ECE 253 HW1

A53280596 Yawen Zhao

Academic Integrity Policy: Integrity of scholarship is essential for an academic commu- nity. The University expects that both faculty and students will honor this principle and in so doing protect the validity of University intellectual work. For students, this means that all academic work will be done by the individual to whom it is assigned, without unauthorized aid of any kind. By including this in my report, I agree to abide by the Academic Integrity Policy men- tioned above.

Problem 1

In [15]:
```
import numpy as np

A = np.array([[3, 9, 5, 1],
              [4, 25, 4, 3],
              [63, 13, 23, 9],
              [6, 32, 77, 0],
              [12, 8, 6, 1]])

B = np.array([[0, 1, 0, 1],
              [0, 1, 1, 0],
              [0, 0, 0, 1],
              [1, 1, 0, 1],
              [0, 1, 0, 0]])
```

In [16]:
```
C = A * B
print(C)
```

```
[[ 0  9  0  1]
 [ 0 25  4  0]
 [ 0  0  0  9]
 [ 6 32  0  0]
 [ 0  8  0  0]]
```

(i) Pointwise multiplication of A and B get C as above

In [17]:
```
np.dot(C[1,:], C[3,:])
```

Out[17]: 800

(ii) The inner product of the 2nd and 4th row of C is 800

In [18]:
```
minn = np.min(C)
maxn = np.max(C)
```

In [19]:
```
print(minn, maxn)
```

```
0 32
```

In [20]:
```
minindex = np.where(C == minn)
maxindex = np.where(C == maxn)
print(minindex)
print(maxindex)
```

```
(array([0, 0, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4]), array([0, 2, 0, 3, 0, 1, 2,
2, 3, 0, 2, 3]))
(array([3]), array([1]))
```

(iii) As above, the corresponding row and column indices of min values in C are shown as above, which shows the positions of min values in C are [0,0],[0,2],[1,0],[1,3],[2,0],[2,1],[2,2],[3,2],[3,3],[4,0], [4,2],[4,3]. And the row and column indice of max value is also shown as above, which shows the position of max value is [3,1].

Problem 2

(i) The color image is shown below.

(ii) The gray image B is shown below. As the min value of the gray-scale is 8 and the max value of it is 252, the values are certainly between 0 and 255. And from the transformation formula, we can also indicate that: As for value of R, G, B are between 0 and 255, then

$$0 < 0.299 * R + 0.587 * G + 0.144 * B < (0.299 + 0.587 + 0.144) * 255 = 255$$

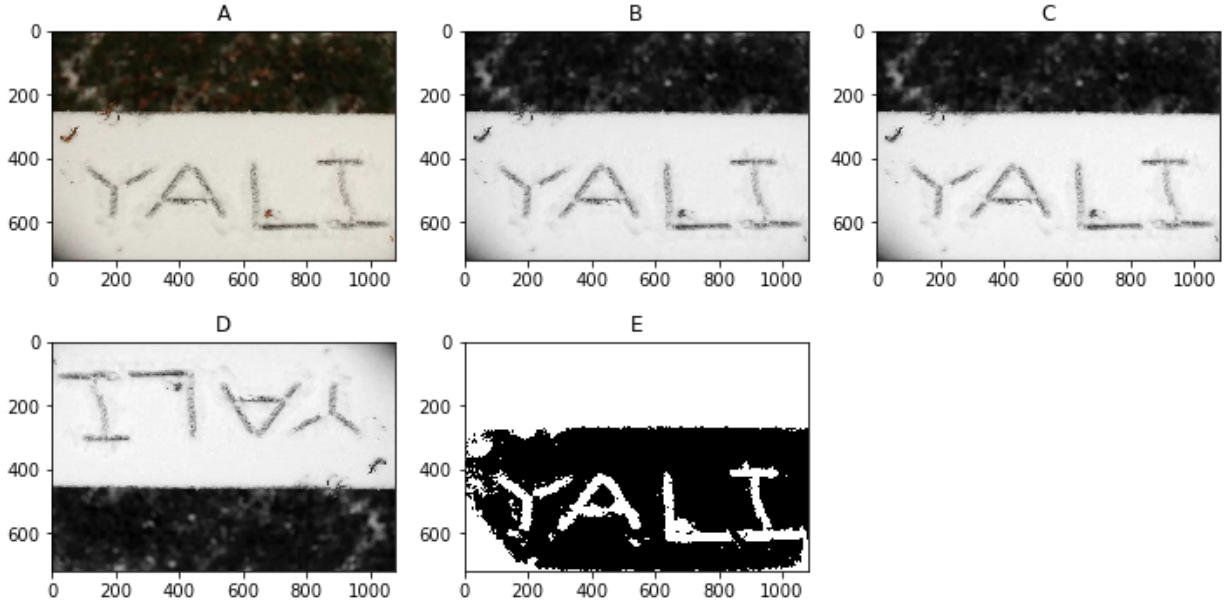
(iii) The image of C is shown below.

(iv) The flipped image D is shown as below.

(v) The binary image E is shown as below.

In [14]:
```
import numpy as np
import imageio
import matplotlib.pyplot as plt

A = imageio.imread('111.jpg')

def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.144]).astype(int)

B = rgb2gray(A)
minn = np.min(B)
maxn = np.max(B)
print("min of B is", minn, ". max of B is", maxn)

C = B + 15
C[C > 255] = 255

D = np.flip(np.flip(C, 0),1)

median = np.median(B)
E = np.zeros(B.shape)
small = B <= median
large = B > median
E[small] = 1
E[large] = 0

f1 = plt.figure(figsize=(12,6))
f1_ax1 = f1.add_subplot(231)
f1_ax2 = f1.add_subplot(232)
f1_ax3 = f1.add_subplot(233)
f1_ax4 = f1.add_subplot(234)
f1_ax5 = f1.add_subplot(235)

f1_ax1.imshow(A)
f1_ax1.title.set_text("A")
f1_ax2.imshow(B, cmap = plt.get_cmap('gray'))
f1_ax2.title.set_text("B")
f1_ax3.imshow(C, cmap = plt.get_cmap('gray'))
f1_ax3.title.set_text("C")
f1_ax4.imshow(D, cmap = plt.get_cmap('gray'))
f1_ax4.title.set_text("D")
f1_ax5.imshow(E, cmap = plt.get_cmap('gray'))
f1_ax5.title.set_text("E")
plt.show()
```

```
min of B is 8 . max of B is 247
```

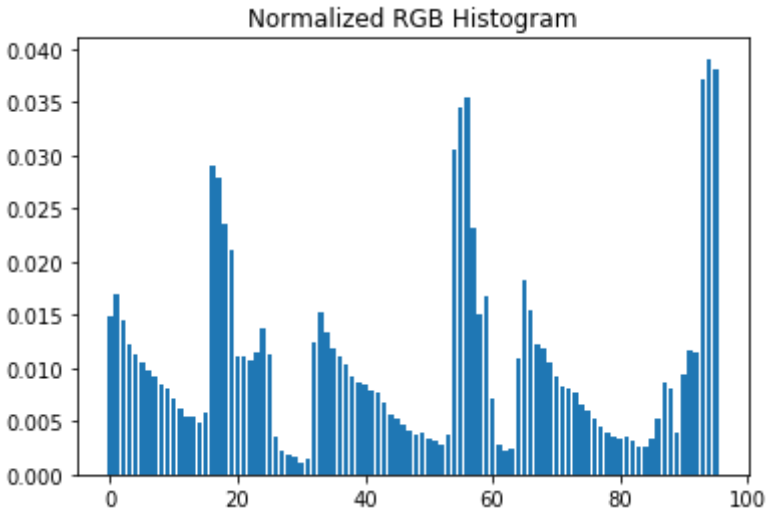


Problem 3

The histogram of the normalized rgb histogram is shown as below

In [8]:
```
def compute_norm_rgb_histogram(A):
    R = []
    G = []
    B = []
    RA = A[:,:,0]
    GA = A[:,:,1]
    BA = A[:,:,2]
    for i in range(32):
        Rnum = RA[RA >= i * 8]
        Rnum = Rnum[Rnum <= (i+1) * 8 -1]
        R.append(len(Rnum))

        Gnum = GA[GA >= i * 8]
        Gnum = Gnum[Gnum <= (i+1) * 8 -1]
        G.append(len(Gnum))

        Bnum = BA[BA >= i * 8]
        Bnum = Bnum[Bnum <= (i+1) * 8 -1]
        B.append(len(Bnum))
    RGB = R + G + B
    sumRGB = sum(RGB)
    RGB = [x / sumRGB for x in RGB]

    return RGB
```

In [9]:
```
image = imageio.imread('geisel.jpg',pilmode="RGB")
RGB = compute_norm_rgb_histogram(image)
plt.bar(range(0, 96), RGB)
plt.title("Normalized RGB Histogram")
plt.show()
print("Sum of the histogram is",sum(RGB))
```



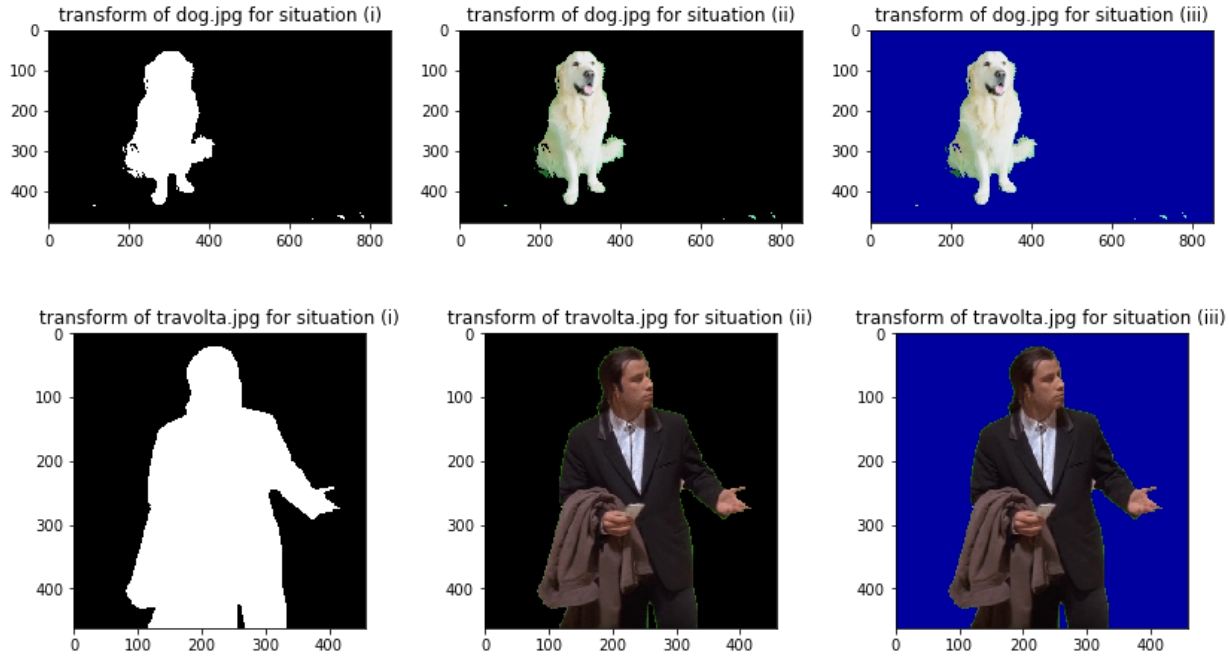

```
Sum of the histogram is 1.0
```

Problem 4

The result for (i), (ii) and (iii) for dog.jpg and travolta.jpg are shown as above.

In [10]:
```
def imageTrans(img):
    res1 = np.zeros(img.shape[0:2])
    res2 = np.zeros(img.shape)
    res3 = np.zeros(img.shape)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if (img[i][j][1] > 120 and ((img[i][j][0] < 100 and img[i][j][2
                                         or (img[i][j][0] < 180 and img[i][j
                res1[i][j] = 0
                res2[i][j] = [0, 0, 0]
                res3[i][j] = [0, 0, 158]
            else:
                res1[i][j] = 255
                res2[i][j] = img[i][j]
                res3[i][j] = img[i][j]
    return res1.astype(int), res2.astype(int), res3.astype(int)
```

In [11]:
```
dog = imageio.imread('dog.jpg',pilmode="RGB")
travolta = imageio.imread('travolta.jpg',pilmode="RGB")

dog1, dog2, dog3 = imageTrans(dog)
travolta1, travolta2, travolta3 = imageTrans(travolta)

f2 = plt.figure(figsize=(14,8))
f2_ax1 = f2.add_subplot(231)
f2_ax2 = f2.add_subplot(232)
f2_ax3 = f2.add_subplot(233)
f2_ax4 = f2.add_subplot(234)
f2_ax5 = f2.add_subplot(235)
f2_ax6 = f2.add_subplot(236)

f2_ax1.imshow(dog1, cmap = plt.get_cmap('gray'))
f2_ax1.title.set_text("transform of dog.jpg for situation (i)")
f2_ax2.imshow(dog2)
f2_ax2.title.set_text("transform of dog.jpg for situation (ii)")
f2_ax3.imshow(dog3)
f2_ax3.title.set_text("transform of dog.jpg for situation (iii)")
f2_ax4.imshow(travolta1, cmap = plt.get_cmap('gray'))
f2_ax4.title.set_text("transform of travolta.jpg for situation (i)")
f2_ax5.imshow(travolta2)
f2_ax5.title.set_text("transform of travolta.jpg for situation (ii)")
f2_ax6.imshow(travolta3)
f2_ax6.title.set_text("transform of travolta.jpg for situation (iii)")
plt.show()
```



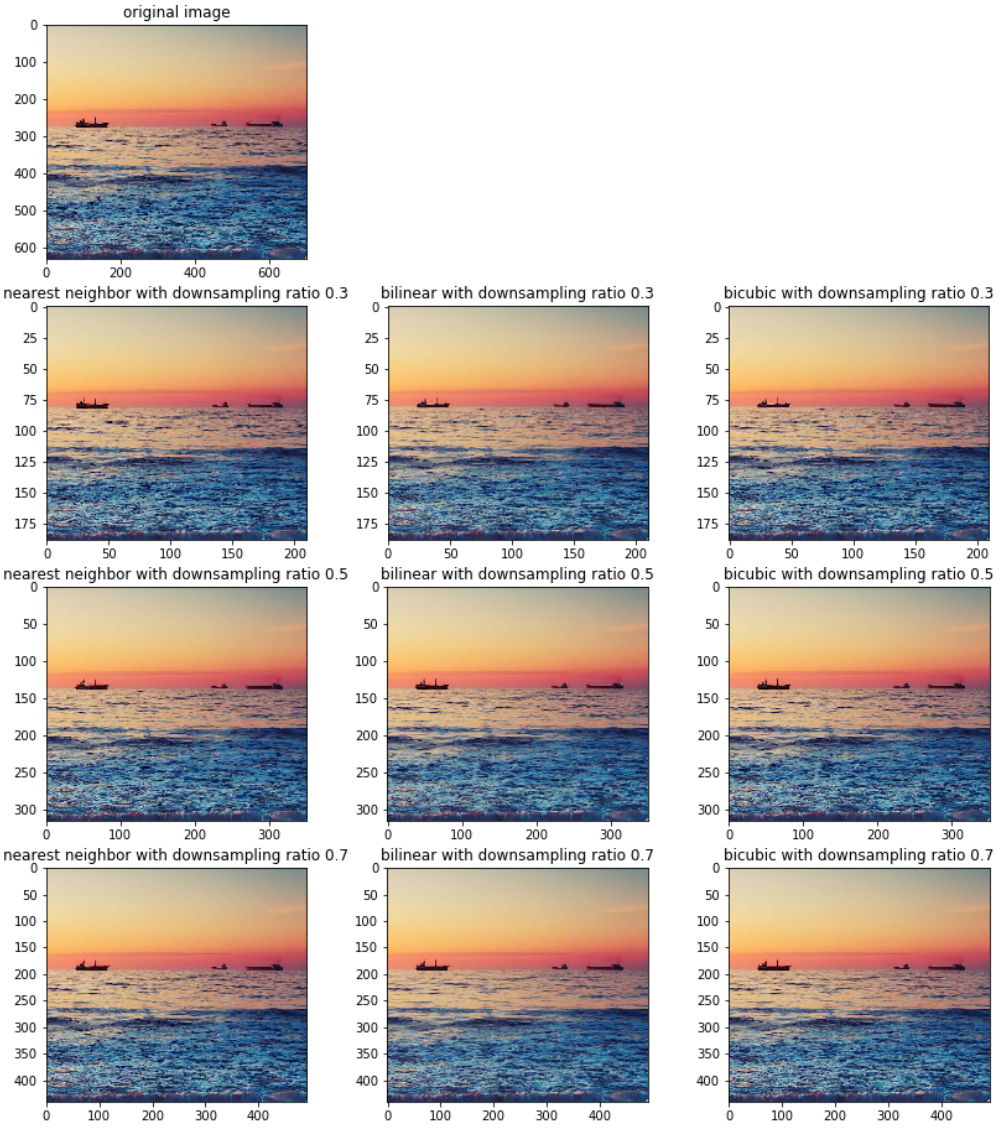
Problem 5

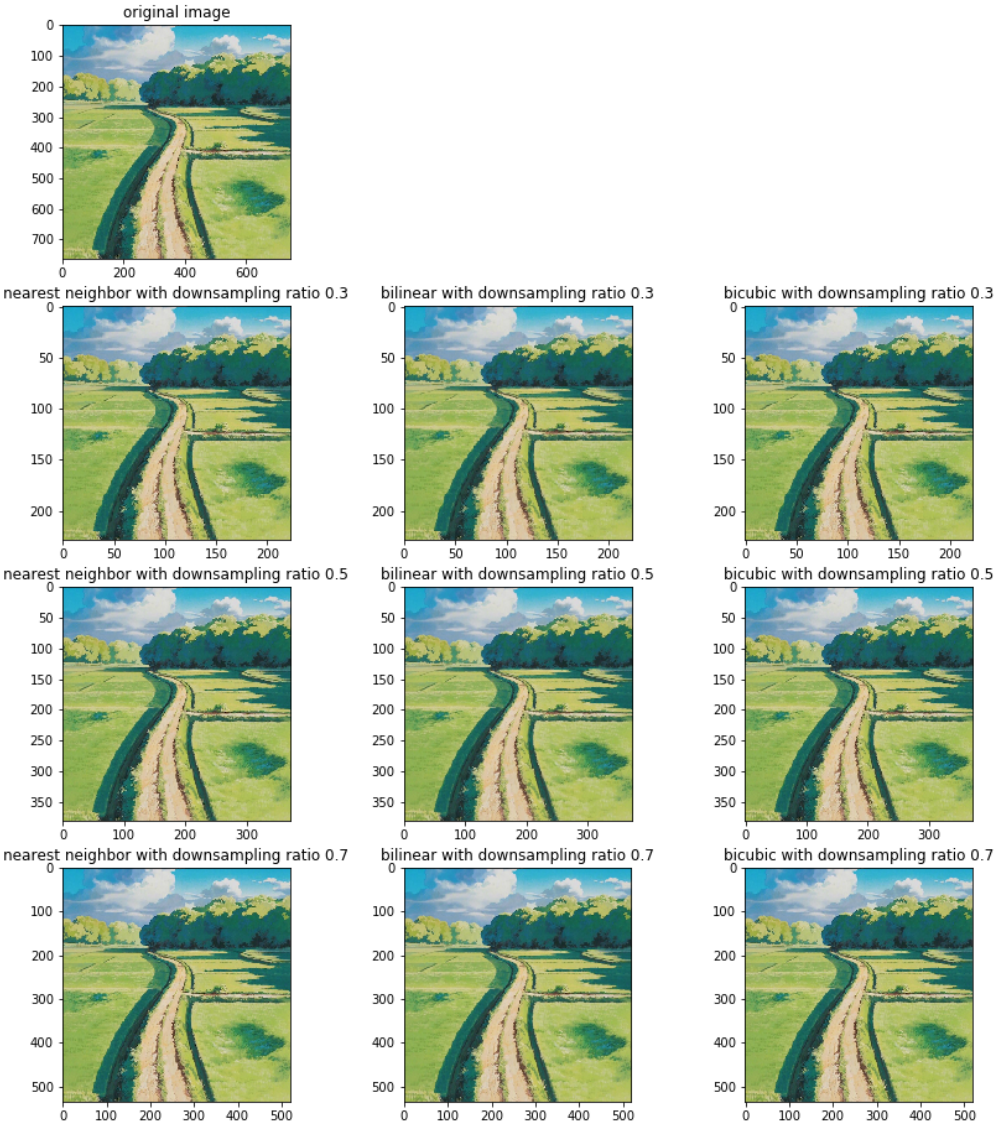
(i) Nearest Neighbor Interpolation, Bilinear Interpolation, Bicubic Interpolation

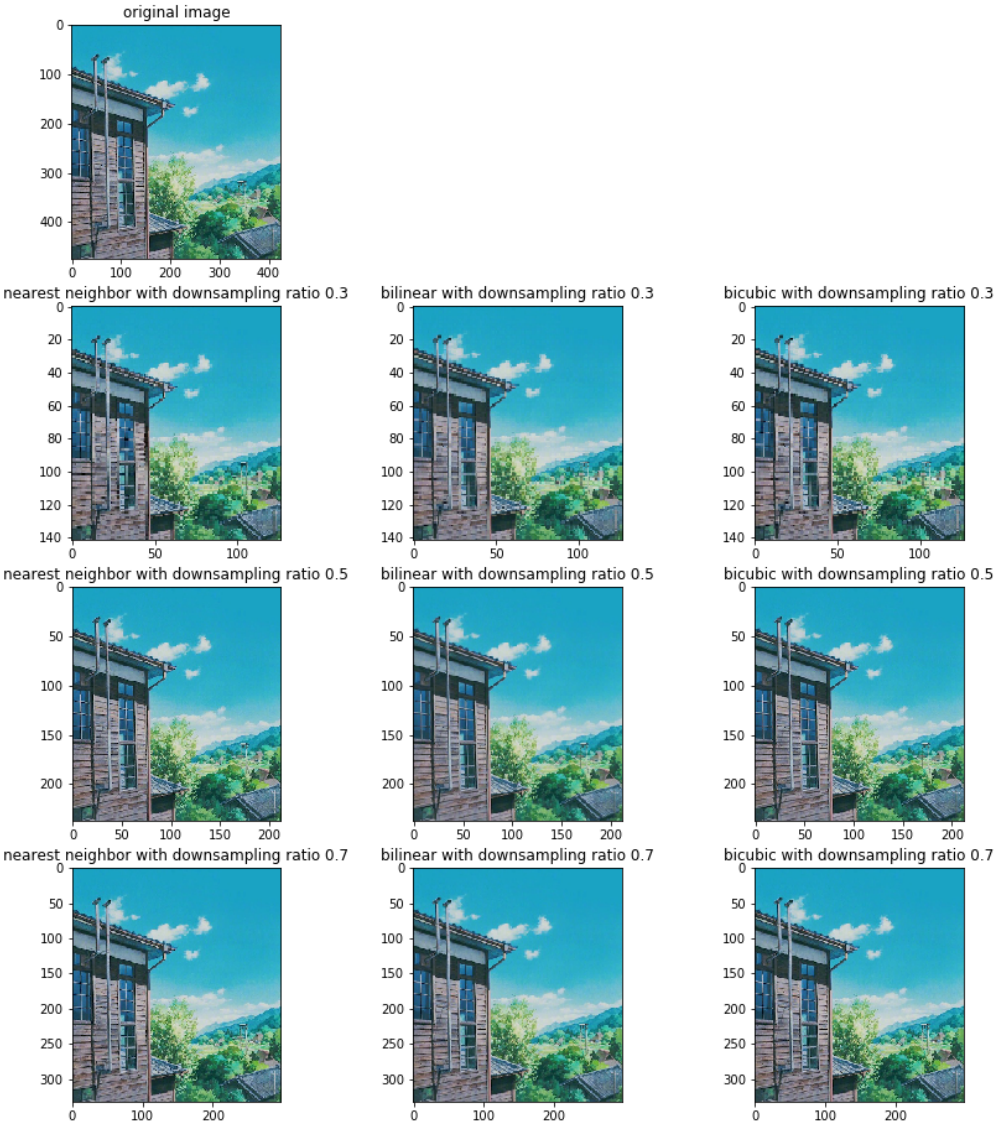
(ii) The nearest neighbor interpolation simply selects the interpolated value as the value of the nearest point and does not consider the values of neighboring points at all. The bilinear interpolation is performed using linear interpolation first in one direction, and then again in the other direction. Bilinear interpolation will consider 4 pixels (2x2) and calculate their weighted average. The bicubic interpolation is similar to bilinear interpolation. But instead of consider 4 pixels, bicubic interpolation will considers 16 pixels (4×4) and calculate their weighted average.

In [ ]:
```
import glob
import cv2

images = [plt.imread(file) for file in glob.glob("interpolation/*.jpg")]
scale_percent = [0.3, 0.5, 0.7]

i = 1
for image in images:
    ref = plt.figure(figsize=(14,16))
    reax0 = ref.add_subplot(431)
    reax0.imshow(image)
    reax0.title.set_text("original image")
#     plt.show()
    na = 4;
    for scale in scale_percent:
        width = int(image.shape[1] * scale)
        height = int(image.shape[0] * scale)
        dim = (width, height)
        nearest = cv2.resize(image, dim, interpolation = cv2.INTER_NEAREST)
        linear = cv2.resize(image, dim, interpolation = cv2.INTER_LINEAR)
        bicubic = cv2.resize(image, dim, interpolation = cv2.INTER_CUBIC)
        ax = ref.add_subplot(4,3,na)
        ax2 = ref.add_subplot(4,3,na+1)
        ax3 = ref.add_subplot(4,3,na+2)
        na += 3
        ax.title.set_text("nearest neighbor with downsampling ratio %s"%sca
        ax.imshow(nearest)
        ax2.title.set_text("bilinear with downsampling ratio %s"%scale)
        ax2.imshow(linear)
        ax3.title.set_text("bicubic with downsampling ratio %s"%scale)
        ax3.imshow(bicubic)
    plt.savefig("question3_%s"%i)
    i += 1
```

(iii) The original image and the result of the 3 interpolation method for 3 different downsampling scale are shown as below. We can tell from the result that nearest neighbor act worse than the two other methods as the detail of image shows less acurate when using nearest neighbor. The difference between bilinear and bicubic are more difficult to tell.

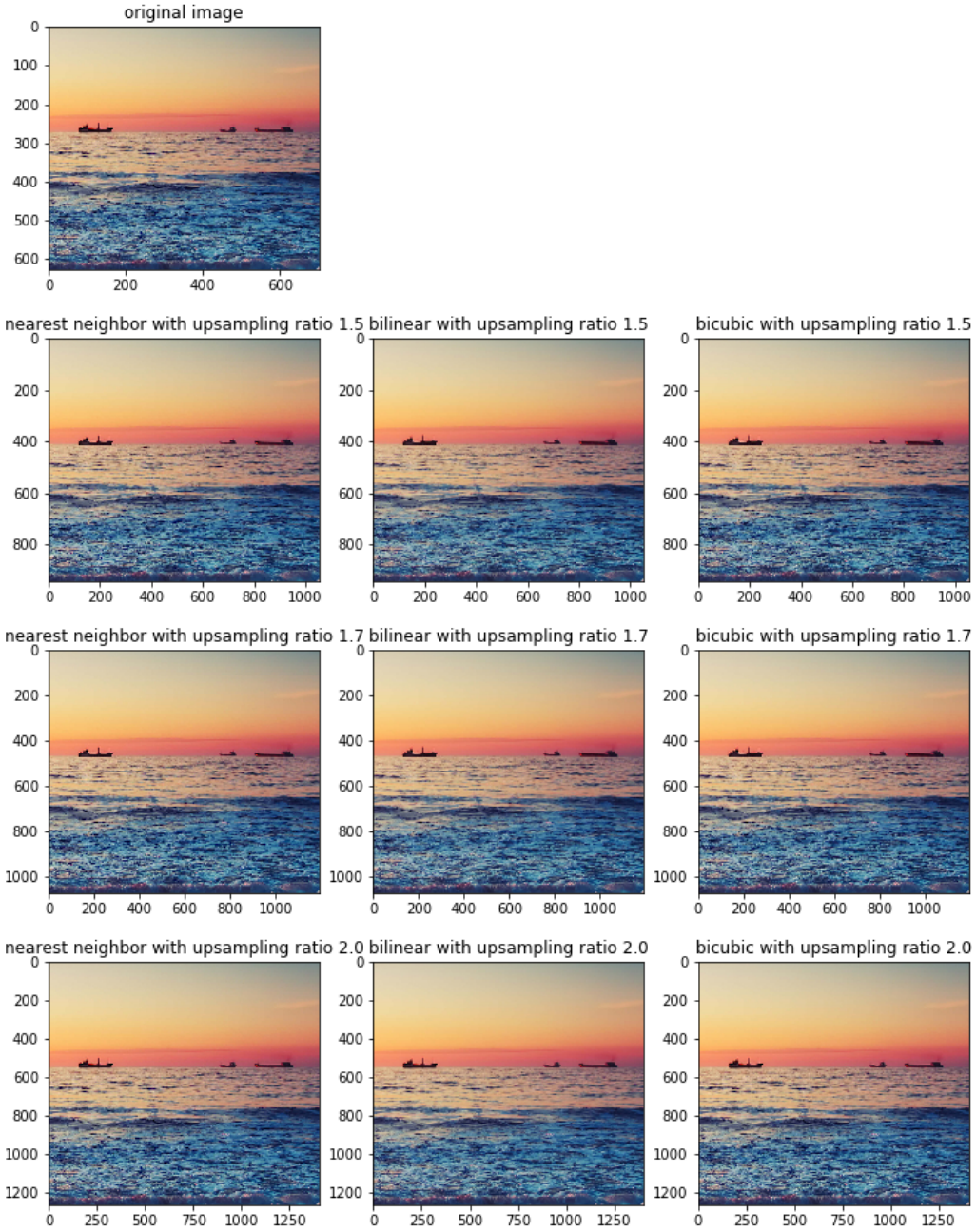

original image
nearest neighbor with downsampling ratio 0.3
bilinear with downsampling ratio 0.3
bicubic with downsampling ratio 0.3
nearest neighbor with downsampling ratio 0.5
bilinear with downsampling ratio 0.5
bicubic with downsampling ratio 0.5
nearest neighbor with downsampling ratio 0.7
bilinear with downsampling ratio 0.7
bicubic with downsampling ratio 0.7

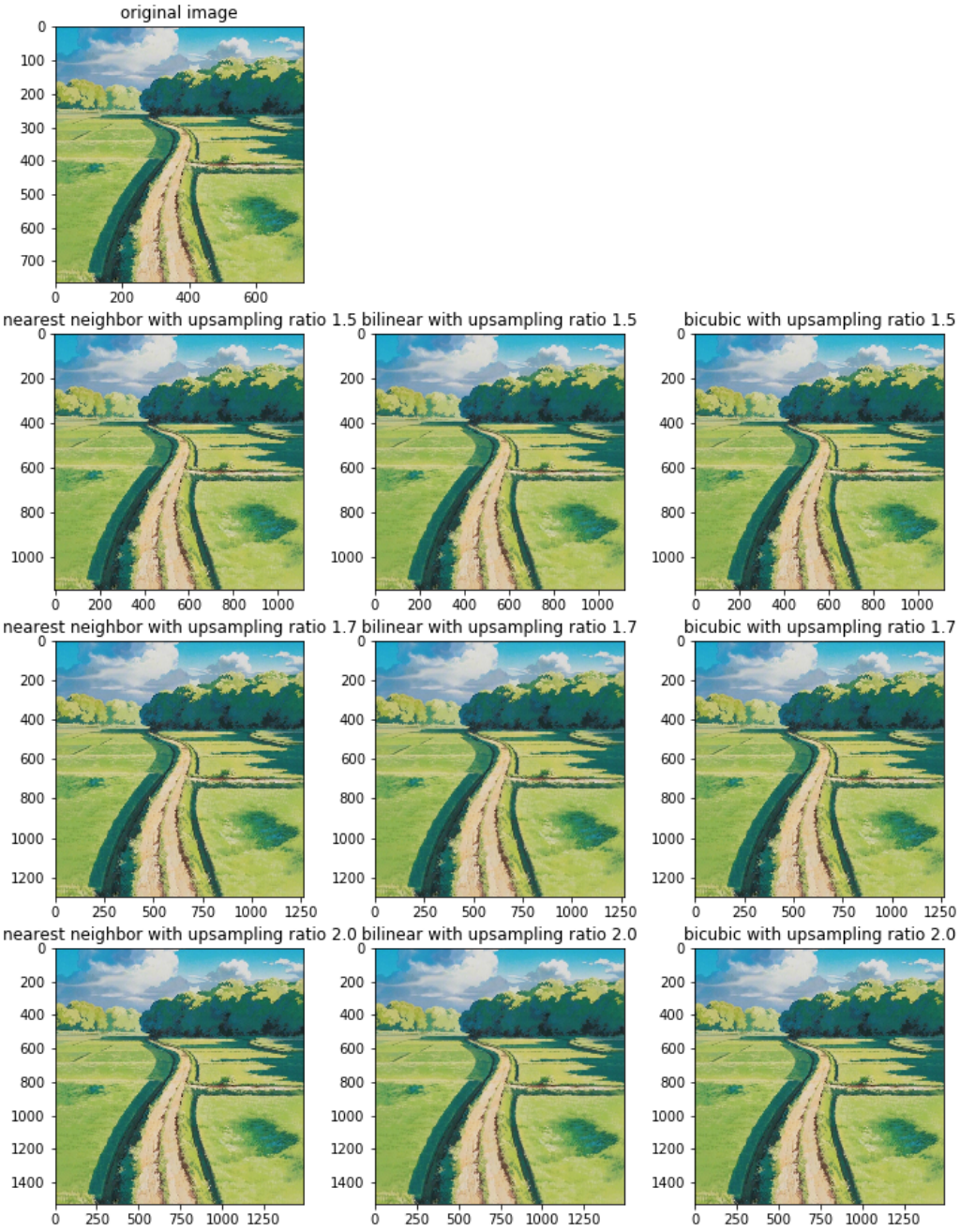

original image
nearest neighbor with downsampling ratio 0.3
bilinear with downsampling ratio 0.3
bicubic with downsampling ratio 0.3
nearest neighbor with downsampling ratio 0.5
bilinear with downsampling ratio 0.5
bicubic with downsampling ratio 0.5
nearest neighbor with downsampling ratio 0.7
bilinear with downsampling ratio 0.7
bicubic with downsampling ratio 0.7

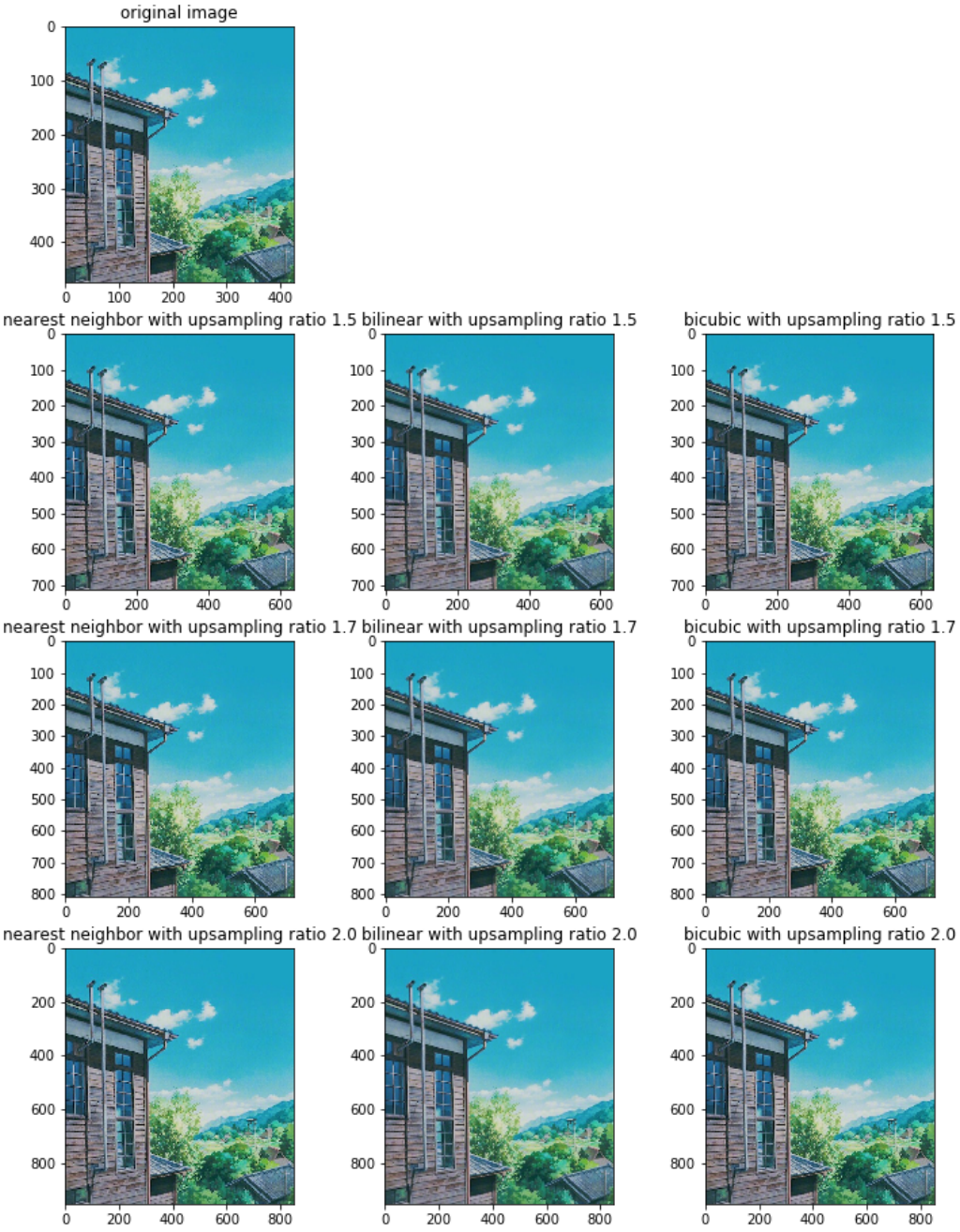

original image
nearest neighbor with downsampling ratio 0.3
bilinear with downsampling ratio 0.3
bicubic with downsampling ratio 0.3
nearest neighbor with downsampling ratio 0.5
bilinear with downsampling ratio 0.5
bicubic with downsampling ratio 0.5
nearest neighbor with downsampling ratio 0.7
bilinear with downsampling ratio 0.7
bicubic with downsampling ratio 0.7

In [ ]:
```
import glob
import cv2

images = [plt.imread(file) for file in glob.glob("interpolation/*.jpg")]
scale_percent = [1.5, 1.7, 2.0]

i = 1
for image in images:
    ref = plt.figure(figsize=(12,16))
    reax0 = ref.add_subplot(431)
    reax0.imshow(image)
    reax0.title.set_text("original image")
#     plt.show()
    na = 4;
    for scale in scale_percent:
        width = int(image.shape[1] * scale)
        height = int(image.shape[0] * scale)
        dim = (width, height)
        nearest = cv2.resize(image, dim, interpolation = cv2.INTER_NEAREST)
        linear = cv2.resize(image, dim, interpolation = cv2.INTER_LINEAR)
        bicubic = cv2.resize(image, dim, interpolation = cv2.INTER_CUBIC)
        ax = ref.add_subplot(4,3,na)
        ax2 = ref.add_subplot(4,3,na+1)
        ax3 = ref.add_subplot(4,3,na+2)
        na += 3
        ax.title.set_text("nearest neighbor with upsampling ratio %s"%scale
        ax.imshow(nearest)
        ax2.title.set_text("bilinear with upsampling ratio %s"%scale)
        ax2.imshow(linear)
        ax3.title.set_text("bicubic with upsampling ratio %s"%scale)
        ax3.imshow(bicubic)
    plt.savefig("question4_%s"%i)
    i += 1
```

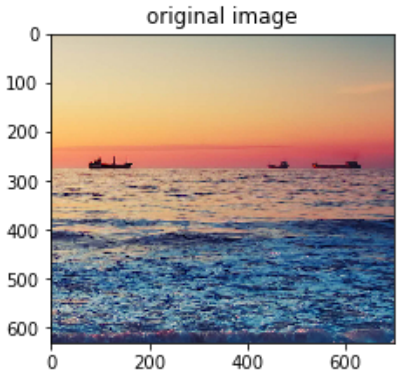
(iv) The original image and the result of the 3 interpolation method for 3 different upsampling scale are shown as below. Samely we can tell from the result that nearest neighbor act worse than the two other methods as interpolated surface shows more smoother than nearest neighbor. The difference between bilinear and bicubic are still hard to tell.

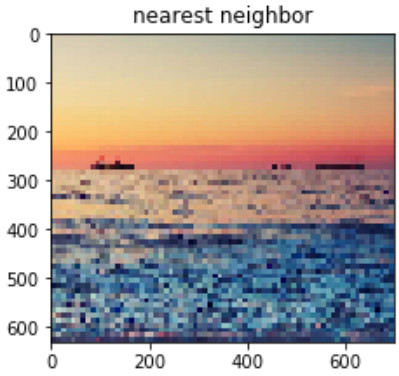

original image
nearest neighbor with upsampling ratio 1.5
bilinear with upsampling ratio 1.5
bicubic with upsampling ratio 1.5
nearest neighbor with upsampling ratio 1.7
bilinear with upsampling ratio 1.7
bicubic with upsampling ratio 1.7
nearest neighbor with upsampling ratio 2.0
bilinear with upsampling ratio 2.0
bicubic with upsampling ratio 2.0

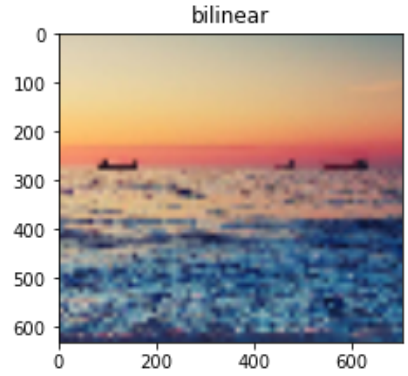

original image
nearest neighbor with upsampling ratio 1.5
bilinear with upsampling ratio 1.5
bicubic with upsampling ratio 1.5
nearest neighbor with upsampling ratio 1.7
bilinear with upsampling ratio 1.7
bicubic with upsampling ratio 1.7
nearest neighbor with upsampling ratio 2.0
bilinear with upsampling ratio 2.0
bicubic with upsampling ratio 2.0

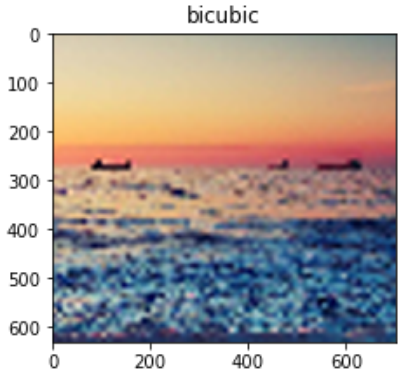

original image
nearest neighbor with upsampling ratio 1.5
bilinear with upsampling ratio 1.5
bicubic with upsampling ratio 1.5
nearest neighbor with upsampling ratio 1.7
bilinear with upsampling ratio 1.7
bicubic with upsampling ratio 1.7
nearest neighbor with upsampling ratio 2.0
bilinear with upsampling ratio 2.0
bicubic with upsampling ratio 2.0

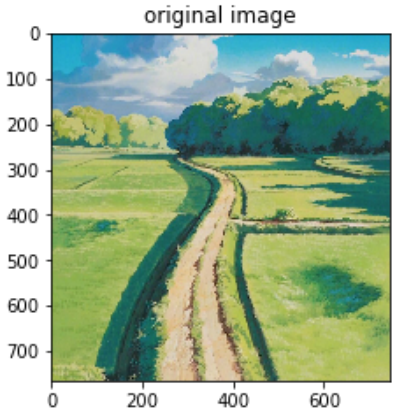
In [12]:
```
import glob
import cv2
import matplotlib.pyplot as plt

images = [plt.imread(file) for file in glob.glob("interpolation/*.jpg")]
i = 1

for image in images:
    ref = plt.figure(figsize=(12,8))
    reax0 = ref.add_subplot(231)
    reax0.imshow(image)
    reax0.title.set_text("original image")
#     plt.show()

    scale = 0.1
    width = int(image.shape[1] * scale)
    height = int(image.shape[0] * scale)
    dim = (width, height)
    ori_dim = (image.shape[1], image.shape[0])

    nearest = cv2.resize(image, dim, interpolation = cv2.INTER_NEAREST)
    linear = cv2.resize(image, dim, interpolation = cv2.INTER_LINEAR)
    bicubic = cv2.resize(image, dim, interpolation = cv2.INTER_CUBIC)

    renearest = cv2.resize(nearest, ori_dim, interpolation = cv2.INTER_NEAR
    relinear = cv2.resize(linear, ori_dim, interpolation = cv2.INTER_LINEAR
    rebicubic = cv2.resize(bicubic, ori_dim, interpolation = cv2.INTER_CUBI
    reax = ref.add_subplot(234)
    reax2 = ref.add_subplot(235)
    reax3 = ref.add_subplot(236)
    reax.title.set_text("nearest neighbor")
    reax.imshow(renearest)
    reax2.title.set_text("bilinear")
    reax2.imshow(relinear)
    reax3.title.set_text("bicubic")
    reax3.imshow(rebicubic)
#     plt.show()
    plt.savefig("question5_%s"%i)
```

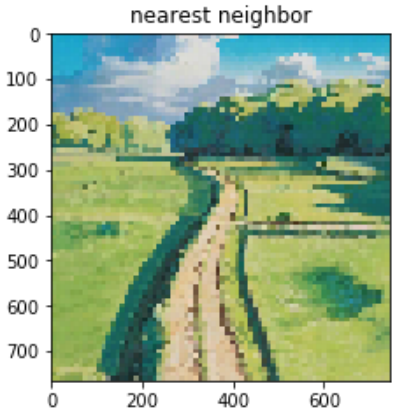

original image

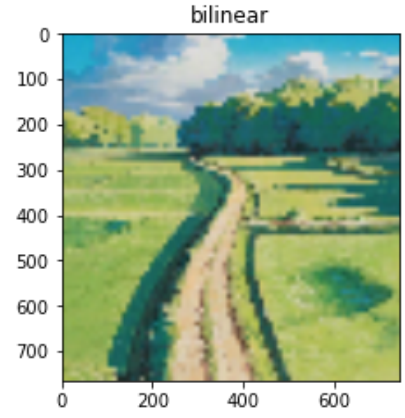

nearest neighbor

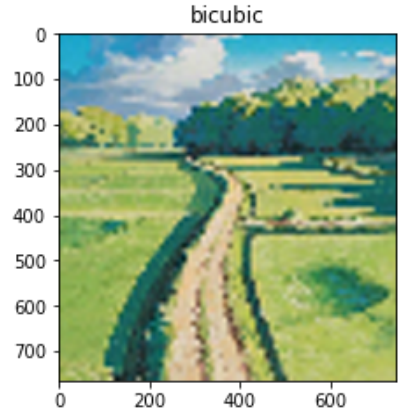

bilinear

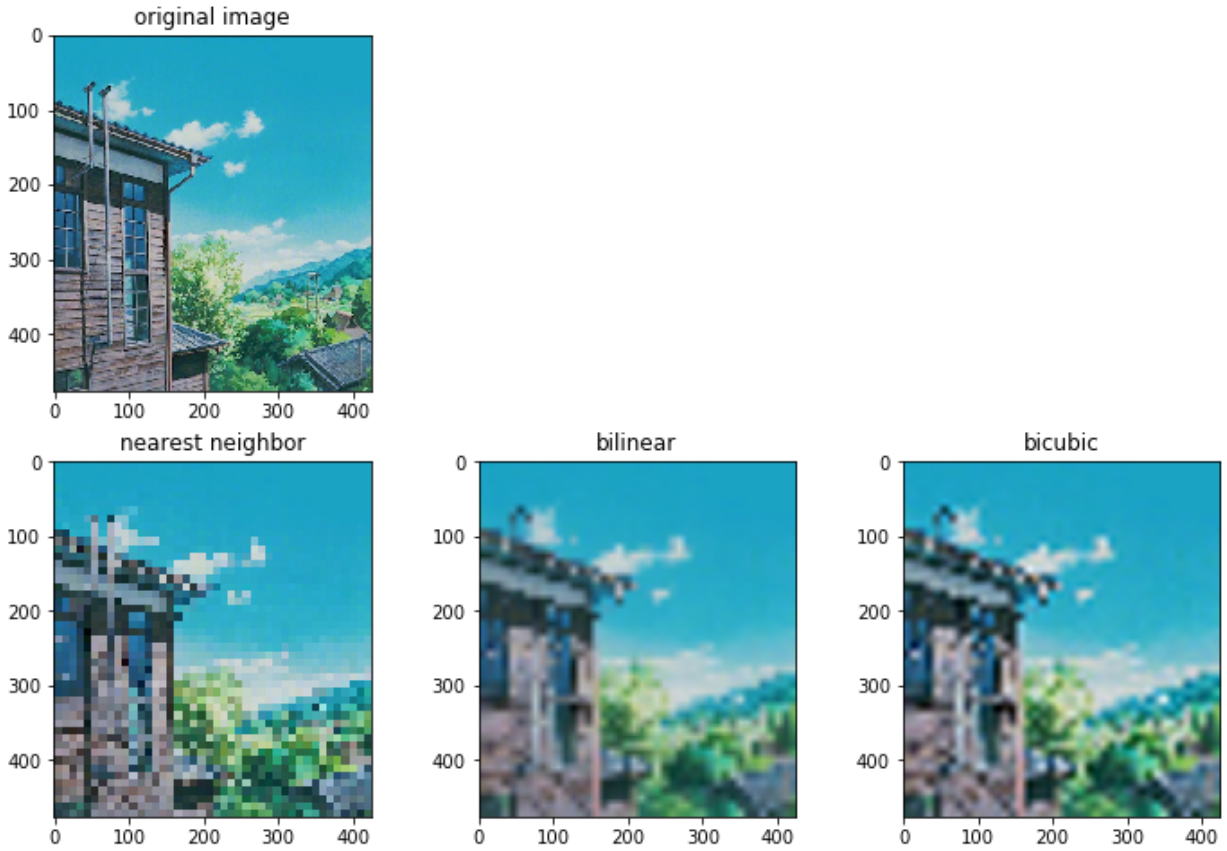

bicubic


original image


nearest neighbor


bilinear


bicubic



(v) The original image and the result of the 3 interpolation method are shown as above. We can tell from the result that bicubic interpolation act better than the two other methods and bilinear interpolation act better than nearest neighbor interpolation. As we can tell from the result, the result of nearest neighbor is not smooth at all and ignored a lot of detail. Bilinear shows much smoother than nearest neighbor but still ignore some of the detail in the image. In contrast, the result of bicubic is not only smoother than the other two, but also shows much more details.