Robust Single Image Super-Resolution via Deep Networks with Sparse Prior

گزارش پیاده سازی مقاله

اریه شده به:

دکتر محمد اسدپور

توسط:

کریم شاهی نیار

درس بینایی کامپیوتر

کارشناسی ارشد

ترم دوم سال تحصیلی 98-99

گروه برق و کامپیوتر دانشگاه تبریز

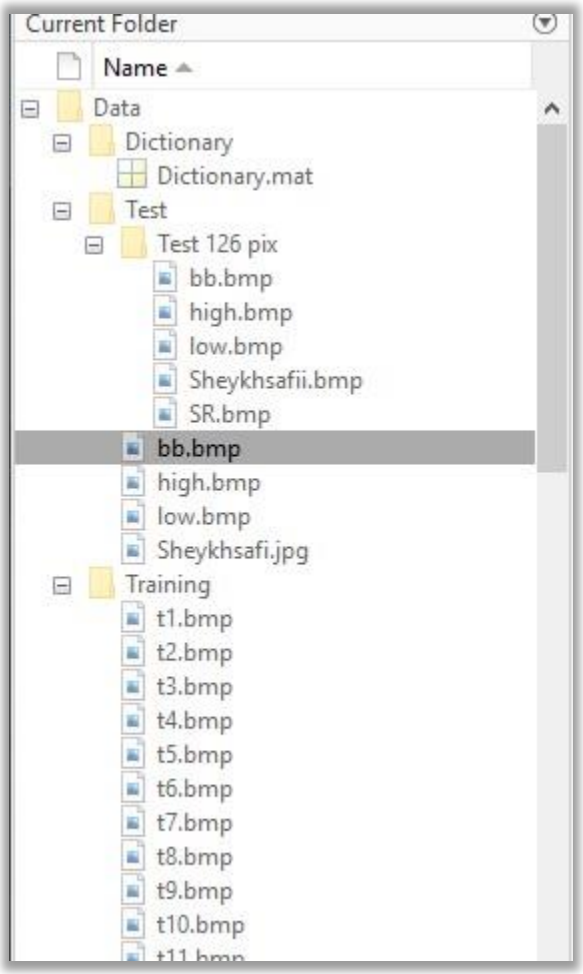
فهرست

فهرست شکلها

Abstract—Single image super-resolution (SR) is an ill-posed problem, which tries to recover a high-resolution image from its low-resolution observation. To regularize the solution of the problem, previousmethods have focused on designing good priors for natural images, such as sparse representation, or directly learning the priors from a large data set with models, such as deep neural networks. In this paper, we argue that domain expertise from the conventional sparse coding model can be combined with the key ingredients of deep learning to achieve further improved results. We demonstrate that a sparse coding model particularly designed for SR can be incarnated as a neural network with the merit of end-to-end optimization over training data. The network has a cascaded structure, which boosts the SR performance for both fixed and incremental scaling factors. The proposed training and testing schemes can be extended for robust handling of images with additional degradation, such as noise and blurring. A subjective assessment is conducted and analyzed in order to thoroughly evaluate various SR techniques. Our proposed model is tested on a wide range of images, and it significantly outperforms the existing state-of-the-art methods for various scaling factors both quantitatively and perceptually. Index Terms—Image super-resolution, deep neural networks, sparse coding

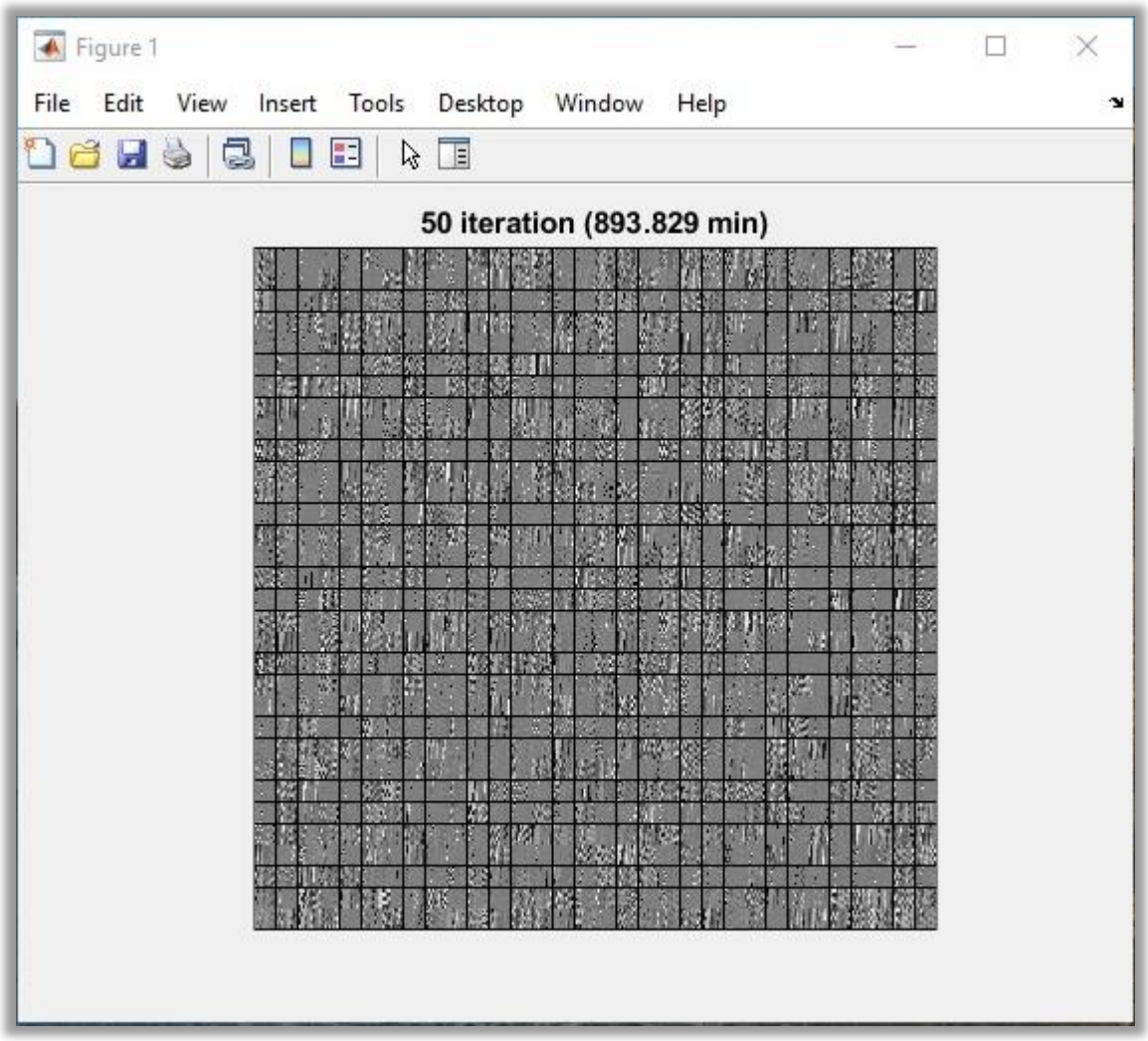
توضیح پیاده سازی

دو فانکشن داریم یکی Train و دیگری Test که از Train برای آموزش استفاده می کنیم.
در پوشه Data دو نوع دیتاست داریم. که در پوشه Training عکس هایی برای آموزش شامل همه نوع تصویر از تصویر گل تا تصویر انسان از ساختمان تا خودرو درونش به چشم میخوره. از این تصاویر برای آموزش استفاده می کنیم، که در اصل براساس سایز از تمام تصاویر فیچرهایی را استخراج می کنیم.



شکل 1 پوشه های حاوی تصاویر آموزش و تست و دیکشنری حاوی آموزش

داخل پوشه Test هم چند تصویر گذاشتیم که برای تست استفاده میکنیم که قابل جایگزینی با هر تصویر دیگر هست.
یک پوشه دیگر هم با نام Dictionary داریم که حاوی دیتای ذخیره شده آموزش برنامه هست، و دیگر لازم نیست ما برای هر تست گرفتن برنامه را آموزش بدهیم.
این فایل دیتای آموزش حاصل ۵۰ تکرار در زمان ۸۹۳ دقیقه است. در تصویر شماره دو زمان اجرا و تعداد تکرار را که در فیگور شماره یک نمایش داده شده را مشاهده می کنید.



شکل 2 نتیجه حاصل از تکرار 50

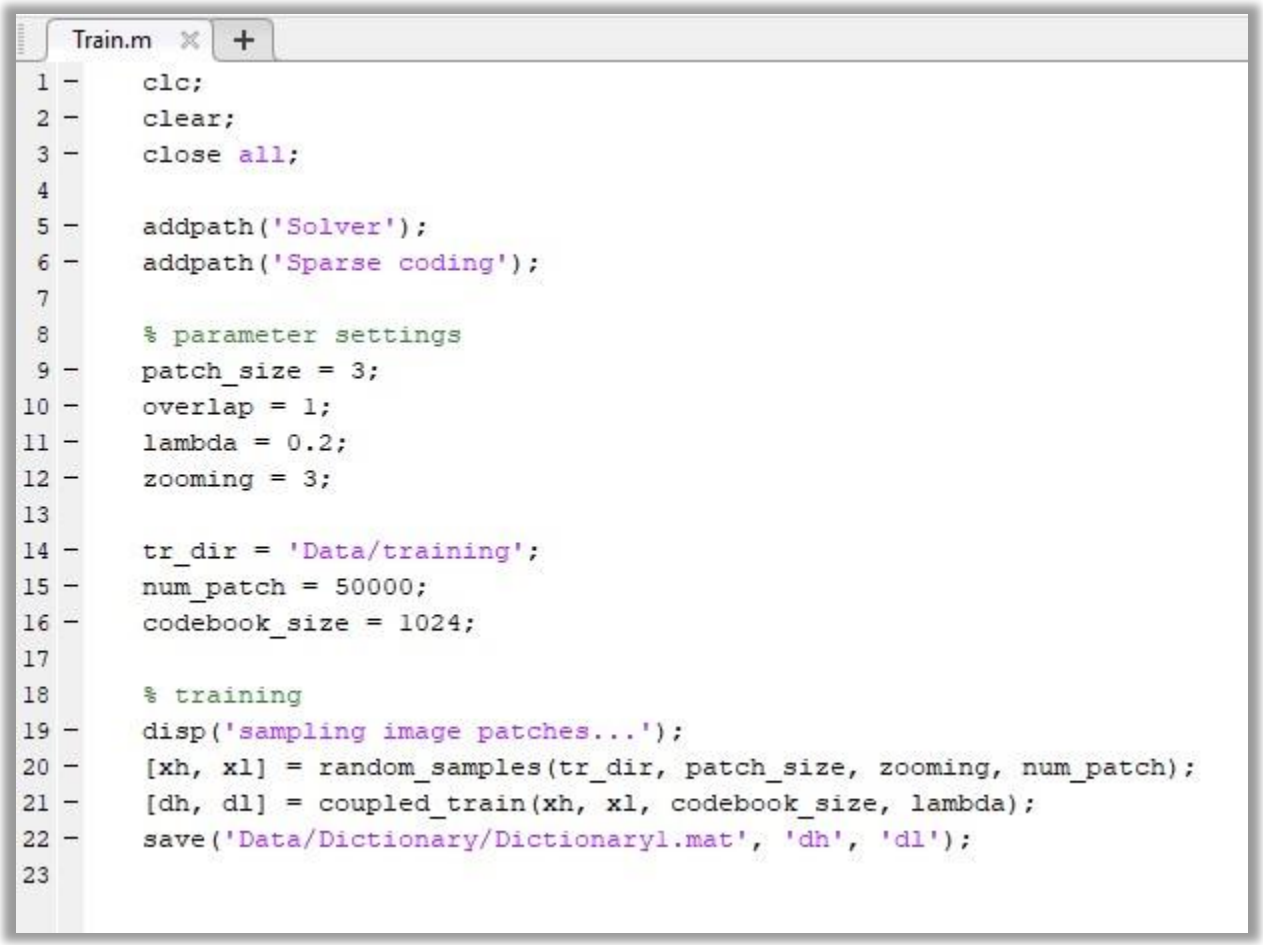
از پروژه های دیگر دو نوع کد برای پیاده سازی این مقاله کپی کرده ام که اولین نوع را در پوشه Solver قرار داده ام که شامل دو فانکشن اصلی هست که درون هرکدام دو الی سه فانکشن مجزا تعریف شده است. فانکشن های SolveLasso.m و feature_sign.m

یک نوع دیگر از فایلهایی که مستقیم از پروژه های دیگر برداشته ام درون پوشه Sparse coding هستند. شامل توابع کتابخانه ای که بر اساس پروژه C هست، که با Mexa۶۴ به صورت dll در متلب استفاده میکنیم، وگرنه خود کتابخانه با زبان C تهیه شده است، که به صورت dll برای استفاده در متلب تبدیل کرده اند.

این کدها به صورت کلی یک نوع مترجم هستند بین نوع داده متلب و نوع داده C، یعنی داده ها را از ما به زبان و فرمت متلب دریافت میکند، به نوع داده زبان تبدیل میکند، با استفاده از کامپایلر C اجرا می کند و خروجی را دریافت می کند، خروجی حاصل را به نوع متلب تبدیل کرده و به ما باز می گرداند.

در این پروژه کدهای coupled_train, random_samples, super_resolution, Test, Train را نوشتم و بقیه را از پروژه های دیگر کپی کرده ام. آنهایی که از پروژه های دیگر برداشتم واقعا پیاده سازی خیلی سنگینی دارد که مشابه کد پایتونش بود ولی ترجمه به زبان متلی هم مشکل و هم شاید خیلی زمانبر بود که ترجیح دادم از پروژه های دیگر قرض بگیرم. در خود مقاله هم خود اسپارس کدینگ از کتابخانه پایتون استفاده کرده بود و خود نویسندگان مقاله Sparse coding را پیاده نکرده اند.

فانکشن Train.m



شکل 3 کد تابع Train

در شروع دو پوشه حاوی فانکشن های پوشه Solver و Sparse coding را با دستور addpatch به پروژه معرفی می کنم با این دستور مسیر فانکشن های مورد نظر که در دو پوشه جداگانه گذاشتم را به عنوان مسیر پروژه معرفی می کنیم و در ادامه میتوانیم از آنها استفاده بکنیم. سطر ۵ و ۷

```
addpath('Solver');
addpath('Sparse coding');
```

یک سری تنظیماتی در اول لازم هست که باید برای آنها متغرهایی را تعریف بکنیم.
Patch_size متغری هست که تعداد پیکسل های تشکیل دهنده patch را در آن ذخیره می کنیم.
overlap یعنی بین دو patch مجاور چند پیکسل روی هم اورلپ شوند را در آن ذخیره میکنیم. یک را مقداردهی کردیم یعنی بین دو patch مجاور یک پیکسل روی هم اورلپ شوند.
lambda پارامتر اسپارس کودینگ هست که مقدارش را ۰.۱ قرار دادیم.
و zooming افزایش زوم را مقداردهی میکنیم، یعنی ما می خواهیم چند برابر افزایش کیفیت بدهیم عکسمان را، فقط توجه کنیم که چون مقدار این متغیر هم در آموزش و هم در تست استفاده می شود، باید در هر دو مشترک باشد، یعنی در آن دیتاست Dictionary بعد از آموزش اطلاعات بر اساس zooming قبلی ثبت می شود و آموزش می بیند، اگر مقدار این متغیر را تغیر بدهیم باید پروسه آموزش را از دوباره انجام بدهیم. سطر ۸ الی ۱۲

```
% parameter settings
patch_size = 3;
overlap = 1;
lambda = 0.2;
zooming = 3;
```

مسیر آموزش را با این خط کد معرفی می کنیم. سطر ۱۴

```
tr_dir = 'Data/training';
```

یک متغیری معرفی می کنیم و تعداد patch ی که از تمام تصاویر آموزش که در پوشه Data/training قراردادیم را به برنامه دستور می دهیم. در اینجا ۵۰۰۰۰ را مقداردهی میکنیم. سطر ۱۵

```
num_patch = 50000;
```

و در سطر 16 متغیری را ایجاد می کنیم تا اندازه دیکشنری خروجی را در آن ذخیره کنیم، این دیکشنری را در سطر 22 می بینید. که من همینجا هم می آورم تا مشاهده کنید.

```
codebook_size = 1024;
save('Data/Dictionary/Dictionary1.mat', 'dh', 'dl');
```

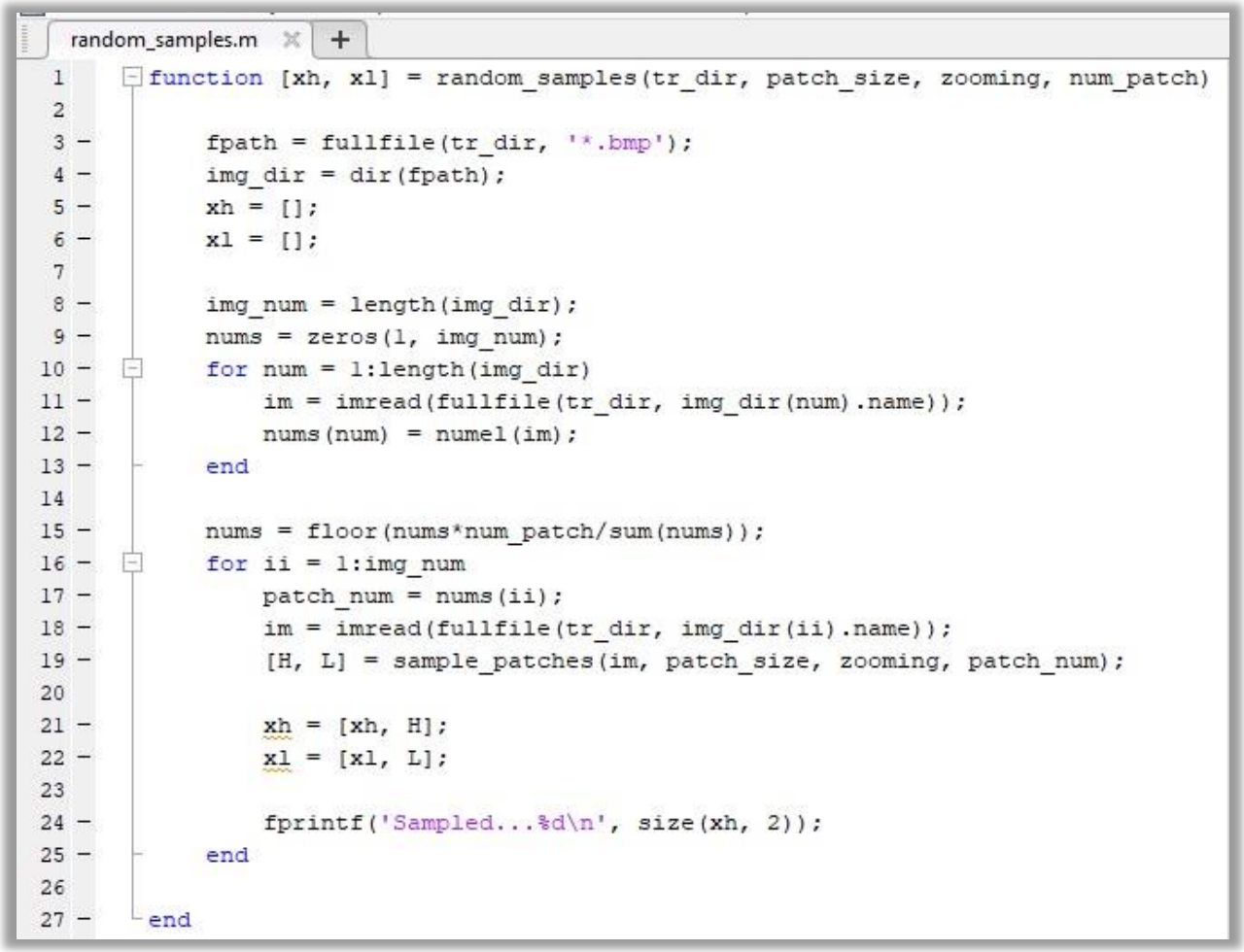
در ادامه از دو فانکش مهم در درون فانکشن Train استفاده میکنیم، فراخوانی می کنیم.

اولی random_samples هست که بر اساس فرودی هایی که ما اعمال می کنیم از درون پوشه عکس های آموزش برای ما patch ها را استخراج می کند.

و فانکشن coupled_train هست که روش Sparse coding را فرواخوانی می کند و 50000 پچ patch استخراج شده را به اسپارس کودینگ ارسال می کند و dh, dl را بازمی گرداند که ضرایب آموزش دیده ما هستند.
سطر های 18 الی 21

```
% training
disp('sampling image patches...');
[xh, xl] = random_samples(tr_dir, patch_size, zooming, num_patch);
[dh, dl] = coupled_train(xh, xl, codebook_size, lambda);
```

فانکشن random_samples



شکل 4 کد random_samples

ورودی که تابع random_samples دریافت می کند tr_dir مسیر حاوی فولدر عکس های آموزش، patch_size که سه هست، zooming که یعنی تصویر نهایی چند برابر زووم بشود و num_patch که 50000 هست.

در اولین خط دستور مسیر کامل را استخراج می کنیم، و فایلهایی که با پسوند بیت مپ هستند را مجاز برای اعمال آموزش معرفی می کنیم. سطر 3

```
fpath = fullfile(tr_dir, '*.bmp');
```

تمامی اسامی عکس ها را در متغیر img_dir ذخیره می کنیم. سطر 4

```
img_dir = dir(fpath);
```

طول img_dir را در متغیر img_num ذخیره می کنم. سطر 8

```
img_num = length(img_dir);
```

یک ماتریس صفر به اندازه img_num تولید می کنم و درون متغیر nums قرار می دهم. سطر 9

```
nums = zeros(1, img_num);
```

حلقه for ایجاد می کنم و بر اساس img_dir تعداد کل پیکسل هایی که در تک تک تصاویر هست را استخراج می کنم.

با دستور imread تصویر ها را فراخوانی کرده و درون متغیر im ذخیره می کنیم.

با دستور numel تمام المانهای تصویر را استخراج کرده و درون nums(num) قرار می دهیم.

حلقه تا اتمام تمام تصاویر درون پوشه اجرا می شود. سطر 10 الی 13

```
    for num = 1:length(img_dir)
        im = imread(fullfile(tr_dir, img_dir(num).name));
        nums(num) = numel(im);
    end
```

به نسبت اندازه تصاویر المانها را استخراج می کنیم یا اینجوری بگویم که 50000 المان که باید استخراج کنیم را به نسبت اندازه تصاویر تقسیم میکنم. یعنی به نوعی می خواهیم که یک نرمالسازی بین المانهای دریافتی از تصاویر برقرار باشد اینطور نشود که از یک تصویر کوچک مثلا 100 المان استخراج شود و از یک تصویر بزرگ 50 المان. سطر 15

```
    nums = floor(nums*num_patch/sum(nums));
```

در ادام یک حلقه for ایجاد می کنیم به تعداد تصاویر موجود در پوشه آموزش. که این تعداد را در متغیر img_num ذخیره کرده ایم.

درون حلقه تک تک تصاویر را می خوانیم و درون im قرار میدهیم. یک فانکشنی را فراخوانی می کنیم که در ورودی عکس، تعداد پچی که باید استخراج شود را در ورودی از ما می گیرد و در خروجی patch ها را بازمی گرداند. سطر 19

بعد از این کار ما patch ها را ذخیره می کنیم. سطر های 16 الی 25

```
    for ii = 1:img_num
        patch_num = nums(ii);
        im = imread(fullfile(tr_dir, img_dir(ii).name));
        [H, L] = sample_patches(im, patch_size, zooming, patch_num);

        xh = [xh, H];
        xl = [xl, L];

        fprintf('Sampled...%d\n', size(xh, 2));
    end
```

فانکشن sample_patches

```
function [HP, LP] = sample_patches(im, patch_size, zooming, patch_num)

    lz = 2;
    if size(im, 3) == 3
        hIm = rgb2gray(im);
    else
        hIm = im;
    end

    if rem(size(hIm,1),zooming)
        nrow = floor(size(hIm,1)/zooming)*zooming;
        hIm = hIm(1:nrow,:);
    end
    if rem(size(hIm,2),zooming)
        ncol = floor(size(hIm,2)/zooming)*zooming;
        hIm = hIm(:,1:ncol);
    end

    lIm = imresize(hIm,1/zooming);
    [nrow, ncol] = size(lIm);

    x = randperm(nrow-patch_size-lz-1);
    y = randperm(ncol-patch_size-lz-1);
    [X,Y] = meshgrid(x,y);

    xrow = X(:);
    ycol = Y(:);
```

شکل 5 کد sample_patches

در این فانکشن برای مثال اولین تصویر را از ورودی دریافت می کند، سایز تصویر 3*176*176از نوع uint8 Patch_size=3, zooming=3 و تعداد پچ ما در این تصویر patch_num=323 هست.

اول تصویر را به حالت گری لول میبریم با دستور rgb2gray(im) و درون متغیر hIm ذخیره می کنیم. سطر 31 الی 36

```
lz = 2;
if size(im, 3) == 3
    hIm = rgb2gray(im);
else
    hIm = im;
end
```

تعداد patch ها در جهت سطر و ستون را استخراج می کنیم. سطر 38 الی 45

```
if rem(size(hIm,1),zooming)
    nrow = floor(size(hIm,1)/zooming)*zooming;
    hIm = hIm(1:nrow,:);
end
if rem(size(hIm,2),zooming)
    ncol = floor(size(hIm,2)/zooming)*zooming;
    hIm = hIm(:,1:ncol);
end
```

در ادامه تصویر دریافتی را به نصبت یک بر روی سایز zoom کوچک میکنیم تا هم تصویر کوچک شده را در دست داشته باشیم و هم تصویر اورجینال را برای استخراج بهتر patch ها

سایز سطر و ستون تصویر کوچک شده را ذخیره می کنیم. سطر 47 و 48

```
lIm = imresize(hIm,1/zooming);
[nrow, ncol] = size(lIm);
```

در ادامه یک meshgrid ایجاد میکنیم که به ازای هر patch به اندازه تصویر کوچک با اعداد تصادفی

```
x = randperm(nrow-patch_size-lz-1);
y = randperm(ncol-patch_size-lz-1);
[X,Y] = meshgrid(x,y);
```

در ادامه براساس سایز patch ها یک xrow و ycol را مقداردهی میکنیم و تصویر اصلی را resize میکنیم، دوبرابر میکنم به صورت خطی 'bicubic'

```
xrow = X(:);
ycol = Y(:);

xrow = xrow(1:patch_num);
ycol = ycol(1:patch_num);

% zoom the original image
lIm = imresize(lIm, lz,'bicubic');
hIm = double(hIm);
lIm = double(lIm);
```

ماتریس صفر H و L را ایجاد میکنم.

```
H = zeros(zooming^2*patch_size^2,patch_num);
L = zeros(lz^2*4*patch_size^2,patch_num);
```

گرادیان ها را تعریف می کنم. هم در جهت سطر و هم درجهت ستون

بعد دو متغیر کانولوشن براسان گرادیان ها تعریف می کنم. سطر 68 الی 73

```
% compute the first and second order gradients
hf1 = [-1,0,1];
vf1 = [-1,0,1]';

lImG11 = conv2(lIm,hf1,'same');
lImG12 = conv2(lIm,vf1,'same');
```

گرادیان را برای مرتبه سوم هم تعریف میکنم و کانولوشن ها رو هم تعریف میکنم. سطر 75 الی 79

```
hf2 = [1,0,-2,0,1];
vf2 = [1,0,-2,0,1]';

lImG21 = conv2(lIm,hf2,'same');
lImG22 = conv2(lIm,vf2,'same')
```

حلقه ای ایجاد می کنیم و patch را استخراج می کنم. یعنی به ترتیب xrow و ycol که بدست آورده بودیم در سطر 57 و 58

به تعداد patch مان سطر و ستون را در hrow و hcol ذخیره میکنم

```
count = 1;
for pnum = 1:patch_num

    hrow = (xrow(pnum)-1)*zooming + 1;
    hcol = (ycol(pnum)-1)*zooming + 1;
    Hpatch = hIm(hrow:hrow+zooming*patch_size-1,hcol:hcol+zooming*patch_size-1);
```

ماتریس Hpatch را ایجاد میکنیم. اینها برای تصویر اورجینال بود برای تصویر کوچک شده هم معادل همین کار را انجام می دهم و patch ها را استخراج میکنم.

```
lrow = (xrow(pnum)-1)*lz + 1;
lcol = (ycol(pnum)-1)*lz + 1;

Lpatch1 = lImG11(lrow:lrow+lz*patch_size-1,lcol:lcol+lz*patch_size-1);
Lpatch2 = lImG12(lrow:lrow+lz*patch_size-1,lcol:lcol+lz*patch_size-1);
Lpatch3 = lImG21(lrow:lrow+lz*patch_size-1,lcol:lcol+lz*patch_size-1);
Lpatch4 = lImG22(lrow:lrow+lz*patch_size-1,lcol:lcol+lz*patch_size-1);
```

Patch های ساتخراج شده تصویر کوچک

```
Lpatch = [Lpatch1(:),Lpatch2(:),Lpatch3(:),Lpatch4(:)];
Lpatch = Lpatch(:);
```

Patch های استخراج شده تصویر بزرگ

```
HP(:,count) = Hpatch(:)-mean(Hpatch(:));
LP(:,count) = Lpatch;
```

که در ادامه patch های تصویر کوچک و تصویر بزرگ را با هم تطابق میدهم و به صورت یک دیکشنری که تصویر کوچ را به تصویر بزرگ مپ کرده است ذخیره میکنم، HP وLP را به عنوان خروجی فانکشن sample_patches به خروجی ارسال می شود. که در random_sample از آنها استفاده می کنم. اندازه دیکشنری حدود 20مگابایت برای این 91 تصویر آموش شد. که در پوشه Dictionary ذخیره میگردد.

فانکشن coupled_train

coupled_train.m

```
function [dh, dl] = coupled_train(xh, xl, codebook_size, lambda)

    addpath('Sparse coding/sc2');
%     addpath('/sc2');

    hDim = size(xh, 1);
    lDim = size(xl, 1);

    % joint learning of the dictionary
    X = [1/sqrt(hDim)*xh; 1/sqrt(lDim)*xl];
    X = X(:, 1:80000);
    Xnorm = sqrt(sum(X.^2, 1));

    clear xh xl;

    X = X(:, Xnorm > 1e-5);
    X = X./repmat(sqrt(sum(X.^2, 1)), hDim+lDim, 1);

    idx = randperm(size(X, 2));
    Binit = X(:, idx(1:codebook_size));

    [D] = sparse_coding(X, codebook_size, lambda/2, 'L1', [], 50, 5000, [], [], Binit);

    dh = D(1:hDim, :);
    dl = D(hDim+1:end, :);

    % normalize the dictionary
    dh = dh./repmat(sqrt(sum(dh.^2, 1)), hDim, 1);
    dl = dl./repmat(sqrt(sum(dl.^2, 1)), lDim, 1);

end
```

شکل 6 کد coupled_train

این فانکشن از Sparse coding استفاده می کند و شبکه ما را آموزش می دهد.

کلاسهای Sparse coding را فراخوانی می کنیم.

```
addpath('Sparse coding/sc2');
```

اینجا ابعاد فیچرهای ما را استخراج می کند و درون دو متغیر ذخیره می کند.

```
    hDim = size(xh, 1);
```

```
lDim = size(xl, 1);
```

Patchها یا همون ویژگی ها رو به هم متصل کرده و یک دیکشنری با فرمت ی متفاوت ایجاد می کنیم. که X دیکشنری جدید ما هست که برای sparse coding ورودی هایی که دارد با ورودی های ما متفاوت هست تلاش می کنیم که ورودی متناسب sparse coding را ایجاد کنیم.

```
% joint learning of the dictionary
X = [1/sqrt(hDim)*xh; 1/sqrt(lDim)*xl];
X = X(:, 1:80000);
Xnorm = sqrt(sum(X.^2, 1));
```

بعد از اینکه فرمت و چیدمان ماتریس را عوض کردیم Sparse coding را فراخوانی می کنیم.

L1 روش رگرسیون هست که می تونیم L2 را هم استفاده کنیم که در روش Sparse coding استفاده شده است.

```
clear xh xl;

X = X(:, Xnorm > 1e-5);
X = X./repmat(sqrt(sum(X.^2, 1)), hDim+lDim, 1);

idx = randperm(size(X, 2));
Binit = X(:, idx(1:codebook_size));

[D] = sparse_coding(X, codebook_size, lambda/2, 'L1', [], 50, 5000, [], [], Binit);
```

Sparse coding به ما یک ماتریس D برمی گرداند که باز ما از ماتریس D دو متغیر dh را برای تصاویر با کیفیت بالا و dl را برای تصاویر کیفیت پایین مقدار دهی میکنیم البته به صورت دیکشینری.

```
dh = D(1:hDim, :);
dl = D(hDim+1:end, :);
```

dh و dl را به نرمالایز شده به خروجی ارسال می کنم.

```
    % normalize the dictionary
    dh = dh./repmat(sqrt(sum(dh.^2, 1)), hDim, 1);
    dl = dl./repmat(sqrt(sum(dl.^2, 1)), lDim, 1);
```

که نتیجه حاصل در فانکشن Train در سطر 21 و 22 ذخیره می کنیم. این پروسه همانطور که در اول گزارش اشاره کردم خیلی زمانبر هست بخاطر همین این دیکشنری را هم به همراه گزارش ارسال خواهم کرد تا دیگر نیازی به آموزش دوباره نباشد.

```
[dh, dl] = coupled_train(xh, xl, codebook_size, lambda);
save('Data/Dictionary/Dictionary1.mat', 'dh', 'dl');
```

فانکشن Test.m

ابتدای فانکشن Test، شبیه فانکشن Train هست

Test.m

```
clc;
clear;
close all;

addpath('Solver');
addpath('Sparse coding');

% parameter settings
patch_size = 3;
overlap = 1;
lambda = 0.3;
zooming = 5;


% load training dictionaries
load('Data/Dictionary/Dictionary.mat');

% test image
fname = 'Data/Test/1.bmp';
testIm = imread(fname);
if rem(size(testIm,1),zooming) ~=0
    nrow = floor(size(testIm,1)/zooming)*zooming;
    testIm = testIm(1:nrow,:,:);
end
if rem(size(testIm,2),zooming) ~=0
    ncol = floor(size(testIm,2)/zooming)*zooming;
    testIm = testIm(:,1:ncol,:);
end
imwrite(testIm, 'Data/Test/high.bmp', 'BMP');
lowIm = imresize(testIm,1/zooming, 'bicubic');
imwrite(lowIm,'Data/Test/low.bmp','BMP');
```

شکل 7 قسمتی از کد Test

دیکشنری را lead میکنیم. سطر 16

```
% load training dictionaries
load('Data/Dictionary/Dictionary.mat');
```

وقتی دیکشینری Load می شود، dl و dh توی workspace قرار می گیرند. سطر 16

شکل 8 workspace بعد از لود کردن دیکشنری دیتاست حاصل از آموزش

مسیر فایل Test رو در متغیر fname قرار می دهیم. سطر 19

```
% test image
fname = 'Data/Test/SheykhSafi.bmp';
```

تصویر تست را باز میکنم و درون متغیر testIm ذخیره میکنم.

تصویر را با نام high.bmp در مسیر مورد نظر ذخیره میکنم.

```
testIm = imread(fname);
if rem(size(testIm,1),zooming) ~=0
    nrow = floor(size(testIm,1)/zooming)*zooming;
    testIm = testIm(1:nrow,:,:);
end
if rem(size(testIm,2),zooming) ~=0
    ncol = floor(size(testIm,2)/zooming)*zooming;
    testIm = testIm(:,1:ncol,:);
end
imwrite(testIm, 'Data/Test/high.bmp', 'BMP');
```

تصویر اورجینال را که در متغیر testIm ذخیره کرده ایم را به اندازه یک سوم کوچک تر می کنم و در متغیر lowIm ذخیره میکنم. تصویر کوچک شده را در مسیر مورد نظر با نام low.bmp ذخیره میکنیم.

```
lowIm = imresize(testIm,1/zooming, 'bicubic');
imwrite(lowIm,'Data/Test/low.bmp','BMP');
interpIm = imresize(lowIm,zooming,'bicubic');
imwrite(uint8(interpIm),'Data/Test/bb.bmp','BMP');
```

دوباره با استفاده از bicubic تصویر کوچک شده را به اندازه ای که در zooming تعیین کرده ایم کیفیت تصویر را افزایش میدهم. و در متغیر interpIm ذخیره کرده و این فایل را هم با نام bb.bmp در مسیر مورد نظر ذخیره می کنیم. این فانکشن از کیفیت اصلی تصویر ما اطلاعاتی ندارد. یک تصویر کم کیفیت دریافت می کند و یک تصویر با کیفیت تحویل ما می دهد.

```
lowIm = imresize(testIm,1/zooming, 'bicubic');
imwrite(lowIm,'Data/Test/low.bmp','BMP');
interpIm = imresize(lowIm,zooming,'bicubic');
imwrite(uint8(interpIm),'Data/Test/bb.bmp','BMP');
```

فقط برای ادامه کار باید دامنه تصویر را عوض کنیم چون Solver و Sparse coding از این دامنه استفاده می کنند. ما هم مجبور هستیم از آن استفاده کنیم. چون تصویر رنگی می شود از یک کانال رنگ استفاده می کنیم.

```
% work with illuminance domain
lowIm2 = rgb2ycbcr(lowIm);
lImy = double(lowIm2(:,:,1));

% bicubic interpolation for the other two channels
interpIm2 = rgb2ycbcr(interpIm);
hImcb = interpIm2(:,:,2);
hImcr = interpIm2(:,:,3);
```

فانکشن super_resolution را فراخوانی می کنیم.

فانکشن super_resolution

super_resolution.m

```
function [hIm] = super_resolution(lIm, zooming, patch_size, overlap, dh,

[lhg, lwd] = size(lIm);
hhg = lhg*zooming;
hwd = lwd*zooming;

mIm = imresize(lIm, 2,'bicubic');
hpatch_size = patch_size*zooming;
mpatch_size = patch_size*2;

% extract gradient feature from lIm
hf1 = [-1,0,1];
vf1 = [-1,0,1]';
hf2 = [1,0,-2,0,1];
vf2 = [1,0,-2,0,1]';

lImG11 = conv2(mIm,hf1,'same');
lImG12 = conv2(mIm,vf1,'same');
lImG21 = conv2(mIm,hf2,'same');
lImG22 = conv2(mIm,vf2,'same');

lImfea(:,:,1) = lImG11;
lImfea(:,:,2) = lImG12;
lImfea(:,:,3) = lImG21;
lImfea(:,:,4) = lImG22;

lgridx = 2:patch_size-overlap:lwd-patch_size;
lgridx = [lgridx, lwd-patch_size];
lgridy = 2:patch_size-overlap:lhg-patch_size;
lgridy = [lgridy, lhg-patch_size];

mgridx = (lgridx - 1)*2 + 1;
```

شکل 9 قسمتی از کد super_resolution

قسمت هایی از فانکشن سوپر رزولووشن را در تصویر مشاهده میکنید.

پارامترها را دریافت میکنیم و یک خروجی به نام hIm خواهیم داشت.

با نسبت zooming می خواهیم ببینیم اندازه تصویر نهایی چقدر خواهد شد. سایز سطر و ستون را ضرب در zooming که سه هست میکنیم و در متغیر های مربوطه ذخیره می کنیم. تصویر نهایی باید به اندازه این دو متغیر hhg وhwd ضرب در هم باشد.

```
[lhg, lwd] = size(lIm);
hhg = lhg*zooming;
hwd = lwd*zooming;
```

یک بار تصویر را دو برابر می کنیم به روش bicubic در متغیر mIm ذخیره می کنیم. این عدد دو ثابت هست یعنی اگر نسبت zooming ما شش هم باشد باز این عدد باید دو باشد، با توجه به الگوریتم.

```
mIm = imresize(lIm, 2,'bicubic');
```

Patch سایز را استخراج می کنیم.

```
hpatch_size = patch_size*zooming;
mpatch_size = patch_size*2;
```

گرادیان های مرتبه سوم و چهارم را استخراج می کنیم.

```
% extract gradient feature from lIm
hf1 = [-1,0,1];
vf1 = [-1,0,1]';
hf2 = [1,0,-2,0,1];
vf2 = [1,0,-2,0,1]';
```

بعد کانولوشن ها را استخراج می کنیم، همان کاری که در train انجام داده بودیم. lImG11 و lImG12کانولوشن مرتبه اول هستند و lImG21 و lImG22 کانولوشن مرتبه دوم هستند.

```
lImG11 = conv2(mIm,hf1,'same');
lImG12 = conv2(mIm,vf1,'same');
lImG21 = conv2(mIm,hf2,'same');
lImG22 = conv2(mIm,vf2,'same');
```

بعد از اینکه patch_size را استخراج کردیم مشگریدی که ایجاد کرده بودیم را مقدار دهی میکنیم.

```
lImfea(:,:,1) = lImG11;
lImfea(:,:,2) = lImG12;
lImfea(:,:,3) = lImG21;
lImfea(:,:,4) = lImG22;

lgridx = 2:patch_size-overlap:lwd-patch_size;
lgridx = [lgridx, lwd-patch_size];
lgridy = 2:patch_size-overlap:lhg-patch_size;
lgridy = [lgridy, lhg-patch_size];

mgridx = (lgridx - 1)*2 + 1;
mgridy = (lgridy - 1)*2 + 1;
```

بعد یک راه حل بر اساس bicubic ایجاد می کنیم. و بعد تغیرات را بر روی آن انجام می دهیم.

```
% find sparse solution
bhIm = imresize(lIm, 3, 'bicubic');
hIm = zeros([hhg, hwd]);
nrml_mat = zeros([hhg, hwd]);
```

درون یک حلقه for سطر و ستون را برمیداریم

به ازای هر 100 گام یک نقطه اضافه می کند.

```
% loop to recover each patch
for xx = 1:length(mgridx)
    for yy = 1:length(mgridy)

        mcolx = mgridx(xx);
        mrowy = mgridy(yy);

        count = count + 1;
        if ~mod(count, 100)
            fprintf('.\n');
        else
            fprintf('.');
        end
```

Patch را استخراج می کنیم و با استفاده از کانولوشن patch اصلاح شده را استخراج می کنیم.

```
        mpatch = mIm(mrowy:mrowy+mpatch_size-1, mcolx:mcolx+mpatch_size-1);
        mmean = mean(mpatch(:));

        mpatchfea = lImfea(mrowy:mrowy+mpatch_size-1, mcolx:mcolx+mpatch_size-1, :);
        mpatchfea = mpatchfea(:);
```

خطای patch را استخراج می کنیم در متغیر mnorm قرار می دهیم.

```
mnorm = sqrt(sum(mpatchfea.^2));
```

بعد مقادیر را به فانکشن SolveLasso ارسال می کنیم که توی پوشه Solver هست که همینطور که در اشاره کرده ام از پروژه های دیگر کپی کرده ام. که برای ما ضریب w را استخراج می کند.

```
w = SolveLasso(dl, y, size(dl, 2), 'nnlasso', [], lambda);
```

که dh را در w و mnorm ضرب می کنیم و نتیجه را در متغیر hpatch ذخیره می کنیم. H منظور در اینجه پیشوند high-quality استفاده کرده ام.

```
hpatch = dh*w*mnorm;
```

در ادامه hpatch را reshape می کنم.

```
hpatch = reshape(hpatch, [hpatch_size, hpatch_size]);
```

hpatch را با مقدار قبلی جمع میکنم ودر تصویر آپدیت می کنم.

```
    hpatch = hpatch + mmean;
```

و در hIm ذخیره میکنم که منظور مخفف تصویر با کیفیت بالا است ذخیره می کنم.

این پروسه رو برای تمام patch ها انجام میدهم. با استفاده حلقه for در سطر 47 درون فانکشن super_resolution

در نهایت تصویر نهایی بدست می آید و تصویر نهایی را نورمالایز می کنیم. و تبدیل می کنیم به uint8 و به خروجی ارسال می کنیم.

این کل کارکرد super_resolution بود. که باز اگر برگردیم به فانکشن Test سطر 47 افزایش کانال اول را به ما میدهد.

```
[hImy] = super_resolution(lImy, zooming, patch_size, overlap, dh, dl, lambda);
```

کانال دوم و سوم هم که به روش bicubic محاسبه کرده بودیم در سطرهای 39 الی 42 فانکشن Test

```
% bicubic interpolation for the other two channels
interpIm2 = rgb2ycbcr(interpIm);
hImcb = interpIm2(:,:,2);
hImcr = interpIm2(:,:,3);
```

در سطر های 49 و 50 از آن دو متغیر hImcb و hImcr برای مقدار دهی کانال دوم و سوم استفاده می کنم.

سطر 48 که کانال اول ما را مقدار دهی می کند را در super_resolution محاسبه کرده بودیم که با استفاده از uint8 در کانال اول مقدار دهی می کنیم.

```
resIm(:,:,1) = uint8(hImy);
resIm(:,:,2) = hImcb;
resIm(:,:,3) = hImcr;
```

در سطر 51 برعکس پروسه illuminance که در سطر 36 انجام داده بودیم برعکسش عمل می کنیم و به RGB تبدیل میکنیم.

```
resIm = ycbcr2rgb(resIm);
```

بعد تصویری که با روش super_Resolution محاسبه کرده ام را نمایش می دهم و عنوانش را هم Supper-Resolution می گذارم. فایل حاصل را در پوشه Test با نام SR ذخیره می کنم.

```
figure,imshow(resIm,[]);
title('Super-Resolution');
imwrite(uint8(resIm),'Data/Test/SR.bmp','BMP');
```

در ادامه یک بار دیگر تصویر lowIm تصویر کیفیت پایین را به اندازه zooming با روش nearest افزایش کیفیت می دهم. که جزو خانواده bicubic هست. هر د تصویر ورودی و افزایش کیفیت داده شده با استفاده از روش bicubic را برای مقایسه به خروجی برای نمایش در صفحه نمایش ارسال می کنم.

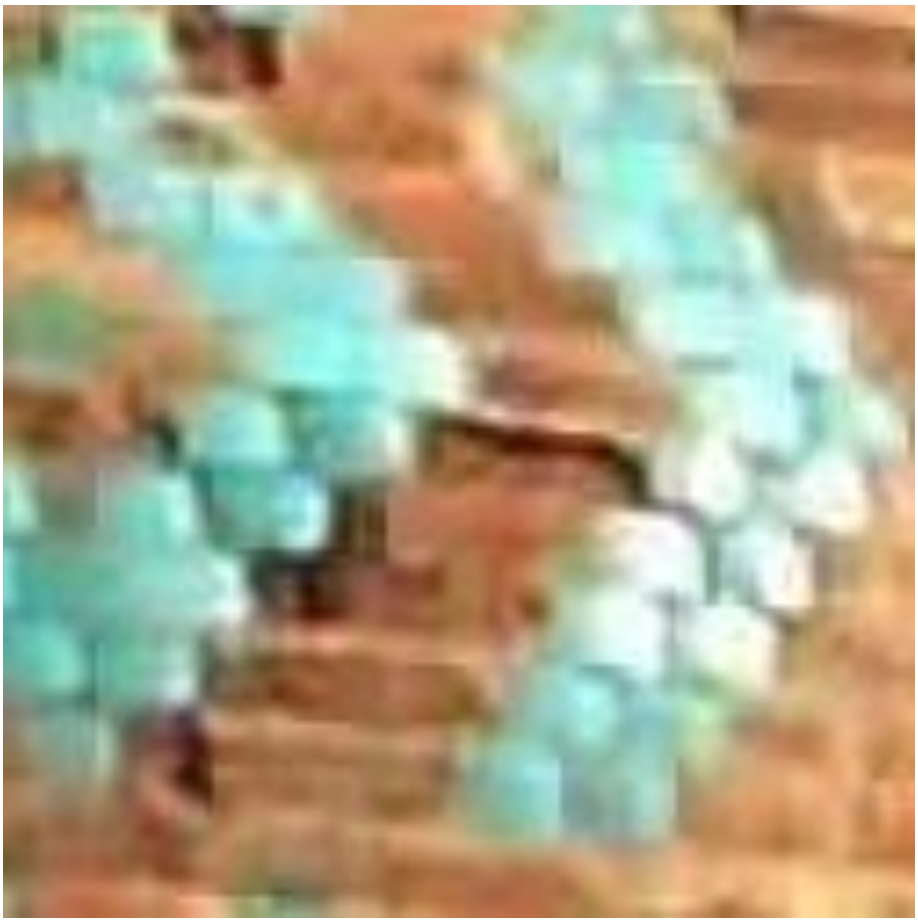
```
nnIm = imresize(lowIm, zooming, 'nearest');
figure, imshow(nnIm);
title('Input image');
pause(1);

figure, imshow(interpIm);
title('Bicubic Interpolation');
pause(1)
```

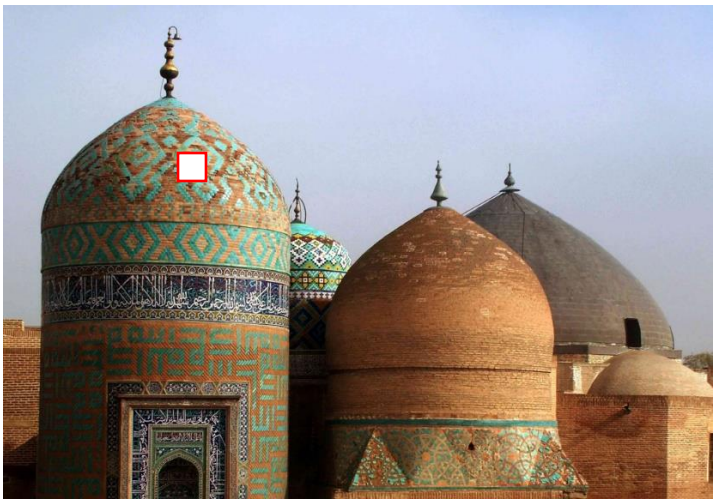
نتیجه گیری.

برای تست یک تکه از تصویر بقعه شیخ صفی الدین اردبیلی را به برنامه دادم و خروجی های زیر را دریافت کردم.

شکل 10 تصویر اورجینال با سایز اصلی

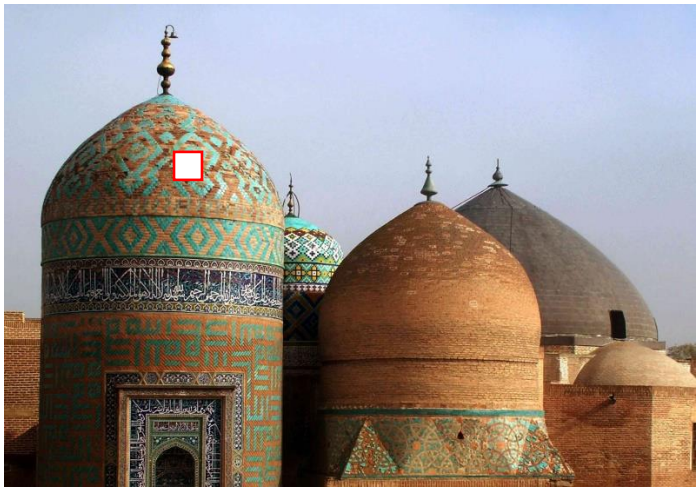


شکل 11 تصویر ورودی، تکه ای به اندازه 126 پیکسل در 126 پیکسل از بقعه شیخ صفی الدین اردبیلی

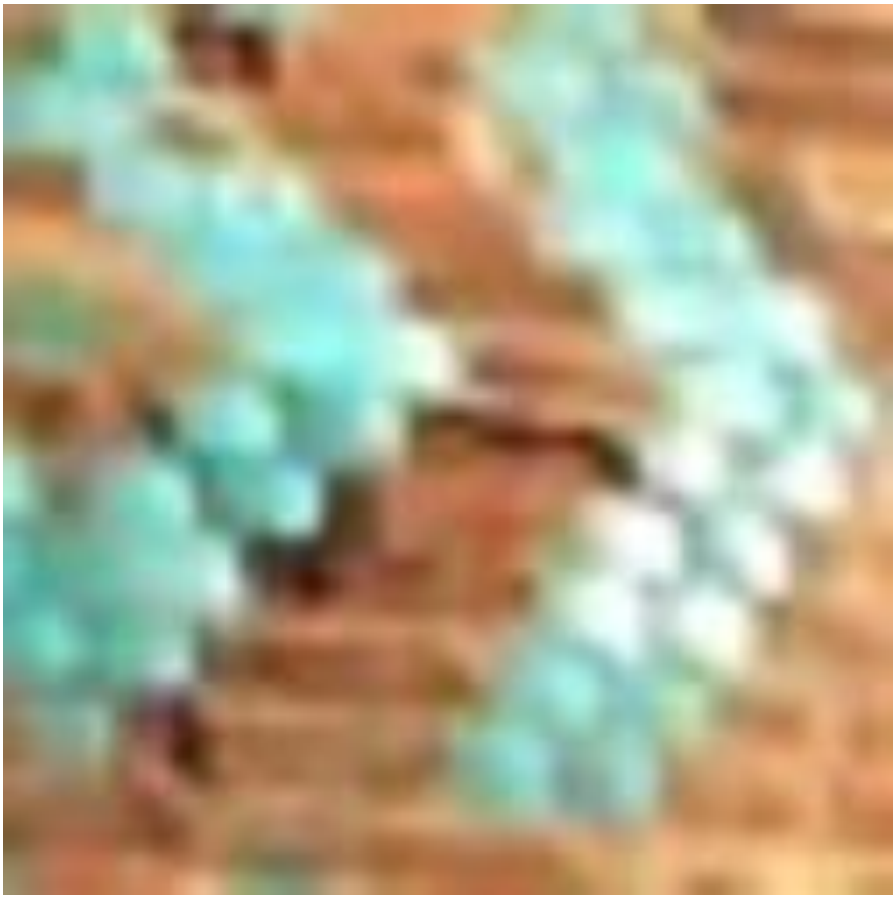


شکل 12 تصویر low سایز شده با سایز اصلی

شکل 13 تصویر Low رزولوشن زوم شده برای بهتر دیده شدن

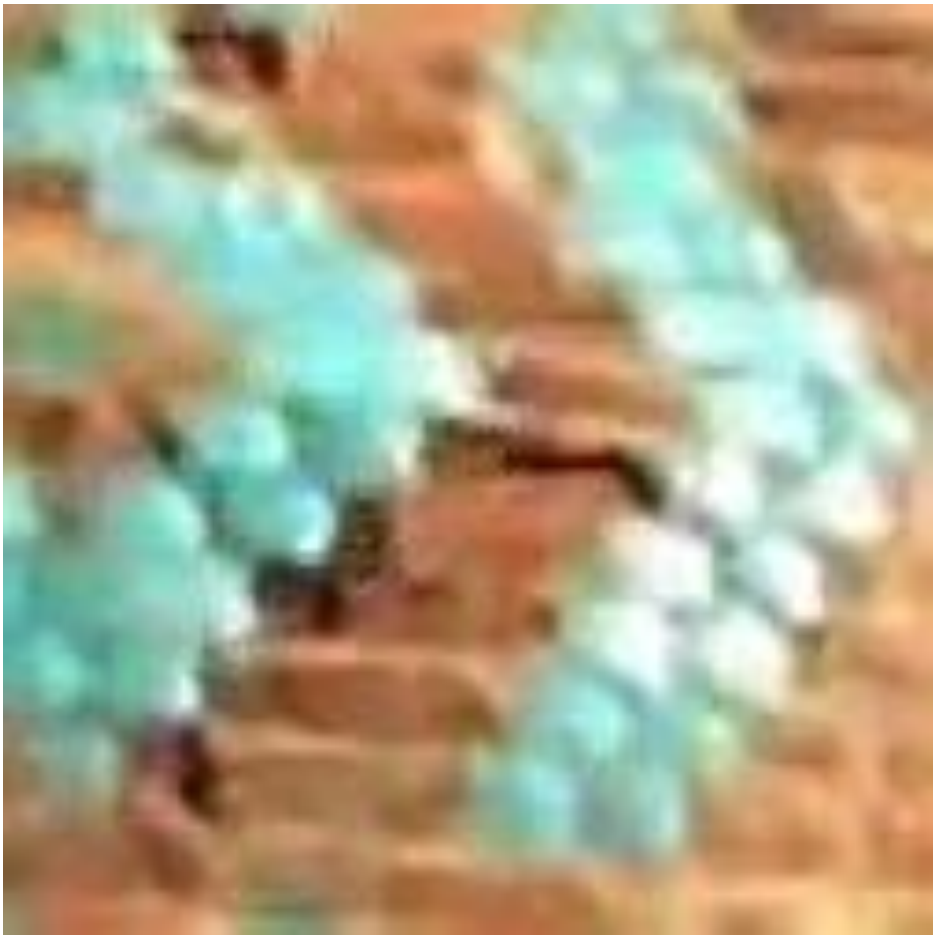


شکل 14 تصویر افزایش یافته رزولوشن با روش bicubic با سایز اصلی



شکل 15 تصویر حاصل از روش bicubic زوم شده برای مشاهده بهتر

شکل 16 تصویر افزایش یافته رزولوشن به رروش SR با سایز اصلی



شکل 17 تصویر زوم شده حاصل از روش Super-Resolution برای مشاهده بهتر

در مقایسه 3 تصویر، تصویر شماره 10 تصویر اورجینال هست برنامه به عنوان تصویر ورودی دریافت می کند. تصویر شماره 12 تصویر کوچک شده با ضریب یک بر اندازه zooming هست که در اینجا zooming برابر 3 مقدار دهی کرده ایم که میشود اندازه یک سوم.

تصویر شماره 14 تصویر حاصل با اعمال روش bicubic هست که در مقایسه با تصویر اورجینال نرم تر شده و لبه ها محو تر شده اند در حقیقت تصویر مات تر به نظر می رسد و مکعب های تشکیل تصویر مات تر شده اند، انگار فوکوس عکس را تغییر داده ایم و جابجا کرده ایم.

تصویر 16 که نتیجه حاصل از این تحقیق و مقاله هست، در تصویر شماره 17 که تصویر حاصل از اعمال SR بر روی تصویر اورجینال را زوم کرده ایم تا بتوانیم نتایج را بهتر مقایسه بکنیم، مشاهده میشود که زرولوشن تصویر افزایش پیدا کرده، در این تصویر هم مشاهده می شود که تصویر صاف تر شده ولی لبه ها آشکارتر از روش bicubic هست و اینطور به نظر می رسد که فوکوس تصویر بهتر تنظیم شده است و تصویر خیلی مات به نظر نمیرسد.