Troy Sanchez
Math 466
Spring 2024
Assignment 11

Photo_Images

a) See Figure 3, "Mean patch as a 20 by 20 image, Photo Images".)
The general features of the 20x20 mean patch (in grayscale to adhere to Photo_Images standards) is of light shades of white and gray on top, which darken towards the bottom of the patch. The image almost resembles black-and-white film grain. This patch indicates overall lighter features in the top half and overall darker features in the bottom half.

b) See Figures 4 and 5.
The most dominant modes of variation in image patches are the eigenvector image patches that look the most similar to the mean patch (Figure 3), in this case eigenvectors 1 through 25 (Figure 4). In these patches (in grayscale to adhere to Photo_Images standards), white round objects, threads, and other discernible entities can be identified against a black backdrop, with the white areas being distinguishable from the black areas. Of eigenvectors 1 through 25, eigenvector 1 is the most straightforward of all in distinguishing white areas from black areas. In the mean patch (Figure 3), the lighter areas and the darker areas can be clearly identified.
The least dominant modes of variation in image patches are the eigenvector image patches that look the most different to the mean patch, in this case eigenvectors 375 through 400 (Figure 5). In these patches (also in grayscale), the gray areas appear to blend in with a black background, which is different from the mean patch's predominantly white background with black squares creeping in. Of eigenvectors 375 through 400, eigenvector 400 is the most dissimilar to the mean in that this patch has an indistinguishable area of white and black squares.

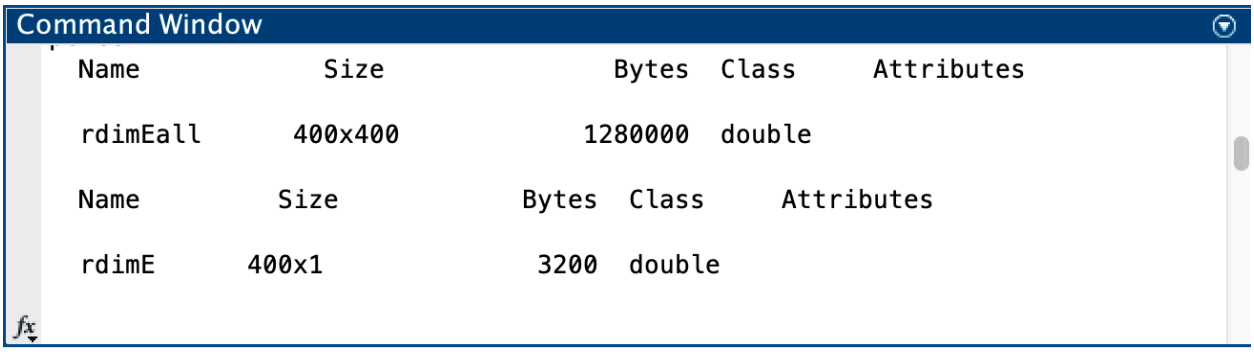
c) See Figure 6.
Here, r = 10.
Both images are set to grayscale to adhere to Photo_Images standards.
While Original image #70 has full resolution and uses all 400 eigenvectors, Re-Patched Image #70 with r = 10 only uses the largest 10 eigenvectors and still has relatively decent resolution, with the 20x20 patches noticeable but still reconstructing the original image fairly well. The Re-Patched image looks a little lighter than the original image and contains slightly indiscernible detail, but overall the r = 10 case is a more cost-conscious and generally accurate representation of the original image.

d) As shown in the MATLAB Command Window, for Photo_Images, r = 10, the percentage of total squared error accounted for by using the first 10 eigenvectors is about 99.938708601070488 percent. The squared error for the first 10 eigenvectors is about 6.8305e+08, and the (total) squared error for all 400 eigenvectors is about 6.8347e+08.

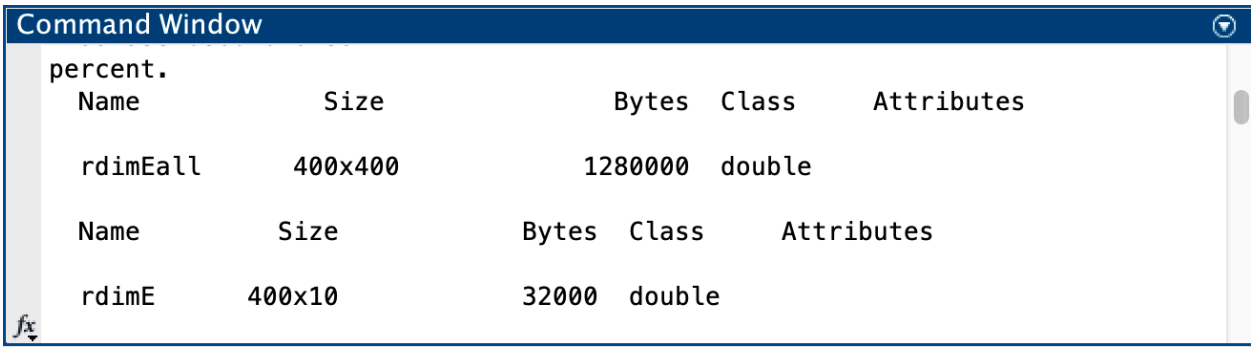
e)
r = 1:
(c) See Figure 7. Both images are set to grayscale to adhere to Photo_Images standards. Similarly to the r = 10 case, Original image #70 has full resolution and uses all 400 eigenvectors, but Re-Patched Image #70 with r = 1 just uses the largest eigenvector. The image has muddled resolution and appears much more pixelated than the original image. Although, the overall color distribution remains intact with the top left corner featuring mostly white and some gray, and the rest of the image being dark gray to black. Still, the r = 1 case would not be an adequately accurate representation of the original image.
(d) As shown in the MATLAB Command Window, for Photo_Images, r = 1, the percentage of total squared error accounted for by using the first eigenvector is about 99.630903286829536 percent. The squared error for the first eigenvector is about 6.8095e+08, and the (total) squared error for all 400 eigenvectors is about 6.8347e+08.

r = 80:
(c) See Figure 8. Both images are set to grayscale to adhere to Photo_Images standards. Original image #70 once again has full resolution and uses all 400 eigenvectors, but this time, Re-Patched image #70 looks almost identical to the original image. Aside from the shading being a bit lighter than the original, the re-patched image is virtually a carbon copy of the original image. If disregarding the storage usage, the r = 80 case is a very accurate representation of the original image.
(d) As shown in the MATLAB Command Window, for Photo_Images, r = 80, the percentage of total squared error accounted for by using the first 80 eigenvectors is about 99.987446521730135 percent. The squared error for the first 80 eigenvectors is about 6.8338e+08, and the (total) squared error for all 400 eigenvectors is about 6.8347e+08.

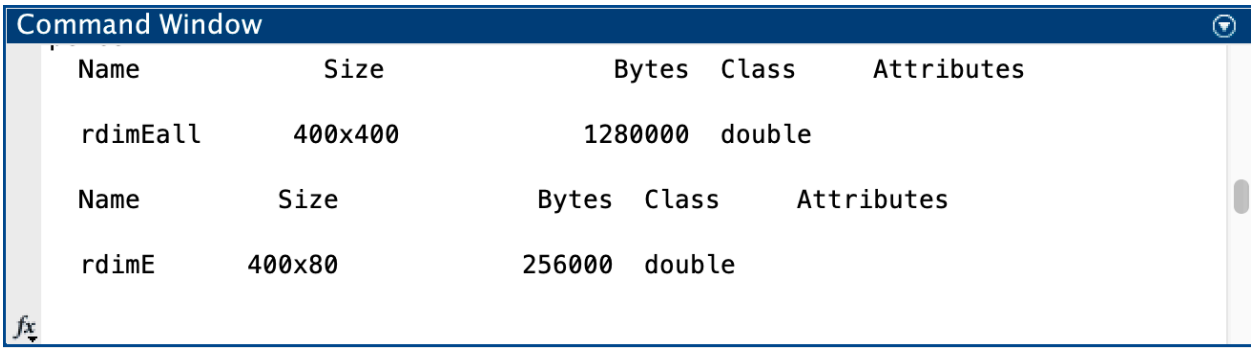
From r = 1 (Figure 7) to r = 10 (Figure 6), the results change in that the Re-Patched image becomes clearer and gains better resolution. The r = 1 case almost looks like a censored version of the original image, and the r = 10 case looks cleaner and more lightly shaded. Many finer details are discovered when switching from r = 1 to r = 10.
From r = 10 (Figure 6) to r = 80 (Figure 8), the Re-Patched image gets much closer to the original image and could pass for the original with a bit lighter shade. The Re-Patched image no longer looks fuzzy, and adding more eigenvectors may not provide much more necessary detail.
We compare the storage of the eigenvectors for each r-dimension.
For r = 1, we have



The case of all eigenvectors takes up 1,280,000 bytes, but if we compress the entire dataset into just the largest eigenvector, only 3,200 bytes are taken up. Hence, we would be saving 1280000 - 3200 = 1,276,800 bytes if we compress the dataset such that r = 1.
Moreover, for r = 10, we have



If we compress the entire dataset into only the largest r = 10 eigenvectors, only 32,000 bytes are taken up, and we would be saving 1280000 - 32000 = 1,248,000 bytes.
Lastly, for r = 80, we have



If we compress the entire dataset into only the largest r = 80 eigenvectors, just 256,000 bytes are taken up, and we would be saving 1280000 - 256000 = 1,024,000 bytes.

Figure 1.

Figure 2.

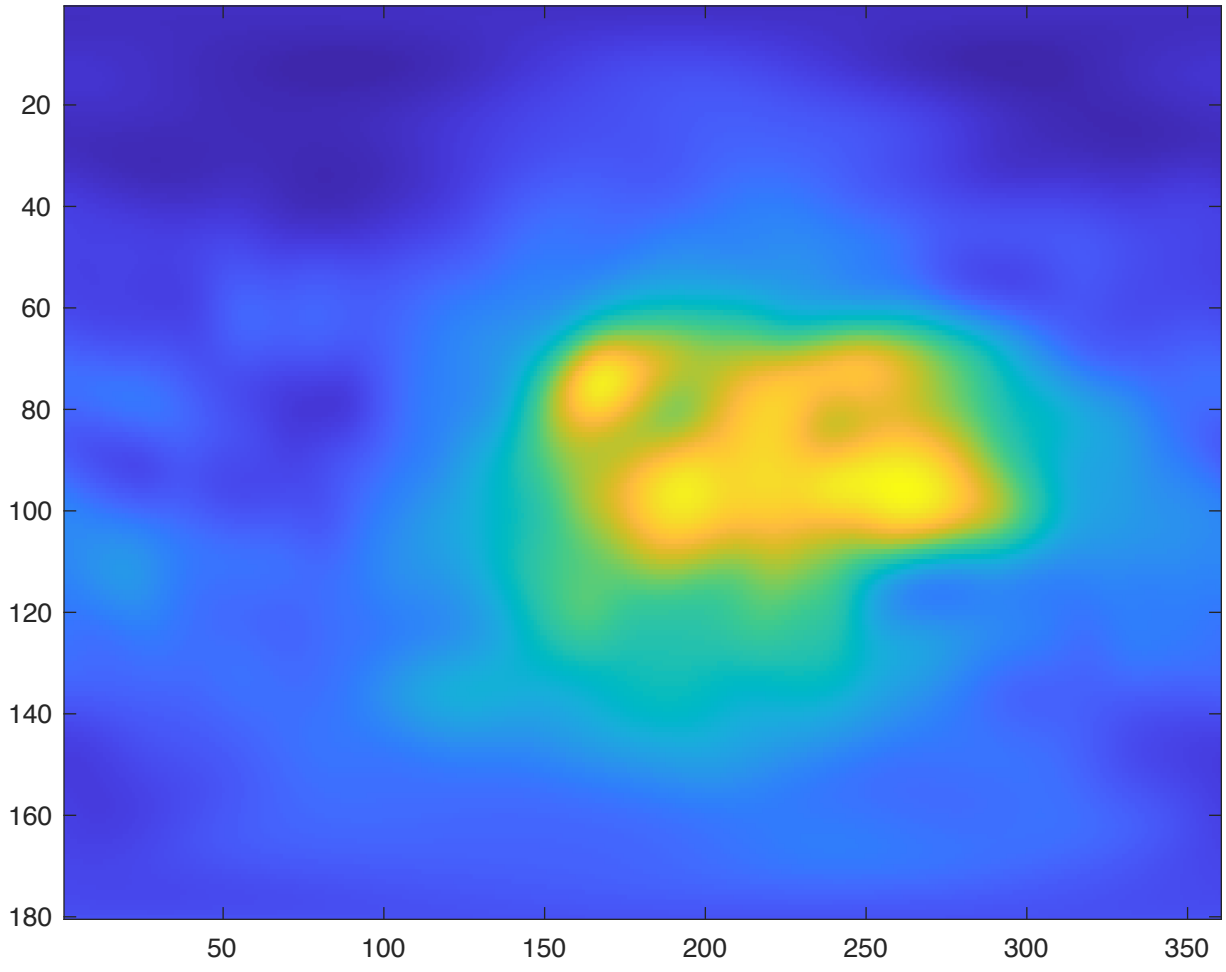


Figure 3.

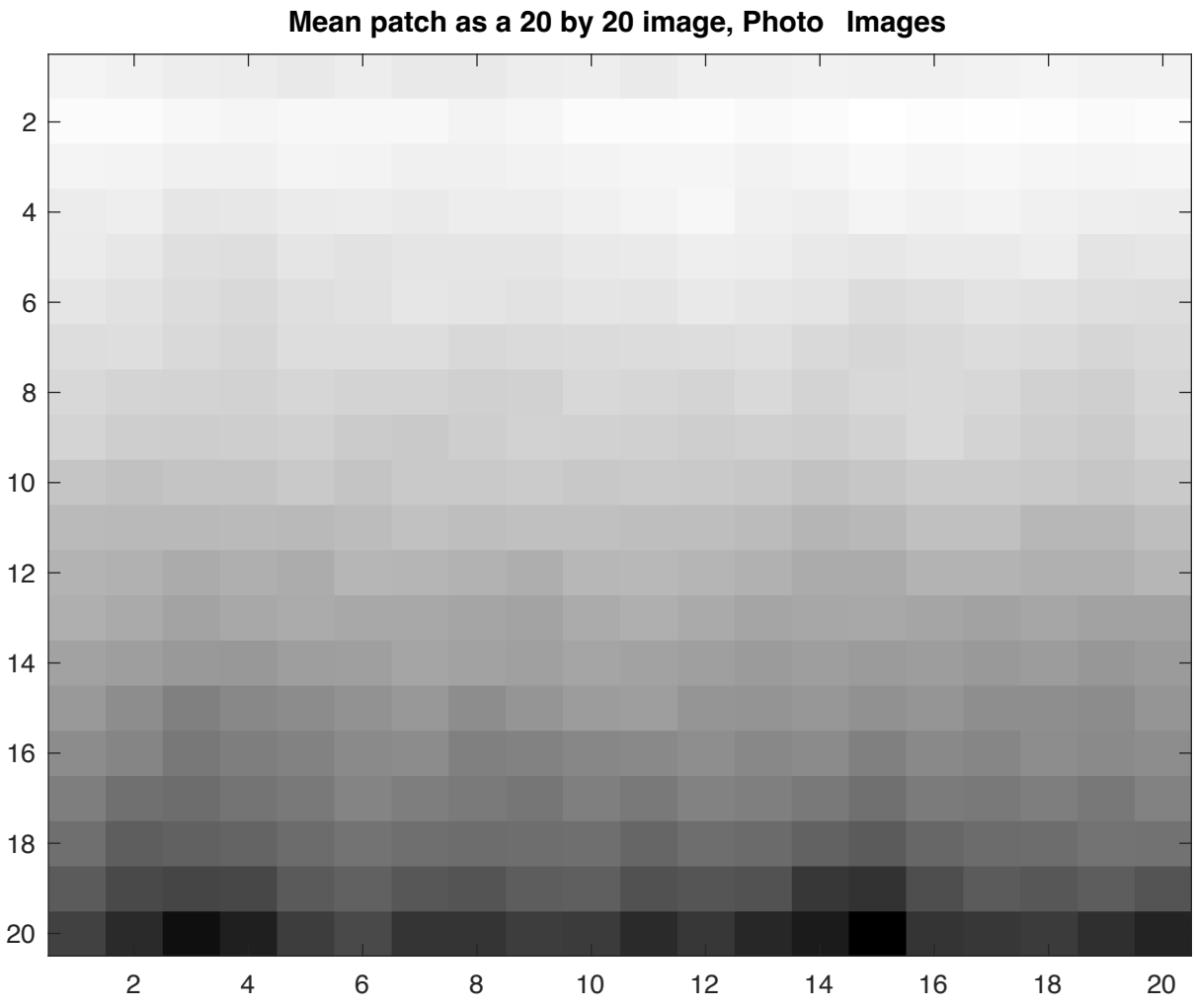

Mean patch as a 20 by 20 image, Photo Images
2
4
6
8
10
12
14
16
18
20
2
4
6
8
10
12
14
16
18
20

Figure 4.

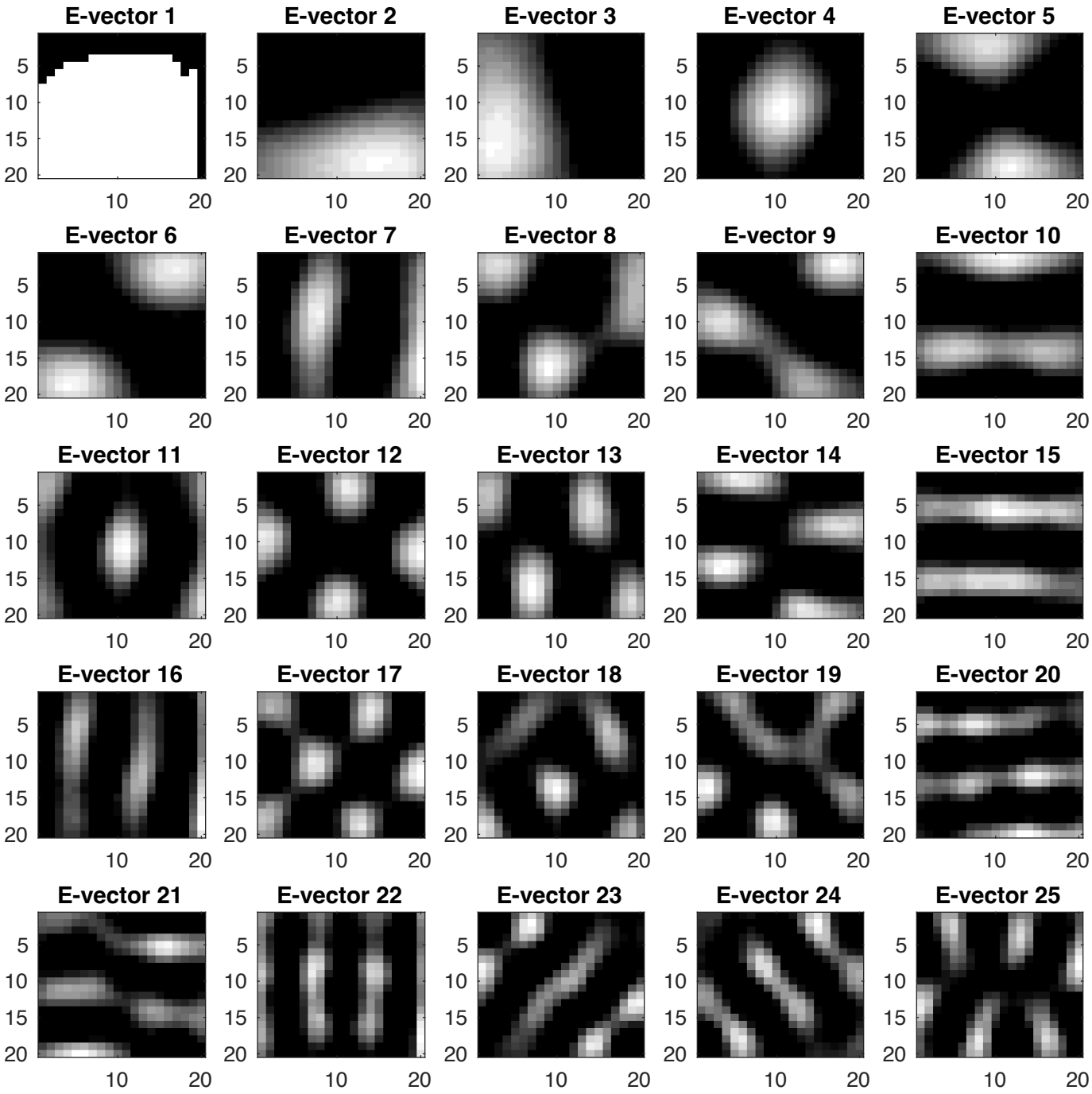

E-vector 1
E-vector 2
E-vector 3
E-vector 4
E-vector 5
E-vector 6
E-vector 7
E-vector 8
E-vector 9
E-vector 10
E-vector 11
E-vector 12
E-vector 13
E-vector 14
E-vector 15
E-vector 16
E-vector 17
E-vector 18
E-vector 19
E-vector 20
E-vector 21
E-vector 22
E-vector 23
E-vector 24
E-vector 25
5
10
15
20
10
20

Figure 5.

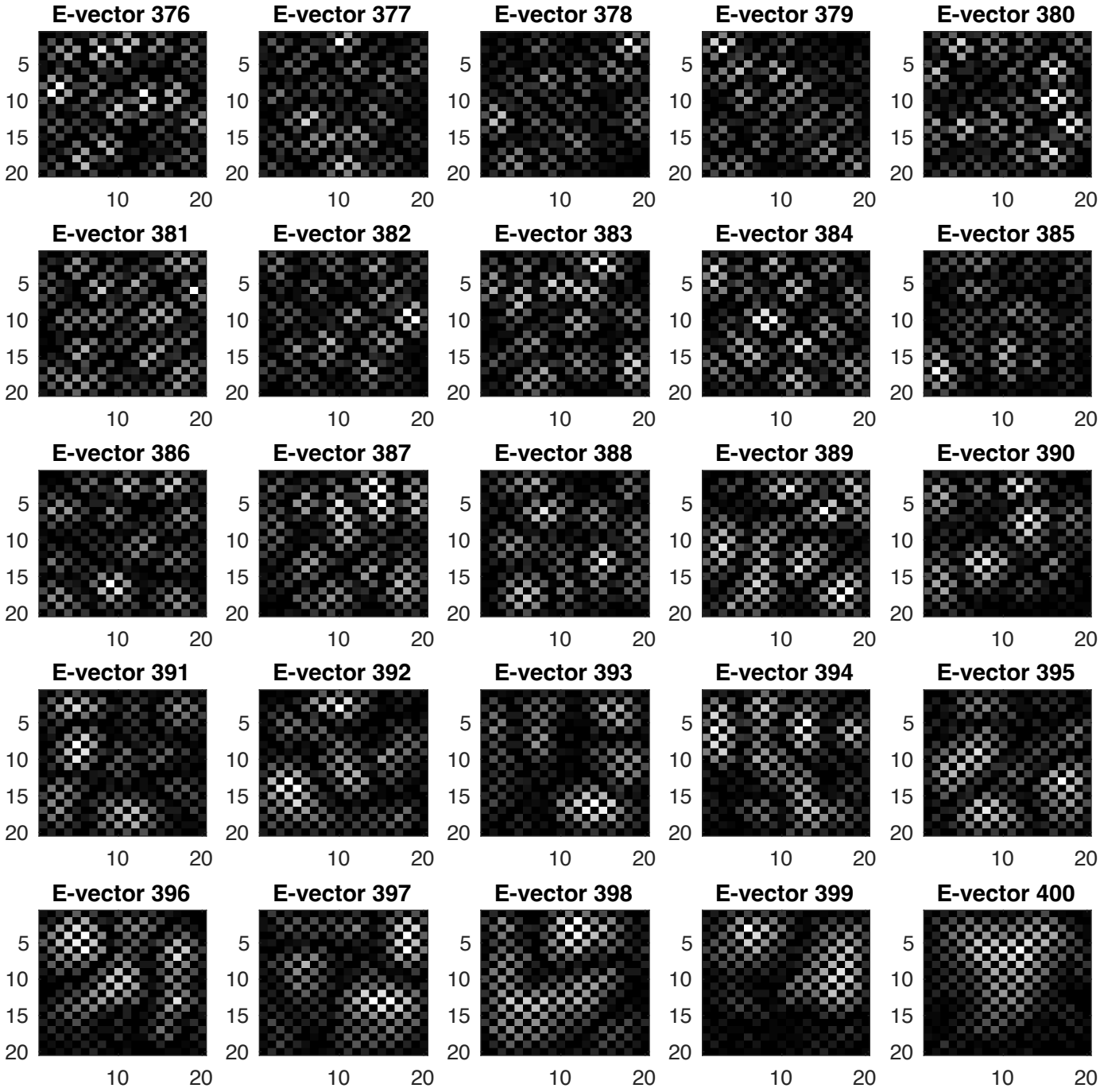

E-vector 376
E-vector 377
E-vector 378
E-vector 379
E-vector 380
E-vector 381
E-vector 382
E-vector 383
E-vector 384
E-vector 385
E-vector 386
E-vector 387
E-vector 388
E-vector 389
E-vector 390
E-vector 391
E-vector 392
E-vector 393
E-vector 394
E-vector 395
E-vector 396
E-vector 397
E-vector 398
E-vector 399
E-vector 400
5
10
15
20
10
20

Figure 6.

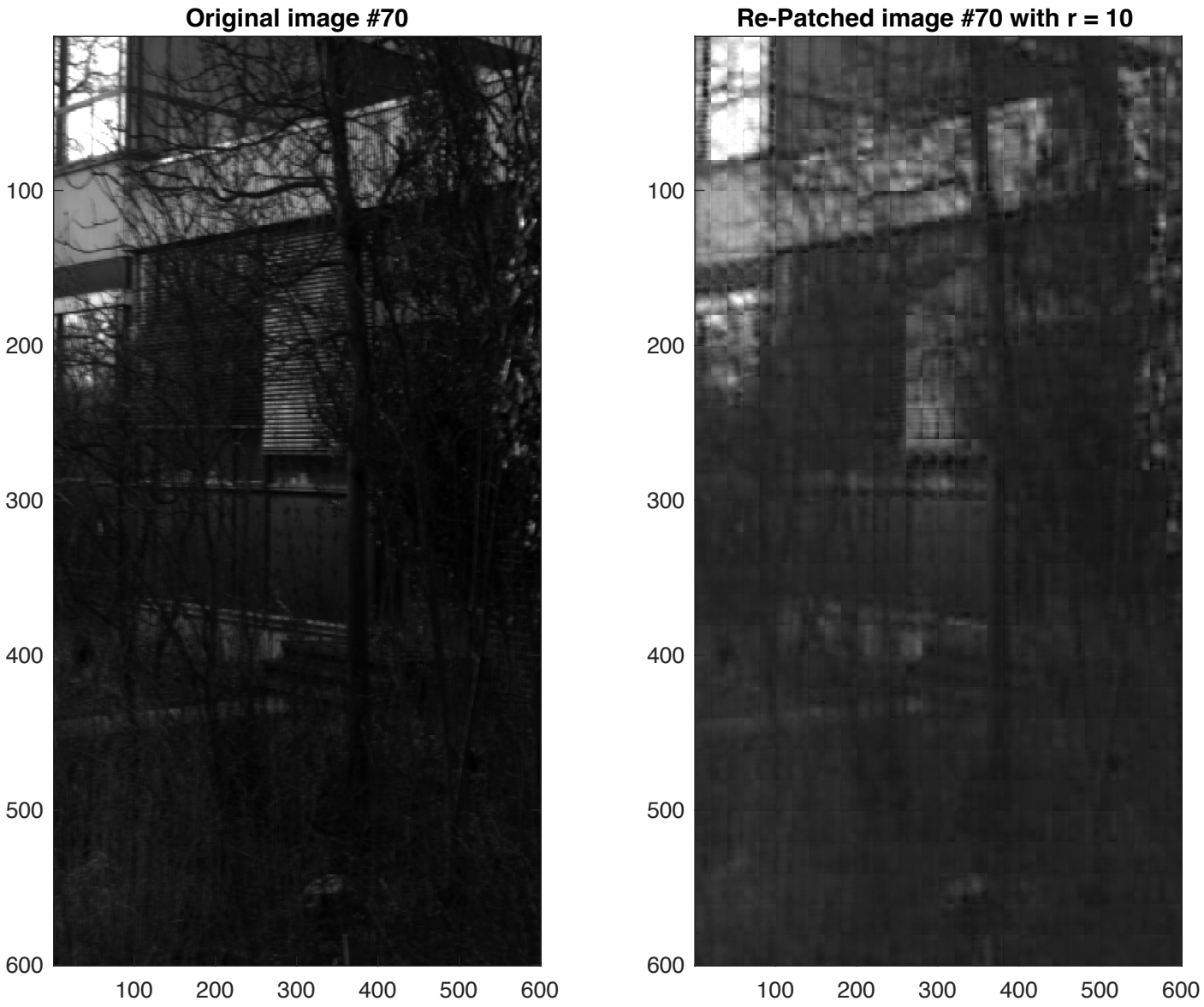

Original image #70
Re-Patched image #70 with r = 10
100
200
300
400
500
600

Figure 7.

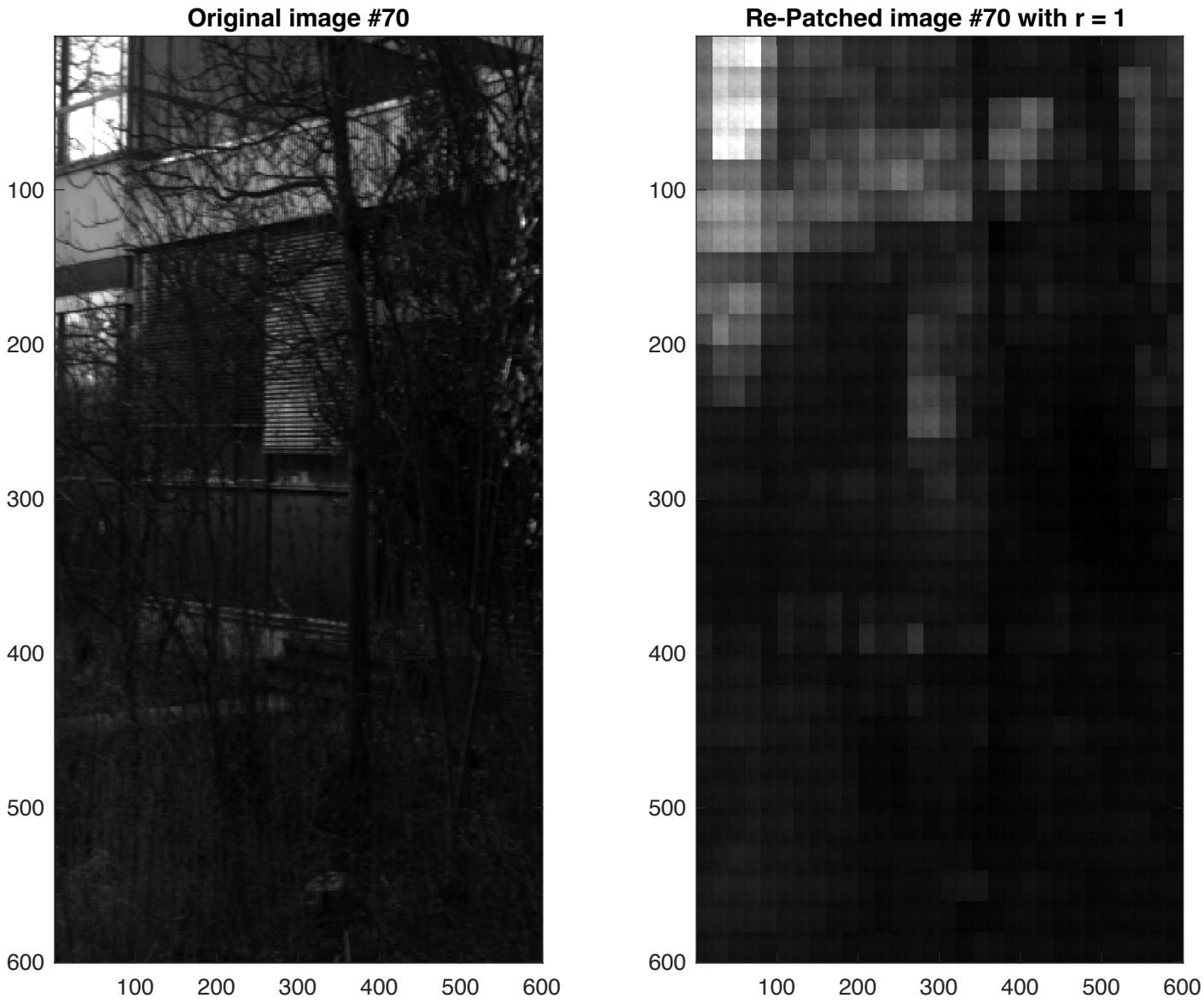
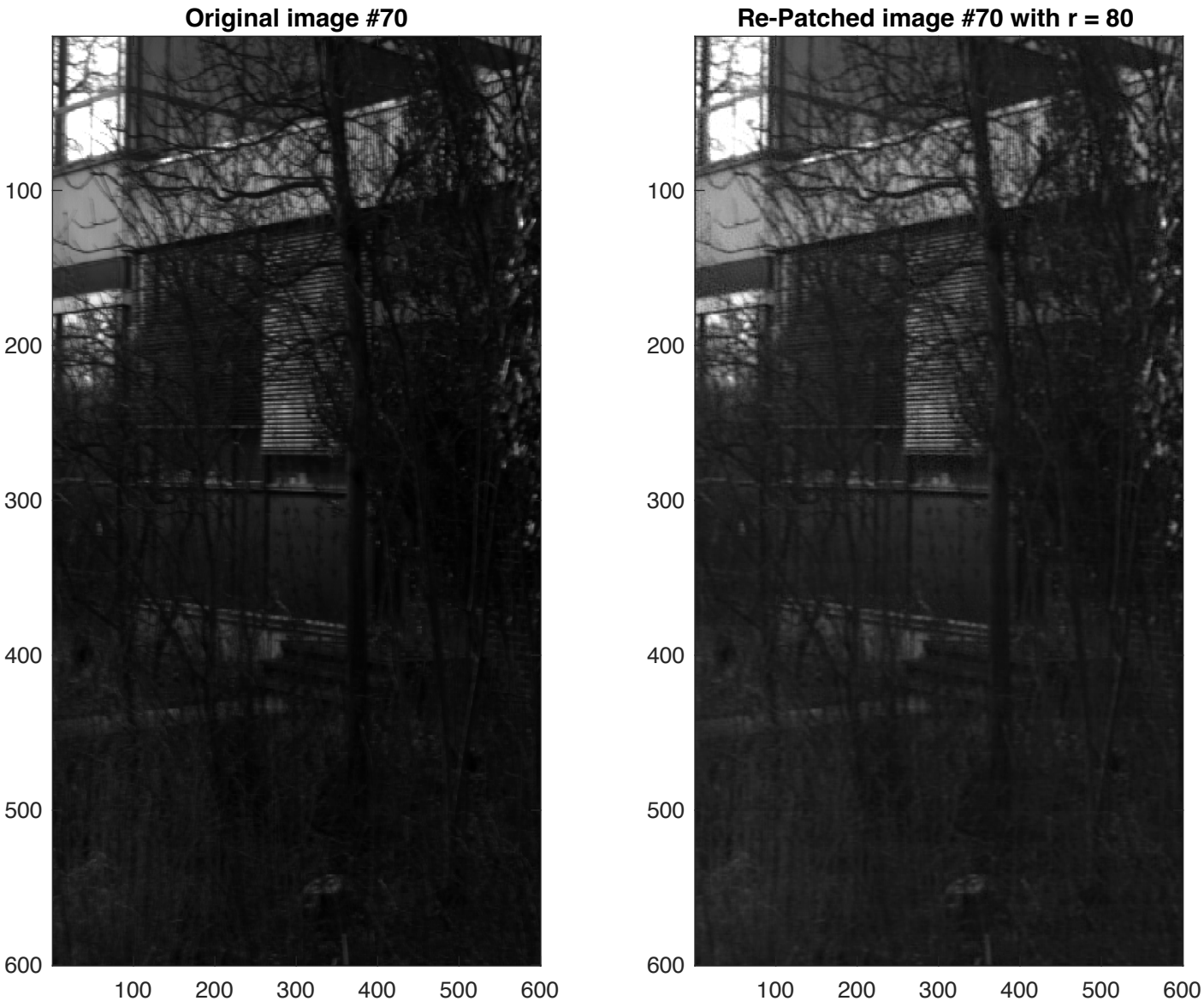

Original image #70
Re-Patched image #70 with r = 1
100
200
300
400
500
600

Figure 8.


Original image #70
Re-Patched image #70 with r = 80
100
200
300
400
500
600

VTEC_Images

a) See Figure 9, "Mean patch as a 20 by 20 image, VTEC Images".
The general features of the 20x20 mean patch (in colormap('default')) to adhere to VTEC_Images standards) is of horizontal bands of a little blue on top, a substantial amount of blue on the bottom, and orange and bright yellow in the middle.

b) See Figures 10 and 11.
The most dominant modes of variation in image patches are, once again, the eigenvector image patches that look the most similar to the general distribution of the mean patch (Figure 9), in this case eigenvectors 1 through 25 (Figure 10). The patch corresponding to eigenvector 1 seems the most similar to the mean patch, as there is a large blob of bright yellow much like the 20x20 mean patch. The different colors can be easily distinguished in the mean patch as well as in the patches of the largest 25 eigenvalues.
The least dominant modes of variation in image patches are, again, the eigenvector image patches that look the most different to the mean patch, in this case, eigenvectors 375 through 400 (Figure 11). In these patches, all the colors are mixed with each other in an ambiguously blue background, which contrasts with the orderly horizontal stripes of the mean patch.

c) See Figure 12.
Here, r = 10.
Original image #170 has full resolution and uses all 400 eigenvectors, but Re-Patched image #170 with r = 10 just uses the largest 10 eigenvectors. For using less than one percent of all eigenvectors, the Re-Patched image looks very accurate and reconstructs the original image extremely well. Aside from a few hardly noticeable areas of slight fuzz like in the bottom left and bottom right, the r = 10 case is heartily recommended as a cost-conscious and accurate reconstruction of the original image.

d) As shown in the MATLAB Command Window, for VTEC_Images, r = 10, the percentage of total squared error accounted for by using the first 10 eigenvectors is about 99.999871231291877 percent. The (total) squared error for all 400 eigenvectors is about 1.4880e+07, and the squared error for the first 10 eigenvectors is about 1.4880e+07 (very close to the error for all eigenvectors).

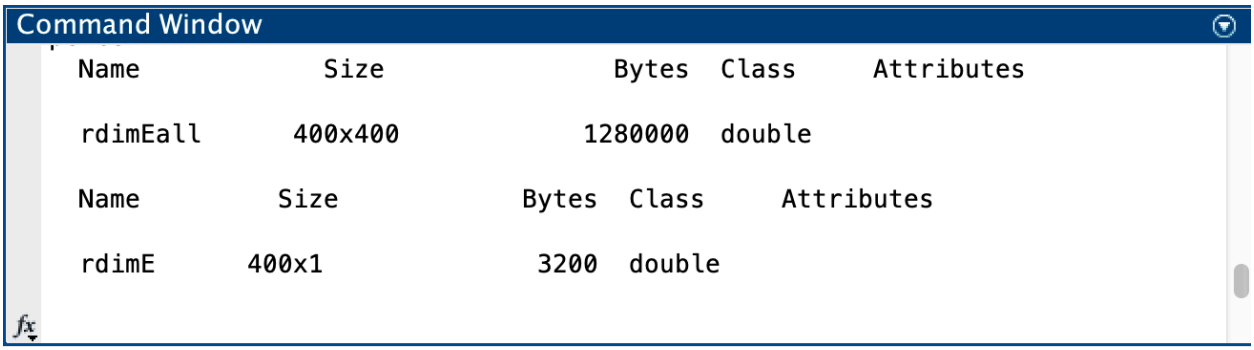
e)
r = 1:
(c) See Figure 13. Compared to Original image #170's full resolution and 400 eigenvectors, Re-Patched image #170 with r = 1 uses only the largest eigenvector. Though the muddled resolution is noticeable here and looks "blocky", the color distribution is still the same and retains the main color features of the original image. Due to the Re-Patched image's muddled resolution, the r = 1 case would not be recommended as a good reconstruction of the original image.
(d) As shown in the MATLAB Command Window, for VTEC_Images, r = 1, the percentage of total squared error accounted for by using the first eigenvector is about 99.978042642193060

percent. The squared error for the first eigenvector is about 1.4877e+07, and the (total) squared error for all 400 eigenvectors is about 1.4880e+07.

r = 80:
(c) See Figure 14. Once more, Original image #170 has full resolution and uses 400 eigenvectors, and Re-Patched image #170 with r = 80 uses just 80 eigenvectors with stunning results. The Re-Patched still looks a little fuzzy like in the r = 10 case, but it is still greatly accurate and is happily recommended as a cost-efficient and reliable reconstruction of the original image.
(d) As shown in the MATLAB Command Window, For VTEC_Images, r = 80, the percentage of total squared error accounted for by using the first 80 eigenvectors is about 99.999973241135123 percent. The (total) squared error for all 400 eigenvectors is about 1.4880e+07, and the squared error for the first 80 eigenvectors is also about 1.4880e+07.

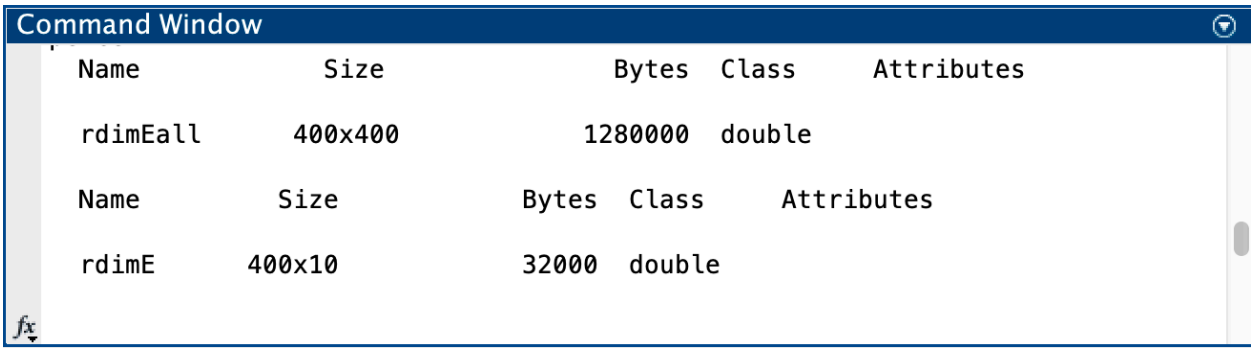
From r = 1 (Figure 13) to r = 10 (Figure 12), the results change in that the Re-Patched image becomes less blocky and, again, gains better resolution. The r = 10 image looks cleaner and more defined than the r = 1 image.
From r = 10 (Figure 12) to r = 80 (Figure 14), the Re-Patched image in the r = 80 case actually does not look much different than the r = 10 case, though when zooming in, the r = 80 case has finer detail. The r = 80 case still looks grainy, but it is clearer than the r = 10 case. Upgrading from r = 10 to r = 80 may not provide significantly better results.

We now compare the storage of the eigenvectors for each r-dimension.
For r = 1, we have

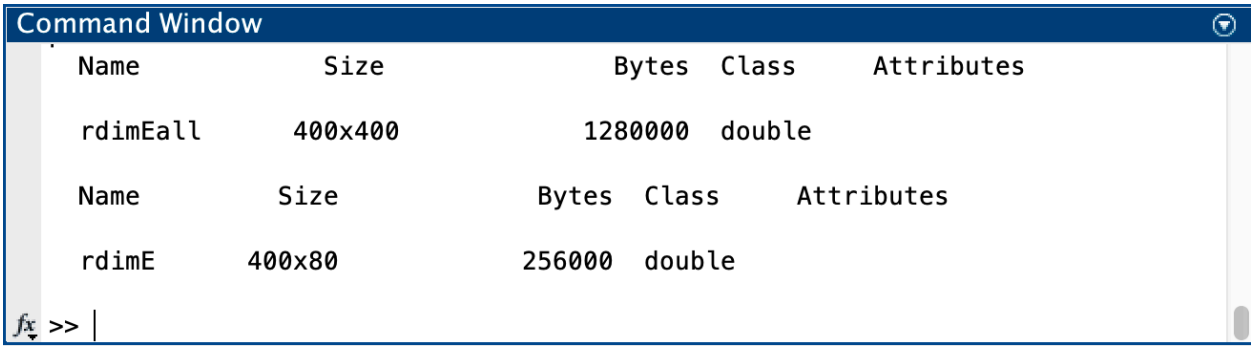


Much like with Photo_Images, the case of all eigenvectors takes up 1,280,000 bytes. If we compress the entire dataset into only the largest eigenvector, only 3,200 bytes are consumed, and we would be saving 1280000 - 3200 = 1,276,800 bytes by compressing the dataset where r = 1.

Also, for r = 10, we have



If we compress the entire dataset into just the largest r = 10 eigenvectors, only 32,000 bytes are consumed, saving 1280000 - 32000 = 1,248,000 bytes.
Finally, for r = 80, we have



Once more, if we compress the whole dataset into just the largest r = 80 eigenvectors, only 256,000 bytes are consumed, saving 1280000 - 256000 = 1,024,000 bytes.

Figure 9.

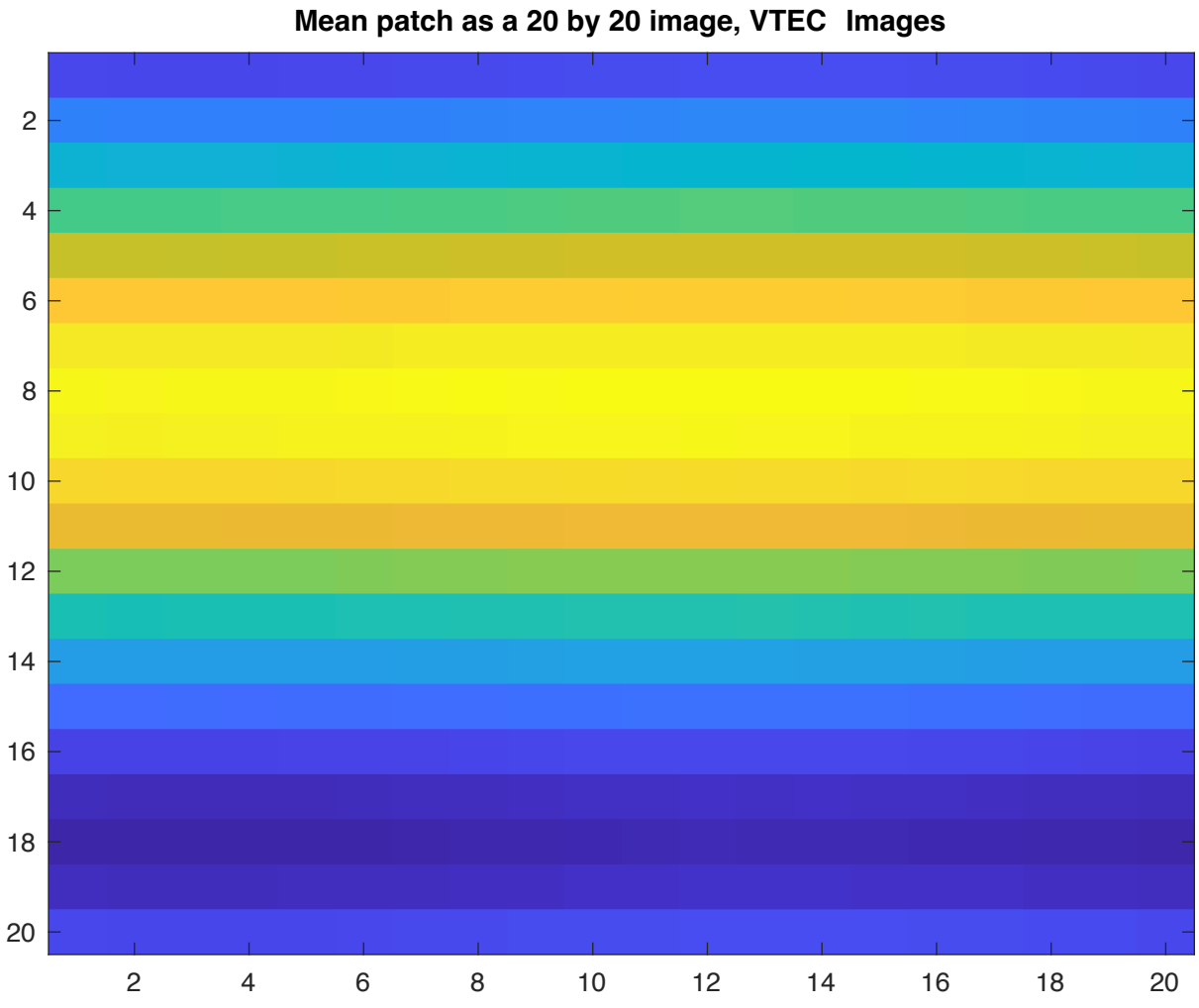

Mean patch as a 20 by 20 image, VTEC Images
2
4
6
8
10
12
14
16
18
20
2
4
6
8
10
12
14
16
18
20

Figure 10.

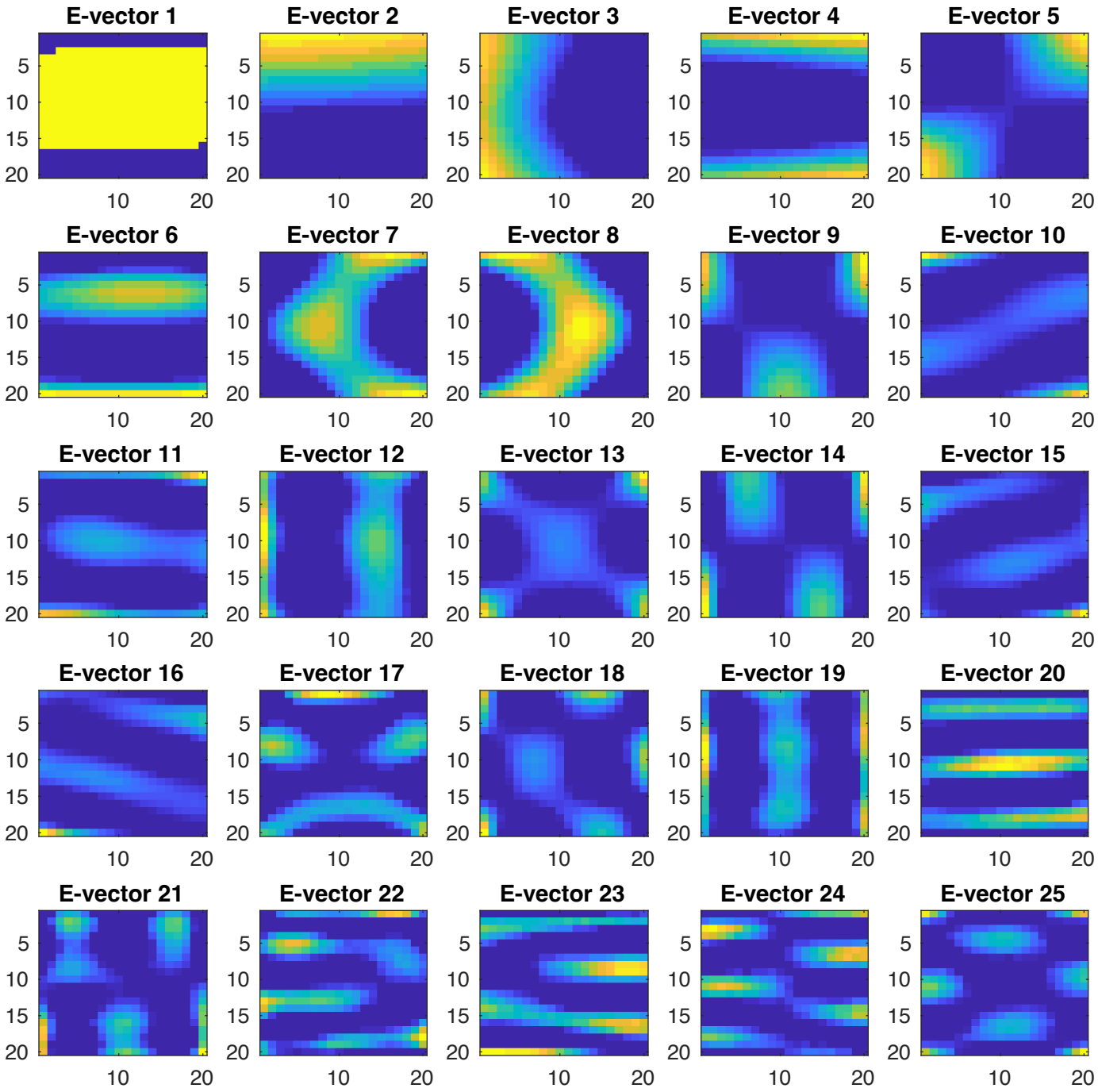

E-vector 1
E-vector 2
E-vector 3
E-vector 4
E-vector 5
E-vector 6
E-vector 7
E-vector 8
E-vector 9
E-vector 10
E-vector 11
E-vector 12
E-vector 13
E-vector 14
E-vector 15
E-vector 16
E-vector 17
E-vector 18
E-vector 19
E-vector 20
E-vector 21
E-vector 22
E-vector 23
E-vector 24
E-vector 25
5
10
15
20
10
20

Figure 11.

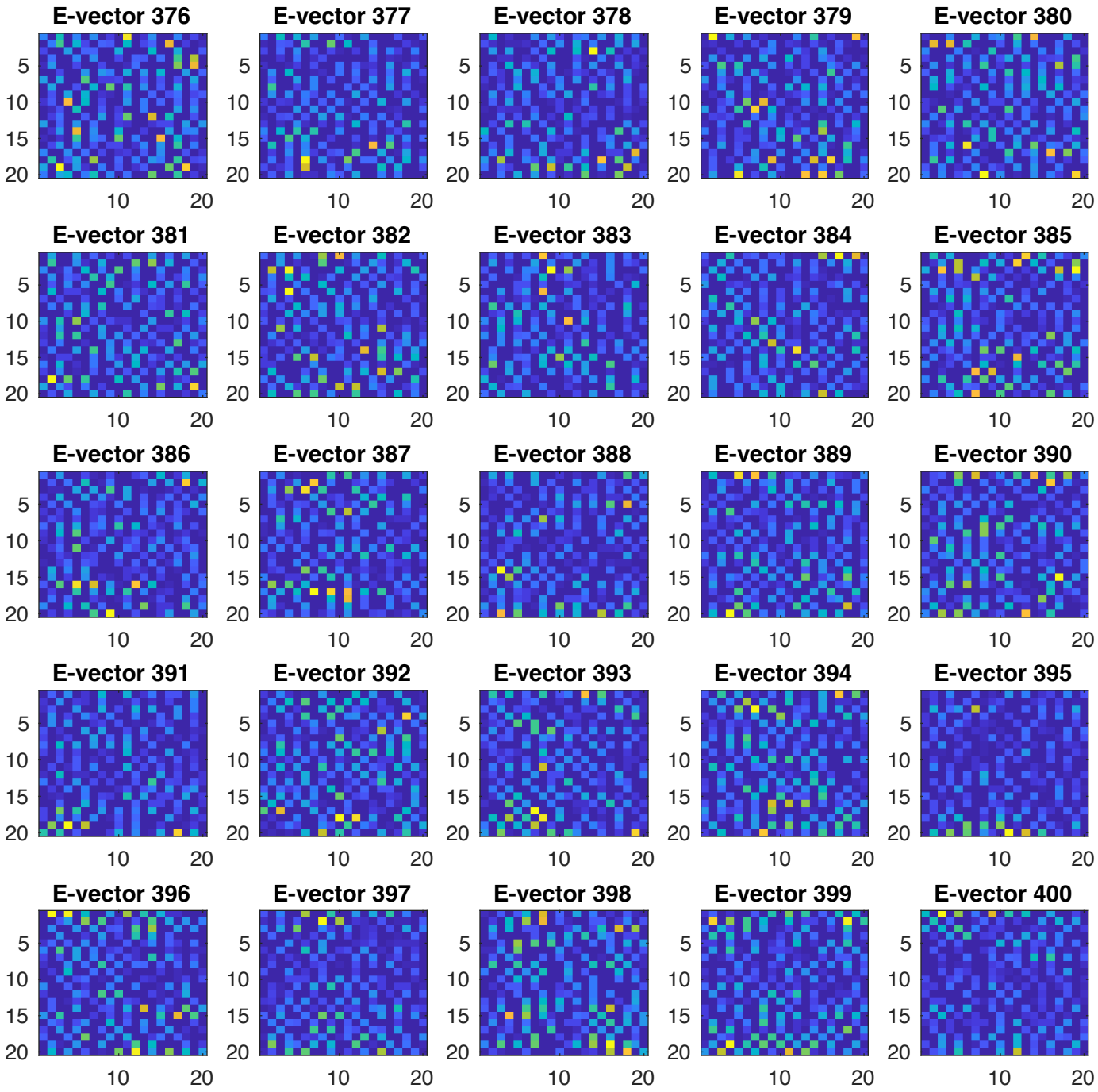

E-vector 376
E-vector 377
E-vector 378
E-vector 379
E-vector 380
E-vector 381
E-vector 382
E-vector 383
E-vector 384
E-vector 385
E-vector 386
E-vector 387
E-vector 388
E-vector 389
E-vector 390
E-vector 391
E-vector 392
E-vector 393
E-vector 394
E-vector 395
E-vector 396
E-vector 397
E-vector 398
E-vector 399
E-vector 400
5
10
15
20
10
20

Figure 12.

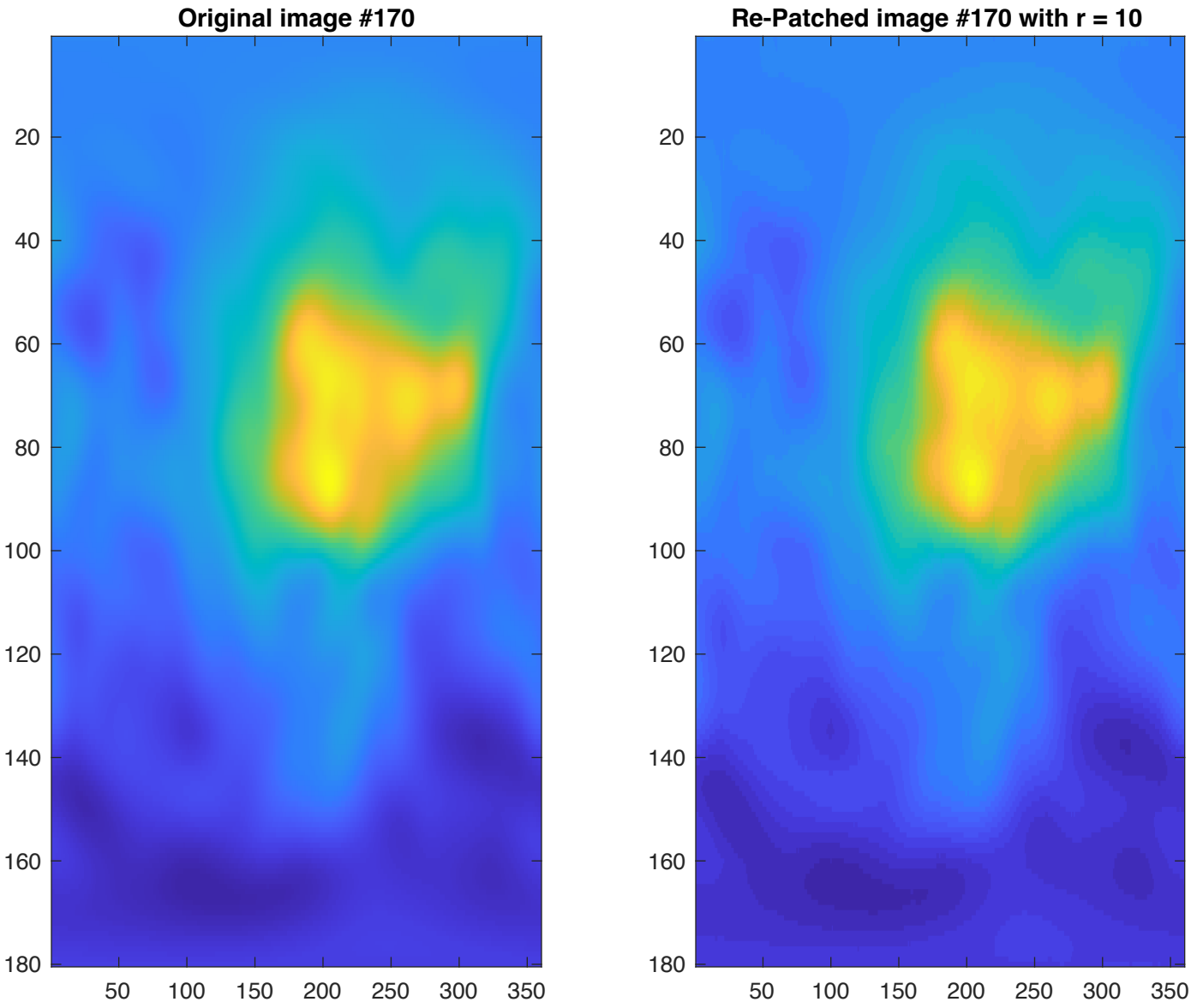

Original image #170
Re-Patched image #170 with r = 10
20
40
60
80
100
120
140
160
180
50
100
150
200
250
300
350

Figure 13.

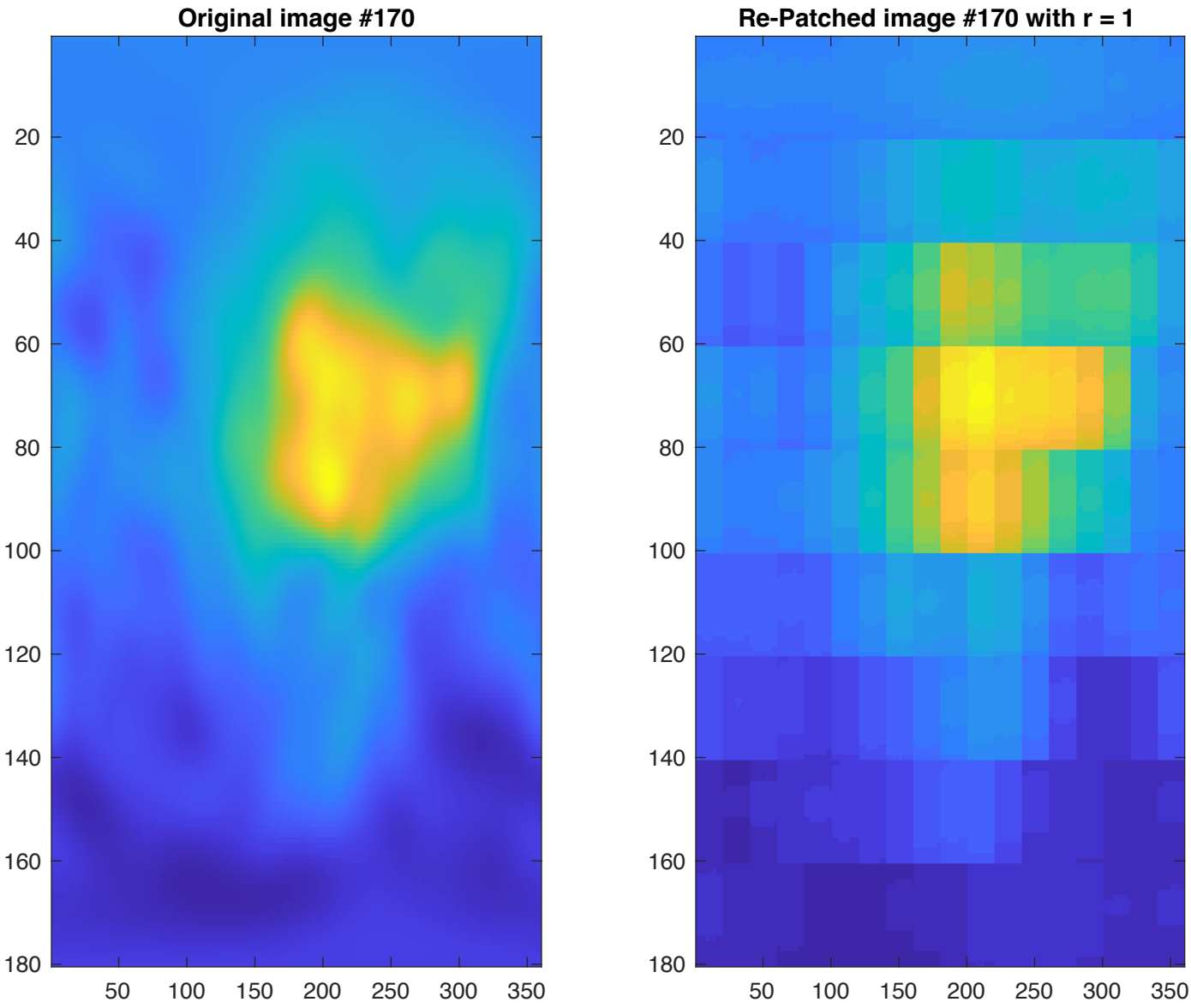
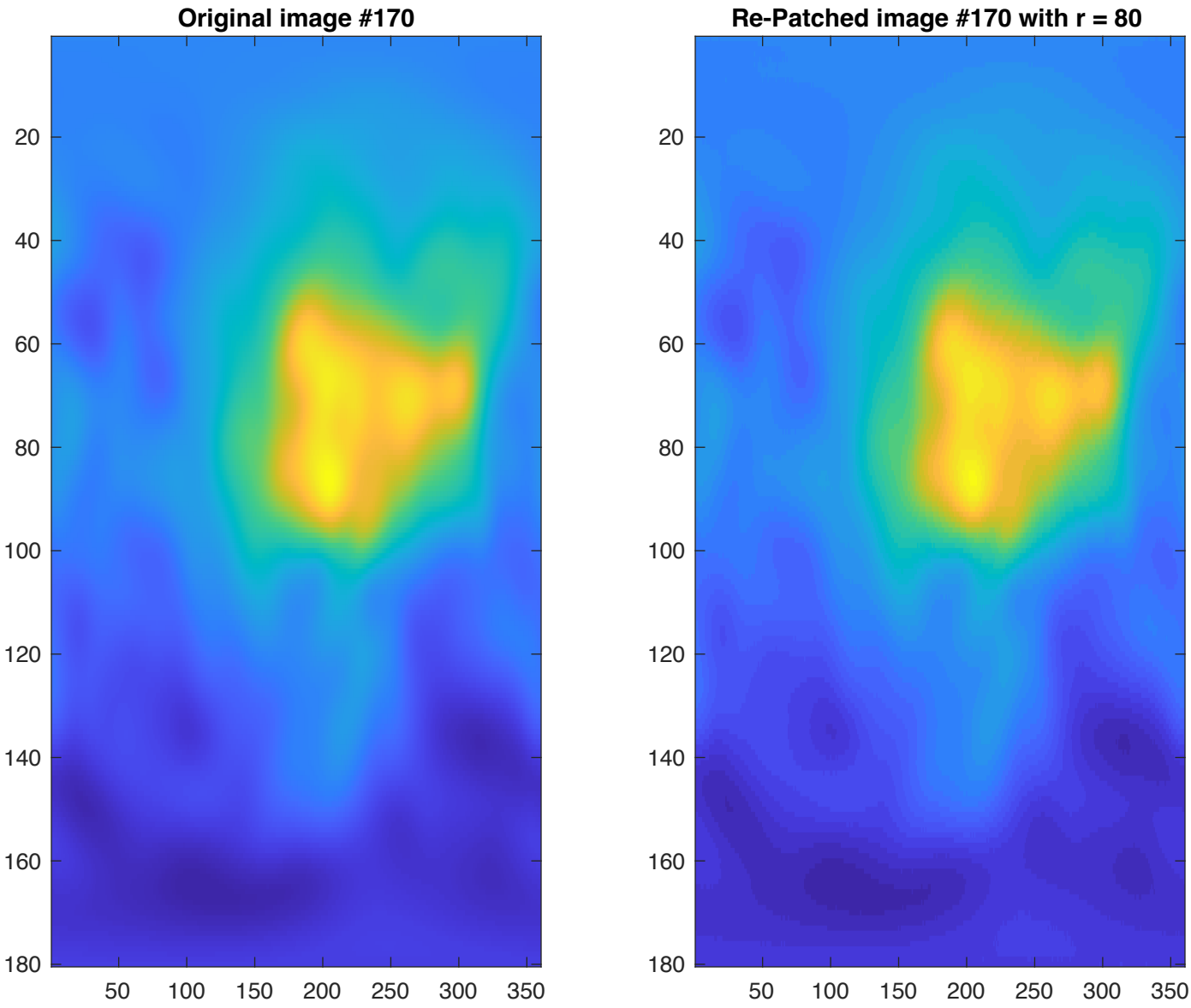

Original image #170
Re-Patched image #170 with r = 1
20
40
60
80
100
120
140
160
180
50
100
150
200
250
300
350

Figure 14.


Original image #170
Re-Patched image #170 with r = 80
20
40
60
80
100
120
140
160
180
50
100
150
200
250
300
350

MATLAB Command Window

```
Your variables are:

Photo_Images  VTEC_Images

ans =
   600   600   100
ans =
   180   360   365

For Photo _ Images, r=10,
the percentage of total squared error accounted for
by using the first 10 eigenvector(s) is about
  99.938708601070488
percent.
  Name            Size               Bytes  Class     Attributes

  rdimEall      400x400            1280000  double

  Name          Size             Bytes  Class     Attributes

  rdimE       400x10             32000  double


For Photo _ Images, r=1,
the percentage of total squared error accounted for
by using the first 1 eigenvector(s) is about
  99.630903286829536
percent.
  Name            Size               Bytes  Class     Attributes

  rdimEall      400x400            1280000  double

  Name          Size             Bytes  Class     Attributes

  rdimE       400x1               3200  double


For Photo _ Images, r=80,
the percentage of total squared error accounted for
by using the first 80 eigenvector(s) is about
  99.987446521730135
percent.
  Name            Size               Bytes  Class     Attributes

  rdimEall      400x400            1280000  double

  Name          Size              Bytes  Class     Attributes

  rdimE       400x80             256000  double


For VTEC _ Images, r=10,
the percentage of total squared error accounted for
by using the first 10 eigenvector(s) is about
  99.999871231291877
percent.
  Name            Size               Bytes  Class     Attributes

  rdimEall      400x400            1280000  double

  Name          Size             Bytes  Class     Attributes
```

```
  rdimE      400x10            32000  double


For VTEC _ Images, r=1,
the percentage of total squared error accounted for
by using the first 1 eigenvector(s) is about
  99.978042642193060
percent.
  Name            Size               Bytes  Class     Attributes

  rdimEall      400x400            1280000  double

  Name         Size            Bytes  Class     Attributes

  rdimE      400x1              3200  double


For VTEC _ Images, r=80,
the percentage of total squared error accounted for
by using the first 80 eigenvector(s) is about
  99.999973241135123
percent.
  Name            Size               Bytes  Class     Attributes

  rdimEall      400x400            1280000  double

  Name         Size             Bytes  Class     Attributes

  rdimE      400x80           256000  double

Warning: The figure is too large for the page and will be cut off. Resize
the figure, adjust the output size by setting the figure's PaperPosition
property, use the 'print' command with either the  '-bestfit' or '-fillpage'
options, or use the 'Best fit' or 'Fill page' options on the 'Print Preview'
window.
> In validate (line 146)
In print (line 67)
In hgexport
In filemenufcn>localSaveExportHelper (line 225)
In filemenufcn>localSaveExport (line 350)
In filemenufcn (line 58)
Warning: The figure is too large for the page and will be cut off. Resize
the figure, adjust the output size by setting the figure's PaperPosition
property, use the 'print' command with either the  '-bestfit' or '-fillpage'
options, or use the 'Best fit' or 'Fill page' options on the 'Print Preview'
window.
> In validate (line 146)
In print (line 67)
In hgexport
In filemenufcn>localSaveExportHelper (line 225)
In filemenufcn>localSaveExport (line 350)
In filemenufcn (line 58)
Warning: The figure is too large for the page and will be cut off. Resize
the figure, adjust the output size by setting the figure's PaperPosition
property, use the 'print' command with either the  '-bestfit' or '-fillpage'
options, or use the 'Best fit' or 'Fill page' options on the 'Print Preview'
window.
> In validate (line 146)
In print (line 67)
In hgexport
In filemenufcn>localSaveExportHelper (line 225)
In filemenufcn>localSaveExport (line 350)
In filemenufcn (line 58)
```

```
Warning: The figure is too large for the page and will be cut off. Resize
the figure, adjust the output size by setting the figure's PaperPosition
property, use the 'print' command with either the  '-bestfit' or '-fillpage'
options, or use the 'Best fit' or 'Fill page' options on the 'Print Preview'
window.
> In validate (line 146)
In print (line 67)
In hgexport
In filemenufcn>localSaveExportHelper (line 225)
In filemenufcn>localSaveExport (line 350)
In filemenufcn (line 58)
Warning: The figure is too large for the page and will be cut off. Resize
the figure, adjust the output size by setting the figure's PaperPosition
property, use the 'print' command with either the  '-bestfit' or '-fillpage'
options, or use the 'Best fit' or 'Fill page' options on the 'Print Preview'
window.
> In validate (line 146)
In print (line 67)
In hgexport
In filemenufcn>localSaveExportHelper (line 225)
In filemenufcn>localSaveExport (line 350)
In filemenufcn (line 58)
Warning: The figure is too large for the page and will be cut off. Resize
the figure, adjust the output size by setting the figure's PaperPosition
property, use the 'print' command with either the  '-bestfit' or '-fillpage'
options, or use the 'Best fit' or 'Fill page' options on the 'Print Preview'
window.
> In validate (line 146)
In print (line 67)
In hgexport
In filemenufcn>localSaveExportHelper (line 225)
In filemenufcn>localSaveExport (line 350)
In filemenufcn (line 58)
Warning: The figure is too large for the page and will be cut off. Resize
the figure, adjust the output size by setting the figure's PaperPosition
property, use the 'print' command with either the  '-bestfit' or '-fillpage'
options, or use the 'Best fit' or 'Fill page' options on the 'Print Preview'
window.
> In validate (line 146)
In print (line 67)
In hgexport
In filemenufcn>localSaveExportHelper (line 225)
In filemenufcn>localSaveExport (line 350)
In filemenufcn (line 58)
Warning: The figure is too large for the page and will be cut off. Resize
the figure, adjust the output size by setting the figure's PaperPosition
property, use the 'print' command with either the  '-bestfit' or '-fillpage'
options, or use the 'Best fit' or 'Fill page' options on the 'Print Preview'
window.
> In validate (line 146)
In print (line 67)
In hgexport
In filemenufcn>localSaveExportHelper (line 225)
In filemenufcn>localSaveExport (line 350)
In filemenufcn (line 58)
Warning: The figure is too large for the page and will be cut off. Resize
the figure, adjust the output size by setting the figure's PaperPosition
property, use the 'print' command with either the  '-bestfit' or '-fillpage'
options, or use the 'Best fit' or 'Fill page' options on the 'Print Preview'
window.
> In validate (line 146)
In print (line 67)
In hgexport
```

```
In filemenufcn>localSaveExportHelper (line 225)
In filemenufcn>localSaveExport (line 350)
In filemenufcn (line 58)
Warning: The figure is too large for the page and will be cut off. Resize
the figure, adjust the output size by setting the figure's PaperPosition
property, use the 'print' command with either the  '-bestfit' or '-fillpage'
options, or use the 'Best fit' or 'Fill page' options on the 'Print Preview'
window.
> In validate (line 146)
In print (line 67)
In hgexport
In filemenufcn>localSaveExportHelper (line 225)
In filemenufcn>localSaveExport (line 350)
In filemenufcn (line 58)
Warning: The figure is too large for the page and will be cut off. Resize
the figure, adjust the output size by setting the figure's PaperPosition
property, use the 'print' command with either the  '-bestfit' or '-fillpage'
options, or use the 'Best fit' or 'Fill page' options on the 'Print Preview'
window.
> In validate (line 146)
In print (line 67)
In hgexport
In filemenufcn>localSaveExportHelper (line 225)
In filemenufcn>localSaveExport (line 350)
In filemenufcn (line 58)
Warning: The figure is too large for the page and will be cut off. Resize
the figure, adjust the output size by setting the figure's PaperPosition
property, use the 'print' command with either the  '-bestfit' or '-fillpage'
options, or use the 'Best fit' or 'Fill page' options on the 'Print Preview'
window.
> In validate (line 146)
In print (line 67)
In hgexport
In filemenufcn>localSaveExportHelper (line 225)
In filemenufcn>localSaveExport (line 350)
In filemenufcn (line 58)
>>
```

MATLAB Workspace
Apr 19, 2024

Name	Value
ans	[180,360,365]
D_Photo_Images	*600x600x100 do...*
image_num_P	70
image_num_V	170
n	20
Photo_Images	*600x600x100 ui...*
VTEC_Images	*180x360x365 do...*

```
clear, clc, clf, close all
load('PCA_Exercise_Images.mat')
who
size(Photo_Images) % 600 600 100 Image 1, Image 2, ... Image 100
% imagesc(Photo_Images(:,:,40))
% colormap(sqrt(gray(256)))
figure(1)
imagesc(Photo_Images(:,:,40))
colormap(sqrt(gray(256)))
size(VTEC_Images) % 180 360 365
figure(2)
imagesc(VTEC_Images(:,:,40))
colormap('default')

% size(Photo_Images,1)/20, size(Photo_Images,2)/20

n = 20; % For the code for parts (b),(c) to run, let n >= 5 AND n^2 > r,
% AND n divides size(..._Images,1), AND n divides size(..._Images,2).
D_Photo_Images = double(Photo_Images);

% Photo_Images introQ
image_num_P = 70; % 70 for the Assignment parameters
A11(n,Photo_Images,D_Photo_Images,image_num_P,"Photo _ Images");
% If wanting to test ONLY VTEC_Images, comment this block out (above 2 lines).

% % VTEC_Images introQ
image_num_V = 170; % 170 for the Assignment parameters
A11(n,VTEC_Images,VTEC_Images,image_num_V,"VTEC _ Images");
% % IF wanting to test ONLY Photo_Images, comment this block out (above 2 lines).

function [] = A11(n,Orig_Images,D_Images,image_num,file)
[allArrays,C,F] = TranscribeArray(n,D_Images);

% Qa
avgPatch = mean(allArrays,2);
newAvgPatch = reshape(avgPatch,[n n]);
figure
% imagesc(im2uint8(newAvgPatch))
imagesc(newAvgPatch)
if file == "Photo _ Images"
    colormap(sqrt(gray(256)))
end
title("Mean patch as a " + n + " by " + n + " image, " + file)

% Qb
Xtilda = allArrays-avgPatch;
[E,D,Eall,Dall] = SortedEig(Xtilda,10); % We only need to save Eall here.
eVecWantFirst = Eall(:,1:25);
figure('Position',[300 200 900 600])
for k = 1:size(eVecWantFirst,2)
    patchedEig = reshape(eVecWantFirst(:,k),[n n]);
    subplot(5,5,k);
    % plot(patchedEig)
    imagesc(im2uint8(patchedEig))
    if file == "Photo _ Images"
        colormap(gray)
    end
    title("E-vector " + k)
end
eVecWantLast = Eall(:,end-24:end);
figure('Position',[300 200 900 600])
for l = 1:size(eVecWantLast,2)
```

```
    patchedEig = reshape(eVecWantLast(:,l),[n n]);
    subplot(5,5,l);
    % plot(patchedEig)
    m = l+(n*n)-25;
    imagesc(im2uint8(patchedEig))
    if file == "Photo _ Images"
        colormap(gray)
    end
    title("E-vector " + m)
end

% Qc
%   Photo_Images: COLUMNS / PATCHES 62101-63000 ARE FOR IMAGE 70
des_stop = image_num*C*F;
des_array = des_stop-(C*F)+1:des_stop;
consArrays = allArrays(:,des_array); % X's
consXtilda = Xtilda(:,des_array); % Xtilda's
% consArrays = allArrays(:,62101:63000); % X's
% consXtilda = Xtilda(:,62101:63000);% Xtilda's

% Qc, r=10
r = 10;
[rdimE,rdimD,rdimEall,rdimDall] = SortedEig(consXtilda,r);
remastXhat = GlueItTogether(rdimE,consArrays,avgPatch,C,F,n,Orig_Images,image_num,r,↙
file);
% Qd, r=10
% first r e-vectors sq error: Photo_ 6.8305e+08, VTEC_ 1.4880e+07
% total sq error: Photo_ 6.8347e+08, VTEC_ 1.4880e+07
Errors(avgPatch,remastXhat,rdimEall,consArrays,file,r,rdimE);

% Qc, r=1
r = 1;
[rdimE,rdimD,rdimEall,rdimDall] = SortedEig(consXtilda,r);
remastXhat = GlueItTogether(rdimE,consArrays,avgPatch,C,F,n,Orig_Images,image_num,r,↙
file);
% Qd, r=1
% first r e-vectors sq error: Photo_ 6.8095e+08, VTEC_ 1.4877e+07
% total sq error: Photo_ 6.8347e+08, VTEC_ 1.4880e+07
Errors(avgPatch,remastXhat,rdimEall,consArrays,file,r,rdimE);

% Qc, r=80
r = 80;
[rdimE,rdimD,rdimEall,rdimDall] = SortedEig(consXtilda,r);
remastXhat = GlueItTogether(rdimE,consArrays,avgPatch,C,F,n,Orig_Images,image_num,r,↙
file);
% Qd, r=80
% first r e-vectors sq error: Photo_ 6.8338e+08, VTEC_ 1.4880e+07
% total sq error: Photo_ 6.8347e+08, VTEC_ 1.4880e+07
Errors(avgPatch,remastXhat,rdimEall,consArrays,file,r,rdimE);

end
```

```
function [V,D,Vsrtd,Dsrtd] = SortedEig(XVtilda,r)
    % Here, we sort and return the largest r eigenvalues and
    %  eigenvectors of XVtilda*XVtilda' .
    [Vorig,Dorig] = eig(XVtilda*XVtilda');
    [d,ind] = sort(diag(Dorig),'descend');
    Dsrtd = Dorig(ind,ind);
    Vsrtd = Vorig(ind,ind);
    % V = zeros(n,n);
    % D = zeros(n,n);
    D = Dsrtd(1:r,1:r);
    V = Vsrtd(:,1:r);
end
```

```
function [allArrays,C,F] = TranscribeArray(n,D_Images)
    % Photo_Images introQ
    for a = 1:size(D_Images,3)
        Array{a} = mat2cell(D_Images(:,:,a), ...
            (n*ones(size(D_Images,1)/n,1)'), ...
            (n*ones(size(D_Images,2)/n,1)'));
    end
    % Now, each image (indexed aa) is partitioned into 20x20 pixel patches.

    % size(Array), ans = 1 365
    % size(Array{1,3}), ans = 9 18. The 3 is from 1 to 365, FIRST 1 SAME
    % size(Array{1,3}{4,3}), ans = 20 20
    %  FIRST 1 SAME, 4 is from 1 to 9 (up-down), 3 from 1 to 18 (left-right)
    B = size(Array,2);
    C = size(Array{1,1},1);
    F = size(Array{1,1},2);
    allArrays = zeros(B*C*F,n*n)';
    i = 1;
    for b = 1:B % BB = 365 loaves
        % Array_V{1,bb} , 9x18 array of 20x20 doubles
        for c = 1:C % bb from 1 to 365, ROWS FROM UP TO DOWN, CC*FF = 252
            % Array_V{1,bb}{cc,:} , 1x18 array of 20x20 doubles
            for f = 1:F % ONE-'COLUMNS' FROM LEFT TO RIGHT
                % Array_V{1,bb}{cc,dd} , finally a 20x20 double.
                newArray = reshape(Array{1,b}{c,f},[n*n 1]);
                allArrays(:,i) = newArray;
                i = i+1;
            end
        end
    end
    % Now, each 20x20 'patch' is reshaped into a d=400 dim column vector.
    % allArraysV(:,ii) is one such 400 dimensional column vector.
    % allArraysV is a collection of 59,130 column vectors. PATCHES ARE RESHAPED.
    % column 1 corresponds to Array_V{1,1}{1,1}, 20x20 into 400x1; Image 1/365
    % column 2 corresponds to Array_V{1,1}{1,2}, 20x20 into 400x1; Image 1/365
    % ...
    % column 18 corresponds to Array_V{1,1}{1,18}, Image 1/365
    % column 19 corresponds to Array_V{1,1}{2,1}, Image 1/365
    % ...
    % column 36 corresponds to Array_V{1,1}{2,18}, Image 1/365
    % ...
    % column 162 corresponds to Array_V{1,1}{9,18}, Image 1/365
    %   COLUMNS / PATCHES 1-162 ARE FOR IMAGE 1


    % column 163 corresponds to Array_V{1,2}{1,1}, Image 2/365
    % ...
    % column 324 corresponds to Array_V{1,2}{9,18}, Image 2/365
    %   COLUMNS / PATCHES 163-324 ARE FOR IMAGE 2

    % ...
    % column 27379 corresponds to Array_V{1,170}{1,1}, Image 170/365
    % ...
    % column 27540 corresponds to Array_V{1,170}{9,18}, Image 170/365
    %   COLUMNS / PATCHES 27379-27540 ARE FOR IMAGE 170

    % ...
    % column 58969 corresponds to Array_V{1,365}{1,1}, Image 365/365
    % ...
    % column 59130 corresponds to Array_V{1,365}{9,18}, Image 365/365
    %   COLUMNS / PATCHES 58969-59130 ARE FOR IMAGE 365
    % EACH "PATCH" IS A SAMPLE.
    %
```

```
    %
    % allArraysV is X
end
```

```
function [remastXhat] = GlueItTogether(conE,consArrays,avgPatch,C,F,n,Orig_Images,↙
image_num,r,file);
    projX70tilda1str = conE*((conE'*conE)^(-1))*conE'*consArrays;
    remastXhat = projX70tilda1str + avgPatch;

    gluedPatch = cell(C,F);
    for row = 1:C
        for col = 1:F
            gluedPatch{row,col} = reshape(remastXhat(:,(row-1)*F+col),[n n]);
        end
    end
    wholePatch = cell2mat(gluedPatch);
    figure('Position',[200 300 1175 500])
    subplot(1,2,1);
    imagesc(Orig_Images(:,:,image_num))
    if file == "Photo _ Images"
        colormap(gray)
    end
    title("Original image #" + image_num);
    subplot(1,2,2);
    imagesc(uint8(wholePatch))
    if file == "Photo _ Images"
        colormap(gray)
    end
    title("Re-Patched image #" + image_num + " with r = " + r);
end
```

```
function [] = Errors(avgPatch,remastXhat,rdimEall,consArrays,file,r,rdimE)
    squaredError1str = (norm(avgPatch-remastXhat))^2;
    projX70tildaAll = rdimEall*((rdimEall'*rdimEall)^(-1))*rdimEall'*consArrays;
    remastXhatAll = projX70tildaAll + avgPatch;
    squaredErrorAll = (norm(avgPatch-remastXhatAll))^2;
    percentage = (squaredError1str/squaredErrorAll) * 100;
    disp(newline + "For " + file + ", r=" + r + "," + newline + ...
        "the percentage of total squared error accounted for" + newline + ...
        "by using the first " + r + " eigenvector(s) is about ")
    format long
    disp(percentage)
    disp("percent.")
    whos rdimEall
    whos rdimE
end
```