Image Sampling and Quantization Project Documentation

Elizabeth Chilcoat, Kelle Clark, Michael Heath

September 16, 2020

1 Usage and System Dependencies

1.1 Dependencies

The user should check to see if the following libraries are installed or do a batch install with

```
pip install -r requirements.txt
```

The libraries and versions used by the team include:

```
numpy==1.19.1
opencv_python_headless==4.4.0.40
matplotlib==3.2.2
```

1.2 Usage

To run in the command line:

```
>python image_sample_quant.py [params] dir
```

where optional parameters are:

```
> python  image_sample_quant.py [-h] [--faster bool] dir

optional arguments:
  -h, --help      show this help message and exit
  --faster bool  The intensity change algorithm speed:
  "1" uses a faster implementation
```

and

```
image_sample_quant  name of the executable
default for --faster is 0
dir  input directory
```

The images in the given directory will be unchanged and new images with intensity values $k = 4$ and $k = 6$ will be added with modified names following key input by the user. To each of these images, new enlarged and reduced image files will be created of the original image files using the linear, bi-linear, bi-cubic and nearest neighbor interpolation techniques. If there are originally n image files in the directory provided, at the end of the run the directory will contain an additional 18n image files. Each of these images will have the name of the original file with a postfix that indicates what method applied. For example,

`SmButterflyk6nn_shrink.png`

will be the name of the image file generated from taking SmButterfly.png in the given directory and changing the intensity resolution to k = 6 and then applying the nearest neighbor interpolation method to decrease the size of the image. The program will terminate when the user has used a key strike to implement the methods to all images in the directory.

Below is the command line used to run the program with the default arguments on the test file ImageCorpus4Rest that contains 4 images of differnent spatial resolutions of the same object. This file is in the Team's GitLab Project image-sampling-and-quantization repository.

```
(base)> python image_sample_quant.py ImageCorpus4Res
```

A GUI displays the images in the directory and metadata is written to the command line showing the progress of the system. The output of the above run is included in the Appendix.

The program is event driven and waits on the user to press a key for each transformation. The intensity levels will first be adjusted to k = 6 and to k = 4 for each file in the directory. New files are created for each of these images with appropriately labeled names. Then, again with each key press, the program will apply the interpolation methods to both increase and decrease the spatial resolution to all images in the given directory and create new files each time.

2 User Requirements Log

The design, implementation and testing of the system are based on satisfying the following stated user requirements.

2.1 Project Requirements:

1. Starting with the same object, represent the continuous image of the object using 4 different sensor arrays, I_i where $i = 1, \ldots, 4$, so that each of these pixel arrays uses a different spatial resolution $M_i \times N_i$.

2. For the four pixel matrices I_i and for each entry $I_i(m, n)$, change the quantization scheme using the original $k = 8$ to $k = 4, 6$ giving three different representations, I_{ik}, for $i = 1, \ldots 4$ and $k = 4, 6, 8$.

3. Give qualitative analysis of these 12 images having different spatial and intensity resolution values.

4. Manipulate these 12 images, I_{ik} using the four interpolation techniques to shrink and enlarge images: linear, bilinear, bicubic, and nearest neighbor.

5. Compare and contrast the effectiveness of each of these interpolation methods.

6. Address the computational complexity of the methods used to shrink and enlarge the images.

7. List the steps the team used to complete the project, including testing.

8. Reflect and document the team experience during the project.

3 Software Model, Development Methodology

The team used the integrate and configure model to solve the problems posed and offer solutions. The previous assignment to build an Image Viewer was reconfigured and scaled to add the desired functionality of manipulating the spatial and intensity resolution of images within the given directory. The team's script, image-browser.py, was reconfigured so that file information is printed to the console to aid in comparing multiple files and new resulting files are placed in the given directory. The GUI was slimmed down to only display the image.

4 Implementation

4.1 Obtain Images of The Same Object using Different Spatial Resolutions

To meet the first requirement, the team began with the image of a butterfly take with an iPhone 11 using iOS 13.6.1. and the phone's native camera and the Live Photo setting. The team then used the phone's email application to send the image as a scaled version (small, medium, large, actual size = Xlarge). This scaling preserved the intensity resolution but reduced the number of columns and rows for the image, thus changing the size of the files. This method produced representations of the same object using different spatial resolutions (sampling rate) that the team labeled as XL, L, Md and Sm. The corresponding dimensions for these images are: 1296 pixels $\times$ 1401 pixels as XLarge, as 1184 pixels $\times$ 1280 pixels, 592 pixels $\times$ 640 pixels and 296 pixels $\times$ 320 pixels. In each of these images, the bit depth is 24 implying that the RGB (Red, Green, and Blue) values in each of the three channels uses 2^3 bits and the depth per square inch

(resolution unit is 2 = inch) is 72dpi. It is noted that the files are in RGB format (see Figure 1) before we open the images with the cv2 image reader. The format of the image files is then in BGR. The team chose not to convert this formatting as we did in the previous assignment. These four images (Figures 1 - 4) are found in the file ImageCorpus4Res of the team's project repository.

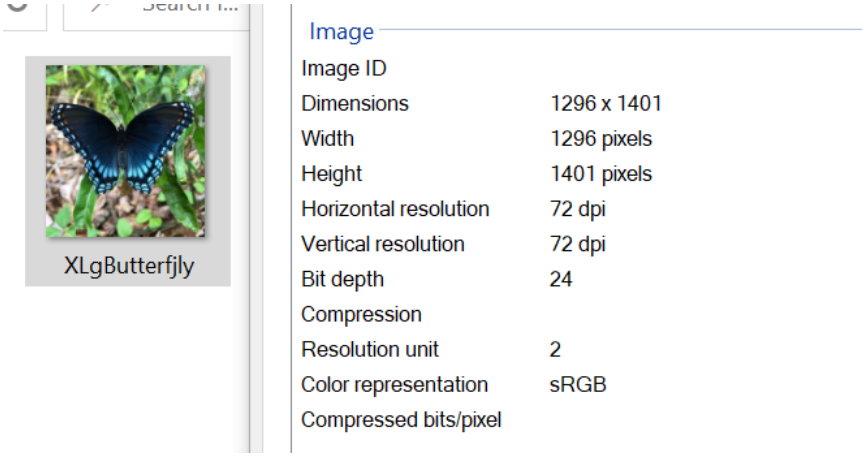


Figure 1: "XLButterfly.jpg File Information"

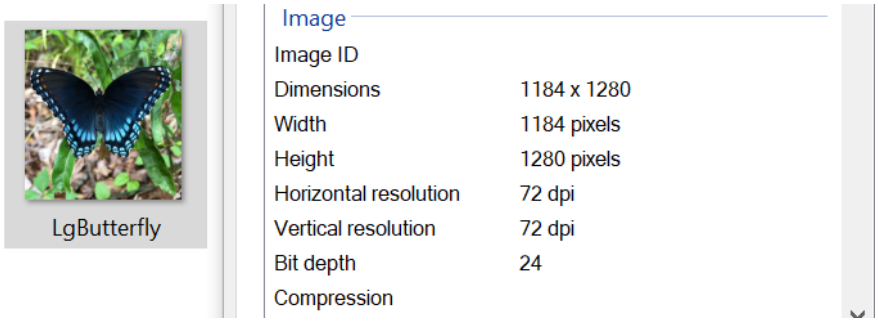


Figure 2: "LButterfly.jpg File Information"

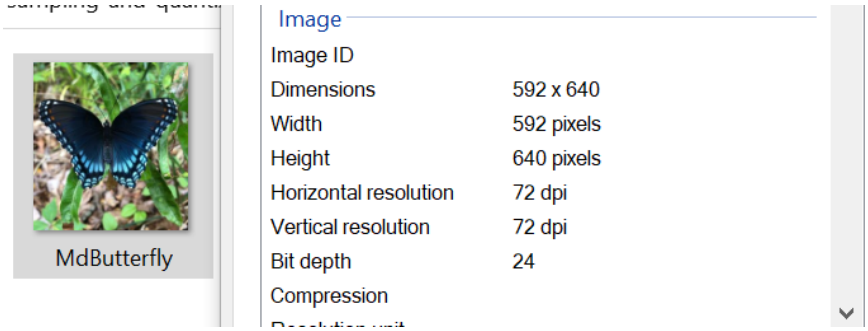


Figure 3: "MdLButterfly.jpg File Information"

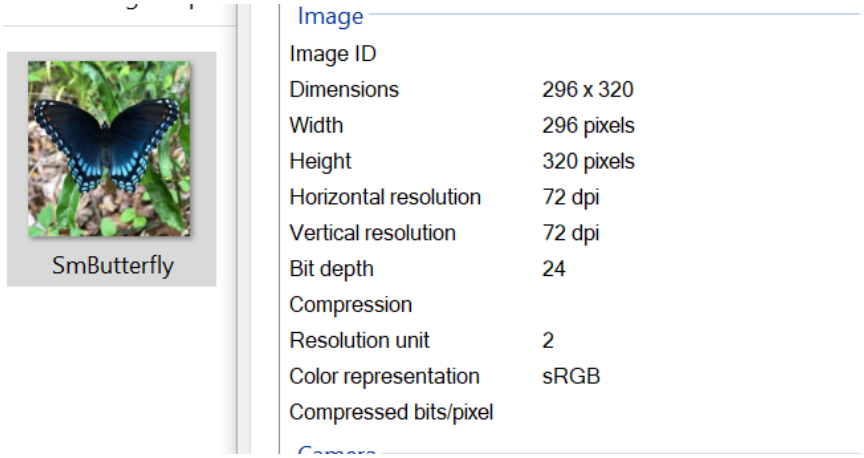


Figure 4: "SmButterfly.jpg File Information"

4.2 Vary the Quantization Scheme

The second requirement to use different intensity resolutions required transformations to be applied to the original images. For the sampling of the continuous function modeling the image as a 2D object, For a color, 2D image the continuous function at each spatial position (x, y) is a 3-dimensional vector $f(x, y)$ where the value represents the 3 channels for red, green and blue (RGB) values. For a color image, the standard quantization scheme is 8 bits, and the intensity of the colors in each of the channels is in $[0, 2^8 - 1]$. Given the 8 bit intensity value for the RGB, the storage for each of these four images ius $M_i(N_i)k$ bits. The team did a quick test of the properties of the files collected in Figures 5 and 6. For the Xlarge (original) image, the M intensity resolution $k = 8$. As noted in the section above, the original color image is modeled with a 3 dimensional vector with 8 bit entries, $f(x, y)$, at each spatial location (x, y) (Figure 7).

To change from one intensity k to another while preserving the ordering of

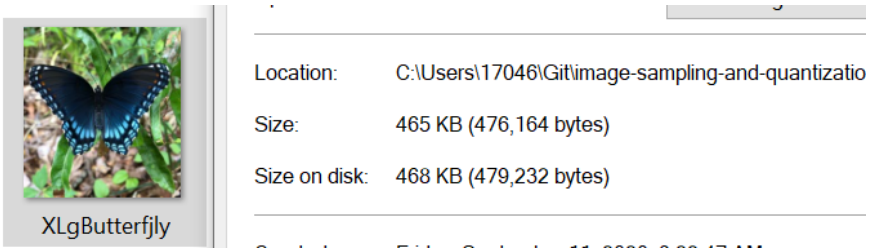


Figure 5: "The Original, XLarge, image required storage"

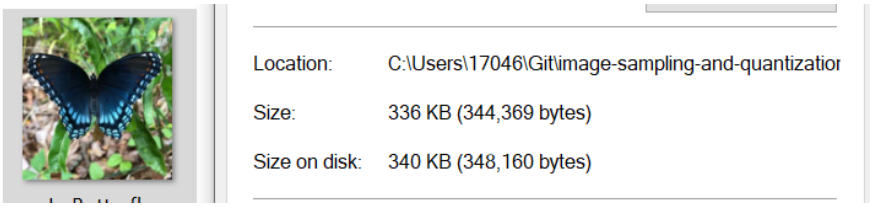


Figure 6: "The Large image required storage"

the set of values, the team used three different implementations a standardizing transformation. Starting with the original color signature in Figure 7, the system applies a transformation to the pixel values in the range of $[0, 255 = 2^8 - 1]$, to the normalized these values to $[0, 1]$.

While this step is not required, it provided an opportunity to visualize the mapping of the colors in a way that preserved all of the original pixel value in each of the BGR channels (see Figure 8). The values were then mapped to $[0, 2^k - 1]$ were the new desired intensity value is k see (Figure 9). Using an integer data type for these values, gives us 2^k possible value for each of the colors but using the floor function to create intervals of values that would map onto the same image value in the interval $[0, 2^k - 1]$ (See Figure 9) The result of this transformation is to literally reduce the possible values used, which in conjunction with a mapping of the standard color display graphics would give us an image with fewer displayed colors but ranging in the same intensities as using 8 bits. The changed number of colors in the final image can be best visualized by comparing the first graphic of the original image in Figure 7 to the graphic of the displayed image with changed k in Figure 10.

Simply using $[0, 2^k - 1]$ values renders a darker image (Figure 11) which is not the desired effect. The darkened image is only using values in the darker hues of Blue, Green and Red before making the final association between the new 2^6 values to the entire standard BGR 2^8 color representation. Taking these 2^6 values to $[0, 2^8 - 1]$, takes any pixel Blue value of in the set $\{0, 1, 2, 3\}$ to a new rendered value of 0. A value in the range of $\{4, 5, 6, 7\}$ in the original image using 2^8, would be translated in the converted image visually as 4. With our mapping of these colors to these classes, We need only 6 bits to represent these

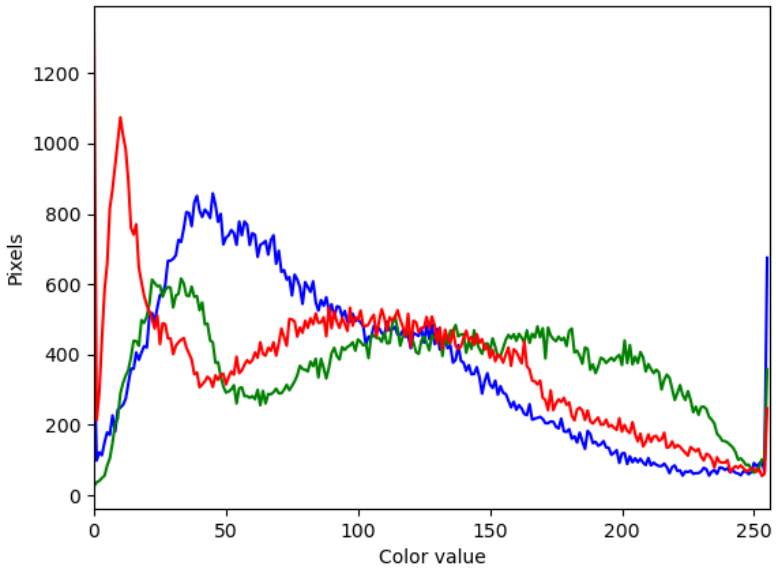


Figure 7: "The Color Signature of the Original Image"

2^6 values in the range $[0, 255]$. These results were verified in the testing of the system with images in ImageTestColorCorpus. These results are included in the Testing Section below.

When implementing the functionality of changing the intensity representation for an image, we have a option for the user to execute this change using a high level and direct operation (–fast 1) that only involves a single transformation and a slower (–fast 0) implementation that produces illustrative intermediate steps.

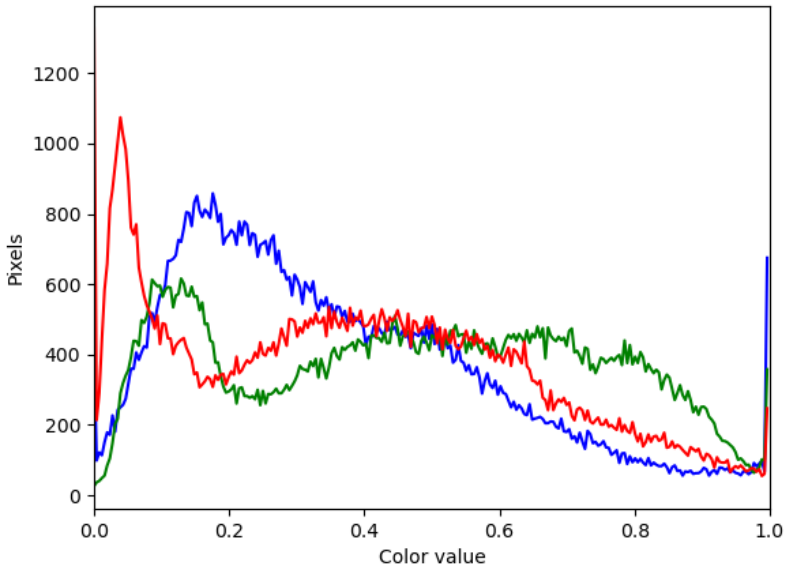


Figure 8: "The Color Signature of the Standardized Image"

The resulting images using different intensity value for the and the BGR distrbutions for these renderings are presented in (Figures 12 - 15):

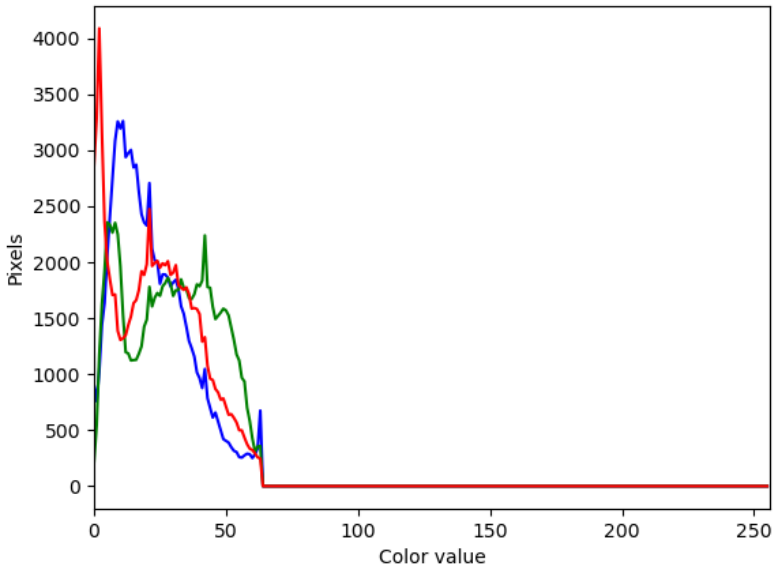


Figure 9: "The Color Signature of the 2nd Transformation on the Image"

4.3 Qualitative Analysis of Images

Visually, it seems that the images with higher spatial resolutions have greater detail than those with lower spatial resolutions. Comparing the large image, say LgButterfly.jpg, to an enlarged version of the medium image, MdButterflybicubic-increase, reveals the difference in details rendered. All of the different interpolations used to enlarge the medium image, while visually very different, had nearly the same amount of detail. In contrast, the large image shows more fine details that are not present in the medium image.

Varying the intensity resolution also has an effect on the level of visual detail. The original images have a k value of 8. Lowering k to 6 produces a grainy images with slight color banding, especially in the butterfly's wings. The grass and leaves do not have as much variation in color, so the effect is less noticeable. At $k = 4$, the images looks even worse. The color bands are now large and clearly distinguishable, with sharp, pixelated edges.

4.4 Interpolation Techniques

The simplest way to transform an image from size $M \times N$ to say $\frac{M}{2} \times \frac{N}{2}$ is to remove every other column and every other row. This does not retain the quality of the image, in fact if the image has relatively small dimensions the image is very distorted. The same can be said if the image were to be enlarged

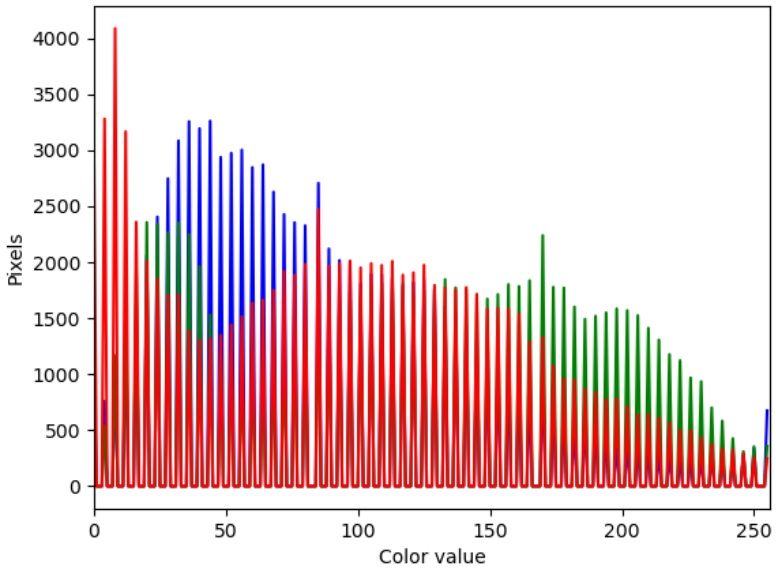


Figure 10: "The Color Signature of the Final Image"

to a size $2M \times 2N$.

The interpolation techniques studied in class range in complexity from the linear to the bicubic in terms of the number of operations required to "fill in" spatial positions that are created when enlarging an image or to "collapse" to fewer positions. The bicubic interpolation method requires the determination of 16 different coefficients using 8 different points, the bilinear uses four points to determine four weights using four points, while the linear only requires calculating two weights for two points. As usual, one expects that with greater cost of complexity comes a greater benefit in the smoothness and fit of the approximation.

The bilinear, bicubic and nearest neighbor interpolation methods are apart of the ImageTk library. These methods take in parameters image and a value such at 0.5 or 2 and shrink or increase the image respectively. These three interpolation techniques were implemented and the fourth, linear, was developed by the team. Each of the four original images from the Image4ResCorpus and the newly created images with k = 4 and 6 were both doubled and halved in size by these methods. The collection of these images is located in the repository, but a representative image is here for illustration and discussion. The team uses MdButterfly.png allowing for both halving and doubling of the spatial resolution to be more easily viewed on all team members view-ports. For comparison, the original image is in Figure 16:

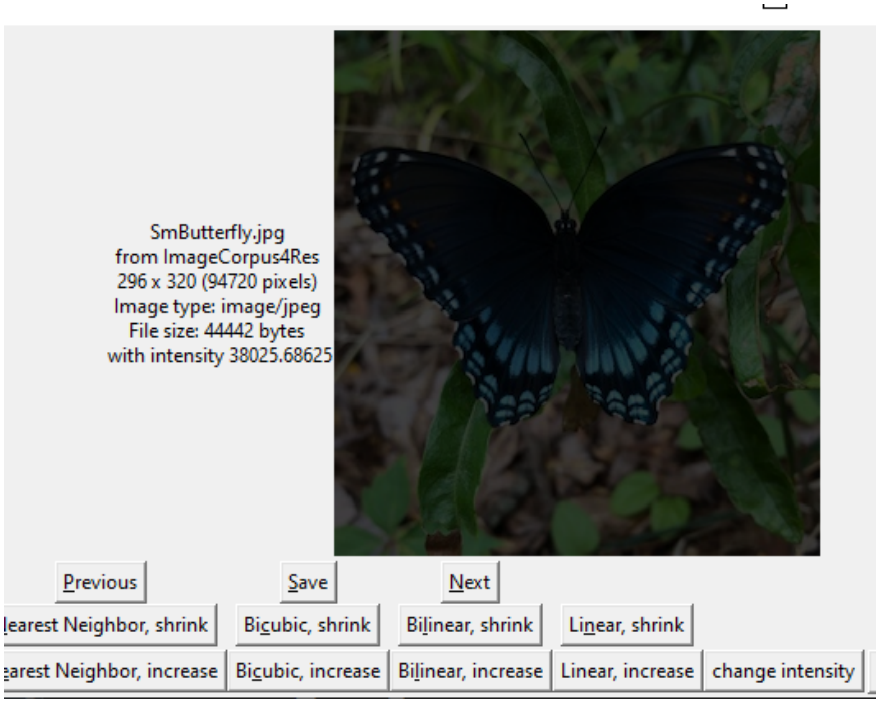


Figure 11: "The Darkened Image before the Last Translation"

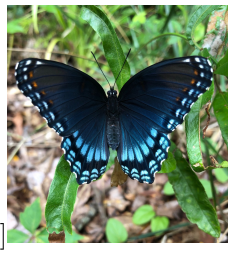
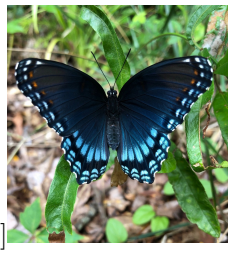
[a]  [b]  [c] 

Figure 12: (a) XL,k = 8 (b) XL,k = 6 (c) XL, k = 4

Figures 17 - 20 are the enlargements of this Medium image using the interpolation methods:

[a]  [b]  [c] 

Figure 13: (a) ,Lg = 8 (b) Lg,k = 6 (c) Lg, k = 4

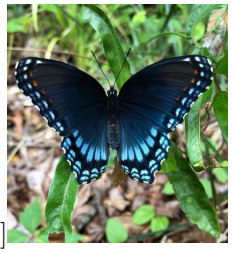
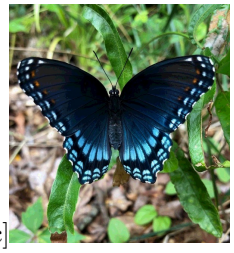
[a]  [b]  [c] 

Figure 14: (a) M,k = 8 (b) M,k = 6 (c) M, k = 4

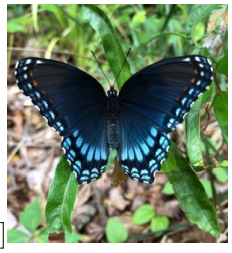
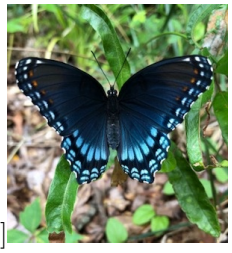
[a]  [b]  [c] 

Figure 15: (a) Sm,k = 8 (b) Sm,k = 6 (c) Sm, k = 4

Figures 21 - 24 are the reduction of the image using the interpolation methods:

Figure 16: "The Medium Butterfly Image"

4.5 Compare and Contrast these Interpolation Results

The interpolation techniques used produced widely varying results. The nearest neighbor technique produced a very "blocky" image. A good example that is also visually interesting of this block phenomenon can be seen by comparing the transformed image in Figures 25

Figure 17: "Increase with Linear Interpolation"

to one of the team's earlier images produced used as the original in Figure 26.

Figure 18: "Increase with Bilinear Interpolation"

The other techniques were all much smoother than nearest neighbor, with smoothness increasing from linear, to bilinear, and finally to bicubic giving the best results. The difference between the techniques was noticeable in the shrunken images, but was much more pronounced in the enlarged images. The linear interpolation of the enlarged image produced an interesting effect: the butterfly's left antenna was more pixelated than the right. This was due to averaging the top right and bottom left neighbors of a pixel not in the same row or column as an existing pixel.

4.6 The Complexity of the Interpolation Results

The complexity of the methods increases with the number of adjacent pixels used in the interpolation process and with each increase in complexity, there is

Figure 19: "Increase with Bicubic Interpolation"

Figure 20: "Increase with Nearest Neighbor"

theoretically better approximation and visual continuity of representation and quality of the image. The functions enlargelinear() and shrinklinear use linear interpolation to estimate the value of the pixels. In the case where the image is doubled, the linear interpolation would take a 2 by 2 section of four pixels of the image and can create a 3 by 3 section of 9 pixels by copying values to the four corners of the new section then averaging along the rows and columns and along the off-main or main diagonal for the remaining elements. Figures 27 and 28 below illustrate how the some of the new pixels may look after using linear interpolation to increase the size of a section in the array versus using

Figure 21: "Reduce with Linear Interpolation"

bilinear interpolation. The bilinear interpolation would require twice as many computations and the bicubic would involve all the points around the center resulting in greater complexity.

Figure 22: "Reduction with BiLinear Interpolation"

Because all of the values to be used to construct approximation lie on a square lattice, each of the above interpolation techniques is equivalent to a nearest neighbor algorithm with either k = 2, 4 or 8.

The method used by the team uses two nested for loops to create the new image. Despite this method having the smallest operation complexity, the implementation and algorithm design leaves this method taking longer to run than the efficient library functions with a greater number of computations to complete. The are also still some errors in the linear shrink function leaving vertical lines of grey/black that have not been resolved.

The time for each computation was measure on a X1 Carbon PC with Core i5 processor. The average time in seconds for all images is used as the time for the algorithm. Note: nn = nearest neighbor method. The command line output of

Figure 23: "Reduction with BiCubic Interpolation"

the call to the program that generated these results is included in the Appendix. It is surprising to see that the bilinear had the smallest average time, beating the bicubic which requires more computations. The linear should have been faster than both the bilinear and bicubic since it requires fewer computations (equations involved), however the linear enlarge and linear reduce algorithms were not optimized for time or memory. These functions were extremely slow compared to the high-level included library functions. The nearest neighbor algorithm is the fastest, as expected, since it requires the fewest operations but is not expected to result in a very high quality approximation to the function values that the method is interpolating.

Figure 24: "Reduction with Nearest Neighbor Interpolation "

5 Testing

5.1 Test Objective: Given a call using the help flag, -h, what is the output?

Test ran:

```
> python  image_sample_quant.py -h
```

Test result:

```
usage: image_sample_quant.py [-h] [--faster bool] dir

Image Manipulation v1.0
```

Figure 25: "Reduction with Nearest Neighbor Interpolation "

```
positional arguments:
  dir             The root directory to view photos in

optional arguments:
  -h, --help      show this help message and exit
  --faster bool   The color algorithm speed: "1" uses a faster implementation
```

Figure 26: "Reduction with Nearest Neighbor Interpolation "

5.2 Test Objective: Determine what happens at the end case where an image has only a two values in a channel, is all white or has a large variance in a channel with many values?.

Test ran

```
> python  image_sample_quantKelle.py --faster 1 TestColorCorpus
```

The TestColorCorpus file contains four images. The values in the channels were output to the screen using the above script that contained added print statements for this purpose. Test result:

```
The Blue Test image (predominately blue, only two values in Blue channel):
```

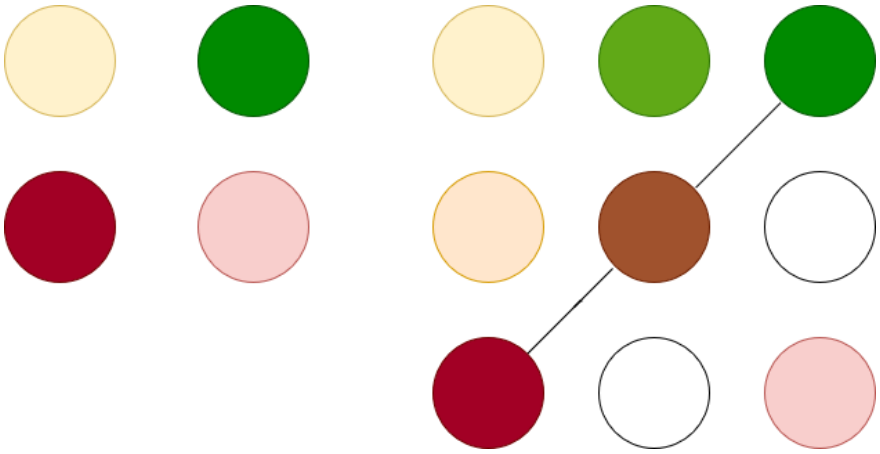


Figure 27: "Using Linear Interpolation to Estimate the New Center Pixel"

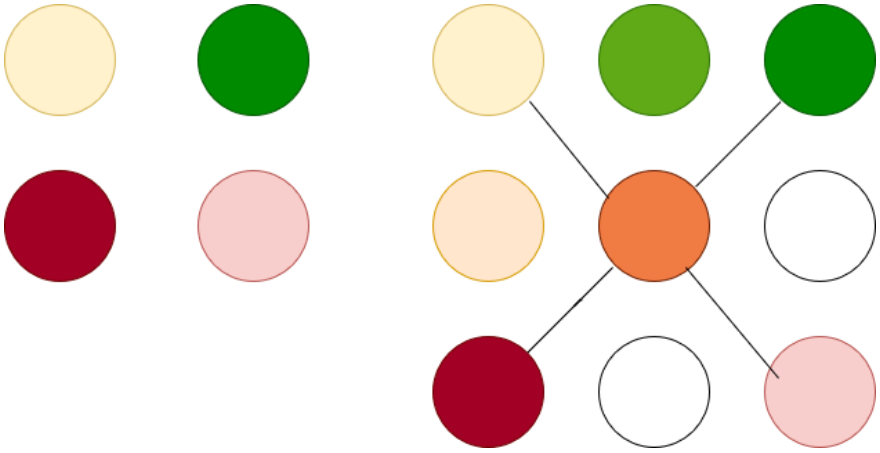


Figure 28: "Using Biinear Interpolation to Estimate the New Center Pixel"

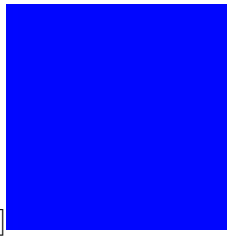
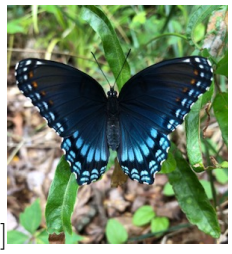
```
the original max red is 1 and the original min red is 1
the original max green is 6 and the original min green is 6
the original max blue is 255 and the original min blue is 254

The first transformation (standardize):
the normalized_max red is 0.00392156862745098
and the normalized_min red is 0.00392156862745098
the normalized_max green is 0.023529411764705882
and the normalized_min green is 0.023529411764705882
the normalized_mmx blue is 1.0
and the normalized_min blue is 0.996078431372549
```

avg. time	nn	linear	bilinear	bicubic
	0.02	0.38	0.04	0.03

Table 1: Enlarge Image

avg. time	nn	linear	bilinear	bicubic
	0.16	1.64	0.06	0.03

Table 2: Reduce Image

[b]  [a]  [c] 

[d]

Figure 29: (a) Butterfly (b) Blue (c) Psycadelic (d) White

```
The second transformation (to only 2^6 values):
the max changed_red is 0
and the min changed_red is 0
the max changed_green is 1
and the min changed_green is 1
the max changed_blue is 63
and the min changed_blue is 62

The third transformation (to 2^6 values in the range [0, 2^8]):
the max clr corrected red is 0
and the min clr corrected red is 0
the max clr corrected green is 4
and the min clr corrected green is 4
the max corrected blue is 255
```

```
and the min clr corrected blue is 250

The PyschoButterfly image with high variation among value in each channel:
the original max red is 255
and the original min red is 0
the original max green is 252
and the original min green is 0
the original max blue is 255
and the original min blue is 0

The first transformation (standardize):
the normalized_max red is 1.0
and the normalized_min red is 0.0
the normalized_max green is 0.9882352941176471
and the normalized_min green is 0.0
the normalized_mmx blue is 1.0
and the normalized_min blue is 0.0

The second transformation (to only 2^6 values):
the max changed_red is 63
and the min changed_red is 0
the max changed_green is 62
and the min changed_green is 0
the max changed_blue is 63
and the min changed_blue is 0

The third transformation (to 2^6 values in the range [0, 2^8]):
the max clr corrected red is 255
and the min clr corrected red is 0
the max clr corrected green is 250
and the min clr corrected green is 0
the max corrected blue is 255
and the min clr corrected blue is 0

For the White image:
the original max red is 255
and the original min red is 255
the original max green is 255
and the original min green is 255
the original max blue is 255
and the original min blue is 255

The first transformation (standardize):
the normalized_max red is 1.0
and the normalized_min red is 1.0
the normalized_max green is 1.0
```

```
and the normalized_min green is 1.0
the normalized_mmx blue is 1.0
and the normalized_min blue is 1.0

The second transformation (to only 2^6 values):
the max changed_red is 63
and the min changed_red is 63
the max changed_green is 63
and the min changed_green is 63
the max changed_blue is 63
and the min changed_blue is 63

The third transformation (to 2^6 values in the range [0, 2^8]):
the max clr corrected red is 255
and the min clr corrected red is 255
the max clr corrected green is 255
and the min clr corrected green is 255
the max corrected blue is 255
and the min clr corrected blue is 255
```

6 Reflecting on the learning experience and teamwork

Reflect on your solution to the problem and the learning experience through this project. Trust building, cohesion, and psychological safety are the foundations elements of teamwork. Reflect on the team dynamic experienced in this project.

6.1 Team member contribution/effort assessment

The following rubric is used by the members of the team in order to rate themselves and their teams members on a scale of 1 to 5 about their individual contribution to the project (a rating of 1 being poor and 5 being outstanding). Rationale should be provided for each rating. These documents are separate.

Table 3: Team Assessment.

	Self-assessment	**Team member 1**	**Team member 2**
attended meetings			
communicated			
participated			
contributed			
provided feedback			
positive attitude			

The rubric for scoring is given in the following image:

	Good	Fair	Poor
Conformance to Specifications (40 points)	The program works correctly and meets all of the specifications. (40 points).	The program works correctly but implements less than 50% of the specifications. (30 points)	The program produces incorrect results or does not compile. (10 points)
Data Structures and Algorithms (20 points)	Appropriate and efficient data structures and algorithms are used. (20 points)	The data structures and algorithms used get the job done but they are neither a natural fit nor efficient. (10 points)	Solution is based on brute force approach. No consideration is given to selection of suitable data structures and algorithms. (5 points)
Testing (20 points)	Coverage of test cases is comprehensive. Following information is provided for test cases: inputs, expected results, pass/fail, and remarks.(20 points)	Coverage of test cases is low. Test case documentation is incomplete. (10 points)	Cursory treatment of testing. No documentation of test cases. (5 points)
Coding (10 points)	The code is compact without sacrificing readability and understandability. (10 points)	The code is fairly compact without sacrificing readability and understandability. (5 points)	The code is brute force and unnecessarily long. (2 points)
Readability (5 points)	The code is well organized and very easy to follow. (5 points)	The code is readable only by someone who knows what it is supposed to be doing. (3 points)	The code is poorly organized and very difficult to read. (1 points)
Documentation (5 points)	The code is self-documenting and obviates the need for elaborate documentation. It is well written and clearly explains what the code is accomplishing and how. (5 points)	The documentation is simply comments embedded in the code with some simple header comments separating functions/methods. (3 points)	The documentation is simply comments embedded in the code and does not help the reader understand the code. (2 points)

Figure 30: "Team member assessment rubric"

6.2 Appendix

A sample run of the system:

```
(base) PS C:\Users\17046\Git\image-sampling-and-quantization> python image_sampl
First, lower intensity to see the effect of intensity resolution
INITAL IMAGE: ImageCorpus4Res/LgButterfly.jpg
k=4
k=6
INITAL IMAGE: ImageCorpus4Res/MdButterfly.jpg
k=4
k=6
INITAL IMAGE: ImageCorpus4Res/SmButterfly.jpg
k=4
k=6
INITAL IMAGE: ImageCorpus4Res/XLgButterfjly.jpeg
k=4
k=6
Next, edit spacial resolution
INITAL IMAGE: ImageCorpus4Res/LgButterfly.jpg
Linear shrink
Shrinking using linear took:  2.5891354084014893  seconds
Linear increase
Shrinking using linear took:  0.8527224222819011  seconds
NN shrink
Shrinking using Nearest Neighbor took:  0.04472419818242391
seconds
NN increase
Enlarging using Nearest Neighbors took:  0.04209882418314616
seconds
Bicubic shrink
Shrinking using bicubic took:  0.04883146286010742  seconds
Bicubic increase
Enlarging using Bicubic took:  0.021650667985280356  seconds
Bilinear shrink
Shrinking using bilinear took:  0.01960525115331014  seconds
Bilinear increase
Enlarging using Bilinear took:  0.02164525588353475  seconds
INITAL IMAGE: ImageCorpus4Res/LgButterflyk4.jpg
Linear shrink
Shrinking using linear took:  2.4286069869995117  seconds
Linear increase
Shrinking using linear took:  0.4972434639930725  seconds
NN shrink
Shrinking using Nearest Neighbor took:  0.0004105130831400553
seconds
NN increase
```

```
Enlarging using Nearest Neighbors took:  0.003070688247680664
seconds
Bicubic shrink
Shrinking using bicubic took:  0.0003747820854187012  seconds
Bicubic increase
Enlarging using Bicubic took:  0.0042016943295796715  seconds
Bilinear shrink
Shrinking using bilinear took:  0.03707644542058309  seconds
Bilinear increase
Enlarging using Bilinear took:  0.033587602774302165  seconds
INITAL IMAGE: ImageCorpus4Res/LgButterflyk6.jpg
Linear shrink
Shrinking using linear took:  2.5055034160614014  seconds
Linear increase
Shrinking using linear took:  0.4989607135454814  seconds
NN shrink
Shrinking using Nearest Neighbor took:  0.0003823121388753255
seconds
NN increase
Enlarging using Nearest Neighbors took:  0.0031541864077250163
seconds
Bicubic shrink
Shrinking using bicubic took:  0.00036626259485880535  seconds
Bicubic increase
Enlarging using Bicubic took:  0.09292356967926026  seconds
Bilinear shrink
Shrinking using bilinear took:  0.03389294147491455  seconds
Bilinear increase
Enlarging using Bilinear took:  0.052468891938527426  seconds
INITAL IMAGE: ImageCorpus4Res/MdButterfly.jpg
Linear shrink
Shrinking using linear took:  0.8479800224304199  seconds
Linear increase
Shrinking using linear took:  0.16222028732299804  seconds
NN shrink
Shrinking using Nearest Neighbor took:  0.0774968942006429
seconds
NN increase
Enlarging using Nearest Neighbors took:  0.03220831155776978
seconds
Bicubic shrink
Shrinking using bicubic took:  0.0340355118115743  seconds
Bicubic increase
Enlarging using Bicubic took:  0.044381606578826 91  seconds
Bilinear shrink
Shrinking using bilinear took:  0.16967305739720662  seconds
```

```
Bilinear increase
Enlarging using Bilinear took:  0.04261620044708252  seconds
INITAL IMAGE: ImageCorpus4Res/MdButterflyk4.jpg
Linear shrink
Shrinking using linear took:  0.8549058437347412  seconds
Linear increase
Shrinking using linear took:  0.16132032871246338  seconds
NN shrink
Shrinking using Nearest Neighbor took:  0.02875842253367106
seconds
NN increase
Enlarging using Nearest Neighbors took:  0.030717945098876952
seconds
Bicubic shrink
Shrinking using bicubic took:  0.029998222986857098  seconds
Bicubic increase
Enlarging using Bicubic took:  0.031226901213328044  seconds
Bilinear shrink
Shrinking using bilinear took:  0.028605687618255615  seconds
Bilinear increase
Enlarging using Bilinear took:  0.03349769910176595  seconds
INITAL IMAGE: ImageCorpus4Res/MdButterflyk6.jpg
Linear shrink
Shrinking using linear took:  0.7197098731994629  seconds
Linear increase
Shrinking using linear took:  0.1272963245709737  seconds
NN shrink
Shrinking using Nearest Neighbor took:  0.00018289486567179362
seconds
NN increase
Enlarging using Nearest Neighbors took:  0.0171304980913798
seconds
Bicubic shrink
Shrinking using bicubic took:  0.01404887040456136  seconds
Bicubic increase
Enlarging using Bicubic took:  0.02211603323618571  seconds
Bilinear shrink
Shrinking using bilinear took:  0.028836270173390705  seconds
Bilinear increase
Enlarging using Bilinear took:  0.030486126740773518  seconds
INITAL IMAGE: ImageCorpus4Res/SmButterfly.jpg
Linear shrink
Shrinking using linear took:  0.22566938400268555  seconds
Linear increase
Shrinking using linear took:  0.033559183279673256  seconds
NN shrink
```

```
Shrinking using Nearest Neighbor took:  0.03644677003224691
seconds
NN increase
Enlarging using Nearest Neighbors took:  0.03240679502487183
seconds
Bicubic shrink
Shrinking using bicubic took:  0.14779818058013916  seconds
Bicubic increase
Enlarging using Bicubic took:  0.02645655075709025  seconds
Bilinear shrink
Shrinking using bilinear took:  0.0387883186340332  seconds
Bilinear increase
Enlarging using Bilinear took:  0.0290617307027181  seconds
INITAL IMAGE: ImageCorpus4Res/SmButterflyk4.jpg
Linear shrink
Shrinking using linear took:  0.23375940322875977  seconds
Linear increase
Shrinking using linear took:  0.03422480821609497  seconds
NN shrink
Shrinking using Nearest Neighbor took:  0.0078858216603597
seconds
NN increase
Enlarging using Nearest Neighbors took:  0.012282506624857584
seconds
Bicubic shrink
Shrinking using bicubic took:  0.0102721373240153  seconds
Bicubic increase
Enlarging using Bicubic took:  0.009168008963267008  seconds
Bilinear shrink
Shrinking using bilinear took:  0.009338951110839844  seconds
Bilinear increase
Enlarging using Bilinear took:  0.008974925676981608  seconds
INITAL IMAGE: ImageCorpus4Res/SmButterflyk6.jpg
Linear shrink
Shrinking using linear took:  0.3426778316497803  seconds
Linear increase
Shrinking using linear took:  0.03449602127075195  seconds
NN shrink
Shrinking using Nearest Neighbor took:  0.033667985598246256
seconds
NN increase
Enlarging using Nearest Neighbors took:  0.0146270751953125
seconds
Bicubic shrink
Shrinking using bicubic took:  0.01744114557902018  seconds
Bicubic increase
```

```
Enlarging using Bicubic took:  0.01712027390797933  seconds
Bilinear shrink
Shrinking using bilinear took:  0.014067153135935465  seconds
Bilinear increase
Enlarging using Bilinear took:  0.013644647598266602  seconds
INITAL IMAGE: ImageCorpus4Res/XLgButterfjly.jpeg
Linear shrink
Shrinking using linear took:  2.9236817359924316  seconds
Linear increase
Shrinking using linear took:  0.6007621010144552  seconds
NN shrink
Shrinking using Nearest Neighbor took:  0.0011760234832763673
seconds
NN increase
Enlarging using Nearest Neighbors took:  0.04698765277862549
seconds
Bicubic shrink
Shrinking using bicubic took:  0.0519892414410909  seconds
Bicubic increase
Enlarging using Bicubic took:  0.06497762600580852  seconds
Bilinear shrink
Shrinking using bilinear took:  0.04236080646514893  seconds
Bilinear increase
Enlarging using Bilinear took:  0.04403071800867717  seconds
INITAL IMAGE: ImageCorpus4Res/XLgButterfjly.jpek4.jpg
Linear shrink
Shrinking using linear took:  2.98809814453125  seconds
Linear increase
Shrinking using linear took:  0.5988335927327474  seconds
NN shrink
Shrinking using Nearest Neighbor took:  0.0005058924357096354
seconds
NN increase
Enlarging using Nearest Neighbors took:  0.0037890116373697916
seconds
Bicubic shrink
Shrinking using bicubic took:  0.000560295581817627  seconds
Bicubic increase
Enlarging using Bicubic took:  12.96173487106959  seconds
Bilinear shrink
Shrinking using bilinear took:  0.2425095796585083  seconds
Bilinear increase
Enlarging using Bilinear took:  0.11151620546976725  seconds
INITAL IMAGE: ImageCorpus4Res/XLgButterfjly.jpek6.jpg
Linear shrink
Shrinking using linear took:  3.022336006164551  seconds
```

```
Linear increase
Shrinking using linear took:  0.9879124124844869  seconds
NN shrink
Shrinking using Nearest Neighbor took:  1.7283301989237467
seconds
NN increase
Enlarging using Nearest Neighbors took:  0.045692157745361325
seconds
Bicubic shrink
Shrinking using bicubic took:  0.026384270191192626  seconds
Bicubic increase
Enlarging using Bicubic took:  0.024783825874328612  seconds
Bilinear shrink
Shrinking using bilinear took:  0.025062660376230877  seconds
Bilinear increase
Enlarging using Bilinear took:  0.031006880601247153  seconds
end
```