Single Image Super-Resolution using Patch Recurrences

A Project Report

submitted by

AKHIL C

in partial fulfilment of the requirements

for the award of the degree of

MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

MAY 2014

THESIS CERTIFICATE

This is to certify that the thesis titled **Single Image Super-Resolution using Patch Recurrences**, submitted by **Akhil C**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Aravind R
Project Guide
Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 5th May 2014

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere gratitude to my guide Dr. Aravind R. I am deeply grateful for his continuous support during my project work, for his motivation, patience and encouragement. His guidance has helped me in completing my work, writing this thesis and also placement preparations. I could not have imagined having a better guide and mentor for my work.

Besides my guide, I would like to thank all the faculties in the Department of Electrical Engineering for teaching me with such patience which has helped me during my work or in my studies. I would like to thank the Department itself for providing such good facilities which were crucial in my work.

I would like to thank my friends and labmates Dileep, Vaisak, Sriram, Vinod, Praneeth, Srinivas and Abhijith for their encouragement and for giving me good company throughout my stay in IIT.

Finally I would like to thank my family. I am indebted to them forever for their never ending support and for giving me the freedom to make the decisions in my life.

ABSTRACT

High resolution always have demand in many areas like medical imaging, surveillance, satellite imaging, military applications etc.. But often we are supplied with images of low resolution and quality. Class of techniques which reconstructs a high resolution (HR) from one or many low resolution images are called super-resolution (SR). An algorithm for single image super-resolution is presented in this thesis. It does not need any additional information like database or multiple images which are essential for some of the SR methods. The algorithm presented uses internal statistics (such as patch recurrence) of image for effectively estimating the HR image. It is also statistically shown in this thesis that there are sufficient amount redundancy within image for this method to work. An iterative method is also discussed which starts with an initial estimate of intended HR image and converges to the actual HR image. The combination of both these methods was also implemented and it is found to be giving good results.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

SR	Super-Resolution
HR	High resolution
LR	Low Resolution
SSD	Sum of Squared Differences
BSD	Berkeley Segmentation Dataset
TV	Total Variation
BTV	Bilateral Total Variation

NOTATION

α	Magnification Factor
I_0, L	Input Image
$I_{-1},$	Input Image Downsampled by α
I_1, H	Desired Output Image
B	Blur Kernel
s	Downsampling Factor
$\mathcal{L}$	LR Patch
$\mathcal{H}$	HR Patch
(m, n)	LR Patch Location
(M, N)	HR Patch Location
C	Blurring Matrix
D	Decimating Matrix
A	Downsampling Matrix
$F(x)$	Stabilizing Function
∇	Gradient Operator
x_i	Estimate at i^{th} iteration

CHAPTER 1

Introduction

Images of higher resolution are always preferred in many areas like computer vision applications, medical imaging, surveillance, satellite imaging and so on. But they are not always available due to several reasons such as hardware limitations of the image acquisition device or due to higher cost or storage space requirement. But increasing demand of high resolution image in different areas motivated researches in improving resolution. Super-resolution refers to the class of techniques which improve the resolution and quality of a low resolution image.

Image interpolation techniques can also magnify an image and give the same amount of pixels that super-resolution can give. But the distinction among them is quite clear. Image interpolation finds the unknown pixel values by fitting the known pixels to a polynomial. Bilinear and bicubic interpolations are examples to this and they are widely used since they are computationally cheap. But the downside to this is that image interpolated are always smooth i.e., they do not have the high frequency details. Super-resolution on the other hand makes use of additional information such as internal and external statistics of image, image prior etc.. Many methods under super-resolution processes more than one image and is computationally complex. But these methods reconstruct the high resolution image quite accurately.

There are several methods for super-resolution. They can be classified into Classical multi-image super-resolution, Example based SR and Regularization based SR. Classical method calculates the unknown pixels using the knowledge from multiple images. All the images must have a subpixel shift between them for this method to work. Drawback of this technique is that one may not always have such images to work with. The second one, example based method form a collection of low and high resolution (LR-HR) patch pairs using a database. And by learning the relation of these pairs it can estimate the HR output of a new LR image. The need of database adds more issues such as more storage and computational requirements. Regularization based methods take super-resolution as an ill-posed problem and try to minimize some error function.

This report presents a complete algorithm for reconstructing the high resolution parent image of a given LR image. The algorithm is capable of super-resolving the image without any additional images. It makes use of informations which are obtained from within the given image. Hence this is a true single image super-resolution technique. It requires relatively lesser computations and storage when compared to example based (which uses large databases). Since no additional external data is needed, this algorithm can be useful to even people who doesn't know the technical aspects of it. Anyone can just input an LR image and get the super-resolved output.

1.1 Organization of thesis

Chapter. 2 explains about the redundant informations available within the image. Specifically patch recurrence is the internal redundancy that will be studied in that chapter. It also explains how we can extract that information so that it can be utilized for super-resolution technique. An algorithm for calculating the redundancy in image and the results obtained for two different databases are also included there.

Chapter. 3 is a continuation of the previous one. It explains how the extracted information can help us to reconstruct the high resolution image. A complete algorithm for super-resolution is presented there. Results obtained for four different images using this algorithm are also included. Algorithm was tested with magnification factors of 2 and 4.

Chapter. 4 deals with a different approach to the problem. It discusses how an iterative method can be implemented to achieve super-resolution. It starts with viewing super-resolution as an ill-posed problem. And shows that the problem can be solved iteratively. The method starts with an initial approximation and converges to the desired HR image. Finally output of chapter. 3 can be used in this method and get good results.

Chapter. 5 discusses the future works and conclusions.

CHAPTER 2

Patch Redundancy in Images

Many image enhancement problems like super-resolution, inpainting etc. are under constrained and ill posed. Such problems often rely on statistical prior learned from a database of images. But many images contain substantial amount of information within itself. An example of such internal redundant data is recurrence of small image patches (eg. 5×5 patches). These internal statistics, if available and exploited, can solve the aforementioned problems without needing an external database Zontak and Irani (2011). This chapter will discuss about patch recurrences of images and results obtained for two different databases.

2.1 Patch Recurrence Test

In this thesis a patch is defined as a small section of an image of 5×5 size. Hence from here on all the reference to patch means it is of dimensions 5×5. A patch can repeat in an image in two ways, either in the same scale or in a downscaled version of the original image. Either way to say that a patch has repeated it must recur 'as is' (without downscaling the patch) in another location. For scaling down the image a magnification factor α = 2 is used. A patch may only contain an edge or corner since they are very small. But results show that such patches are found multiple times in different scales of the same image.

For testing similarity of two patches Gaussian-weighted SSD (Sum of Squared Differences) is used as a distance measure. This is simply difference of the two patches multiplied by a Gaussian window which is then squared and summed. By using a Gaussian window the local structure of the texture is preserved, since it will give more weight to the centre pixels. And variance of this window is set as 1 for this test. High variance patches will give much larger SSD errors than low variance or smooth patches when compared to very similar looking patches. This is because even slight misalignments will be noticeable for high variance textured patches. So each patch requires a specific

threshold for testing similarity. For textured patches this threshold value will be higher and for smooth ones it will be lower.

The threshold is calculated as follows. For each patch Gaussian weighted SSD is measured with a slightly shifted copy of itself (0.5 pixel shift in x and y directions). So when a particular patch is compared against another one, it is considered similar if the SSD error is below it's specific threshold value. We have conducted patch recurrence test on two different sets of images. The algorithm of this test is given in the next section and the results for the two databases are included in the following sections.

2.2 Algorithm for Patch Recurrence Test

The algorithm given below explains how the patches are counted for a given image.

- If the given image is colour then convert it to YUV format and use intensity channel(Y) for the test. Let's call this image as I_0.
- Remove the DC offset since it doesn't contribute anything to variations in patches.
- Take patches one by one and find their 0.5 pixel shifted versions.
- Find the Gaussian weighted SSD between patches and their shifted versions.
- Store these obtained values as thresholds.
- Compare all the patches against each other in the same image I_0 using Gaussian weighted SSD. This is for testing in-scale patch recurrence. Increment count whenever SSD is found to be lower than the threshold calculated in the previous step.
- Down scale I_0 by α = 2 to obtain I_{-1} (MATLAB inbuilt function *imresize* is used here with default interpolation method). Compare all the patches in I_0 against patches in I_{-1}. These counts will quantify the cross-scale patch recurrences.
- The image can be further down scaled to obtain I_{-2}, I_{-3} etc. And again the procedure can be repeated.

2.3 Results

Patch recurrence test was conducted over two databases containing 20 images each. The result obtained are explained in the following two sections.

2.3.1 Dataset - 1 Results

First dataset used for testing consists of 20 images. They are tiger stripes images of size 255×255. Fig. 2.1 shows two sample images of size 255×255 taken from the dataset.

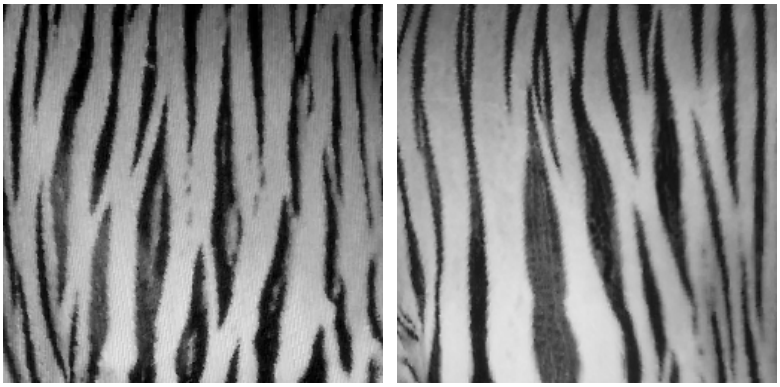


Figure 2.1: Two sample images from the database 1

All 20 images were tested to find patch recurrence in same scale and 3 downscaled versions with downscaling factors α = 2, α = 4 and α = 8. So patches in original image I_0 is compared against patches from I_{-1} (of size 127×127), I_{-2} (of size 63×63) and I_{-3} (of size 31×31). Total number of patches in I_0 is 63001 (including the overlap) and the final image I_{-3} has only 729 patches. So we can't expect much redundancy in those highly downscaled versions. The results obtained can be easily inferred from Fig. 2.2.

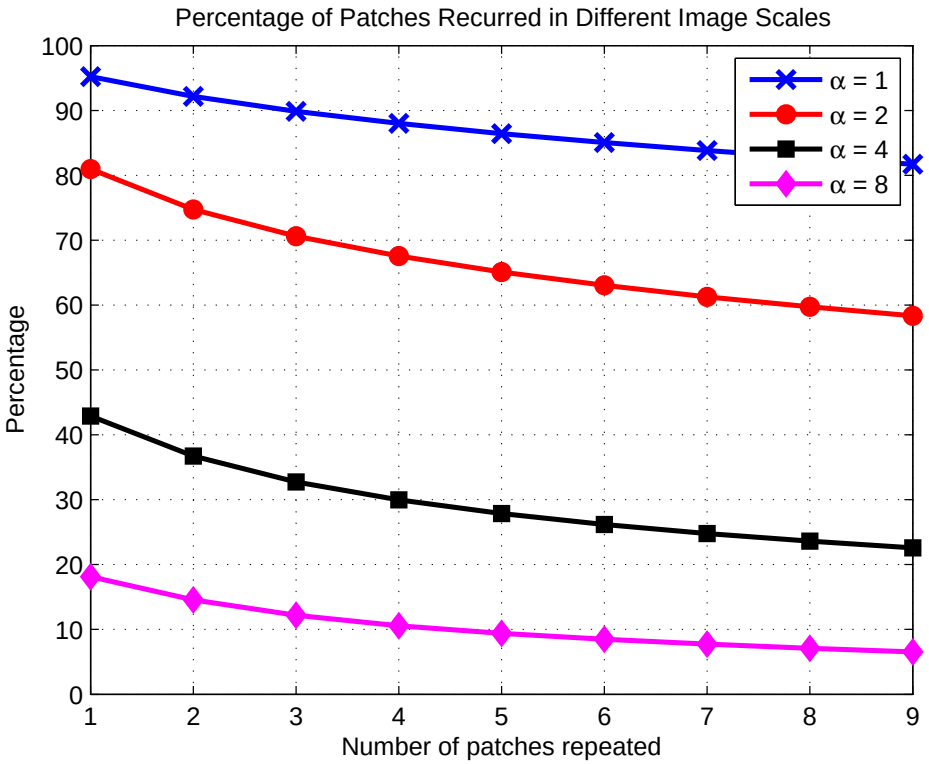


Figure 2.2: Results for first dataset - 1

Four curves represent different scales of I_0. x axis implies the number of patches that has repeated. For example, Third red circle in second curve from top (for α = 2) says that 70 % of the patches in I_0 has repeated in I_{-1} 3 more times. In another words 3 or more matches have been found in I_{-1} for 70% of the patches in I_0.

2.3.2 Dataset -2 Results

This dataset is also consists of 20 images taken from Berkeley Segmentation Dataset (BSD300). All the images are of size 481×321 or 321×481 and are natural images unlike the first dataset where it was cropped portion of tiger patches. So these images does not show any obvious patch redundancy. Fig. 2.3 shows two sample images from the dataset.

Figure 2.3: Two sample images from the database 2

For this dataset also patches from original image I_0 (α = 1 i.e., same scale) of size 481×321, were tested against patches in I_{-1} (α = 2) of size 240×160 , I_{-2} (α = 4) of size 120×80 and I_{-3} (α = 8) of size 60×40. The results obtained are given in Fig. 2.4

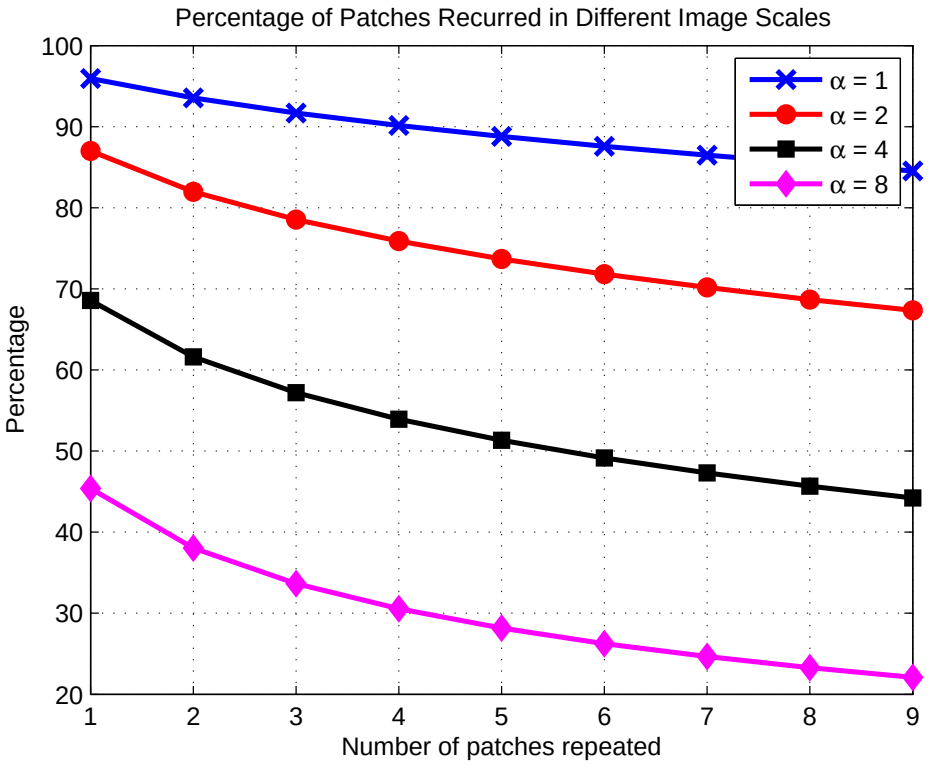


Figure 2.4: Results for first dataset - 2

The results are quite similar to what was obtained with dataset - 1 for α = 1 and α = 2. But for α = 4 and α = 8 the percentage of patches repeated are higher for dataset - 2. This may seem confusing considering tiger stripes in dataset - 1 looked to have more recurring patterns. The reason for this is that images in this dataset are larger in dimensions, hence for $\alpha = 8$ I_{-3} is of size 60×40 as opposed to 31×31 in dataset - 1. So 2016 patches are available in this case where as we had only 729 patches in the former case. Although number of patches in original image is also higher here still the ratio of total patches in original to that in downscaled image will be higher for an image with larger dimension. Hence bigger image even when downscaled will have more patches to search against. And also another important point which needs to be re-emphasized is that patches considered are of very small size. Hence the patch redundancy which we intuitively thought there is in dataset - 1, may not be relevant for such small size of patch. All it can contain is a corner or a smooth area.

CHAPTER 3

Single Image Super-Resolution using Patch Recurrence

Example-based super-resolution works by learning the correspondences between low resolution and high resolution image patches. To do so it requires a database of low and high resolution image pairs. After learning the correspondence it can be applied to a new low resolution image Freeman *et al.* (2002). The main issue with conventional example-based super-resolution is that it requires a database. And clearly it needs more memory and computational power to store and process through the database containing h. A true single image super-resolution, which does not need any information other than the given image, hence can work faster. Because it uses only internal statistics with in the image. The advantage of this technique does not end here. It has been shown by Zontak and Irani (2011) that often internal statistics (like patch recurrence within the image) are more powerful than statistics obtained from a database. This may seem counter intuitive, but a patch in an image will almost surely recur in the same image. We saw in previous chapter that patches will repeat at least once in the same image more than 95 % of the time. But the same patch may not occur in another image and it has been shown by Zontak and Irani (2011) that for all the patches in a given image to repeat we may need hundreds of images in the database. And this is again computationally demanding.

3.1 Cross-Scale Patch Recurrence

In the previous chapter two types of patch recurrences were discussed. In-scale (repetitions in the same scale) and cross-scale patch recurrences (repetitions in downscaled versions of image). As already mentioned example-based super-resolution requires a database to work with. The technique implemented in this thesis also work in a similar fashion but only that it forms the database using patches within the images. For the purpose of forming such a database cross-scale patch recurrence is exploited.

The given image I_0 is downscaled to get I_{-1} (Using *imresize* function for a factor of α) and a cross-scale search is done to find similar patches. For every patch found in I_{-1} we can map this location back to I_0 and find it's high resolution parent. This idea is illustrated in Fig. 3.1.

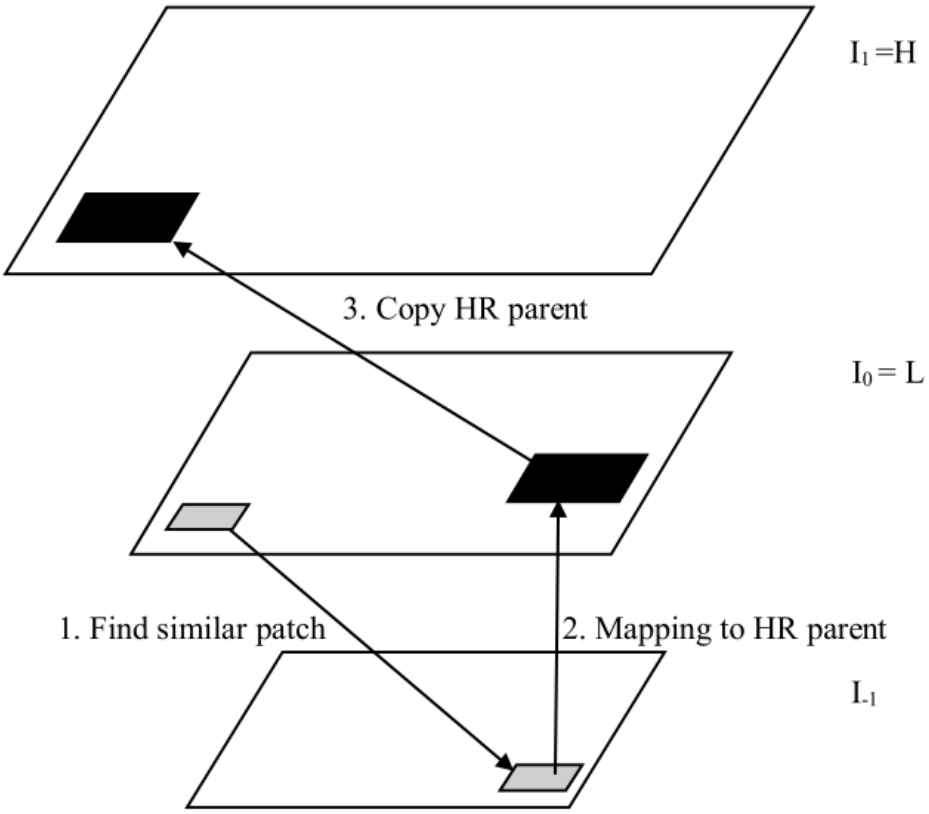


Figure 3.1: Finding LR-HR pair and copying it to appropriate location

3.2 Super-Resolution

The main idea of super-resolution is depicted in Fig. 3.1. The image in the middle is the given image I_0 (L) and the image on the top is the high resolution image I_1 (H) we intend to reconstruct. Grey colour patch in I_0 is a low resolution patch and our aim is to find an HR patch from which it must have originated. This is actually an under constrained problem, because many solution exists for such an HR patch. So additional information or constraints are necessary to make sure that the reconstructed patch is correct one (Yue *et al.* (2013)). For this additional constraint a similar patch is found

in I_{-1} and (grey patch in I_{-1}, i.e. last image in Fig. 3.1). We can map the location of this image back to I_0 to find an HR patch (black patch in I_0). These two patches (grey in I_{-1} and black in I_0) form an LR-HR pair. We can use this relation to find HR parent of the actual patch we started with. Glasner *et al.* (2009) use this technique for super-resolution reconstruction.

We may copy the HR parent that we found to the corresponding location since the grey patches are similar and black patch is it's HR parent. Two important points to be noted here is that several HR parents could create the same LR patch. And also we're working under the assumption that downsampling operation to generate I_{-1} is same as the one with which I_0 is formed from I_1.

$$L = (H * B) \downarrow s \tag{3.1}$$

Eq. 3.1 represents image formation process in very basic form. HR image H is convolved with blur kernel B and then subsampled by a factor s to obtain low resolution image L. We do not know the blur kernel B but it can be approximated by gaussian kernel Glasner *et al.* (2009). In this thesis MATLAB inbuilt *imresize* function is used as approximation and it gives good results with default kernel.

3.3 Algorithm for Single Image Super-Resolution

Let's denote given image, super-resolved output, magnification factor by I_0, I_1 and α respectively. And let I_{-1} be the intermediately generated downscaled image. We'll start by interpolating I_0 to I_1 using MATLAB *imresize* function which by default uses bicubic interpolation. This may be used as a background on which further work is done. If an HR patch is obtained at a particular location it will be placed there. Even if no HR patch is found at some locations, interpolated image will be there in those area. So there's an assurance that even at the worst case i.e., no HR patch is found, we'll still end up with interpolated image.

- Convert I_0 to YUV format if it is a colour image. U and V channels contain only chroma information and a simple bicubic interpolation by factor α is sufficient for

them. Y channel where as contain important greyscale details hence SR technique is applied to this channel. If the image is greyscale then consider it as is.

- Interpolate I_0 by a magnification factor α to get I_1. This will serve as initial SR image. Patches will be replaced in this image wherever they're found.

- Downscale I_0 to obtain I_{-1} and using the method discussed in section 2.2 find similar patches in I_{-1}.

- Store the location of all similar patches that was found in I_{-1} (or store at least 10 patches if available). Location of similar patches in I_{-1} for every patch in I_0 will be available at the end of this step.

- Using the locations obtained in the previous step get the HR parents of LR patches in I_0 by simple mapping.

  For example let's say $\mathcal{L}_0$ is a patch in I_0 at location (m, n). And let $\mathcal{L}_{01}$, $\mathcal{L}_{02}$, ..., $\mathcal{L}_{0n}$ be n patches in I_{-1} which are similar to $\mathcal{L}_0$. And their locations be denoted by (m_1, n_1), (m_2, n_2), ..., (m_n, n_n). Let $\mathcal{H}_{01}$, $\mathcal{H}_{02}$, ..., $\mathcal{H}_{0n}$ be their respective HR parents in I_0 and their location be denoted by (M_1, N_1), (M_2, N_2), ..., (M_n, N_n). (M_i, N_i) are obtained by the simple mapping given in Eq. 3.2 and 3.3.

$$M_i = \alpha * (m_i - 1) + 1 \tag{3.2}$$

$$N_i = \alpha * (n_i - 1) + 1 \tag{3.3}$$

- From the available HR patches find the one which fits the best in I_1. Any of the n HR patches available can be HR parent of $\mathcal{L}_0$. For finding best HR patch see which one is agreeing the most with the neighbouring patches.

  As mentioned earlier patches are overlapping with each other. So the HR patch which we are trying to place might be overlapping with a previously placed patch. But sometimes there may not be any overlapping at all. For example if no matching patches were found for previous locations (4 consecutive LR patch recurrence

test should fail for this to happen) then no patch would have been replaced and we won't find any overlapping area. And also the first patch in image won't have any overlap either. In these cases we are forced to choose HR parent whose corresponding LR patch was most similar to the original patch.

- Measure the euclidean distance between the current and previous patch in this overlapping area. The best HR patch is the one which has least distance with the overlapping area.

- Based on the criteria above mentioned best HR patch can be selected. And this patch has to be placed at (M, N) which are the locations mapped from (m, n) using Eq. 3.2 and 3.3.

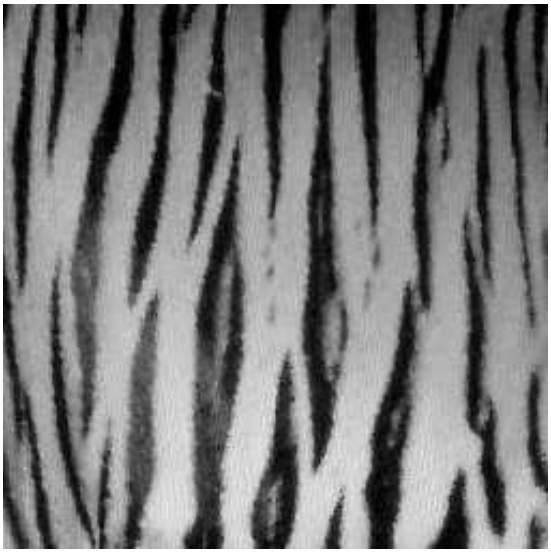
  As we have statistically shown in 2.3 more than 80 % of the patches will be repeated at least once in image downscaled by a factor of 2. So it can be said the same percentage of patches will be replaced in I_1. Results obtained are presented in the next section.

3.4 Results

Algorithm was implemented in MATLAB and tested using images from database - 1 and 2 which was used in 2.3. Tests were conducted for α = 2 and 4. Results are compared against output of bicubic interpolation (using MATLAB *imresize* function).

3.4.1 Inputs

Fig. 3.2 shows 4 images given as input to super-resolution algorithms. They are of different sizes and first one is greyscale image whereas the rest are colour images.



(a) Input-1 (255×255)

(b) Input-2 (481×321)

(c) Input-3 (321×481)

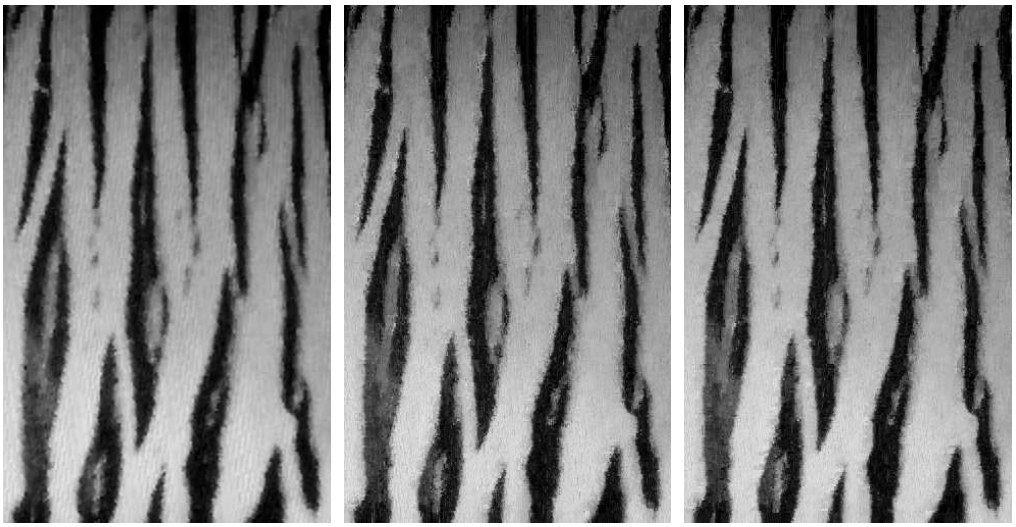
(d) Input-4 (321×481)

Figure 3.2: Test images. Dimensions are mentioned below each figure.

3.4.2 Outputs

For each input 3 outputs are displayed. First one is output of bicubic interpolation (using MATLAB *imresize* function). Second and third images are outputs of super-resolution algorithms. SR test was conducted in two ways, one without checking for best HR patch of all the available ones (It just copies HR parent most similar LR patch). Output of this is the second figure. And the last figure is output of complete algorithm i.e., it looks for the best HR patch.

SR Outputs for α = 2

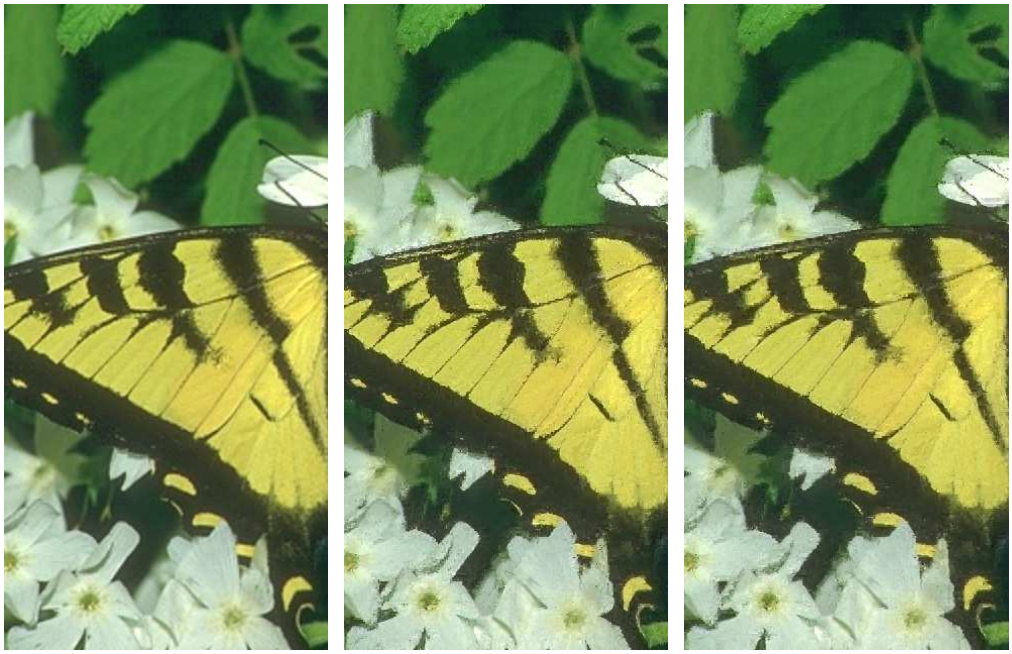


(a) Bicubic Interpolation (b) SR output-1 (c) SR output-2

Figure 3.3: Output for Input-1

(a) Bicubic Interpolation (b) SR output-1 (c) SR output-2

Figure 3.4: Output for Input-2



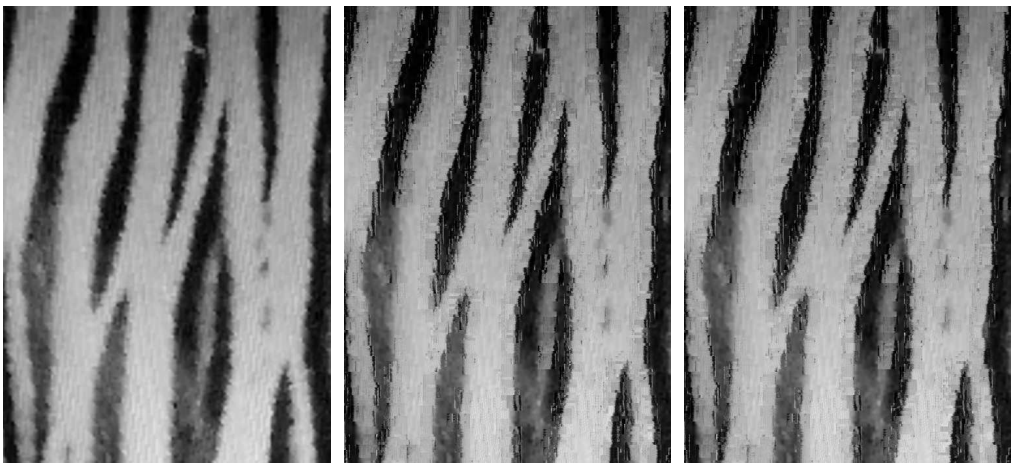
(a) Bicubic Interpolation (b) SR output-1 (c) SR output-2

Figure 3.5: Output for Input-3

(a) Bicubic Interpolation (b) SR output-1 (c) SR output-2

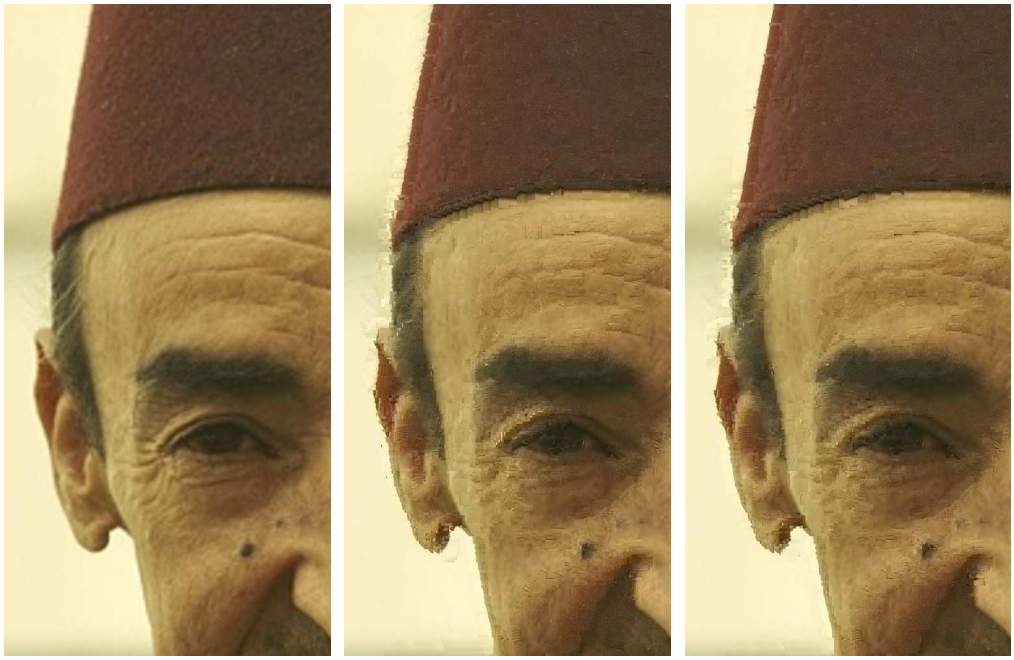
Figure 3.6: Output for Input-4

SR Output for α = 4



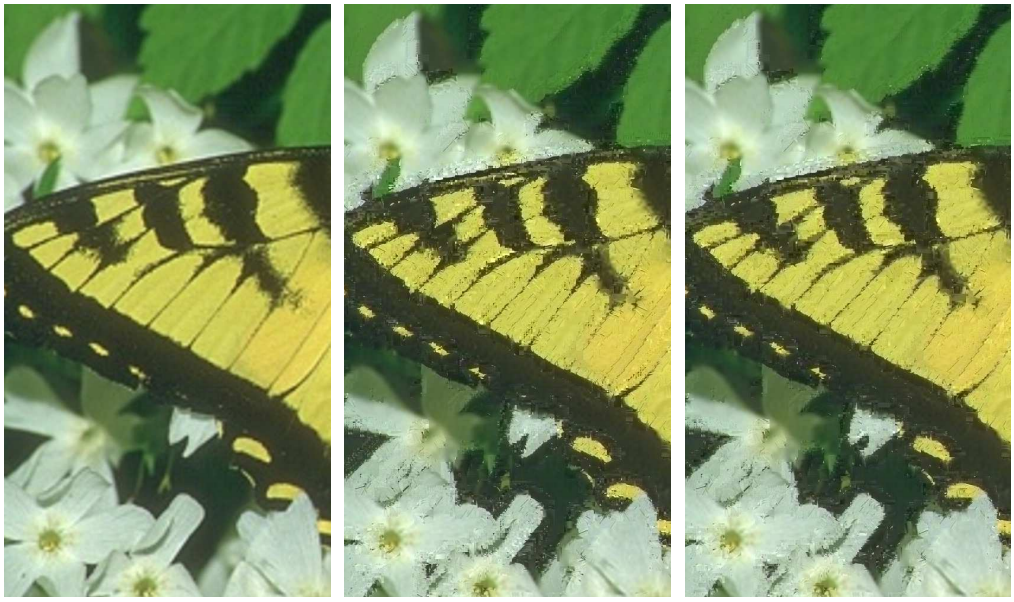
(a) Bicubic Interpolation (b) SR output-1 (c) SR output-2

Figure 3.7: Output for Input-1



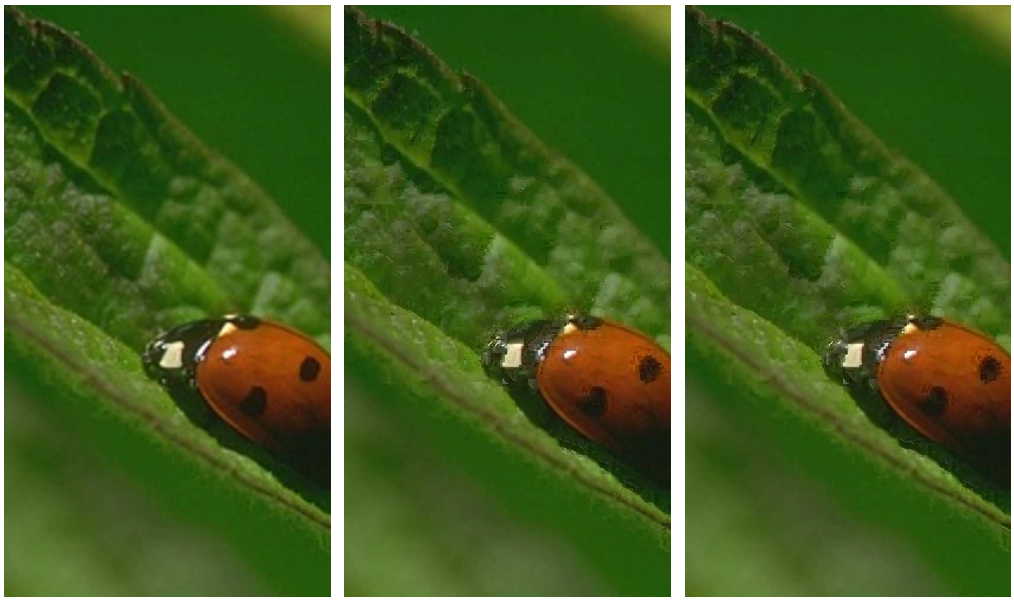
(a) Bicubic Interpolation (b) SR output-1 (c) SR output-2

Figure 3.8: Output for Input-2



(a) Bicubic Interpolation (b) SR output-1 (c) SR output-2

Figure 3.9: Output for Input-3



(a) Bicubic Interpolation (b) SR output-1 (c) SR output-2

Figure 3.10: Output for Input-4

It can be seen that results of SR algorithm is looking sharper than interpolated output for α = 2. There is a considerable amount of distortion when α is increased to 4. This is expected since search space is considerably reduced when magnification factor is increased. Even for α = 2 the available area to search is only $1/4^{th}$ of the input image. For a magnification factor of α available search area is shrunk by α^2. Hence for higher magnifications results may not be satisfactory. Also there is not much difference between

second and third image for α = 4. This is also a consequence of the aforementioned problem. Since area is less probability of finding more than one patch is less. So we may not have lot of HR patches to choose from.

CHAPTER 4

Regularization Method for Super-Resolution

4.1 Regularization

A low resolution image is formed by blurring and downsampling a high resolution image. Furthermore noise may have also been added to it. Aim of a super-resolution technique is to reverse these processes. Mathematically formation of an LR image can be represented as given below.

$$y = DCx + n \tag{4.1}$$

where y is the LR image (of size $m \times n$) arranged column wise as an $N \times 1$ vector ($N = mn$). x is the HR image (of size $\alpha m \times \alpha n$) arranged column wise as a $\alpha^2 N \times 1$ vector. C is the blur matrix of size $\alpha^2 N \times \alpha^2 N$ and D is the downsampling matrix which is of size $N \times \alpha^2 N$. Finally n is noise vector of size $N \times 1$. We may write Eq. 4.1 as,

$$y = Ax + n \tag{4.2}$$

where $A = DC$ is of size $N \times \alpha^2 N$. We have to solve for x from Eq. 4.2. But this equation is ill-posed and a small error in input vector y can cause huge change in approximation of x if Eq. 4.2 is solved directly. Fortunately there exist other methods for solving this. We can get a good estimate of x by taking a regularized minimization approach. The solution for Eq. 4.2 is found as,

$$x = \arg\min_x ||Ax - y||_2^2 + aF(x) \tag{4.3}$$

Here $F(x)$ is a stabilizing term and a > 0 is the coefficient of regularization. Tikhonov function, Total variation(TV) and Bilateral TV (BTV) are all good stabilizer functions.

In this thesis TV function was used.

$$TV(x) = ||\nabla x||_1 \tag{4.4}$$

where ∇ is the gradient operator.

4.2 Iterative Algorithm

To solve Eq. 4.3 with stabilizer defined in Eq. 4.4, iterative steepest descent method is used. The update equation which is given below will converge to desired HR image x.

$$x_{n+1} = x_n - b(A^T sign(Ax - y)) + a||\nabla x||_1 \tag{4.5}$$

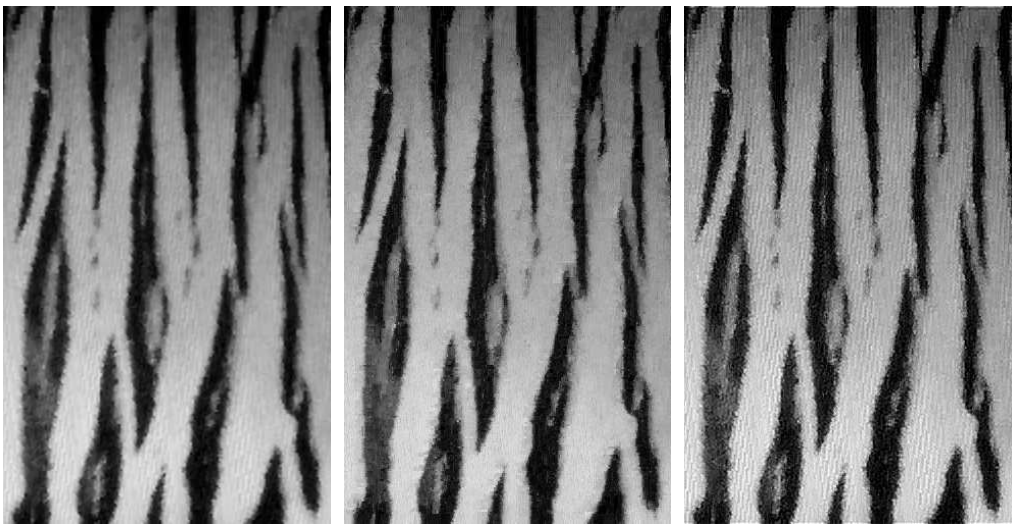
A^T implies upsampling operation followed by filtering. Algorithm for implementing iterative method given in Eq. 4.5 is given below.

- We have to start with an initial guess of x. Output of SR method discussed in Chapter. 3 was taken as initial approximation x_0 in this project.
- Downsample the current approximation of HR image x_i to get y_i. A gaussian kernel can be used as blurring matrix C and then downsample it according to magnification factor α.
- Find the difference between the downsampled image y_i and actual LR image y. This is the $error$ value.
- Take $sign$ of the $error$ and enlarge it to the size of x using same blurring matrix C.
- Add the upscaled error to current approximation x_i, after multiplying with coefficient b.
- Add the regularization term multiplied with coefficient a, to x_i, to obtain new estimate x_{i+1}.

- keep iterating through the above steps until convergence.

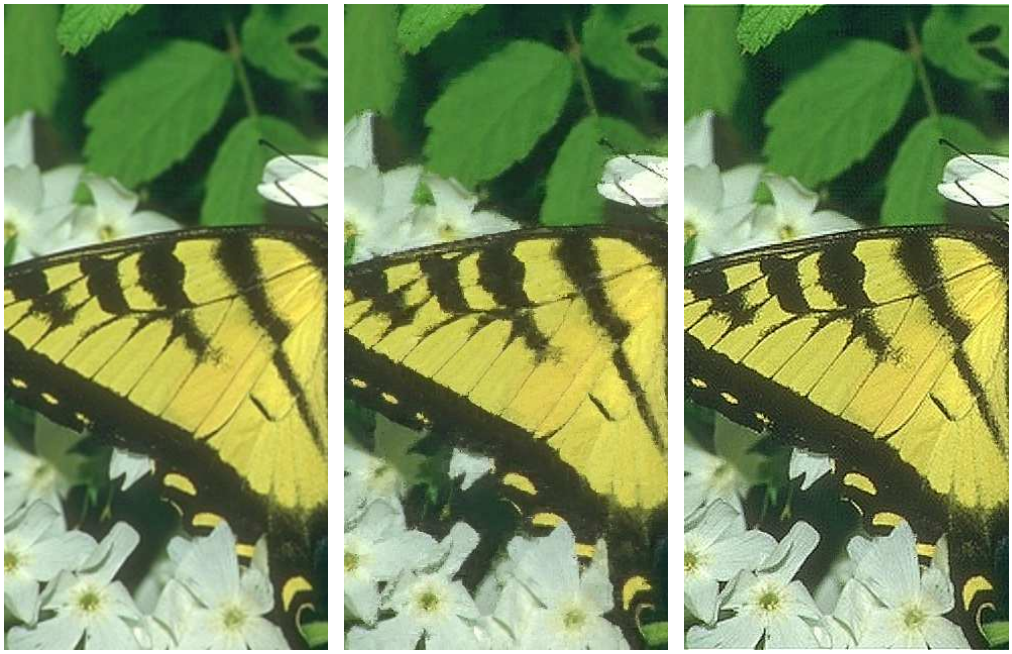
4.3 Results

The above algorithm was implemented in MATLAB and tested against the same images which was used in Chapter. 3 for testing super-resolution algorithm. A magnification factor of 2 was used for testing and the output of algorithm. 3.3 explained in Chapter. 3 was used as initial approximation x_0. Corresponding to each input three outputs are displayed which are bicubic interpolated output, super-resolution using patch recurrence (output of algorithm. 3.3) and output of iterative method presented in this chapter.



(a) Bicubic Interpolation (b) SR by patch recurrence (c) SR by iteration

Figure 4.1: Output for Input-1



(a) Bicubic Interpolation (b) SR by patch recurrence (c) SR by iteration

Figure 4.2: Output for Input-2

(a) Bicubic Interpolation (b) SR by patch recurrence (c) SR by iteration

Figure 4.3: Output for Input-3

(a) Bicubic Interpolation (b) SR by patch recurrence (c) SR by iteration

Figure 4.4: Output for Input-4

The improvement in image details are clearly visible in all the four output images. The images converged reasonably fast to give outputs. The values of coefficient a and b are found by trial error and. The optimum values which are used in this test are $b = 1.5$ and $a = 0.001$.

CHAPTER 5

Conclusions

An algorithm for implementing single image super-resolution was presented. It was based on the internal statistics of an image such as patch recurrences. Tests discussed in chapter. 2 statistically prove that there is sufficient amount of redundancy within an image which if exploited can be used to improve the image quality. It was also noted in the same chapter that for higher value of magnification factor α, patch redundancy diminishes quite drastically. Algorithm presented in chapter. 3 made use of the internal statistics to reconstruct high resolution image. The result obtained is reasonably good for $\alpha = 2$ but for higher values than that distortion was quite high. It can be concluded that this is due to the reduction in search area when higher magnification factor is used. Final chapter discussed regularization method for super-resolution. It used output from previous chapter as an initial approximation. The converged results after iteration were visibly sharper and detailed.

The main drawback of algorithm is relatively higher computation time. For an image of dimensions 481x321 it took approximately 17 minutes to give the output on a desktop computer (core i7 processor running at 3.4ghz). This is a serious issue when it comes to practical usefulness. Especially for real time applications algorithm has to work much faster than this. Bottleneck in the program comes at searching for similar patches section. A simple brute force approach was used in this project to find similar patches. More intelligent approaches for searching can reduce the run-time considerably. Another improvement which can be made is in algorithm. 4.2. Different stabilizer functions can be tried in the iteration section.

APPENDIX A

A SAMPLE APPENDIX

A.1 MATLAB Code for Super-Resolution

```
%% Super Resolution
% akhil cholayil

%% Initialization

close all
clear all
clc

% parallel processing
nCores = 2;
disp('set cores according to your processor in line 11')
disp(['current no of cores = ',num2str(nCores)])
if matlabpool('size') == 0      % checking to see if my pool is already open
    matlabpool('open',nCores);
end

magFactor = 4;
patSize = 5;
% gWindow = fspecial('gaussian',5,1);
gWindow = zeros(patSize,patSize);
gWindow(:) = 1/numel(gWindow);

%% Reading image

InA = imread('35010.jpg');
[rowA,colA,~] = size(InA);
rowA = floor(rowA/magFactor)*magFactor;
colA = floor(colA/magFactor)*magFactor;
```

```
InA = InA(1:rowA,1:colA,:);
colour = false;

if size(InA,3) == 3
    colour = true;
    InAycc = rgb2ycbcr(InA);
    InA = InAycc(:,:,1);
    InAcb = InAycc(:,:,2);
    InAcr = InAycc(:,:,3);
end

HR = imresize(InA,magFactor);
HR = double(HR);
InA = double(InA);
DCoffset = mean(mean(InA));
InA = InA-DCoffset;

%% Calculating patch-specific-threshold

fprintf('calculating patch-specific-threshold...')

Ipad = zeros(rowA+1,colA+1);
Ipad(1:end-1,1:end-1) = InA;
Ishift = (InA+Ipad(1:end-1,2:end)+Ipad(2:end,1:end-1)+Ipad(2:end,2:end))/4;
diffMat = (InA-Ishift);
thresholdMat = zeros((rowA-(patSize-1)),(colA-(patSize-1)));

for j = 1:colA-(patSize-1)
    k = j:j+(patSize-1);
    for i = 1:rowA-(patSize-1)
        thresholdMat(i,j) = sum(sum((gWindow.*diffMat(i:i+(patSize-1),k)).^2));
    end
end

fprintf('done\n');

%% Patch redundancy test in downscaled image

fprintf('running patch redundancy test...')
```

```
simPatches = cell((rowA-(patSize-1)),(colA-(patSize-1)));
InB = imresize(InA,1/magFactor);
[rowB,colB] = size(InB);
rowB = floor(rowB/2)*2;
colB = floor(colB/2)*2;
InB = InB(1:rowB,1:colB,:);
patchLong = zeros(patSize,patSize,(rowB-(patSize-1))*(colB-(patSize-1)));

index = 1;
for j = 1:colB-(patSize-1)
    k = j:j+(patSize-1);
    for i = 1:rowB-(patSize-1)
        patchLong(:,:,index) = gWindow.*InB(i:i+(patSize-1),k);
        index = index+1;
    end
end
simPatNo = 51;
parfor j = 1:colA-(patSize-1)
    k = j:j+(patSize-1);
    for i = 1:rowA-(patSize-1)
        temp1 = bsxfun(@minus,(gWindow.*InA(i:i+(patSize-1),k)),patchLong);
        temp2 = sum(sum(temp1.^2));
        patCount = sum(sum(temp2 <= thresholdMat(i,j)));
        if patCount > 1
            [~,patIndex] = sort(temp2);
            simPatches{i,j} = patIndex(2:min(patCount,simPatNo));
        elseif patCount == 1
            [~,patIndex] = min(temp2);
            simPatches{i,j} = patIndex;
        end
    end
end

fprintf('done\n')

%% Replacing LR patches with HR patches

fprintf('replacing patches...')

InA = InA+DCoffset;
```

```
OlapMask = zeros(size(HR));
k = magFactor*patSize-1;

for i = 1:rowA-(patSize-1)
    l = magFactor*(i-1)+1:magFactor*(i-1+patSize);
    for j = 1:colA-(patSize-1)
        patCount = length(simPatches{i,j});
        colL = magFactor*(j-1)+1;
        if patCount == 1
            [rowH,colH] = ind2sub([rowB-(patSize-1),...
                colB-(patSize-1)],simPatches{i,j});
            rowH = magFactor*(rowH-1)+1;
            colH = magFactor*(colH-1)+1;
            HR(l,colL:colL+k) = InA(rowH:rowH+k,colH:colH+k);
            OlapMask(l,colL:colL+k) = ones;
        elseif patCount > 1
            leastDiff = inf;
            mask = OlapMask(l,colL:colL+k);
            currentPatch = HR(l,colL:colL+k);
            [rowsH,colsH] = ind2sub([rowB-(patSize-1),...
            colB-(patSize-1)],simPatches{i,j});
            rowsH = magFactor*(rowsH-1)+1;
            colsH = magFactor*(colsH-1)+1;
            for m = 2:patCount
                rowH = rowsH(m);
                colH = colsH(m);
                tempHR = InA(rowH:rowH+k,colH:colH+k);
                tempDiff = sum(sum((mask.*(tempHR-currentPatch)).^2));
                if tempDiff < leastDiff
                    leastDiff = tempDiff;
                    bestHR = tempHR;
                end
            end
            HR(l,colL:colL+k) = bestHR;
            OlapMask(l,colL:colL+k) = ones;
        end
    end
end

fprintf('done\n')
```

```
%% Iterating section

fprintf('iteration started...')

Filter = fspecial('gaussian',5,1);
y = InA;
x = HR;     % initial approximation
a = 0.001;
b = 1.5;
noIter = 1000;    % no of iterations
prevErr = inf;

for i = 1:noIter
    var1 = imfilter(x,Filter);
    var2 = downsample(downsample(var1,magFactor)',magFactor)';
    errorIm = var2-y;
    var3 = sign(errorIm);
    var4 = upsample(upsample(var3,magFactor)',magFactor)';
    discrepancy = imfilter(var4,Filter);

    kernel = [-1 1 0];
    diffImageLeft = imfilter(x, kernel);
    kernel = [0 1 -1];
    diffImageRight = imfilter(x, kernel);
    kernel = [-1 1 0]';
    diffImageTop = imfilter(x, kernel);
    kernel = [0 1 -1]';
    diffImageBottom = imfilter(x, kernel);
    stabilizer = sqrt(diffImageLeft.^2 + diffImageRight.^2 + ...
        diffImageTop.^2 + diffImageBottom.^2);

     error = sum(sum(abs(errorIm)));
     if prevErr*1.01 < error
         break
     end
     x = x-b*discrepancy+a*stabilizer;
     prevErr = error;
end
HR = x;
```

```
fprintf('done\n');

%% Output

if colour == true
    InA(:,:,1) = InA;            % input image reassembled
    InA(:,:,2) = InAcb;
    InA(:,:,3) = InAcr;
    InA = ycbcr2rgb(uint8(InA));

    Out(:,:,1) = uint8(HR);
    Out(:,:,2) = imresize(InAcb,magFactor);
    Out(:,:,3) = imresize(InAcr,magFactor);
else
    InA = uint8(InA);
    Out = uint8(HR);
end

Inter = imresize(InA,magFactor);    % interpolated image

imshow(InA)
title('Input Image')

rangex = 160:480;

figure
subplot(1,3,1)
imshow(Inter(:,rangex,:))
title('Interpolated output')

figure
subplot(1,3,2)
imshow(OutwoIter(:,rangex,:))
title('SR simp')

figure
subplot(1,3,3)
imshow(Out(:,rangex,:))
title('SR output')
```

REFERENCES

1. **Elad, M.** and **A. Feuer** (1996). Super-resolution reconstruction of an image. *IEEE Transactions on Image Processing*.

2. **Freeman, W. T.**, **T. R.Jones**, and **Egon.C.Pasztor** (2002). Example-based super-resolution. *Computer Graphics and Applications, IEEE (Volume: 22, Issue: 2)*.

3. **Glasner, D.**, **S. Bagon**, and **M. Irani** (2009). Super-resolution from a single image. *IEEE 12th International Conference on Computer Vision (ICCV)*.

4. **Irani, M.** and **S. Peleg** (1991). Improving resolution by image registration. *CVGIP: Graphical Models and Image Processing, Vol: 53*.

5. **Lukin, A.**, **A. S. Krylov**, and **A. Nasonov** (2005). Image interpolation by super-resolution. *Moscow State University , Russia*.

6. **Martin, D.**, **C. Fowlkes**, **D. Tal**, and **J. Malik**, A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. *In Proc. 8th Int'l Conf. Computer Vision*, volume 2. 2001.

7. **Nguyen, N.** and **P. Milanfar** (2001). A computationally efficient superresolution image reconstruction algorithm. *IEEE Transactions on Image Processing, Vol: 10, No: 4*.

8. **Yue, H.**, **X. Sun**, **J. Yang**, and **F. Wu** (2013). Landmark image super-resolution by retrieving web images. *IEEE Transactions on Image Processing, Volume: 22*.

9. **Zontak, M.** and **M. Irani** (2011). Internal statistics of a single natural image. *SIAM Journal on Scientific Computing*.