In [1]:

```
##William Argus A12802324
## ECE 253 HW1
'''
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.
'''
```

Out[1]:

```
'\nAcademic Integrity Policy: Integrity of scholarsh
ip is essential for an academic commu-\nnity. The Un
iversity expects that both faculty and students will
honor this principle and\nin so doing protect the va
lidity of University intellectual work. For students
, this means\nthat all academic work will be done by
the individual to whom it is assigned, without\nunau
thorized aid of any kind.\nBy including this in my r
eport, I agree to abide by the Academic Integrity Po
licy men-\ntioned above.\n'
```

In [2]:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
from copy import copy, deepcopy
```

In [3]:

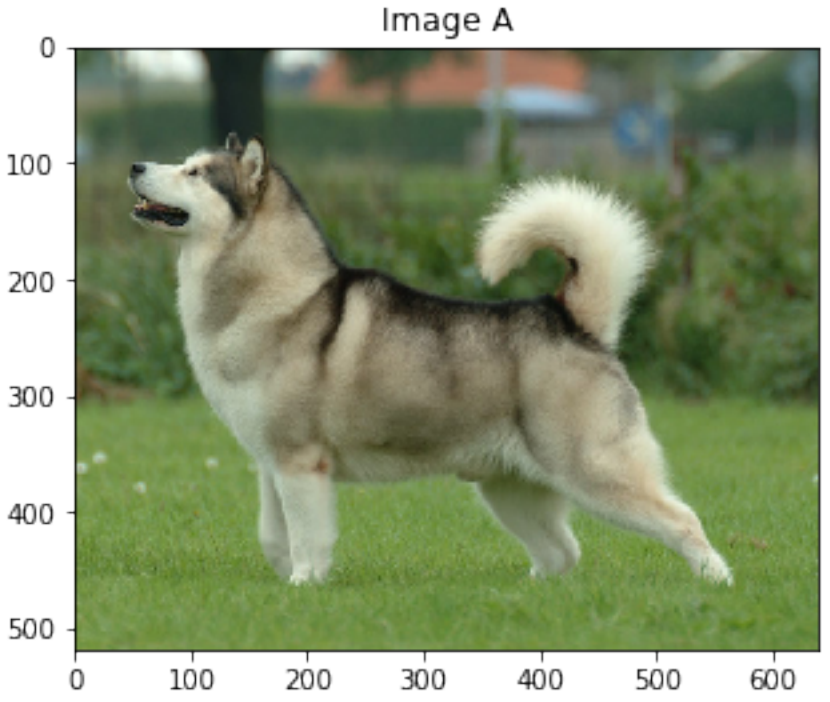
```
##Problem 1

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

B = np.matrix([[0, 1, 0, 1 ],
               [0, 1, 1, 0],
               [0, 0, 0, 1],
               [1, 1, 0, 1],
               [0, 1, 0, 0]])

C = np.multiply(A, B)
print('Pointwise multiplication of A and B set to C:')
print(C)
print('\n')

innerProduct = np.inner(C[1,:], C[2,:])[0,0]
print('Inner product of 2nd and 3rd rows of C:', innerProduct)
print('\n')

maxVal = np.amax(C)
getMaxVal = np.where(C == maxVal)
maxValLoc = np.transpose(np.matrix([ getMaxVal[0][:], getMaxVal[
1][:] ]))
print('Max value of C:', maxVal)
print('indicies of maximum value in C:')
print(maxValLoc)
print('\n')

minVal = np.amin(C)
getMinVal = np.where(C == minVal)
minValLoc = np.transpose(np.matrix([ getMinVal[0][:], getMinVal[
1][:] ]))
print('Min value of C:', minVal)
print('indicies of minimum value in C:')
print(minValLoc)
print('\n')
```

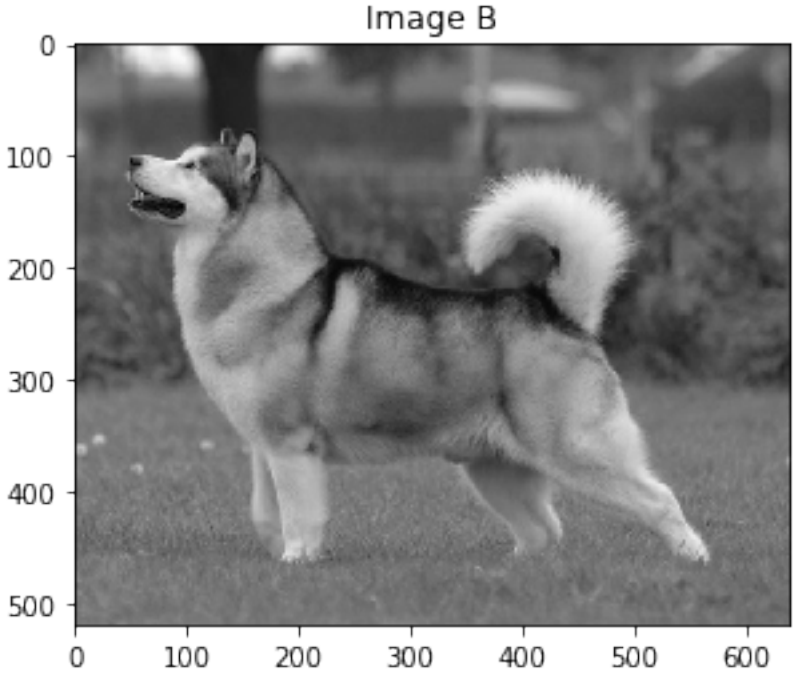
```
Pointwise multiplication of A and B set to C:
[[ 0  9  0  1]
 [ 0 25  4  0]
 [ 0  0  0  9]
 [ 6 32  0  0]
 [ 0  8  0  0]]


Inner product of 2nd and 3rd rows of C: 0


Max value of C: 32
indicies of maximum value in C:
[[3 1]]


Min value of C: 0
indicies of minimum value in C:
[[0 0]
 [0 2]
 [1 0]
 [1 3]
 [2 0]
 [2 1]
 [2 2]
 [3 2]
 [3 3]
 [4 0]
 [4 2]
 [4 3]]
```

In [31]:

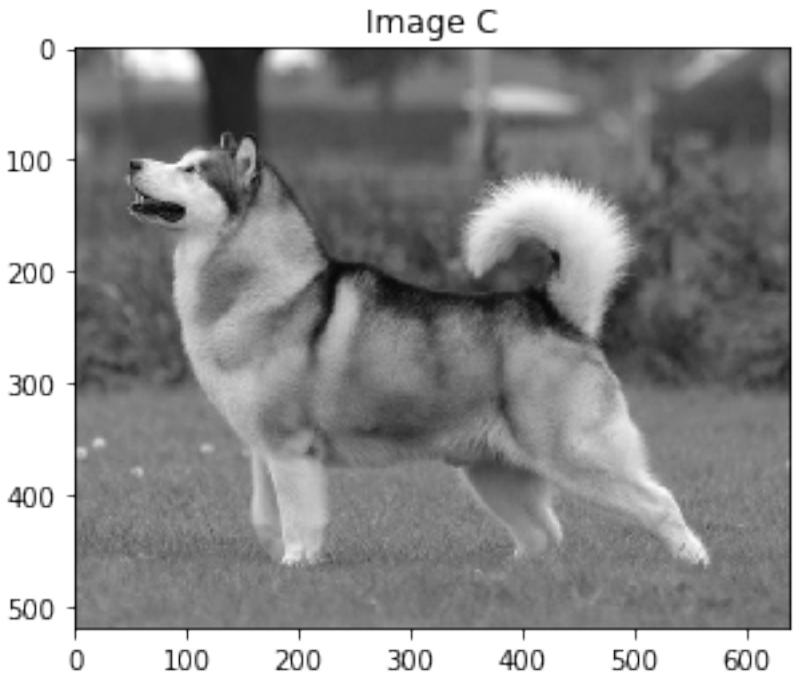
```
'''
Problem 2
Picture: Alaskan_Malamute.jpg
'''


A = plt.imread('Alaskan_Malamute.jpg')
plt.figure( figsize=(5,5))
plt.imshow(A)
plt.title('Image A')
```

```
plt.show()

size = np.shape(A)

#B1 = np.zeros((size[0], size[1]))
#B1[:,:] = (A[:,:,0] + A[:,:,1] + A[:,:,2])
#plt.imshow(B1, cmap="gray")
#plt.show()

B = cv2.cvtColor(A, cv2.COLOR_BGR2GRAY)
B = B-3
B = (B/np.amax(B))*255
plt.title('Image B')
plt.imshow(B, cmap="gray")
plt.show()
print('Min of B:', np.amin(B))
print('Max of B:', np.amax(B))

C = B+15
C[C > 255] = 255
plt.imshow(C, cmap="gray")
plt.title('Image C')
plt.show()
print('Min of C:', np.amin(C))
print('Max of C:', np.amax(C))

D = np.flipud(np.fliplr(B))
plt.imshow(D, cmap="gray")
plt.title('Image D')
plt.show()
print('Min of D:', np.amin(C))
print('Max of D:', np.amax(C))

median = np.median(B)
E = deepcopy(B)
E[E <= median] = 0
E[E > median] = 1
plt.imshow(E, cmap ="binary")
plt.title('Image E')
plt.show()
print('Min of E:', np.amin(C))
print('Max of E:', np.amax(C))
```

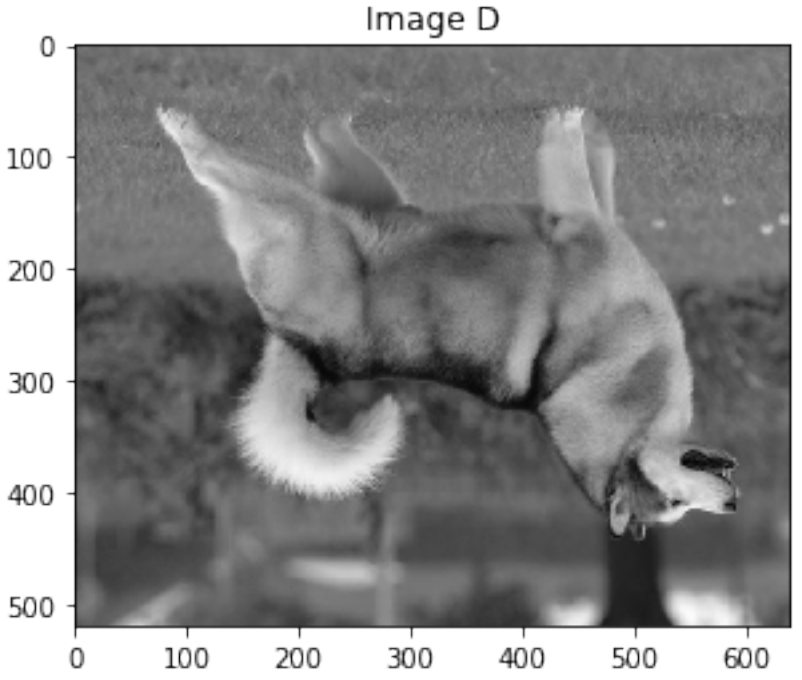




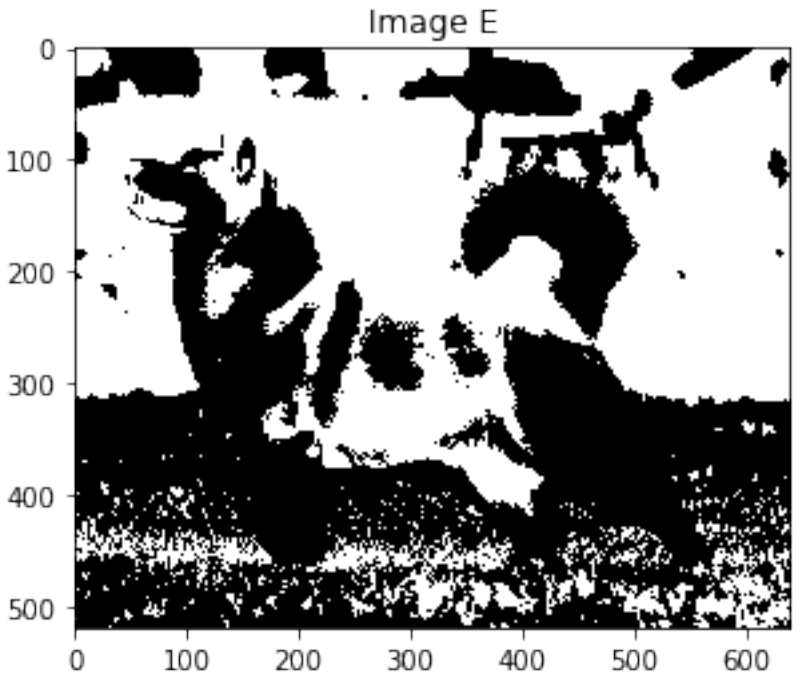

```
Min of B: 0.0
Max of B: 255.0
```




```
Min of C: 15.0
Max of C: 255.0
```




```
Min of D: 15.0
Max of D: 255.0
```

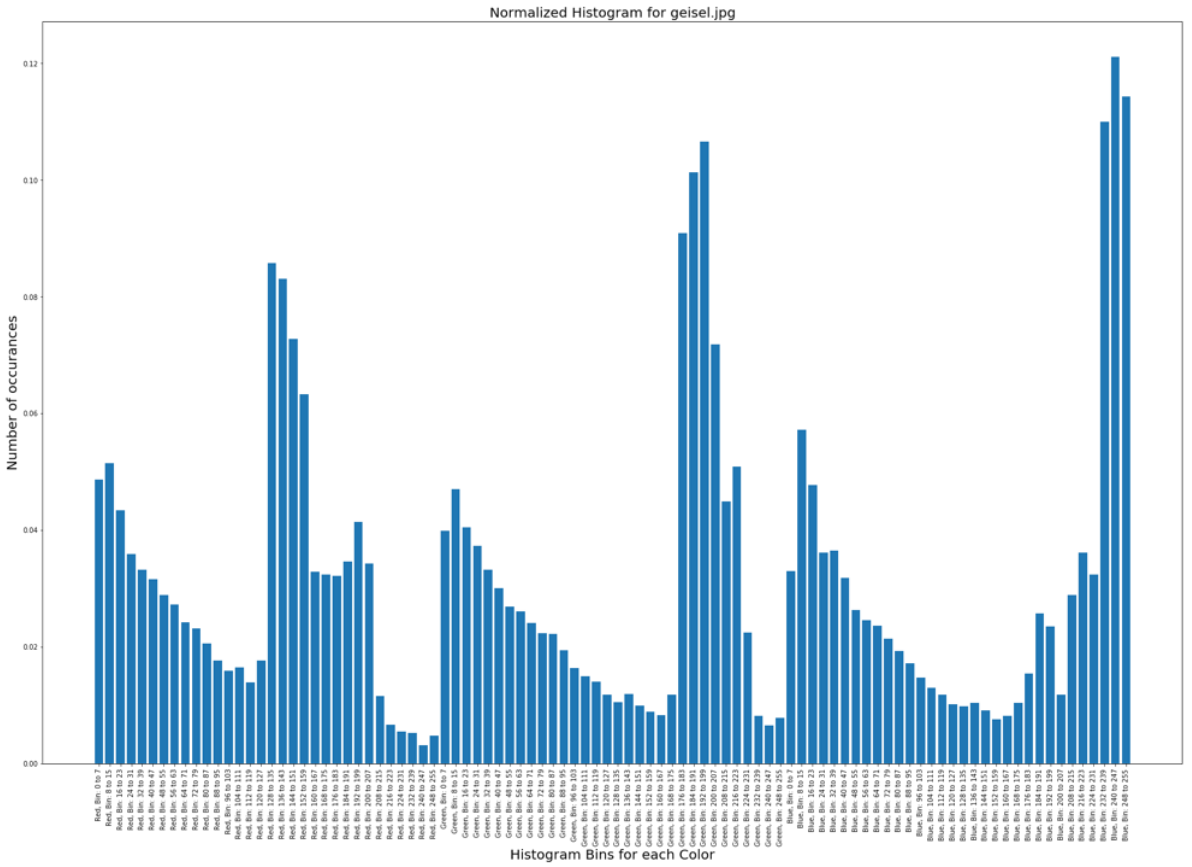



```
Min of E: 15.0
Max of E: 255.0
```

In [5]:

```
'''
Problem 3
'''
##Helper function
def populateBins (colorVector, colorMatrix):
    increment = 256/32
    for k in range(0,32):
        cntLower = colorMatrix < k*increment
        cntUpper = colorMatrix < (k+1)*increment
        A, B = cntUpper.sum(), cntLower.sum()
        cnt = cntUpper.sum() - cntLower.sum()
        colorVector[k] = cnt
    return colorVector

def compute_norm_rgb_histogram (image):
    redVector = np.zeros((32))
    greenVector = np.zeros((32))
    blueVector = np.zeros((32))

    redVector = populateBins(redVector, image[:,:,0])
    greenVector = populateBins(greenVector, image[:,:,1])
    blueVector = populateBins(blueVector, image[:,:,2])
    redVector = redVector/np.sum(redVector)
```

```
    greenVector = greenVector/np.sum(greenVector)

    blueVector = blueVector/np.sum(blueVector)
    q=np.sum(redVector) + np.sum(greenVector) + np.sum(blueVecto
r)
    concat = np.concatenate((redVector, greenVector, blueVector)
)
    q=0
    return concat #(1x96 vector)

image = plt.imread('geisel.jpg')

histo = compute_norm_rgb_histogram(image)


### Creat Histogram Axis ###
axis = []
for k in range(0,96):
    color = 'Red'
    start = 8*k
    end = 8*(k+1)-1
    if k>31:
        color = 'Green'
        start = 8*(k - 32)
        end = 8*(k+1 - 32)-1
    if k>63:
        color = 'Blue'
        start = 8*(k - 64)
        end = 8*(k+1 - 64)-1
    label = color + ', Bin: ' + str(start) + ' to ' + str(end)
    axis.append(label)




plt.figure(num=None, figsize=(30, 20))
y_pos = np.arange(len(axis))
plt.bar(y_pos, histo)
plt.xticks(y_pos, axis)
plt.ylabel('Number of occurances', size = 20)
plt.xlabel('Histogram Bins for each Color', size = 20)
plt.title('Normalized Histogram for geisel.jpg', size = 20)
plt.xticks(rotation=90)
plt.savefig('Histo.png', dpi = 100)
plt.show()
```



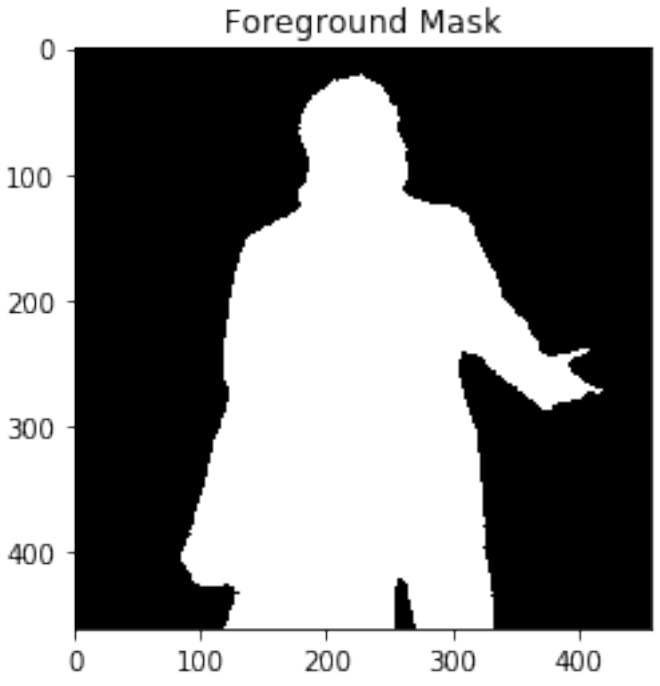
In [33]:

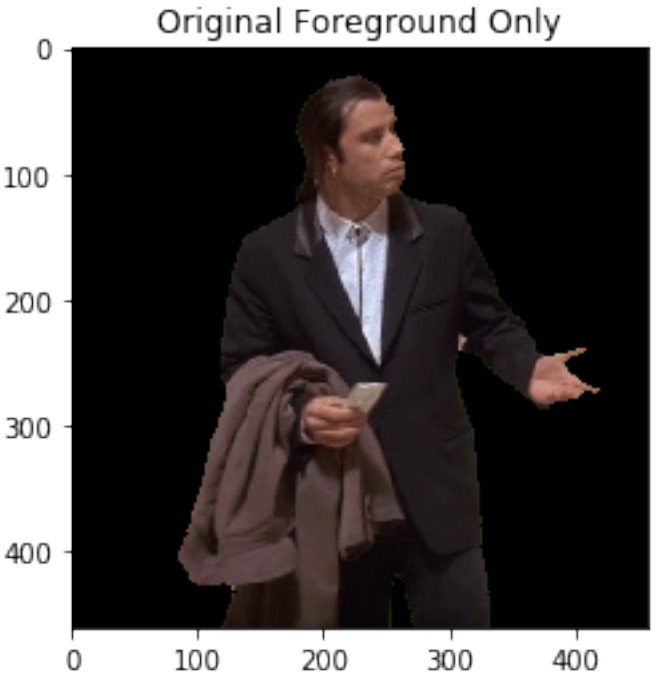
```
'''
Problem 4
'''
travolta = plt.imread('travolta.jpg')
copyTravolta = deepcopy(travolta)
myGreen = np.uint8([[[29,208,0]]])
hsvGreen = cv2.cvtColor(myGreen, cv2.COLOR_BGR2HSV)
hsvTravolta = cv2.cvtColor(copyTravolta, cv2.COLOR_BGR2HSV)
greenU = np.array([100,255,255])
greenL = np.array([60, 50, 50])


binaryImage = cv2.inRange(hsvTravolta, greenL, greenU)
binaryImage[binaryImage == 0 ] = 1
binaryImage[binaryImage == 255] = 0
copyTravolta[binaryImage == 0] = 0
plt.figure(1)
plt.title('Foreground Mask')
plt.imshow(binaryImage, cmap='gray')
plt.figure(2)
plt.title('Original Foreground Only')
plt.imshow(copyTravolta)
```

```
geisel = plt.imread('geisel.jpg')
gCopy = deepcopy(geisel)
gSZ = np.shape(geisel)
tSZ = np.shape(travolta)

dsize = (gSZ[1], gSZ[0])
bigTravolta = cv2.resize(copyTravolta, dsize)
bigMask = cv2.resize(binaryImage, dsize)
gCopy[bigMask==1] = bigTravolta[bigMask==1]
plt.figure(6)
plt.title('Travolta Resized with Geisel as Background')
plt.imshow(gCopy)

dsize = (tSZ[1], tSZ[0])
smallG = cv2.resize(gCopy, dsize)
smallG[binaryImage==1] = travolta[binaryImage==1]
plt.figure(7)
plt.title('Travolta with Resized Geisel as Background')
plt.imshow(smallG)
```

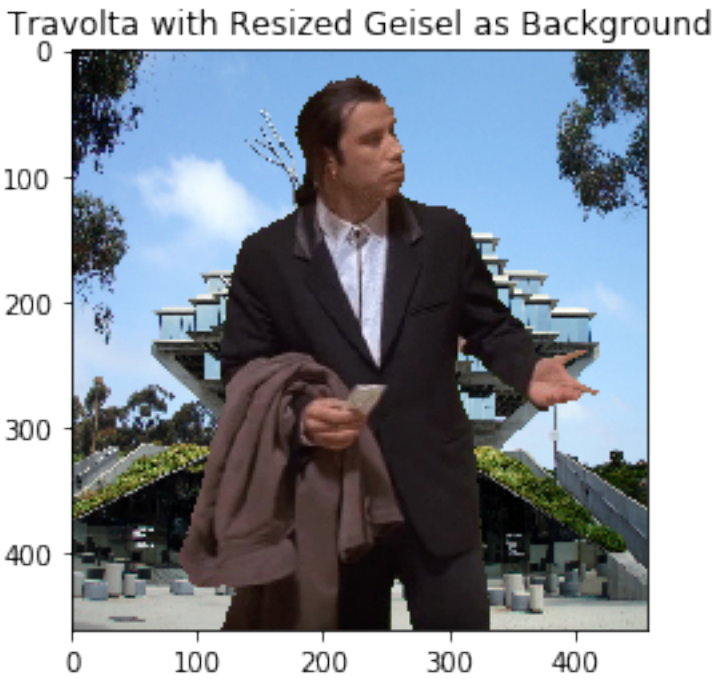
Out[33]:

```
<matplotlib.image.AxesImage at 0x128c0ded0>
```




Original Foreground Only
0
100
200
300
400
0
100
200
300
400


Travolta Resized with Geisel as Background
0
500
1000
1500
2000
0
500
1000
1500
2000
2500
3000



In [35]:

```
'''
Problem 5
'''

'''
3 interpolation methods listed and described

Nearest Neighbor interpolation is an algorithm for interpolation
that takes the value of the nearest
point to the given location and ignores the values of other poin
ts around that are farther away. This
method is simple, however it often produces undesirable aspects
in the image, a trade-off for its simplicity.
Linear interpolation is used to approximate a value for an unkno
wn location by estimating the value at
that location to be the value of a line segment passing that loc
ation drawn between 2 points (the 2 nearest
neighbors) near to the location whose values are known. This giv
es better results than the simpler
nearest neighbor method, however it does result an increase in c
omputational complexity to acheive this
improvement.
Bicubic Interpolation is a method that uses a four by four block
of pixel values (the 16 nearest neighbors of the
point in question) nearest to the unknown location in question i
n order to best estimate the pixel value at that
```

```
location. This method is good for preserving the fine details of

the image if the previous 2 methods do
not do a good enough job. As expected, using bicubic interpolati
on is the most computationally costly of the three.
'''

methods = np.array(['cv2.INTER_NEAREST', 'cv2.INTER_LINEAR', 'cv
2.INTER_CUBIC'])

image1 = cv2.imread("image1.jpg")
image2 = cv2.imread("image2.jpg")
image3 = cv2.imread("image3.jpg")

image1 = cv2.cvtColor(image1, cv2.COLOR_BGR2RGB)
image2 = cv2.cvtColor(image2, cv2.COLOR_BGR2RGB)
image3 = cv2.cvtColor(image3, cv2.COLOR_BGR2RGB)
'''
sz = np.shape(image1)
image2 = cv2.resize(image2, (int(i*sz[1]),int(i*sz[0])))
image3 = cv2.resize(image3, (int(i*sz[1]),int(i*sz[0])))
'''

## Downsample
ratio1 = np.array([.3,.5,.7])
k=1
for i in ratio1:
    image = image1
    sz = np.shape(image)
    resize= cv2.resize(image, (int(i*sz[1]),int(i*sz[0])), inter
polation=cv2.INTER_NEAREST)
    plt.figure(k)
    #plt.figure(num=None, figsize=(8, 8))
    title = 'Image 1 downsample for: nearest neighbor interpolat
ion, with ratio: ' + str(i)
    plt.title(title)
    plt.imshow(resize)
    name = 'Fig: ' + str(k)
    plt.savefig(name)
    k=k+1
    sz = np.shape(image)
    resize= cv2.resize(image, (int(i*sz[1]),int(i*sz[0])), inter
polation=cv2.INTER_LINEAR)
    plt.figure(k)
    #plt.figure(num=None, figsize=(8, 8))
```

```
    title = 'Image 1 downsample for: linear interpolation, with
ratio: ' + str(i)
    plt.title(title)
    plt.imshow(resize)
    name = 'Fig: ' + str(k)
    plt.savefig(name)
    k=k+1
    sz = np.shape(image)
    resize= cv2.resize(image, (int(i*sz[1]),int(i*sz[0])), inter
polation=cv2.INTER_CUBIC)
    plt.figure(k)
    #plt.figure(num=None, figsize=(8, 8))
    title = 'Image 1 downsample for: bicubic interpolation, with
ratio: ' + str(i)
    plt.title(title)
    plt.imshow(resize)
    name = 'Fig: ' + str(k)
    plt.savefig(name)
    k=k+1
    ##############################
    image = image2
    sz = np.shape(image)
    resize= cv2.resize(image, (int(i*sz[1]),int(i*sz[0])), inter
polation=cv2.INTER_NEAREST)
    plt.figure(k)
    #plt.figure(num=None, figsize=(8, 8))
    title = 'Image 2 downsample for: nearest neighbor interpolat
ion, with ratio: ' + str(i)
    plt.title(title)
    plt.imshow(resize)
    name = 'Fig: ' + str(k)
    plt.savefig(name)
    k=k+1
    sz = np.shape(image)
    resize= cv2.resize(image, (int(i*sz[1]),int(i*sz[0])), inter
polation=cv2.INTER_LINEAR)
    plt.figure(k)
    #plt.figure(num=None, figsize=(8, 8))
    title = 'Image 2 downsample for: linear interpolation, with
ratio: ' + str(i)
    plt.title(title)
    plt.imshow(resize)
    name = 'Fig: ' + str(k)
    plt.savefig(name)
    k=k+1
```

```
    sz = np.shape(image)

    resize= cv2.resize(image, (int(i*sz[1]),int(i*sz[0])), inter
polation=cv2.INTER_CUBIC)
    plt.figure(k)
    #plt.figure(num=None, figsize=(8, 8))
    title = 'Image 2 downsample for: bicubic interpolation, with
ratio: ' + str(i)
    plt.title(title)
    plt.imshow(resize)
    name = 'Fig: ' + str(k)
    plt.savefig(name)
    k=k+1
    ############################
    image = image3
    sz = np.shape(image)
    resize= cv2.resize(image, (int(i*sz[1]),int(i*sz[0])), inter
polation=cv2.INTER_NEAREST)
    plt.figure(k)
    #plt.figure(num=None, figsize=(8, 8))
    title = 'Image 3 downsample for: nearest neighbor interpolat
ion, with ratio: ' + str(i)
    plt.title(title)
    plt.imshow(resize)
    name = 'Fig: ' + str(k)
    plt.savefig(name)
    k=k+1
    sz = np.shape(image)
    resize= cv2.resize(image, (int(i*sz[1]),int(i*sz[0])), inter
polation=cv2.INTER_LINEAR)
    plt.figure(k)
    #plt.figure(num=None, figsize=(8, 8))
    title = 'Image 3 downsample for: linear interpolation, with
ratio: ' + str(i)
    plt.title(title)
    plt.imshow(resize)
    name = 'Fig: ' + str(k)
    plt.savefig(name)
    k=k+1
    sz = np.shape(image)
    resize= cv2.resize(image, (int(i*sz[1]),int(i*sz[0])), inter
polation=cv2.INTER_CUBIC)
    plt.figure(k)
    #plt.figure(num=None, figsize=(8, 8))
    title = 'Image 3 downsample for: bicubic interpolation, with
```

```
ratio: ' + str(i)

    plt.title(title)
    plt.imshow(resize)
    name = 'Fig: ' + str(k)
    plt.savefig(name)
    k=k+1
```

```
/Users/femw90/opt/anaconda3/lib/python3.7/site-packa
ges/ipykernel_launcher.py:47: RuntimeWarning: More t
han 20 figures have been opened. Figures created thr
ough the pyplot interface (`matplotlib.pyplot.figure
`) are retained until explicitly closed and may cons
ume too much memory. (To control this warning, see t
he rcParam `figure.max_open_warning`).
/Users/femw90/opt/anaconda3/lib/python3.7/site-packa
ges/ipykernel_launcher.py:59: RuntimeWarning: More t
han 20 figures have been opened. Figures created thr
ough the pyplot interface (`matplotlib.pyplot.figure
`) are retained until explicitly closed and may cons
ume too much memory. (To control this warning, see t
he rcParam `figure.max_open_warning`).
/Users/femw90/opt/anaconda3/lib/python3.7/site-packa
ges/ipykernel_launcher.py:69: RuntimeWarning: More t
han 20 figures have been opened. Figures created thr
ough the pyplot interface (`matplotlib.pyplot.figure
`) are retained until explicitly closed and may cons
ume too much memory. (To control this warning, see t
he rcParam `figure.max_open_warning`).
/Users/femw90/opt/anaconda3/lib/python3.7/site-packa
ges/ipykernel_launcher.py:79: RuntimeWarning: More t
han 20 figures have been opened. Figures created thr
ough the pyplot interface (`matplotlib.pyplot.figure
`) are retained until explicitly closed and may cons
ume too much memory. (To control this warning, see t
he rcParam `figure.max_open_warning`).
/Users/femw90/opt/anaconda3/lib/python3.7/site-packa
ges/ipykernel_launcher.py:91: RuntimeWarning: More t
han 20 figures have been opened. Figures created thr
ough the pyplot interface (`matplotlib.pyplot.figure
`) are retained until explicitly closed and may cons
ume too much memory. (To control this warning, see t
he rcParam `figure.max_open_warning`).
/Users/femw90/opt/anaconda3/lib/python3.7/site-packa
ges/ipykernel_launcher.py:101: RuntimeWarning: More
```

```
than 20 figures have been opened. Figures created th
rough the pyplot interface (`matplotlib.pyplot.figur
e`) are retained until explicitly closed and may con
sume too much memory. (To control this warning, see
the rcParam `figure.max_open_warning`).
/Users/femw90/opt/anaconda3/lib/python3.7/site-packa
ges/ipykernel_launcher.py:111: RuntimeWarning: More
than 20 figures have been opened. Figures created th
rough the pyplot interface (`matplotlib.pyplot.figur
e`) are retained until explicitly closed and may con
sume too much memory. (To control this warning, see
the rcParam `figure.max_open_warning`).
```

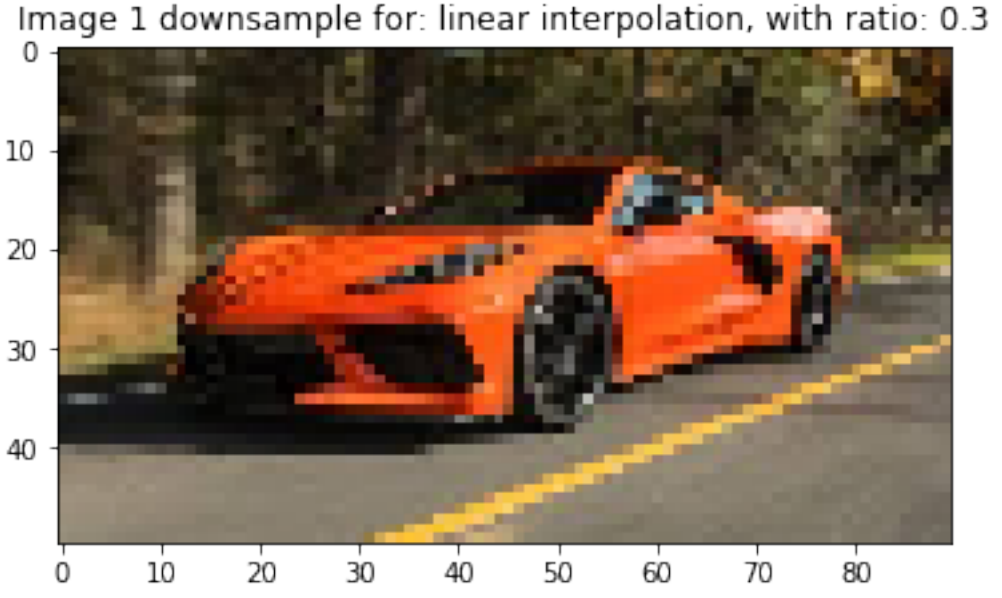


Image 1 downsample for: bicubic interpolation, with ratio: 0.3

Image 2 downsample for: nearest neighbor interpolation, with ratio: 0.3

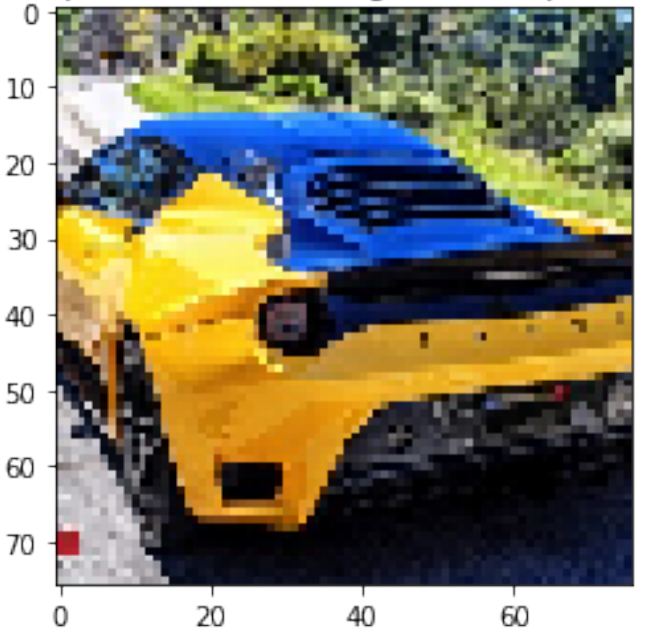


Image 2 downsample for: linear interpolation, with ratio: 0.3

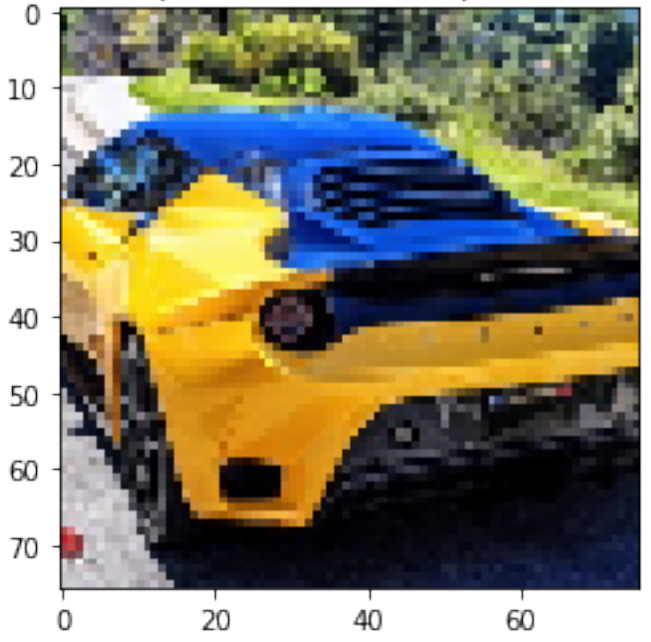


Image 2 downsample for: bicubic interpolation, with ratio: 0.3

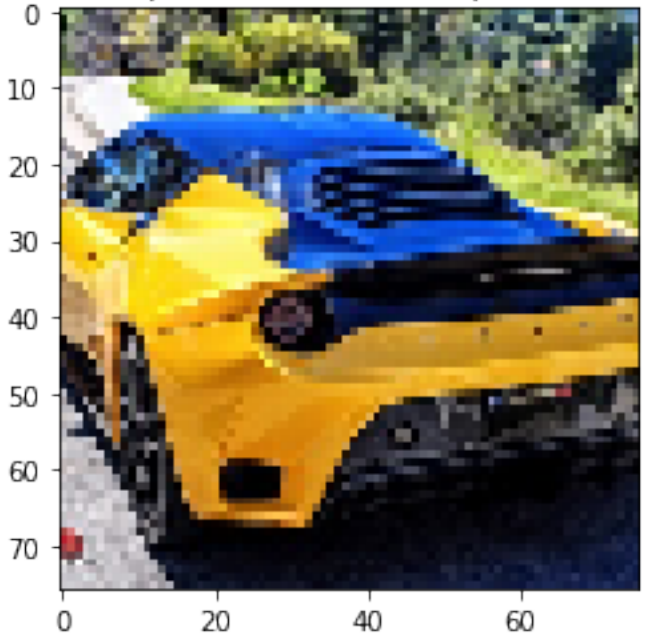


Image 3 downsample for: nearest neighbor interpolation, with ratio: 0.3

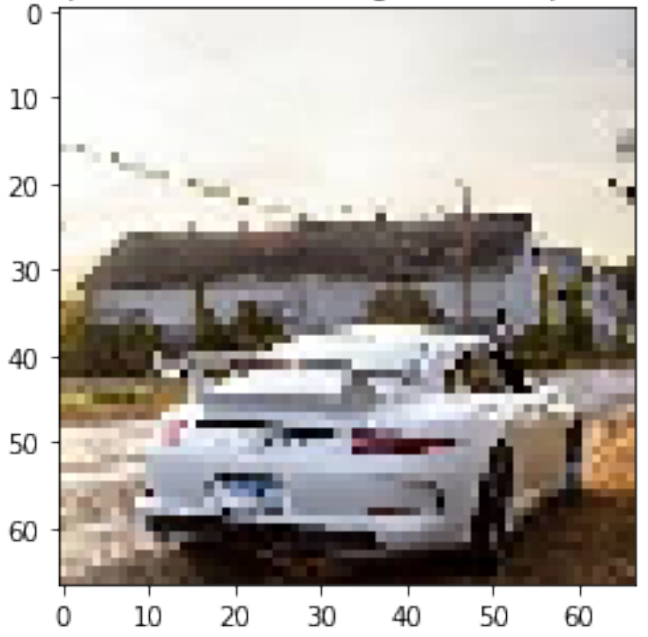


Image 3 downsample for: linear interpolation, with ratio: 0.3

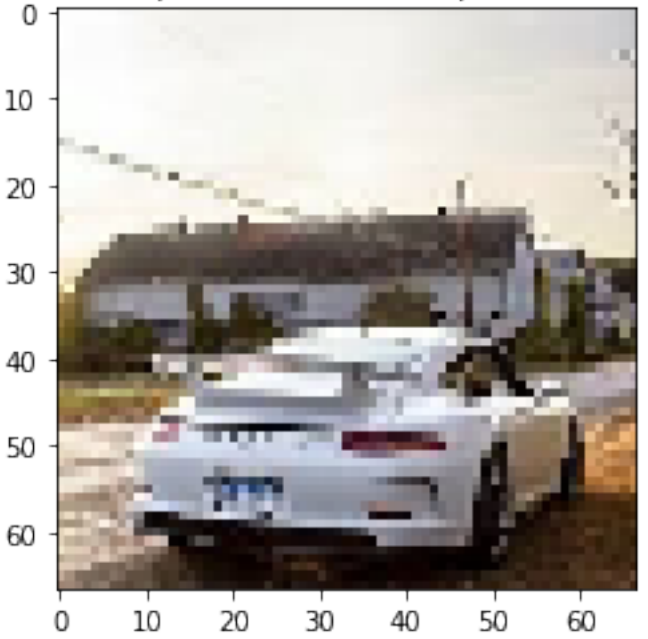


Image 3 downsample for: bicubic interpolation, with ratio: 0.3

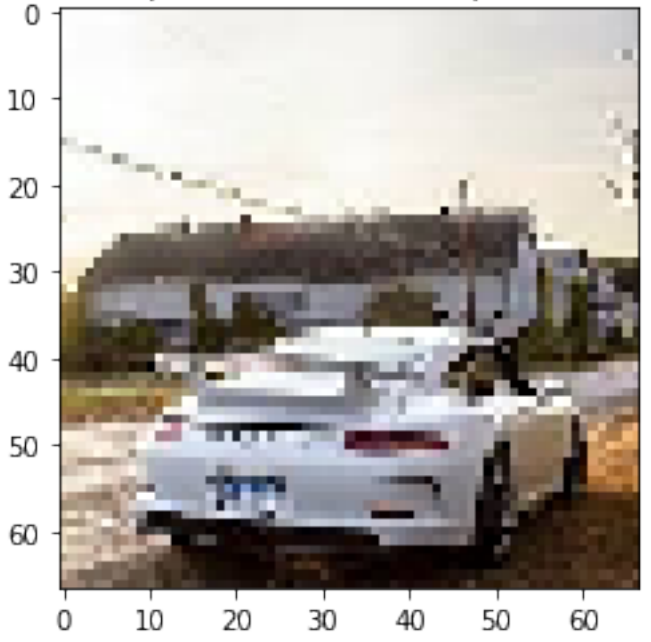


Image 1 downsample for: nearest neighbor interpolation, with ratio: 0.5

Image 1 downsample for: linear interpolation, with ratio: 0.5

Image 1 downsample for: bicubic interpolation, with ratio: 0.5

Image 2 downsample for: nearest neighbor interpolation, with ratio: 0.5

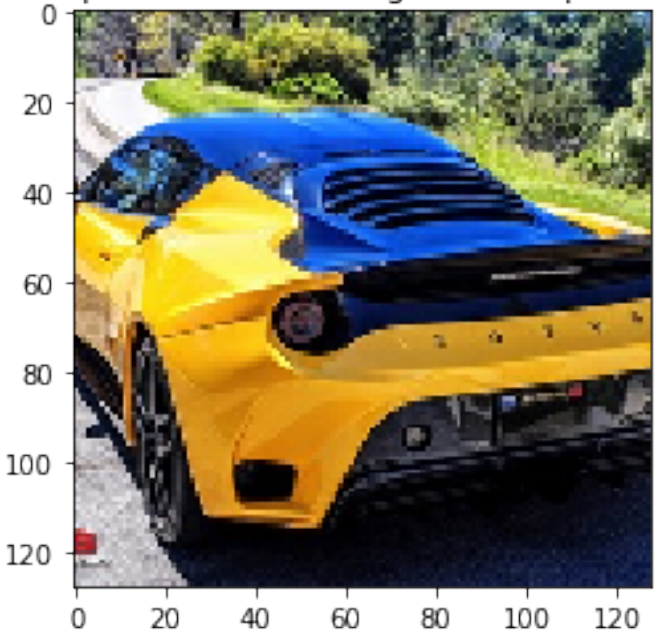


Image 2 downsample for: linear interpolation, with ratio: 0.5

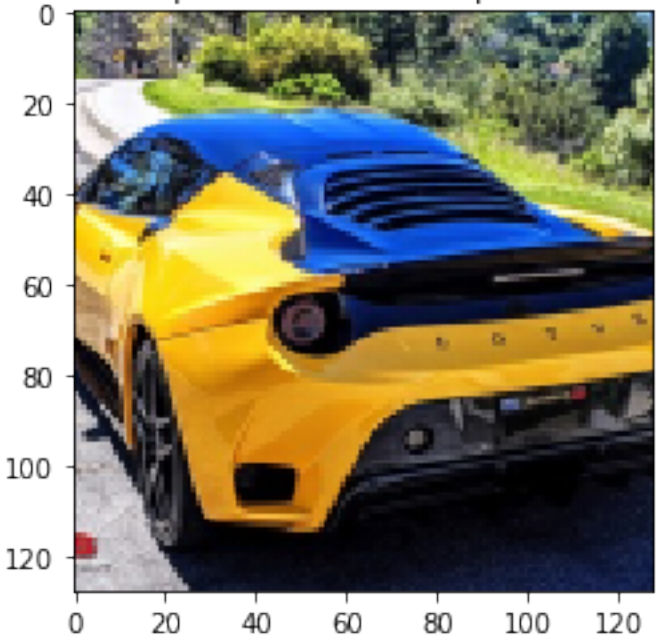


Image 2 downsample for: bicubic interpolation, with ratio: 0.5

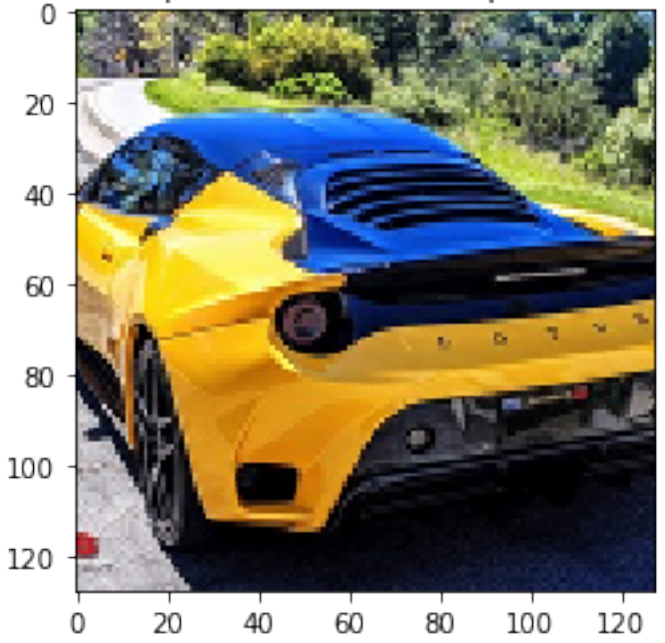


Image 3 downsample for: nearest neighbor interpolation, with ratio: 0.5

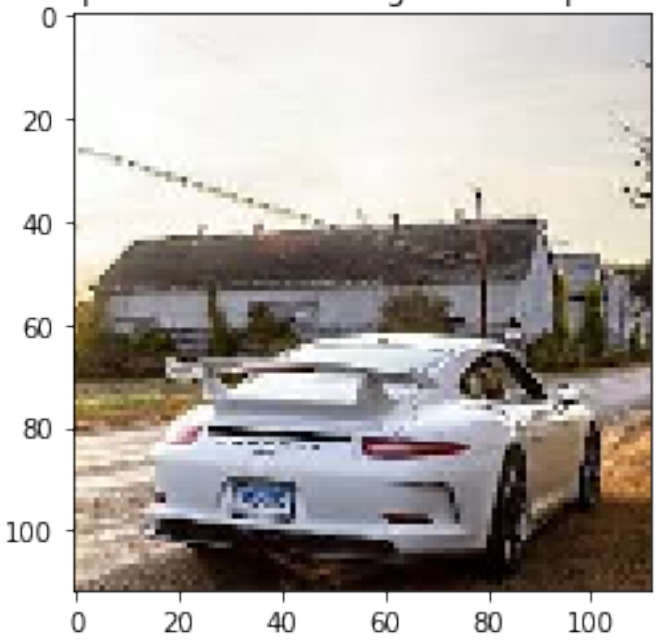


Image 3 downsample for: linear interpolation, with ratio: 0.5

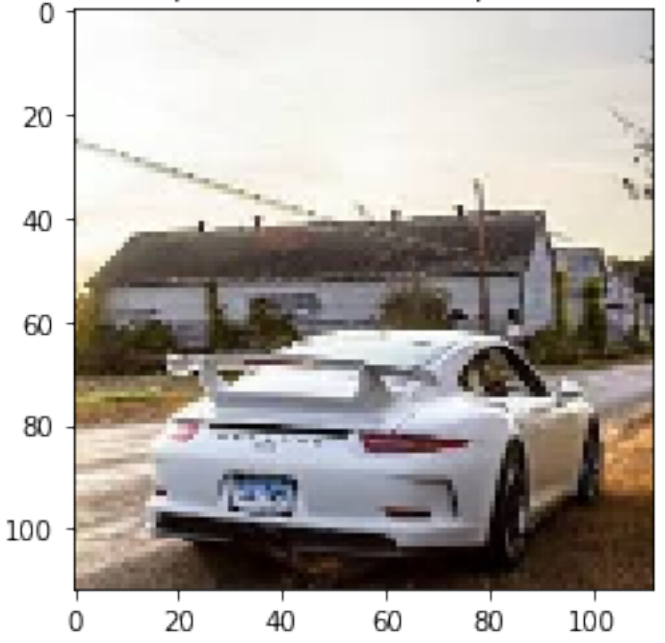


Image 3 downsample for: bicubic interpolation, with ratio: 0.5

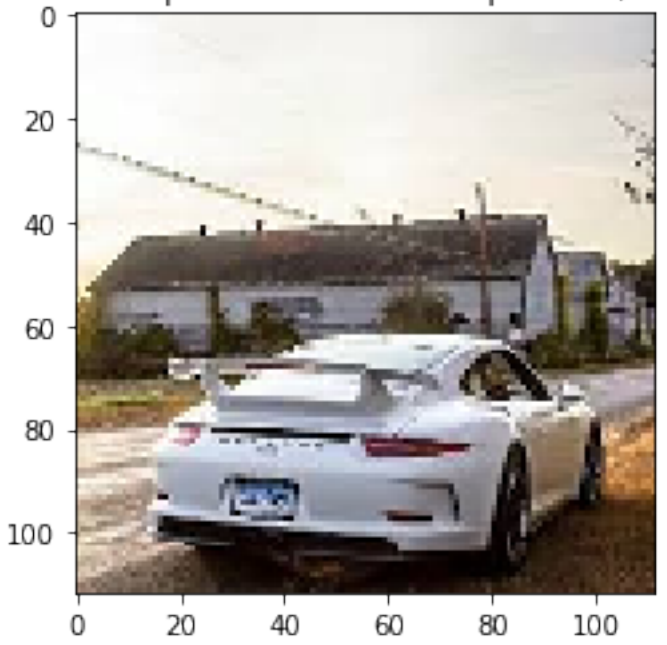


Image 1 downsample for: nearest neighbor interpolation, with ratio: 0.7

Image 1 downsample for: linear interpolation, with ratio: 0.7

Image 1 downsample for: bicubic interpolation, with ratio: 0.7

Image 2 downsample for: nearest neighbor interpolation, with ratio: 0.7

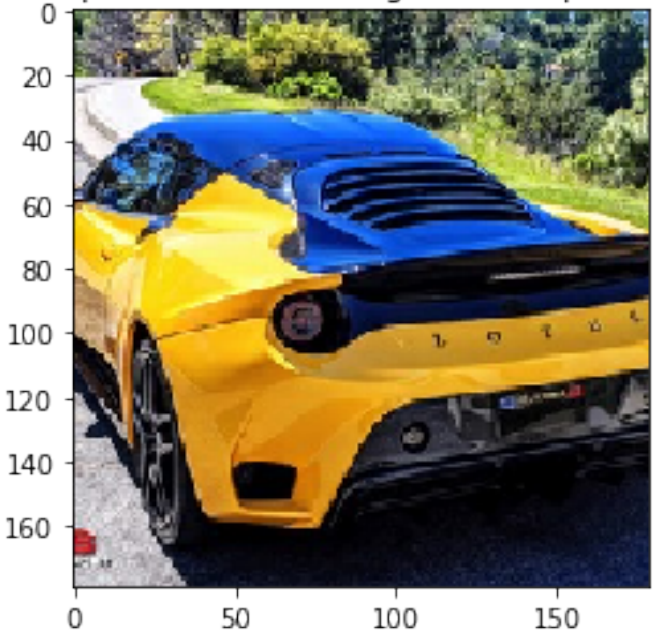


Image 2 downsample for: linear interpolation, with ratio: 0.7

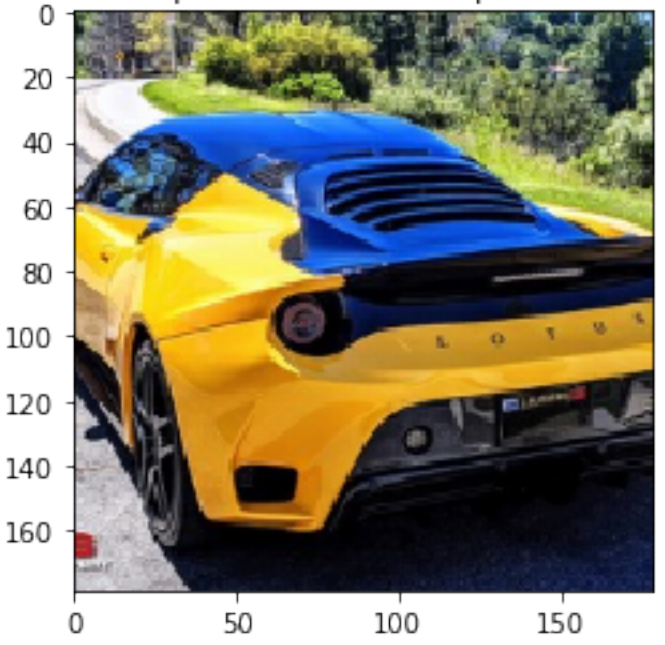


Image 2 downsample for: bicubic interpolation, with ratio: 0.7

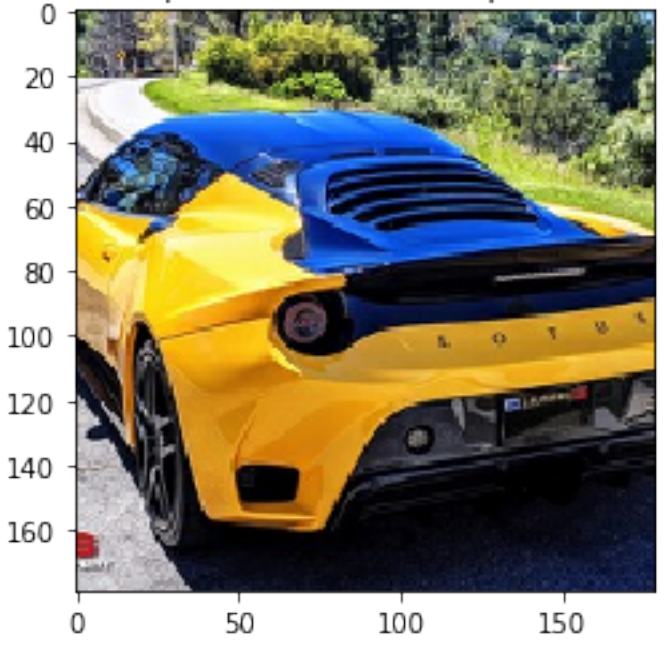


Image 3 downsample for: nearest neighbor interpolation, with ratio: 0.7

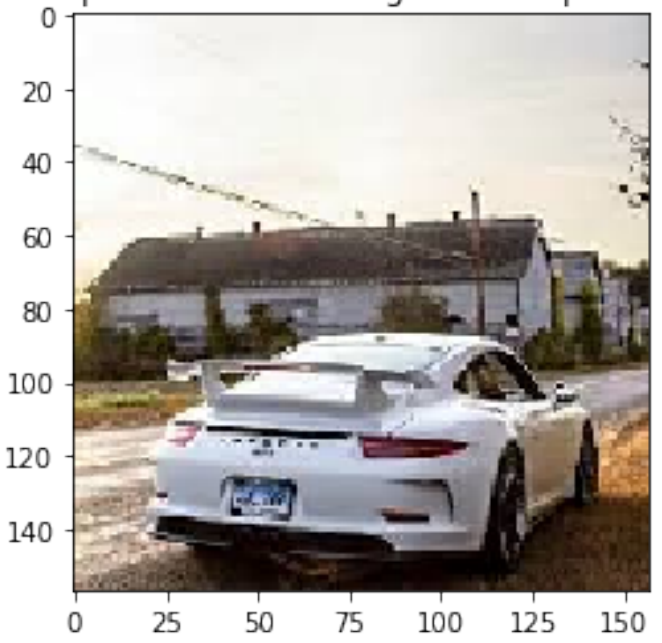
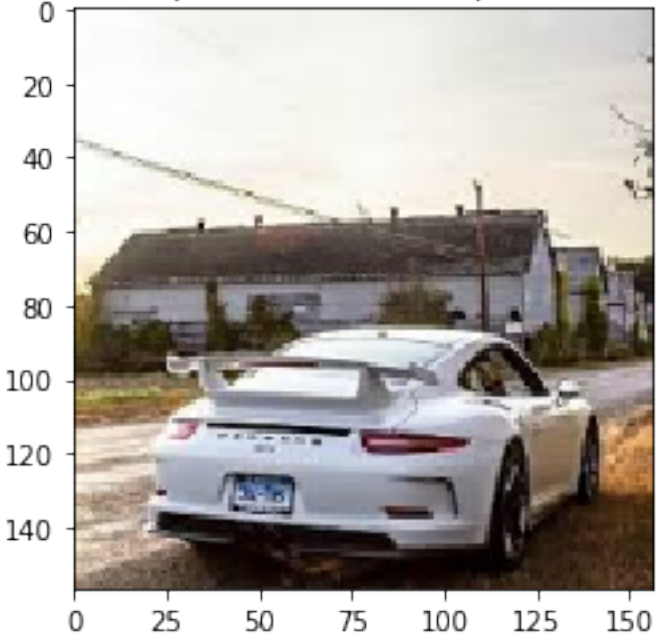
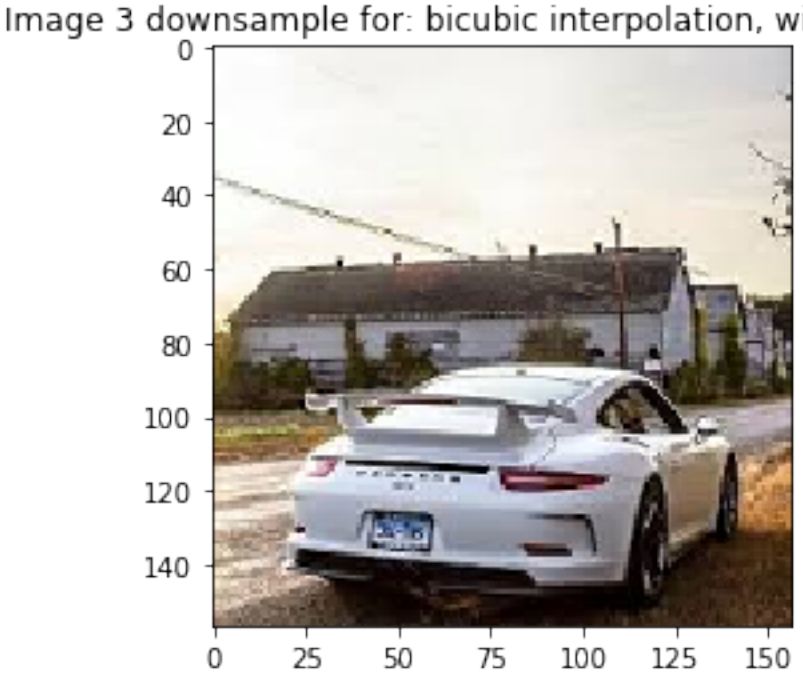


Image 3 downsample for: linear interpolation, with ratio: 0.7





In [ ]:

```
'''
Discussion for First part of problem 5
Rather than discuss each of the 27 images, this author will leav
e it up to the reader to
visually examine the 27 images and will instead focus on what is
beleived to be the goal
of this problem: identifying and discussing the differences in t
he results for each
interpolation method and why they are so.

Firstly, the biggest differences from the original image can be
seen in the images with
the most downsampling (0.3), which makes sense those are the lar
gest altered from the original,
and hence the affects of the downsampling are the most noticable
. It should be noted, however,
that the affects of downsampling are still noticable at the rati
o of 0.7.
These affects are blockiness, as essentially the amount of pixel
s in the image was changed,
and these new pixels were then assigned values from the original
image based on the interpolation
method used. Clearly, there will be blockiness when this is done
,as can clearly be seen, as there
is both a reduction in pixels, and an estimation of what the val
ues of the new pixels should be.
```

```
In terms of interpolation methods, it was consistent across all pictures and methods that the
result quality (essentially how closely it resembled the original image, less blocky means higher
quality) was the best for the highest ratio (0.7), followed by the (0.5), and lowest at (0.3),
which makes sense as that order is least changed resolution to most changed. Across all pictures,
the nearest neighbor interpolation was the lowest quality, which makes sense as it is the simplest.
This is most visible in areas of high contrast, which as shrubbery or lines. The linear interpolation
was a step above, which was expected, and the bicubic was the best. This makes sense as bicubic takes
into consideration the most information, so it works the best. These differences in quality were most
noticable at the lowest ratio of 0.3.
'''
```

In [34]:

```
'''
Problem 5
'''

## Upsample
ratio2 = np.array([1.5,1.7,2])
k=28
for i in ratio2:
    image = image1
    sz = np.shape(image)
    resize= cv2.resize(image, (int(i*sz[1]),int(i*sz[0])), interpolation=cv2.INTER_NEAREST)
    plt.figure(k)
    #plt.figure(num=None, figsize=(8, 8))
    title = 'Image 1 upsample for: nearest neighbor interpolation, with ratio: ' + str(i)
    plt.title(title)
    plt.imshow(resize)
    name = 'Fig: ' + str(k)
    plt.savefig(name)
    k=k+1
    sz = np.shape(image)
```

```
    resize= cv2.resize(image, (int(i*sz[1]),int(i*sz[0])), inter
polation=cv2.INTER_LINEAR)
    plt.figure(k)
    #plt.figure(num=None, figsize=(8, 8))
    title = 'Image 1 upsample for: linear interpolation, with ra
tio: ' + str(i)
    plt.title(title)
    plt.imshow(resize)
    name = 'Fig: ' + str(k)
    plt.savefig(name)
    k=k+1
    sz = np.shape(image)
    resize= cv2.resize(image, (int(i*sz[1]),int(i*sz[0])), inter
polation=cv2.INTER_CUBIC)
    plt.figure(k)
    #plt.figure(num=None, figsize=(8, 8))
    title = 'Image 1 upsample for: bicubic interpolation, with r
atio: ' + str(i)
    plt.title(title)
    plt.imshow(resize)
    name = 'Fig: ' + str(k)
    plt.savefig(name)
    k=k+1
    ##############################
    image = image2
    sz = np.shape(image)
    resize= cv2.resize(image, (int(i*sz[1]),int(i*sz[0])), inter
polation=cv2.INTER_NEAREST)
    plt.figure(k)
    #plt.figure(num=None, figsize=(8, 8))
    title = 'Image 2 upsample for: nearest neighbor interpolatio
n, with ratio: ' + str(i)
    plt.title(title)
    plt.imshow(resize)
    name = 'Fig: ' + str(k)
    plt.savefig(name)
    k=k+1
    sz = np.shape(image)
    resize= cv2.resize(image, (int(i*sz[1]),int(i*sz[0])), inter
polation=cv2.INTER_LINEAR)
    plt.figure(k)
    #plt.figure(num=None, figsize=(8, 8))
    title = 'Image 2 upsample for: linear interpolation, with ra
tio: ' + str(i)
```

```
    plt.title(title)
    plt.imshow(resize)
    name = 'Fig: ' + str(k)
    plt.savefig(name)
    k=k+1
    sz = np.shape(image)
    resize= cv2.resize(image, (int(i*sz[1]),int(i*sz[0])), inter
polation=cv2.INTER_CUBIC)
    plt.figure(k)
    #plt.figure(num=None, figsize=(8, 8))
    title = 'Image 2 upsample for: bicubic interpolation, with r
atio: ' + str(i)
    plt.title(title)
    plt.imshow(resize)
    name = 'Fig: ' + str(k)
    plt.savefig(name)
    k=k+1
    ##############################
    image = image3
    sz = np.shape(image)
    resize= cv2.resize(image, (int(i*sz[1]),int(i*sz[0])), inter
polation=cv2.INTER_NEAREST)
    plt.figure(k)
    #plt.figure(num=None, figsize=(8, 8))
    title = 'Image 3 upsample for: nearest neighbor interpolatio
n, with ratio: ' + str(i)
    plt.title(title)
    plt.imshow(resize)
    name = 'Fig: ' + str(k)
    plt.savefig(name)
    k=k+1
    sz = np.shape(image)
    resize= cv2.resize(image, (int(i*sz[1]),int(i*sz[0])), inter
polation=cv2.INTER_LINEAR)
    plt.figure(k)
    #plt.figure(num=None, figsize=(8, 8))
    title = 'Image 3 upsample for: linear interpolation, with ra
tio: ' + str(i)
    plt.title(title)
    plt.imshow(resize)
    name = 'Fig: ' + str(k)
    plt.savefig(name)
    k=k+1
    sz = np.shape(image)
    resize= cv2.resize(image, (int(i*sz[1]),int(i*sz[0])), inter
```

```
polation=cv2.INTER_CUBIC)
    plt.figure(k)
    #plt.figure(num=None, figsize=(8, 8))
    title = 'Image 3 upsample for: bicubic interpolation, with r
atio: ' + str(i)
    plt.title(title)
    plt.imshow(resize)
    name = 'Fig: ' + str(k)
    plt.savefig(name)
    k=k+1
```

```
/Users/femw90/opt/anaconda3/lib/python3.7/site-packa
ges/ipykernel_launcher.py:32: RuntimeWarning: More t
han 20 figures have been opened. Figures created thr
ough the pyplot interface (`matplotlib.pyplot.figure
`) are retained until explicitly closed and may cons
ume too much memory. (To control this warning, see t
he rcParam `figure.max_open_warning`).
/Users/femw90/opt/anaconda3/lib/python3.7/site-packa
ges/ipykernel_launcher.py:44: RuntimeWarning: More t
han 20 figures have been opened. Figures created thr
ough the pyplot interface (`matplotlib.pyplot.figure
`) are retained until explicitly closed and may cons
ume too much memory. (To control this warning, see t
he rcParam `figure.max_open_warning`).
/Users/femw90/opt/anaconda3/lib/python3.7/site-packa
ges/ipykernel_launcher.py:54: RuntimeWarning: More t
han 20 figures have been opened. Figures created thr
ough the pyplot interface (`matplotlib.pyplot.figure
`) are retained until explicitly closed and may cons
ume too much memory. (To control this warning, see t
he rcParam `figure.max_open_warning`).
/Users/femw90/opt/anaconda3/lib/python3.7/site-packa
ges/ipykernel_launcher.py:64: RuntimeWarning: More t
han 20 figures have been opened. Figures created thr
ough the pyplot interface (`matplotlib.pyplot.figure
`) are retained until explicitly closed and may cons
ume too much memory. (To control this warning, see t
he rcParam `figure.max_open_warning`).
/Users/femw90/opt/anaconda3/lib/python3.7/site-packa
ges/ipykernel_launcher.py:76: RuntimeWarning: More t
han 20 figures have been opened. Figures created thr
ough the pyplot interface (`matplotlib.pyplot.figure
`) are retained until explicitly closed and may cons
ume too much memory. (To control this warning, see t
```

```
he rcParam `figure.max_open_warning`).
/Users/femw90/opt/anaconda3/lib/python3.7/site-packa
ges/ipykernel_launcher.py:86: RuntimeWarning: More t
han 20 figures have been opened. Figures created thr
ough the pyplot interface (`matplotlib.pyplot.figure
`) are retained until explicitly closed and may cons
ume too much memory. (To control this warning, see t
he rcParam `figure.max_open_warning`).
/Users/femw90/opt/anaconda3/lib/python3.7/site-packa
ges/ipykernel_launcher.py:96: RuntimeWarning: More t
han 20 figures have been opened. Figures created thr
ough the pyplot interface (`matplotlib.pyplot.figure
`) are retained until explicitly closed and may cons
ume too much memory. (To control this warning, see t
he rcParam `figure.max_open_warning`).
```

Image 1 upsample for: bicubic interpolation, with ratio: 1.5

Image 2 upsample for: nearest neighbor interpolation, with ratio: 1.5

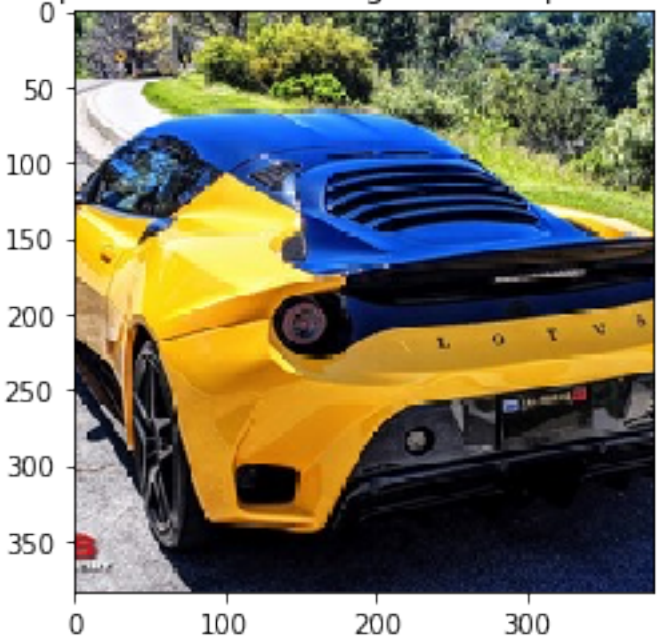


Image 2 upsample for: linear interpolation, with ratio: 1.5

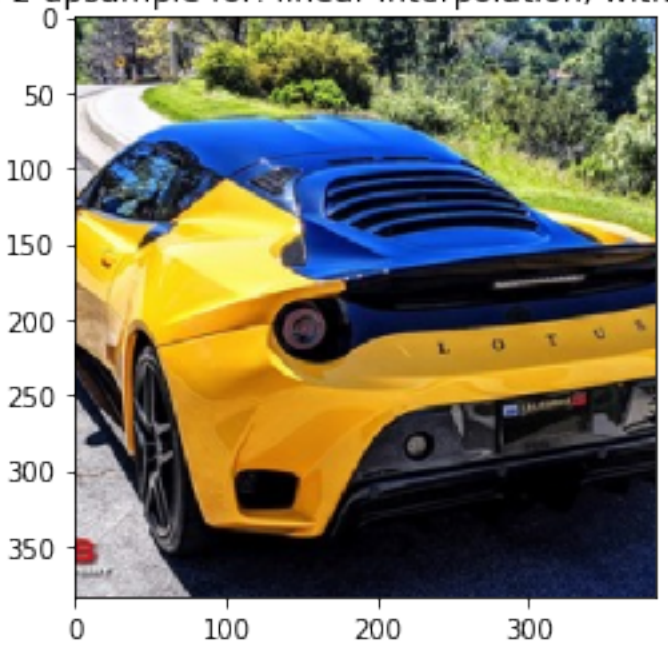


Image 2 upsample for: bicubic interpolation, with ratio: 1.5

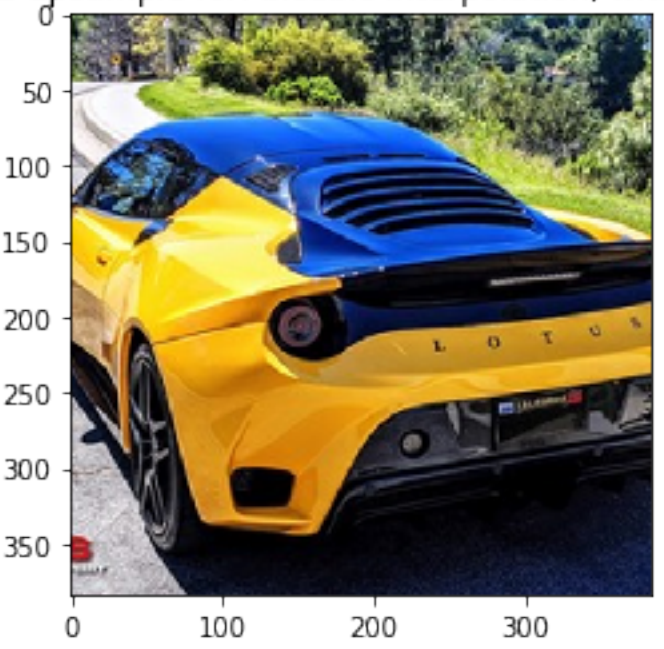


Image 3 upsample for: nearest neighbor interpolation, with ratio: 1.5

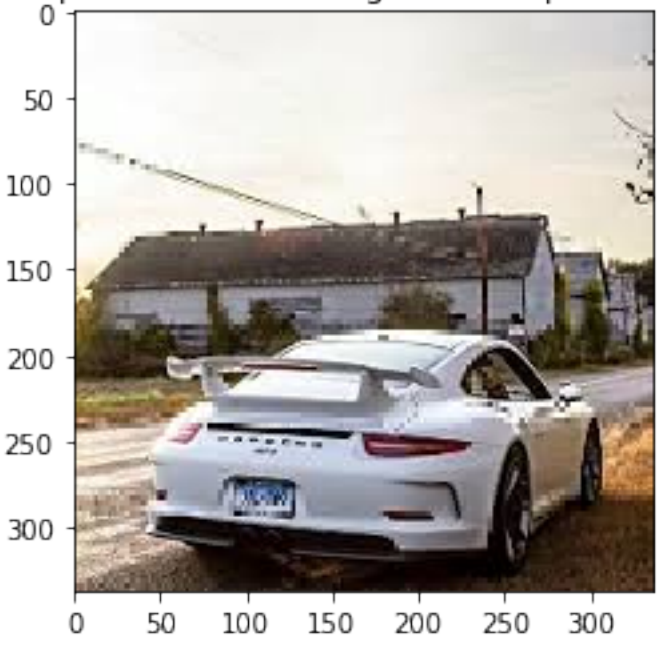


Image 3 upsample for: linear interpolation, with ratio: 1.5

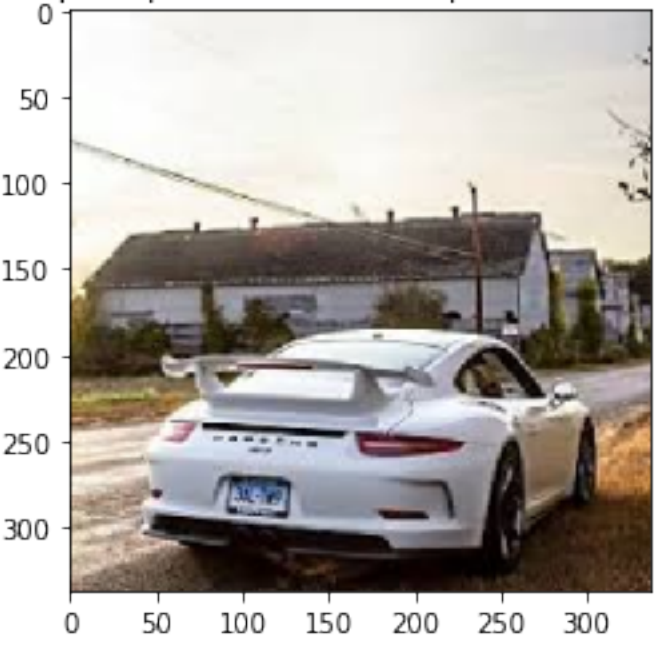


Image 3 upsample for: bicubic interpolation, with ratio: 1.5

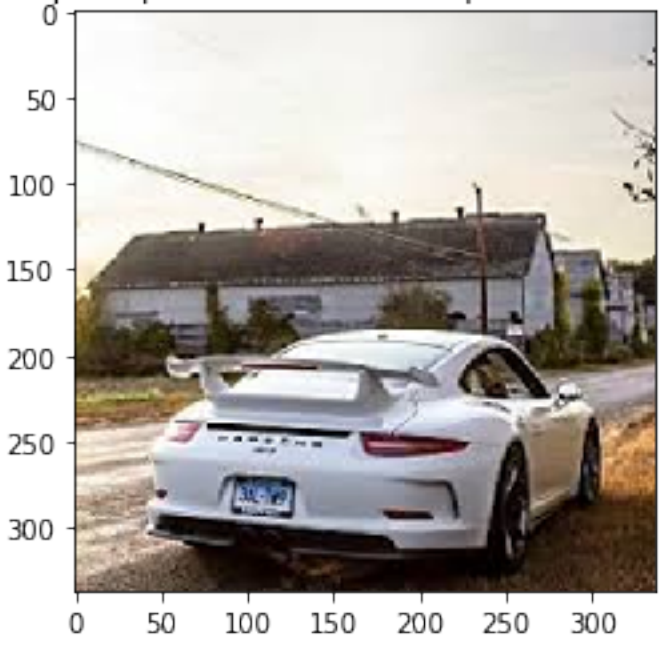


Image 1 upsample for: nearest neighbor interpolation, with ratio: 1.7


Image 1 upsample for: linear interpolation, with ratio: 1.7
0
50
100
150
200
250
0
100
200
300
400
500


Image 1 upsample for: bicubic interpolation, with ratio: 1.7
0
50
100
150
200
250
0
100
200
300
400
500

Image 2 upsample for: nearest neighbor interpolation, with ratio: 1.7

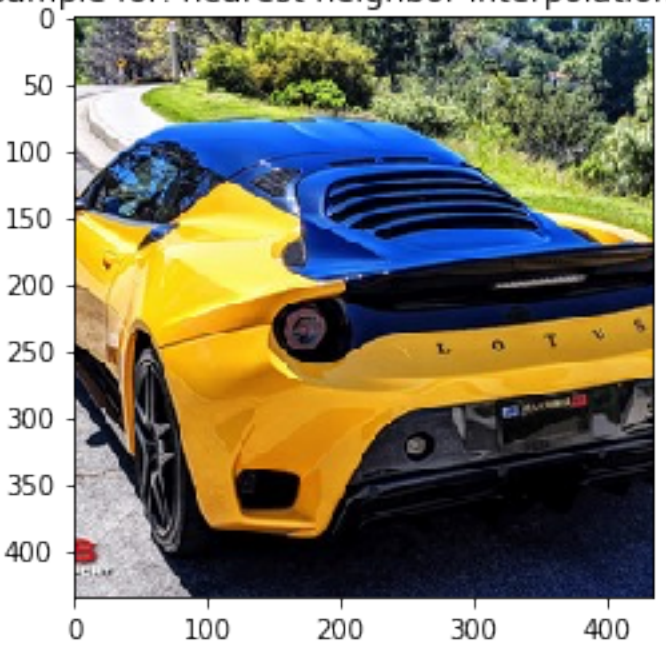


Image 2 upsample for: linear interpolation, with ratio: 1.7

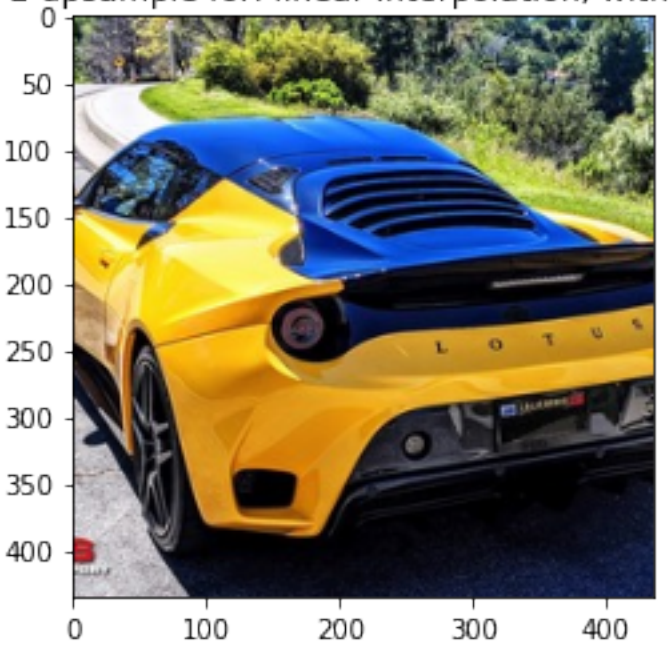


Image 2 upsample for: bicubic interpolation, with ratio: 1.7

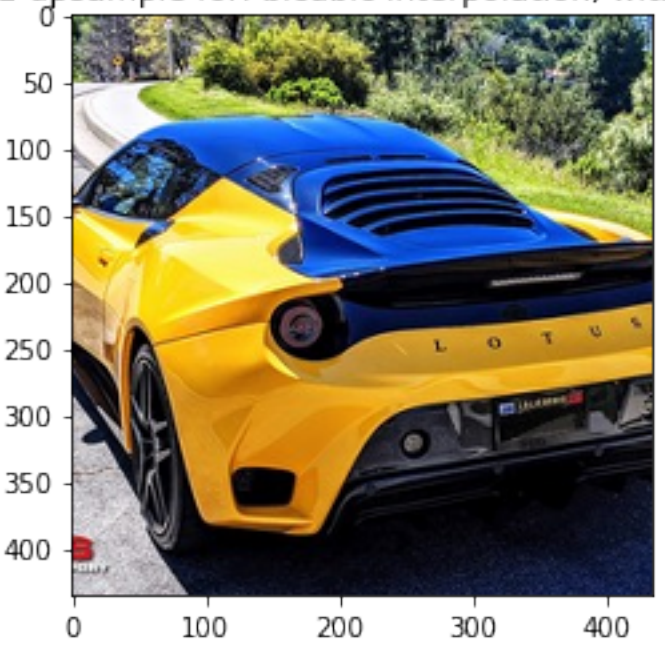


Image 3 upsample for: nearest neighbor interpolation, with ratio: 1.7

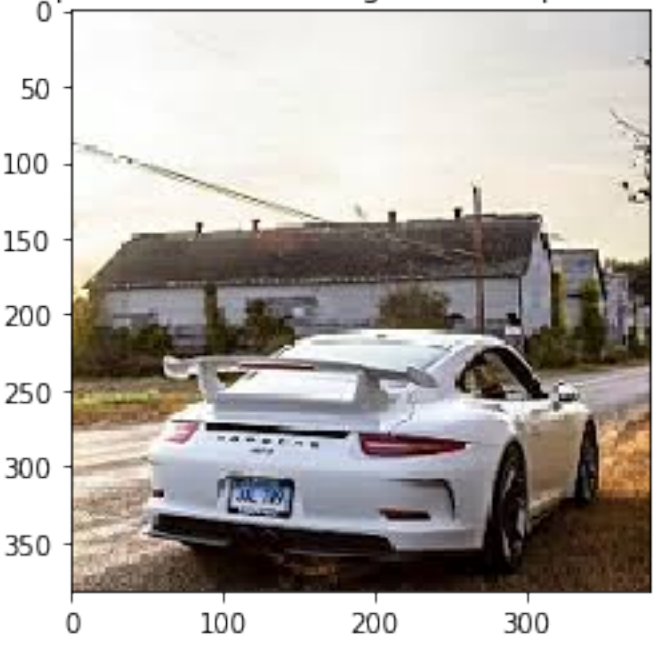


Image 3 upsample for: linear interpolation, with ratio: 1.7

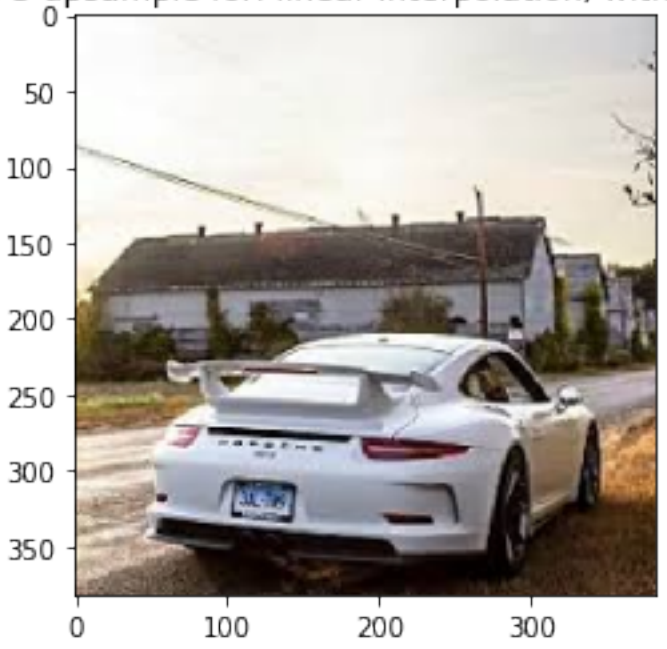


Image 3 upsample for: bicubic interpolation, with ratio: 1.7

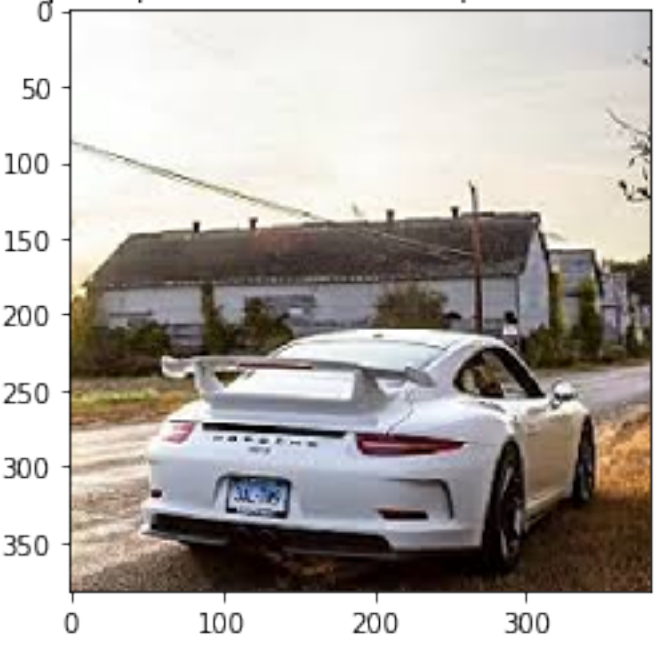


Image 1 upsample for: nearest neighbor interpolation, with ratio: 2.0

Image 1 upsample for: linear interpolation, with ratio: 2.0

Image 1 upsample for: bicubic interpolation, with ratio: 2.0

Image 2 upsample for: nearest neighbor interpolation, with ratio: 2.0

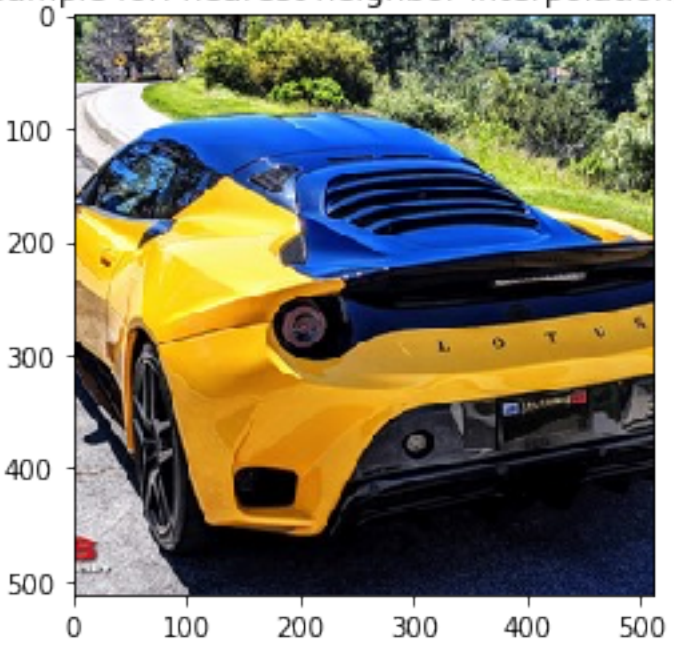


Image 2 upsample for: linear interpolation, with ratio: 2.0

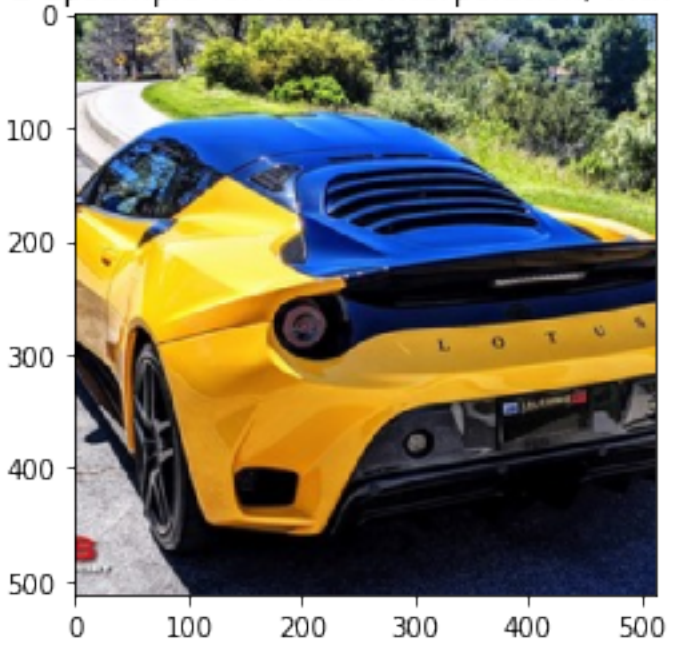


Image 2 upsample for: bicubic interpolation, with ratio: 2.0

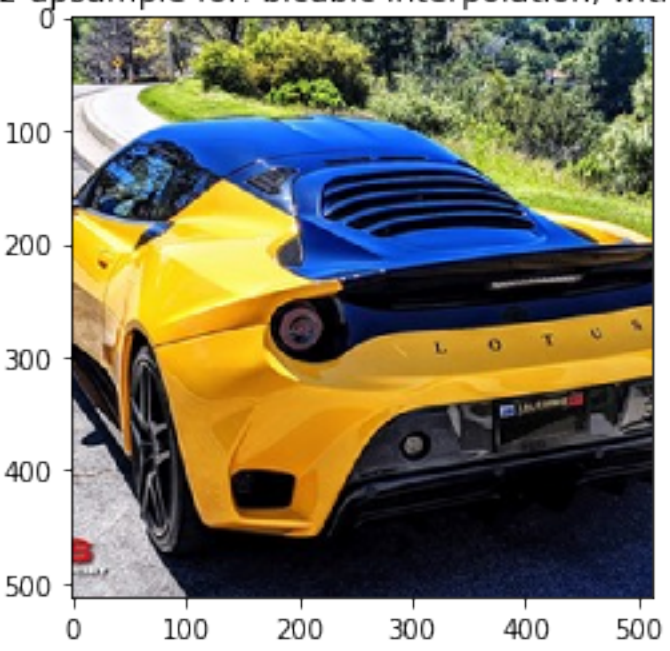


Image 3 upsample for: nearest neighbor interpolation, with ratio: 2.0

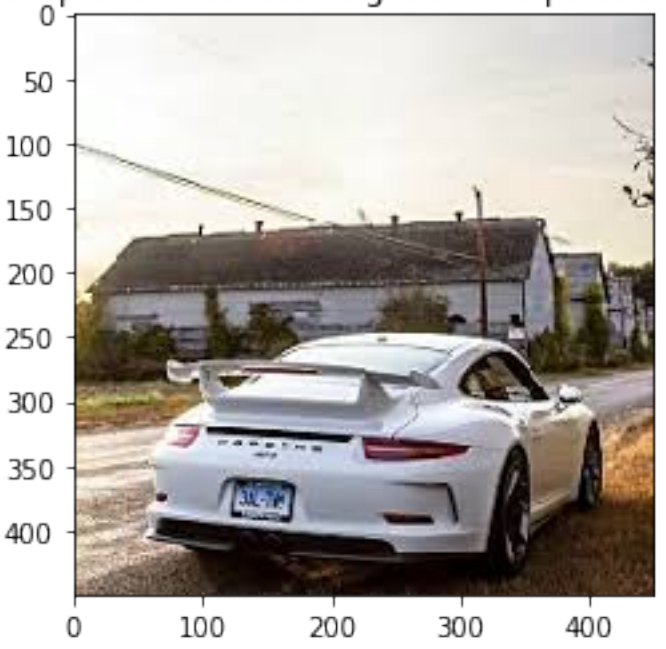


Image 3 upsample for: linear interpolation, with ratio: 2.0

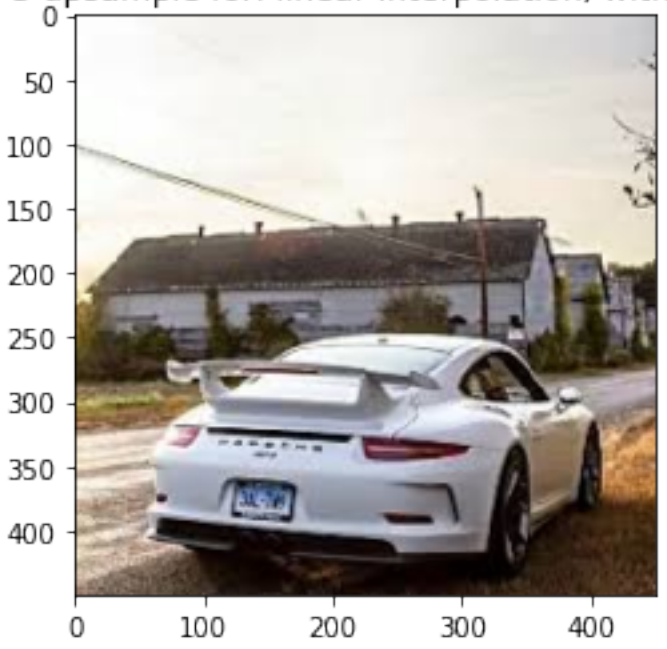
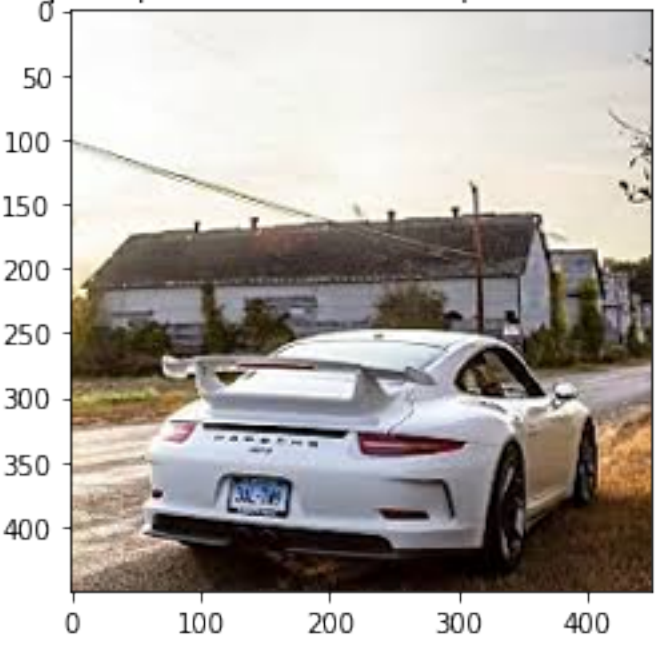


Image 3 upsample for: bicubic interpolation, with ratio: 2.0



In [ ]:

```
'''
Discussion for Second part of problem 5

The results of this part are as follows. There is a decrease in
quality (blockiness) in upsampling compared
to the original picture. This makes sense, as new pixels were cr
eated in the upsample and have to be estimated
from the pixels around them, resulting in some blockiness.
Upsampling with ratio 2.0 visually seemed to look the best, like
ly as the size was doubled so the result
is more likely to produce a smoother image. This follows with wh
at was seem in the first part.
The quality from best to worst of methods is bicubic, linear, an
d nearest neighbor. This is expected, and in line
with what was seen before as bicubic uses the most infomation, r
esulting in a smooth image, and linear uses the
least, creating the most blockiness. This blockiness is most app
arent in image 1, along the lines, as the new
pixels are populated with estimates from the interpolation metho
d used, and since bicubic gives the smoothest
results, it works the best.
'''
```

In [39]:

```
'''
Problem 5
'''
#Downsample then upsample
k=55

image = image1
sz = np.shape(image)
resize= cv2.resize(image, (int(.1*sz[1]),int(.1*sz[0])), interpo
lation=cv2.INTER_NEAREST)
resize= cv2.resize(image, (int(sz[1]),int(sz[0])), interpolation
=cv2.INTER_NEAREST)
plt.figure(k)
#plt.figure(num=None, figsize=(8, 8))
title = 'Image 1 downsample, then upsample for: nearest neighbor
interpolation'
plt.title(title)
```

```
plt.imshow(resize)

name = 'Fig: ' + str(k)
plt.savefig(name)
k=k+1
sz = np.shape(image)
resize= cv2.resize(image, (int(.1*sz[1]),int(.1*sz[0])), interpo
lation=cv2.INTER_LINEAR)
resize= cv2.resize(image, (int(sz[1]),int(sz[0])), interpolation
=cv2.INTER_LINEAR)
plt.figure(k)
#plt.figure(num=None, figsize=(8, 8))
title = 'Image 1 downsample, then upsample for: linear interpola
tion'
plt.title(title)
plt.imshow(resize)
name = 'Fig: ' + str(k)
plt.savefig(name)
k=k+1
sz = np.shape(image)
resize= cv2.resize(image, (int(.1*sz[1]),int(.1*sz[0])), interpo
lation=cv2.INTER_CUBIC)
resize= cv2.resize(image, (int(sz[1]),int(sz[0])), interpolation
=cv2.INTER_CUBIC)
plt.figure(k)
#plt.figure(num=None, figsize=(8, 8))
title = 'Image 1 downsample, then upsample for: bicubic interpol
ation'
plt.title(title)
plt.imshow(resize)
name = 'Fig: ' + str(k)
plt.savefig(name)
k=k+1

image = image2
sz = np.shape(image)
resize= cv2.resize(image, (int(.1*sz[1]),int(.1*sz[0])), interpo
lation=cv2.INTER_NEAREST)
resize= cv2.resize(image, (int(sz[1]),int(sz[0])), interpolation
=cv2.INTER_NEAREST)
plt.figure(k)
#plt.figure(num=None, figsize=(8, 8))
title = 'Image 2 downsample, then upsample for: nearest neighbor
interpolation'
plt.title(title)
```

```
plt.imshow(resize)
name = 'Fig: ' + str(k)
plt.savefig(name)
k=k+1
sz = np.shape(image)
resize= cv2.resize(image, (int(.1*sz[1]),int(.1*sz[0])), interpo
lation=cv2.INTER_LINEAR)
resize= cv2.resize(image, (int(sz[1]),int(sz[0])), interpolation
=cv2.INTER_LINEAR)
plt.figure(k)
#plt.figure(num=None, figsize=(8, 8))
title = 'Image 2 downsample, then upsample for: linear interpola
tion'
plt.title(title)
plt.imshow(resize)
name = 'Fig: ' + str(k)
plt.savefig(name)
k=k+1
sz = np.shape(image)
resize= cv2.resize(image, (int(.1*sz[1]),int(.1*sz[0])), interpo
lation=cv2.INTER_CUBIC)
resize= cv2.resize(image, (int(sz[1]),int(sz[0])), interpolation
=cv2.INTER_CUBIC)
plt.figure(k)
#plt.figure(num=None, figsize=(8, 8))
title = 'Image 2 downsample, then upsample for: bicubic interpol
ation'
plt.title(title)
plt.imshow(resize)
name = 'Fig: ' + str(k)
plt.savefig(name)
k=k+1

image = image3
sz = np.shape(image)
resize= cv2.resize(image, (int(.1*sz[1]),int(.1*sz[0])), interpo
lation=cv2.INTER_NEAREST)
resize= cv2.resize(image, (int(sz[1]),int(sz[0])), interpolation
=cv2.INTER_NEAREST)
plt.figure(k)
#plt.figure(num=None, figsize=(8, 8))
title = 'Image 3 downsample, then upsample for: nearest neighbor
interpolation'
plt.title(title)
plt.imshow(resize)
```

```
name = 'Fig: ' + str(k)
plt.savefig(name)
k=k+1
sz = np.shape(image)
resize= cv2.resize(image, (int(.1*sz[1]),int(.1*sz[0])), interpo
lation=cv2.INTER_LINEAR)
resize= cv2.resize(image, (int(sz[1]),int(sz[0])), interpolation
=cv2.INTER_LINEAR)
plt.figure(k)
#plt.figure(num=None, figsize=(8, 8))
title = 'Image 3 downsample, then upsample for: linear interpola
tion'
plt.title(title)
plt.imshow(resize)
name = 'Fig: ' + str(k)
plt.savefig(name)
k=k+1
sz = np.shape(image)
resize= cv2.resize(image, (int(.1*sz[1]),int(.1*sz[0])), interpo
lation=cv2.INTER_CUBIC)
resize= cv2.resize(image, (int(sz[1]),int(sz[0])), interpolation
=cv2.INTER_CUBIC)
plt.figure(k)
#plt.figure(num=None, figsize=(8, 8))
title = 'Image 3 downsample, then upsample for: bicubic interpol
ation'
plt.title(title)
plt.imshow(resize)
name = 'Fig: ' + str(k)
plt.savefig(name)
k=k+1
```

Image 1 downsample, then upsample for: nearest neighbor interpolation

Image 1 downsample, then upsample for: linear interpolation

Image 1 downsample, then upsample for: bicubic interpolation

Image 2 downsample, then upsample for: nearest neighbor interpolation

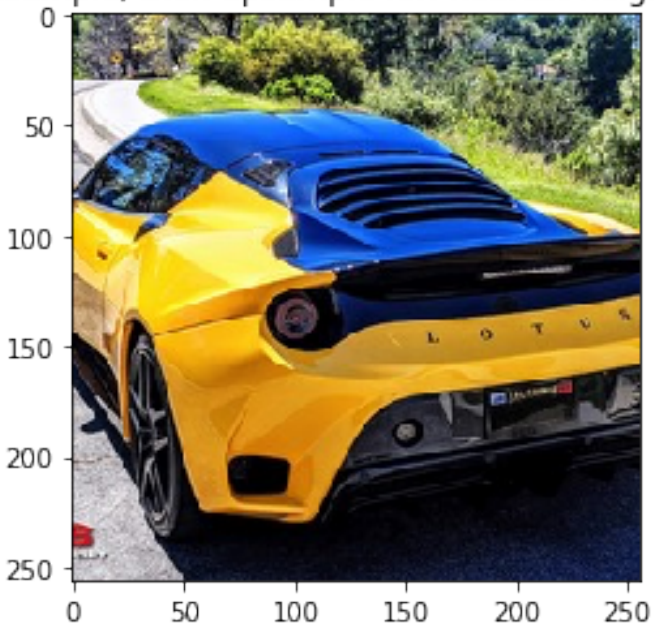


Image 2 downsample, then upsample for: linear interpolation

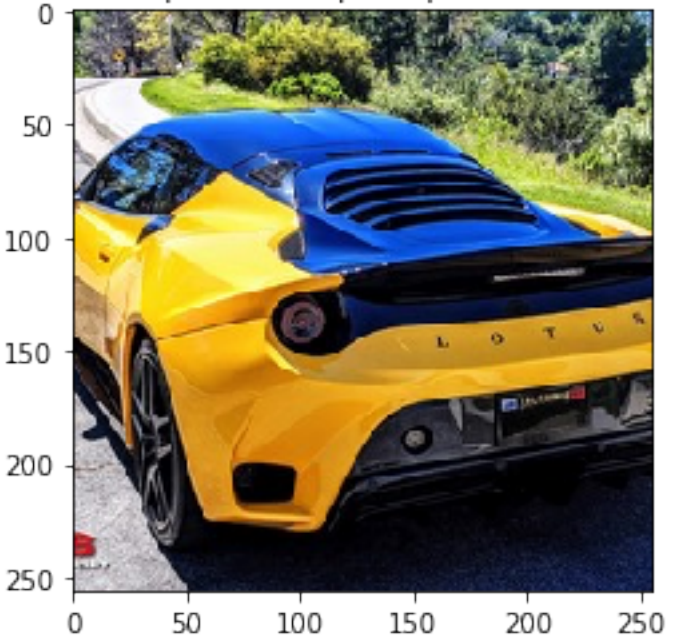


Image 2 downsample, then upsample for: bicubic interpolation

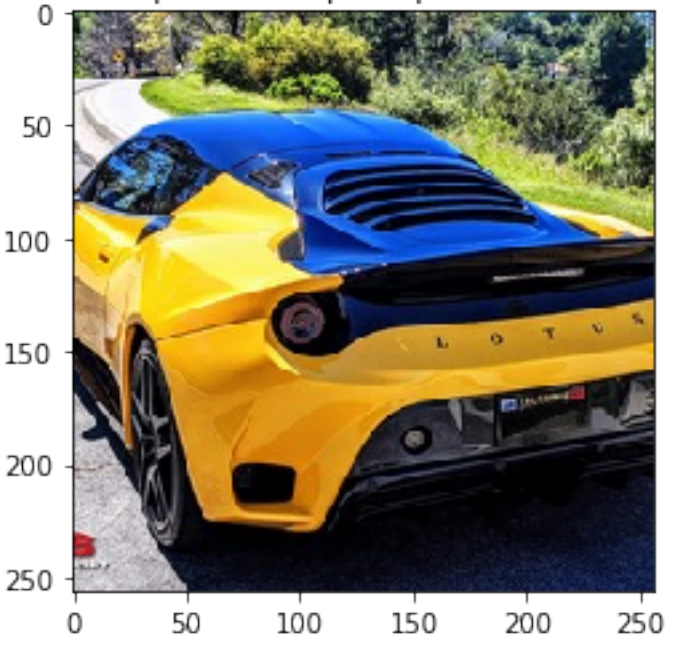


Image 3 downsample, then upsample for: nearest neighbor interpolation

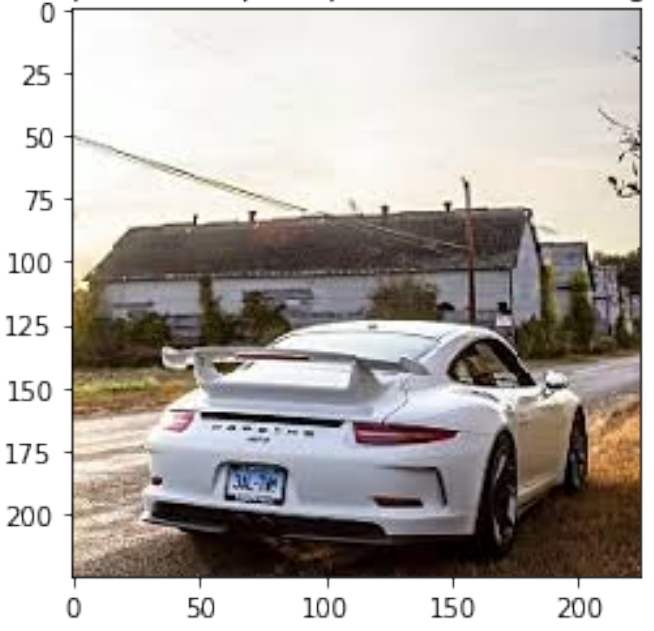


Image 3 downsample, then upsample for: linear interpolation

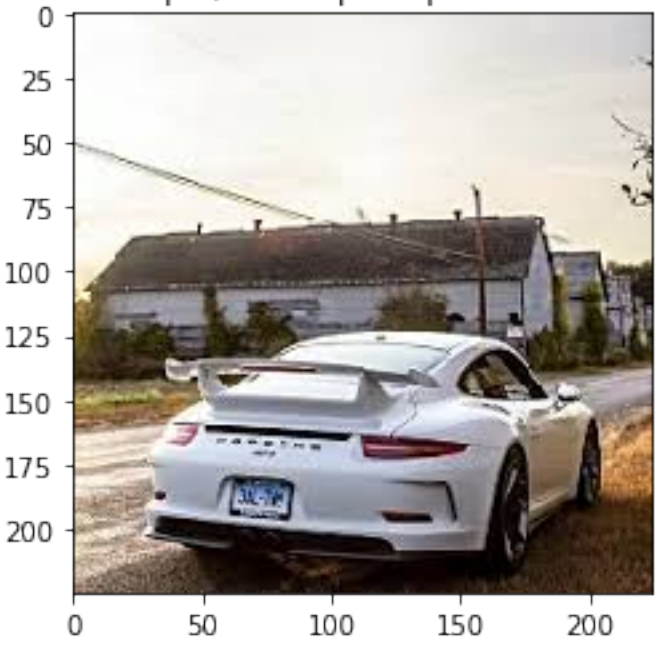
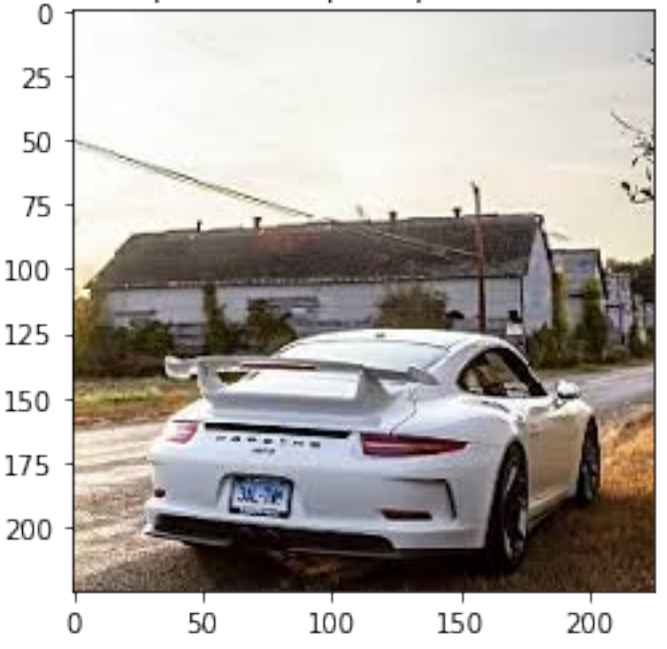


Image 3 downsample, then upsample for: bicubic interpolation



In [ ]:

```
'''
Discussion for Third part of problem 5

Visually, the images look very similar across methods. Even with small regions cropped, they appear the same,
so this author left them uncropped. However, it can be conjectured that the bicubic interpolation works the
best, as it uses 16 pixels around the point in question to determine the pixel value, unlike nearest neightbor,
which uses 1, and linear, which uses 2. This increase in the number of pixels to estimate should result in a
smoother image, making bicubic the best choice to reconstruction the image.
'''
```

In [ ]:

```

```