changes to cross hairs .
2. Define the rectangular crop region, by clicking and dragging the mouse over the image. You can fine-tune the crop rectangle by moving and resizing the crop rectangle using the mouse.
3. When you are finished defining the crop region, perform the crop operation. Double-click the left mouse button or right-click inside the region and select ***Crop Image*** from the context menu. The Image Viewer displays the cropped image.
4. To save the cropped image, use the Save as option or the Export to Workspace option on the Image Viewer File menu.

Now display the image using the *imshow* function.

**Using the imcrop Function:**

By using the *imcrop* function, you can specify the crop region interactively using the mouse or programmatically by specifying the size and position of the crop region.

This example illustrates an interactive syntax. The example reads an image into the MATLAB workspace and calls *imcrop* specifying the image as an argument. *imcrop* displays the image in a figure window and waits for you to draw the crop rectangle on the image. When you move the pointer over the image, the shape of the pointer changes to cross hairs . Click and drag the pointer to specify the size and position of the crop rectangle. You can move and adjust the size of the crop rectangle using the mouse. When you are satisfied with the crop rectangle, double-click to perform the crop operation, or right-click inside the crop rectangle and select Crop Image from the context menu. *imcrop* returns the cropped image.

C = imcrop(A); figure imshow(C);

**Raw MATLAB:** For advanced users, the native MATLAB commands can be used. You can specify the size and position of the crop rectangle as parameters when you call *imcrop*. Specify the crop rectangle as a four-element position vector, [xmin ymin width height].

In this example, you call *imcrop* specifying the image to crop, A, and the crop rectangle.

*imcrop* returns the cropped image in D.

D = imcrop(A,[160 140 110 180]);

figure imshow(D);

* 1. ***Getting Image Pixel Values***

You can get information about specific image pixels such as RGB values. Type:

A(2,15,:)

which returns the RGB (red, green, and blue) color values of the pixel (2,15). R=66; G= 88; B= 174.

ans(:,:,1) = 66

ans(:,:,2) = 88

ans(:,:,3) = 174

Now try:

A(40:100,10:20,:)

You can also use the *impixel* function which will determine the values of one or more pixels in an image and return the values in a variable. Select the pixels interactively using a mouse. *impixel* returns the value of specified pixels in a variable in the MATLAB workspace.

The following example illustrates how to use *impixel* to get pixel values.

1. Display an image.

imshow(A);

1. Call *impixel*. When called with no input arguments, *impixe*l associates itself with the image in the current axes.

vals = impixel

1. Select the points you want to examine in the image by clicking the mouse. *impixel*

places a star at each point you select.



1. When you are finished selecting points, press Return. *impixel* returns the pixel values in an n-by-3 array, where n is the number of points you selected. The stars used to indicate selected points disappear from the image.

|  |  |  |
| --- | --- | --- |
| vals =  46 | 71 | 155 |
| 80 | 96 | 184 |
| 95 | 107 | 193 |

* 1. ***Changing Image Pixel Values***

You can change the values of specific image pixels. Type:

A(40:100,10:20,:) = 0;

figure imshow(A);

which changes the colors of the selected pixels into black color.

Now try:

A(40:100,10:20,:) = 255;

figure imshow(A);

* 1. ***Image Intensity Adjustment***

Image intensity adjustment is used to improve an image, Read *image1.jpg* again.

A = imread('image1.jpg');

Multiply the image pixels values by two.

E = A.\*2;

figure imshow(E);

Now try:

F = A.\*0.75;

figure imshow(F);

Then, try:

F = A.\*7.5;

figure imshow(F);

* 1. ***Detecting Edges Using the edge Function***

In an image, an edge is a curve that follows a path of rapid change in image intensity. Edges are often associated with the boundaries of objects in a scene. Edge detection is used to identify the edges in an image. To find edges, you can use the edge function. This function looks for places in the image where the intensity changes rapidly, using one of these two criteria:

* + - Places where the first derivative of the intensity is larger in magnitude than some threshold.
    - Places where the second derivative of the intensity has a zero crossing.

edge provides a number of derivative estimators, each of which implements one of the definitions above. For some of these estimators, you can specify whether the operation should be sensitive to horizontal edges, vertical edges, or both. edge returns a binary image containing 1’s where edges are found and 0’s elsewhere.

The most powerful edge-detection method that edge provides is the Canny method. The Canny method differs from the other edge-detection methods in that it uses two different thresholds (to detect strong and weak edges), and includes the weak edges in the output only if they are connected to strong edges. This method is therefore less likely than the others to be fooled by noise, and more likely to detect true weak edges.

The following example illustrates the power of the Canny edge detector by showing the results of applying the Sobel and Canny edge detectors to the same image:

1. Read the image and display it. G = imread('image2.png'); imshow(G);
2. Apply the Sobel and Canny edge detectors to the image and display them.

BW1 = edge(G,'sobel'); BW2 = edge(G,'canny');

figure imshow(BW1);

figure imshow(BW2);

* 1. ***Removing Noise from an Image***

Digital images are prone to a variety of types of noise. Noise is the result of errors in the image acquisition process that result in pixel values that do not reflect the true intensities of the real scene. There are several ways that noise can be introduced into an image, depending on how the image is created. For example:

* If the image is scanned from a photograph made on film, the film grain is a source of noise. Noise can also be the result of damage to the film, or be introduced by the scanner itself.
* If the image is acquired directly in a digital format, the mechanism for gathering the data (such as a CCD detector) can introduce noise.
* Electronic transmission of image data can introduce noise.

You can use linear filtering to remove certain types of noise. Certain filters, such as averaging or Gaussian filters, are appropriate for this purpose. For example, an averaging filter is useful for removing grain noise from a photograph. Because each pixel gets set to the average of the pixels in its neighborhood, local variations caused by grain are reduced.

Median filtering is similar to using an averaging filter, in that each output pixel is set to an average of the pixel values in the neighborhood of the corresponding input pixel. However, with median filtering, the value of an output pixel is determined by the median of the neighborhood pixels, rather than the mean. The median is much less sensitive than the mean to extreme values (called outliers). Median filtering is therefore better able to remove these outliers without reducing the sharpness of the image.

The following example compares the use of a linear Gaussian filter and a median filter to remove salt and pepper noise for the same image:

1. Read the image and display it. H = imread('image2.png'); imshow(H);
2. Add salt and pepper noise to the image and then display it.

I = imnoise(H,'salt & pepper',0.02);

figure imshow(I);

1. Filter the noisy image using a linear Gaussian filter.
   * Create a Gaussian filter using the fspecial function.

filter = fspecial('gaussian',[3 3], 0.5);

* + Filter the image using the created filter and then display the filtered image.

J = imfilter(I, filter, 'replicate');

figure imshow(J);

1. Filter the noisy image using a median filter by applying the medfilt2 function and then display the filtered image.

K = medfilt2(I,[3 3]);

figure imshow(K);

1. **Getting Help in MATLAB**

For reference information about any of the functions, type in the MATLAB command window:

help functionname

For example:

help imread

یک سری کاربرد دیگر:

