Assignment 2

Part 1

Part 1

1. Give an example of how one can exploit the associative property of convolution to filter an image.
   a. Applying filters is computationally very expensive if we have to use many filters—the associative property of convolution to cut down the number of filters before applying them to an image.
   b. The associated property can help reduce the computation by convolving the filters first, then applying the convolved filter to the image.
   c. Given the filter size, this way is smaller than the image , and the associative property of convolution will reduce the computational complexity.

Part 2

1. This is the input image: [0 0 1 1 0 0 1 1]. What is the result of dilation with a structuring element [1 1 1]?
   a. `[1 1 1 1 1 0 1 1]`

Part 3

1. The filter f°= [0 -1/2 0 1/2 0] is used as the filter to compute an estimate of the first derivative of the image in the x direction. What is the corresponding second derivative filter f"? (Hint: Asymmetric filters must be flipped before convolution
    a. `[-0.25 0 .5 0 -0.25]`

Part 4

1. Name two specific ways in which one could reduce the amount of fine, detailed edges that are detected with the Canny edge detector.
    a. `Gaussian Kernel:` Canny edge detection algorithm utilizes a Gaussian filter to reduce noise. However, the too-big kernel will smooth out some of the detail in the input image. In that case, the output image would have fewer fine edges.
    b. `Thresholding`: Setting high and low thresholds would reduce the amount of delicate and detailed edges. Setting too high threshold values would miss some delicate, precise edges since they might have lower gradient magnitude values than high threshold values. Likewise, too high threshold values would impact the performance of the hysteresis edge tracking step.

Part 5

1. Describe a possible flaw in using additive Gaussian noise to represent image noise.
    a. The noise in an image may vary depending on many factors; namely, the lighting conditions of the environment ***that will result in modeling with Gaussian noise may not depict the actual world condition.***

Part 6

1. Design a method that takes video data from a camera perched above a conveyor belt at an automotive equipment manufacturer and reports any flaws in the assembly of a part. Specify any critical assumptions your method makes. Your response should be a list of concise, specific steps and incorporate several techniques covered in class thus far.

- Let's assume that the part is in an environment where it can be easily distinguished from the conveyor belt and any other surroundings to reduce noise.
- The camera is above the conveyor belt and produces uniform-sized parts.
    - Image `Pre-Processing [Image Preperation]`. The raw image is converted to grayscale.
    - The image is then simplified into a more binary image by thresholding with techniques like dilation and erosion.
- `Noise reduction techniques`
    - Image Processing Analysis [Image Analysis]object segmentation, background subtraction.
    - Analyze color, texture, boundary
    - Feature-based detection.
- geometrical properties,

> Combined Techniques ***can be used to detect flaws in object shapes. For example, if the texture is different, this could signal a defect. Depth data will also help on the surface imperfections.*** These are some way to detect faults in the product.

Part 2: Question 1 `[SeamCarvingReduceWidth.py]`

- The below images are reduced by 100 pixels in width

Reduce Width

```
Original image size (480, 640, 3)
Reduce width of image (480, 540, 3)
Original image size (769, 775, 3)
Reduce width of image (769, 675, 3)
```

Prague

Original Image (480 X 640)

Reduce Width (480 X 540)

Mall

Question 2 `[SeamCarvingReduceHeight.py]`

- The below images are reduced by 100 pixels in height

Reduce Height

```
Original image size (480, 640, 3)
Reduce width of image (380, 640, 3)
Original image size (769, 775, 3)
Reduce width of image (669, 775, 3)
```

Prague Image

Original Image

Reduced Height Image

Mall Image

Original Image

Reduced Height Image

Question 3

- Energy Function of the `inputSeamCarvingPrague.jpg`

Energy Map `[ energyImage.py ]`

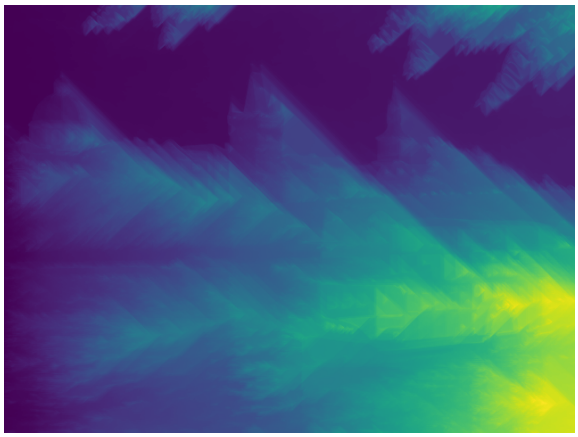
```
image = cv2.imread('inputSeamCarvingPrague.jpg')
im = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
energyImage = energy_image(im)

#plot the energy image
plt.imsave('outputPragueEnergy.png', energyImage)
plt.imshow(energyImage)
```

- The energy function output shows the outlines of the edges in the image.
- The white areas indicate high energy costs on the right of the horizontal energy map and at the bottom of the vertical energy map.
- Highlights the key components (context) in the image which shoud be avoided while using seam-carving of the image.

Horizonatal Seam Cumulative Energy Map

- The horizontal cumulative energy map shows ***high energy in the large boat, buildings, and bridges. It indicates we cannot compress carve these objects during the seam carving process.***

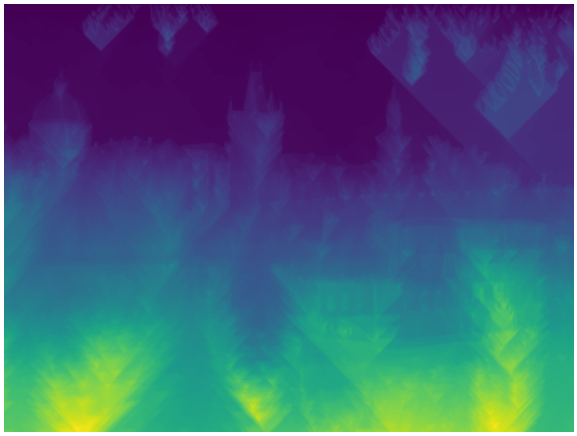
```
#plot the horizontal cumulative energy map
cum_energy_h = cumulative_minimum_energy_map(energyImage, 'HORIZONTAL')
plt.imshow(cum_energy_h)

#save the cumulative energy map
plt.imsave('outputCumEnergyHorizontal.png', cum_energy_h)
```



Vertical Seam Cumulative Energy Map

- The vertical cumulative energy map shows high energy costs in the areas such as the boat and the highly reflective parts of the water.
- We also avoid carving into the ships for this one

```
cum_energy_v = cumulative_minimum_energy_map(energyImage, 'VERTICAL')
plt.imshow(cum_energy_v)

#save the cumulative energy map
plt.imsave('outputCumEnergyVertical.png', cum_energy_v)
```



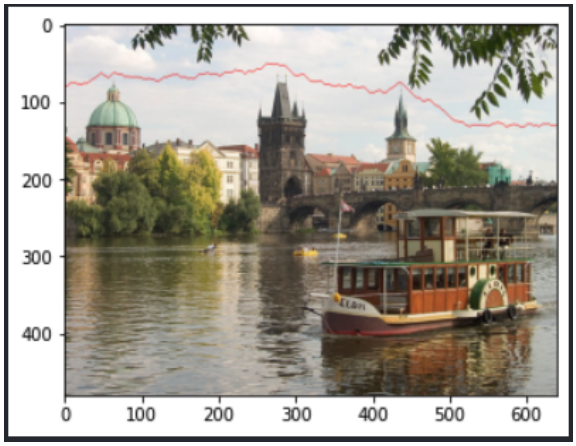
Question 4

Horizontal Seam

- The first horizontal seam appears in the sky which has the same color and little contrast, so there is little cost to cut this seam out.

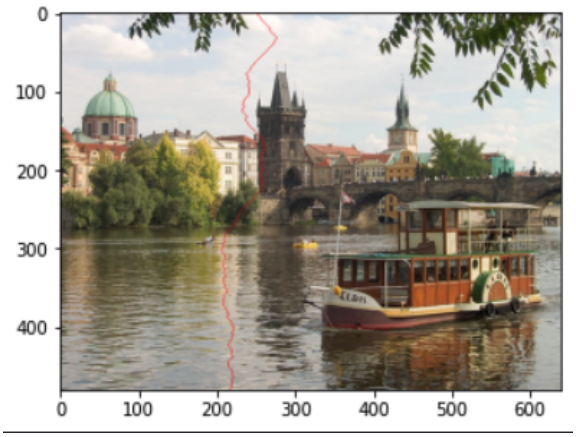
Vertical Seam

- The first vertical seam appears right through the edge of between the dark tower and the white building



the seam would be a near vertical line.

- The vertical seam passes through the reflection of the white building because the colors are the same there, indicating low cost.



Question 5

💡 `Difficult to see the red line in the image`

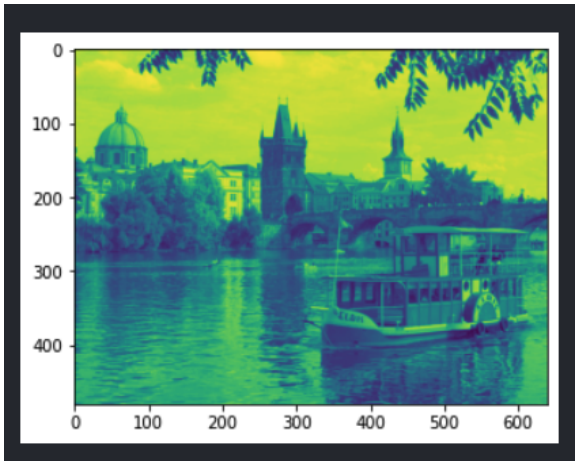
- Used Gaussian Filters

```
grad_dx = gaussian_filter1d(im, axis=0, sigma=1)
grad_dy = gaussian_filter1d(im, axis=1, sigma=1)

energyImage = norm(grad_dx, axis=2) + norm(grad_dy, axis=2)
```

- Here we do not see clear features highlighted whille it was clearly visible while

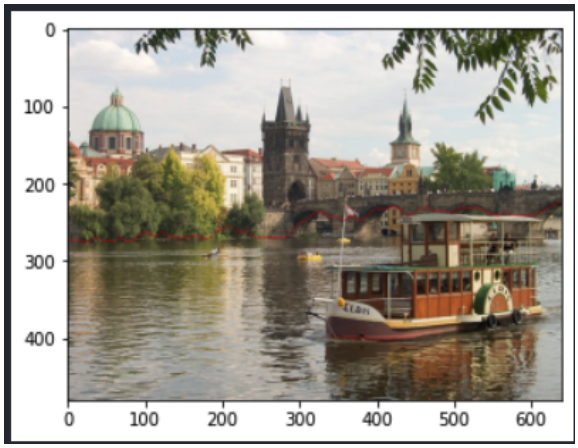
Energy Map with Gaussian Filtres

Energy Map with Sobel Features



Horizontal Seam With Gaussian Filter

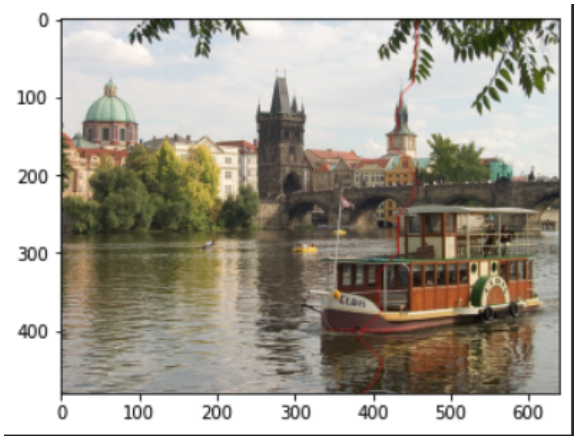
- Horizontal seam is passing through the bridge



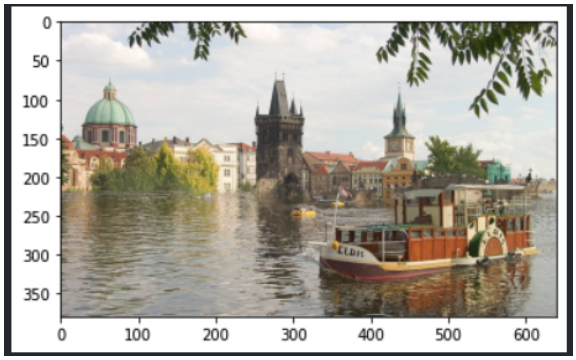
`Reduced Heeight With Gaussian Filter`

Vertical Seam with Gaussian Filter

- Vertical Seam passes through the tower on the right crosses the bridge and somee part of the boat as well



`Reduced Height With Sobel Filter`



The Image did not resize properly with the Gaussian Filter

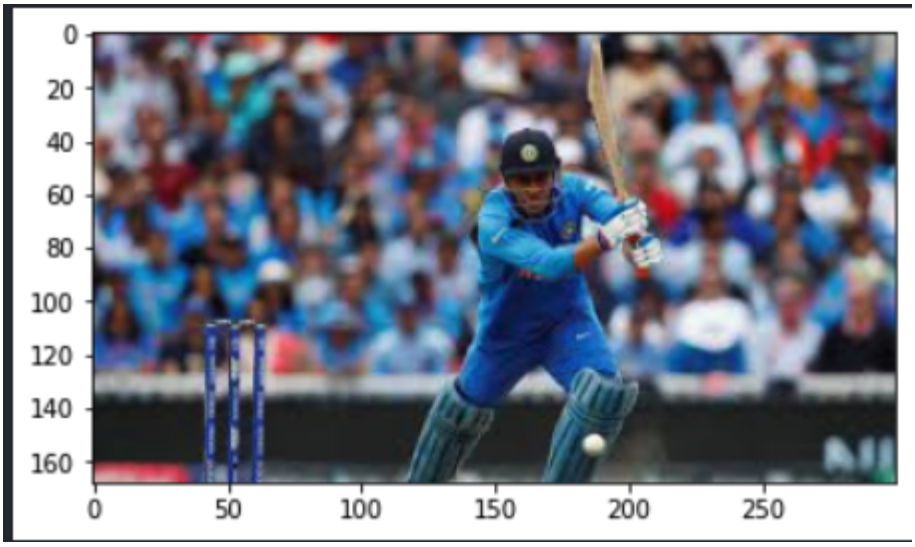
Reduced Height Image

Questions 6

Picture 1: Original Image

- This image includes the background blurred with spectators therefore is prefect to test for width decrease followed by height decrease as well.
- The sides of the image has lower context and that is why it is cut.

```
dhoni = cv2.imread('dhoni.jpeg')
dhoni_im = cv2.cvtColor(dhoni, cv2.COLOR_BGR2RGB)
plt.imshow(dhoni_im)
```



Original Image Dimension

```
print(f'Image shape: {dhoni_im.shape}')
Image shape: (168, 300, 3)
```

Energy Image

```
dhoni_energy = energy_image(dhoni_im)
plt.imshow(dhoni_energy)
```

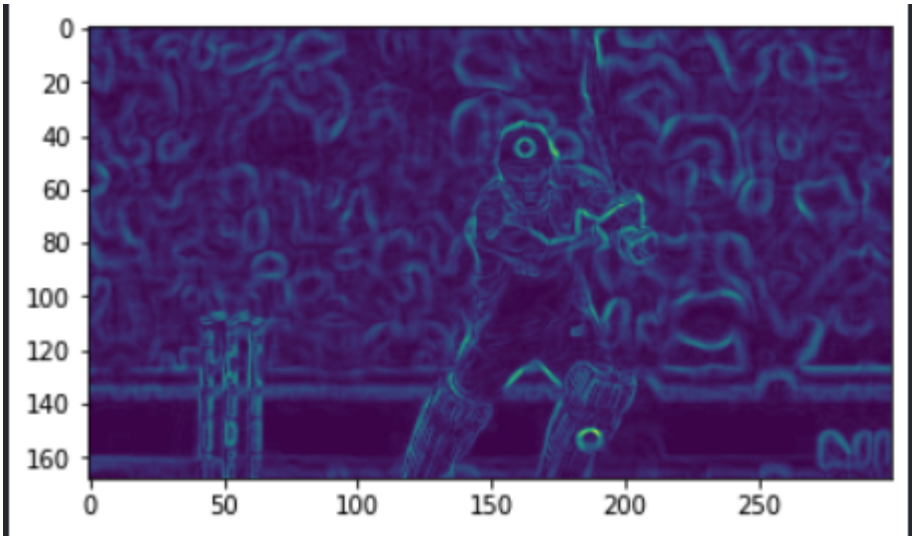


Image Transformations

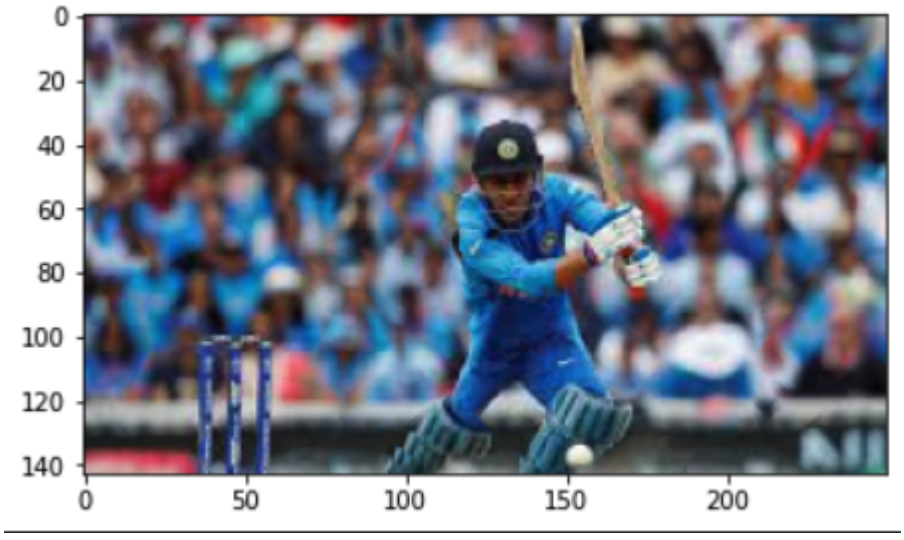
- *reduce the width of the dhoni image by 50 pixels*

```
#reduce the width of the dhoni image by 50 pixels
dhoni_reduced = dhoni_im
for i in range(50):
    dhoni_reduced = reduceWidth(dhoni_reduced, energy_image(dhoni_reduced))
plt.imshow(dhoni_reduced)

print(f'Image shape: {dhoni_reduced.shape}')
Image shape: (168, 250, 3)
```

Transformed Image after reducing the width by 50 pixel

- *reduce the height of the dhoni image by 25 pixels*

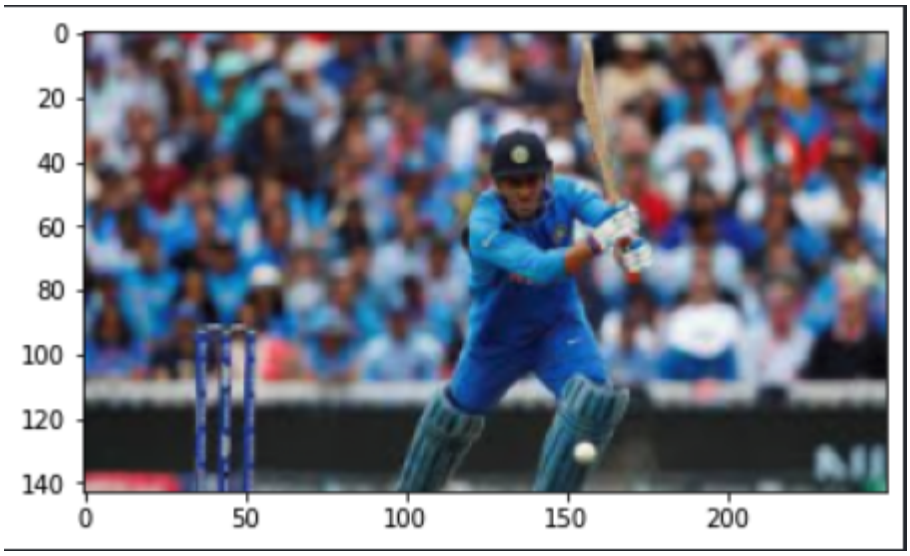
```
# reduce the height of the dhoni image by 50 pixels
for i in range(25):
    dhoni_reduced = reduceHeight(dhoni_reduced, energy_image(dhoni_reduced))
plt.imshow(dhoni_reduced)

print(f'Image shape: {dhoni_reduced.shape}')
Image shape: (143, 250, 3)
```



Transformed Image after reducing the height by 25 pixel

Random Scaling

```
# pillow to resize the image
#import image from PIL
import matplotlib.pyplot as plt

from skimage import data, color
from skimage.transform import rescale, resize, downscale_local_mean

#redccue the sixee of the original image
dhoni_resized = resize(dhoni_im, dhoni_reduced.shape)

#display the resized image

plt.imshow(dhoni_resized)
```



Beach Image

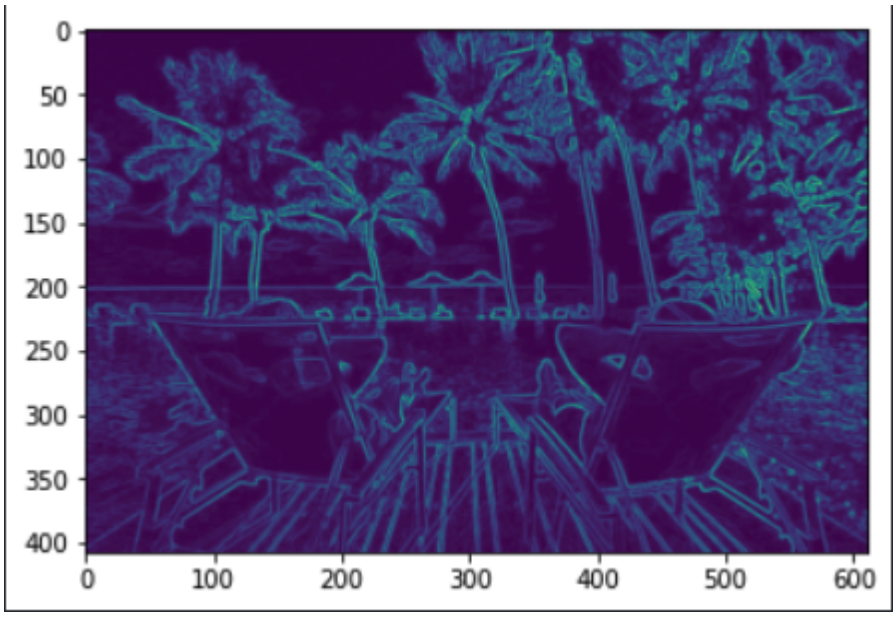
- This is a tricky image with lot of context and contours but the seam carving does a good job of preserving the context when both the height and the width is reduced.

Original Image


```
beach = cv2.imread('beach.jpeg')
beach_im = cv2.cvtColor(beach, cv2.COLOR_BGR2RGB)
plt.imshow(beach_im)
```

Energy Image for the beach

```
# beach energy image

beach_energy = energy_image(beach_im)
plt.imshow(beach_energy)
```

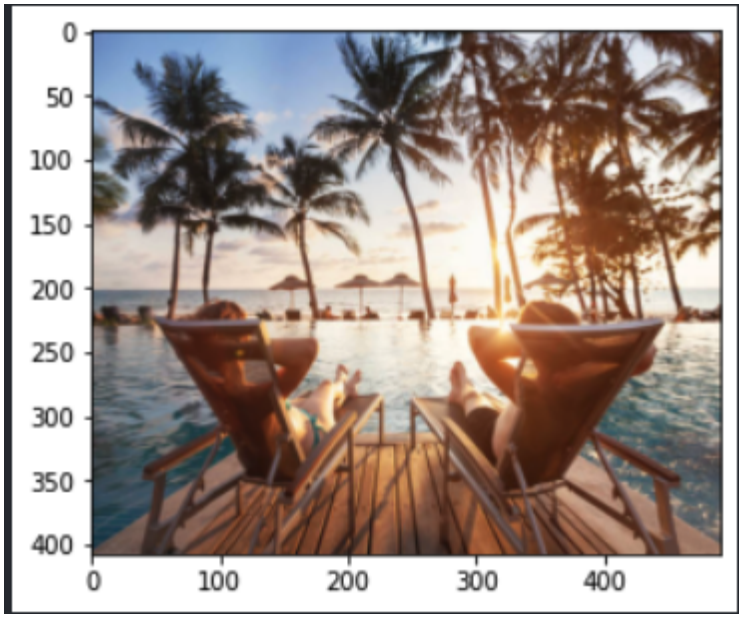


```
print(f'Image shape: {beach_im.shape}')
Image shape: (408, 612, 3)
```

Image Transformations

- *reduce the width of the beach image by 120 pixels*

```
beach_reduced = beach_im
for i in range(120):
    beach_reduced = reduceWidth(beach_reduced, energy_image(beach_reduced))
plt.imshow(beach_reduced)
```




```
print(f'Image shape: {beach_reduced.shape}')
Image shape: (408, 492, 3)
```

- *# reduce the height of the beach image by 150 pixels*

```
for i in range(150):
    beach_reduced = reduceHeight(beach_reduced, energy_image(beach_reduced))
plt.imshow(beach_reduced)
```

Final reduced size of the image

```
print(f'Image shape: {beach_reduced.shape}')
Image shape: (258, 492, 3)
```

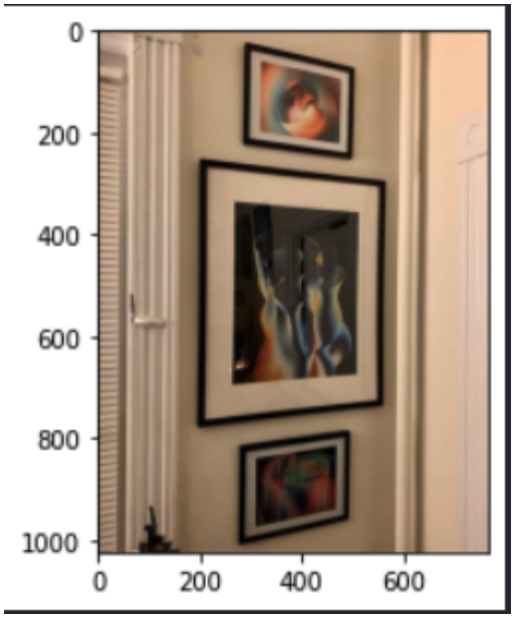
Resized Image

- There is a clear difference between the resize function and the seeam carving algorithm, with when compared ; the seam carving has a lot of tree cover on the left while a lot more cloud on the top-left on the re-sized image
- Trees much densee in seam-carving, while it looks like a lot of tree pixels have been removed from the resized image.

```
#redccue the sixee of the original image
beach_resized = resize(beach_im, beach_reduced.shape)

#display the resized image

plt.imshow(beach_resized)
Image shape: (258, 492, 3)
```

Paintings Image `[GONE WRONG]`

```
paint_image = cv2.imread('paintings.jpeg')
paint_image = cv2.cvtColor(paint_image, cv2.COLOR_BGR2RGB)
plt.imshow(paint_image)
```



Original Image Shape

```
print(f'Image shape: {paint_image.shape}')
Image shape: (1024, 768, 3)
```

Energy Image

```
#energy image of the painting image
paint_energy = energy_image(paint_image)
plt.imshow(paint_energy)
```

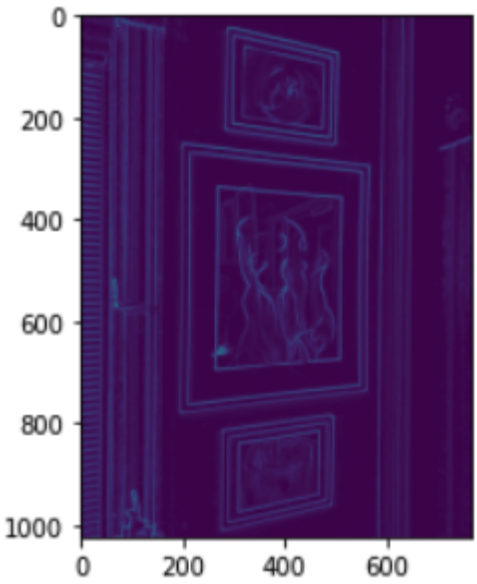
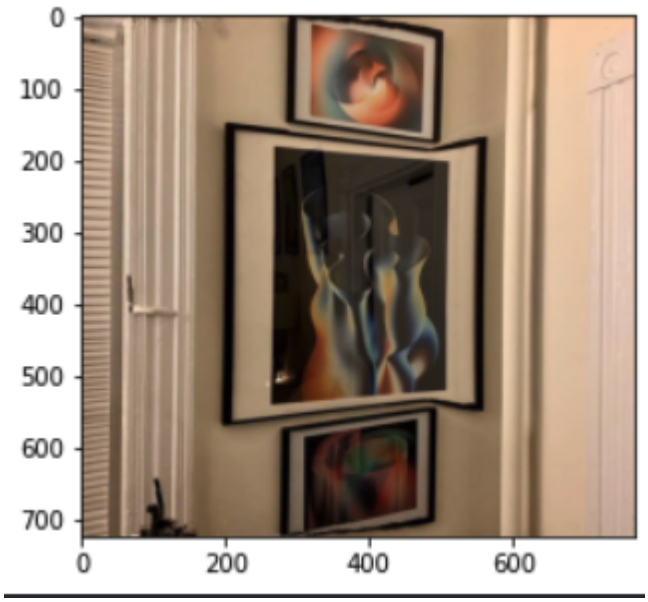


Image Transformations

- R*educe height of the painting image by 300 pixels*
- In this image, when the width was reduced by 300 pixels ; the painting got distorted instead of the sidee walls which is not the desired outcome we are looking from seam carving.

```
for i in range(300):
    paint_image = reduceHeight(paint_image, energy_image(paint_image))
plt.imshow(paint_image)

Output Dimension (724, 768, 3)
```



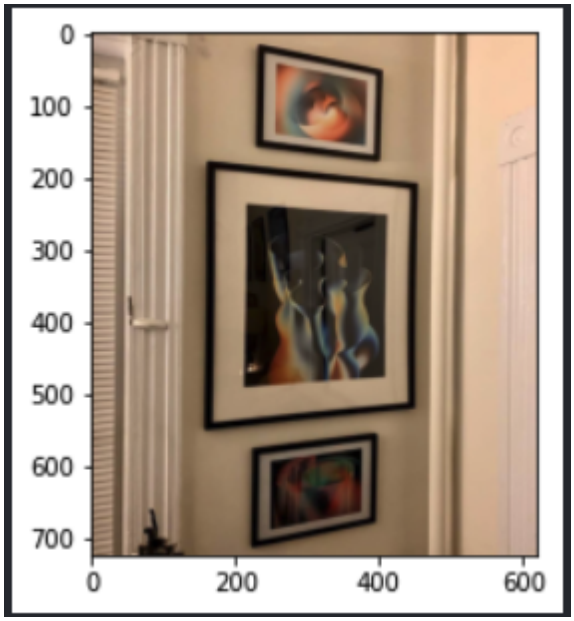
Transformation that went bad with seam carving

- *reduce the width of the painting image by 150 pixels*

```
for i in range(150):
    paint_image = reduceWidth(paint_image, energy_image(paint_image))
plt.imshow(paint_image)

Output Dimension (724, 618, 3)
```

Resized Image

- The resize function does a better job in this scenario as it does not distort the painting.




```
(724, 618, 3)
```

CODE

SeamCarvingReduceHeight.py

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import sys
import numpy as np
from imageio import imread, imwrite
from scipy.ndimage.filters import convolve
from tqdm import trange
from find_optimal_vertical_seam import *
from find_optimal_horizontal_seam import *
from reduce_Width import *
from reduce_Height import *
from  energyImage import *
from cumulativeMinimumEnergyMap import *

#load the image
#reduce  of the image by 100 pixels
image = cv2.imread('inputSeamCarvingPrague.jpg')
im1 = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
print(f'Original image size {im1.shape}')
energyImage = energy_image(im1)
for i in range(100):
```

```
    im1,_ = reduceHeight(im1, energyImage)
    energyImage = energy_image(im1)
plt.imshow(im1)
print(f'Reduce width of image {im1.shape}')
cv2.imwrite('outputReduceHeightPrague.png', cv2.cvtColor(im1, cv2.COLOR_RGB2BGR))


#reduce width of the image by 100 pixels
image2 = cv2.imread('inputSeamCarvingMall.jpg')
im2 = cv2.cvtColor(image2, cv2.COLOR_BGR2RGB)
print(f'Original image size {im2.shape}')
energyImage = energy_image(im2)
for i in range(100):
    im2,_ = reduceHeight(im2, energyImage)
    energyImage = energy_image(im2)
plt.imshow(im2)
print(f'Reduce width of image {im2.shape}')
cv2.imwrite('outputReduceHeightMall.png', cv2.cvtColor(im2, cv2.COLOR_RGB2BGR))
```

SeamCarvingReduceWidth.py

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import sys
import numpy as np
from imageio import imread, imwrite
from scipy.ndimage.filters import convolve
from tqdm import trange
from find_optimal_vertical_seam import *
from find_optimal_horizontal_seam import *
from reduce_Width import *

from  energyImage import *
from cumulativeMinimumEnergyMap import *

#load the image
#reduce height of the image by 100 pixels
image = cv2.imread('inputSeamCarvingPrague.jpg')
im1 = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
print(f'Original image size {im1.shape}')
energyImage = energy_image(im1)
for i in range(100):
    im1,_ = reduceWidth(im1, energyImage)
    energyImage = energy_image(im1)
plt.imshow(im1)

print(f'Reduce width of image {im1.shape}')
cv2.imwrite('outputReduceWidthPrague.png', cv2.cvtColor(im1, cv2.COLOR_RGB2BGR))


#reduce width of the image by 100 pixels
```

```
image2 = cv2.imread('inputSeamCarvingMall.jpg')
im2 = cv2.cvtColor(image2, cv2.COLOR_BGR2RGB)
print(f'Original image size {im2.shape}')
energyImage = energy_image(im2)
for i in range(100):
    im2,_ = reduceWidth(im2, energyImage)
    energyImage = energy_image(im2)
plt.imshow(im2)
print(f'Reduce width of image {im2.shape}')
cv2.imwrite('outputReduceWidthMall.png', cv2.cvtColor(im2, cv2.COLOR_RGB2BGR))
```

reduceWidth.py

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import sys
import numpy as np
from imageio import imread, imwrite
from scipy.ndimage.filters import convolve
from tqdm import trange
from cumulativeMinimumEnergyMap import cumulative_minimum_energy_map
from find_optimal_vertical_seam import find_optimal_vertical_seam

def reduceWidth(im, energyImage):
    """
    Function to reduce the width of the image by 1 pixel

    :param im: image
    Copying the pixels after the seam.
    :param energyImage: energy image of the image
    :return: image with reduced width
    """
    # get the cumulative minimum energy map
    # get the optimal vertical seam
    # remove the seam from the image
    # return the image with reduced width
    cumulativeEnergyMap = cumulative_minimum_energy_map(energyImage, 'VERTICAL')
    seam = find_optimal_vertical_seam(cumulativeEnergyMap)

    #delete the seam from the image from each chanel
    r,c,_ = im.shape
    newImg = np.zeros((r,c,3))
    for i,j in enumerate(seam):
        newImg[i,0:j,:] = im[i,0:j,:] #copy the pixels before the seam
        newImg[i,j:c-1,:] = im[i,j+1:c,:] #copy the pixels after the seam

    newCumulativeEnergyMap = np.zeros((r,c))
    for i,j in enumerate(seam):
        newCumulativeEnergyMap[i,0:j] = cumulativeEnergyMap[i,0:j]
        newCumulativeEnergyMap[i,j:c-1] = cumulativeEnergyMap[i,j+1:c]
```

```
    return newImg[:,:-1,:].astype(np.uint8), newCumulativeEnergyMap[:,:-1]
```

reduceHeight.py

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import sys
import numpy as np
from imageio import imread, imwrite
from scipy.ndimage.filters import convolve
from tqdm import trange
from find_optimal_horizontal_seam import *
from energyImage import *
from find_optimal_vertical_seam import *
from cumulativeMinimumEnergyMap import cumulative_minimum_energy_map



def reduceHeight(im, energyImage):
    """
    Function to reduce the height of the image by 1 pixel

    :param im: image
    :param energyImage: energy image of the image
    :return: reduced image
    """
    # get the cumulative minimum energy map
    cumulativeEnergyMap = cumulative_minimum_energy_map(energyImage, 'HORIZONTAL')
    # get the seam
    seam = find_optimal_horizontal_seam(cumulativeEnergyMap)
    # remove the seam
    # return the image with reduced height
    r,c,_ = im.shape
    newImg = np.zeros((r,c,3))
    for i,j in enumerate(seam):
        newImg[0:j,i,:] = im[0:j,i,:] #copy the first part of the image
        newImg[j:r-1,i,:] = im[j+1:r,i,:] #copy the second part of the image

    newCumulativeEnergyMap = np.zeros((r,c))
    for i,j in enumerate(seam):
        newCumulativeEnergyMap[0:j,i] = cumulativeEnergyMap[0:j,i]
        newCumulativeEnergyMap[j:r-1,i] = cumulativeEnergyMap[j+1:r,i]

    return newImg[:-1,:,:].astype(np.uint8), newCumulativeEnergyMap[:-1,:]
```

displaySeam.py

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import sys
import numpy as np
from imageio import imread, imwrite
from scipy.ndimage.filters import convolve
from tqdm import trange



def display_seam(im,seam, type):
    """
    Function to display the seam on the image

    :param im: image
    :param seam: seam to be displayed
    :param type: type of seam to be displayed
    :return: image with seam displayed
    """
    if type == 'VERTICAL':
        for i in range(len(seam)):
            im[i][seam[i]] = [255, 0, 0]
    if type == 'HORIZONTAL':
        for i in range(len(seam)):
            im[seam[i]][i] = [255, 0, 0]

    #return the image with seam displayed

    return plt.imshow(im)
```

cumulativeMimimumEnergyMap.py

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import sys
import numpy as np
from imageio import imread, imwrite
from scipy.ndimage.filters import convolve
from tqdm import trange
from find_optimal_vertical_seam import *
from find_optimal_horizontal_seam import *


def cumulative_minimum_energy_map(energyImage, seamDirection):
    """
    Function to compute the cumulative minimum energy map of an image
    :param energyImage: energy image of the image
    :param seamDirection: string indicating the direction of seam to be removed
```

```
    :return: cumulative minimum energy map of the image
    """
    row = energyImage.shape[0]
    col = energyImage.shape[1]
    dp_energy = []

    if seamDirection == 'VERTICAL':
        dp_energy.append(energyImage[0].tolist())

        for i in range(1, row):
            temp = []
            for j in range(col):
                if j == 0:
                    temp.append(energyImage[i][j] + min(dp_energy[i-1][j], dp_energy[i
-1][j+1]))
                elif j == col - 1:
                    temp.append(energyImage[i][j] + min(dp_energy[i-1][j], dp_energy[i
-1][j-1]))
                else:
                    temp.append(energyImage[i][j] + min(dp_energy[i-1][j-1], dp_energy
[i-1][j], dp_energy[i-1][j+1]))
            dp_energy.append(temp)
        dp_return = np.asarray(dp_energy, dtype=np.double)


    if seamDirection == 'HORIZONTAL':
        dp_energy.append(energyImage[:,0].tolist())

        for i in range(1, col):
            temp = []
            for j in range(row):
                if j == 0:
                    temp.append(energyImage[j][i] + min(dp_energy[i-1][j], dp_energy[i
-1][j+1]))
                elif j == row - 1:
                    temp.append(energyImage[j][i] + min(dp_energy[i-1][j], dp_energy[i
-1][j-1]))
                else:
                    temp.append(energyImage[j][i] + min(dp_energy[i-1][j-1], dp_energy
[i-1][j], dp_energy[i-1][j+1]))
            dp_energy.append(temp)
        dp_return = np.asarray(dp_energy, dtype=np.double).T
    return dp_return
```

energyImage.py

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import sys
import numpy as np
from imageio import imread, imwrite
```

```
from scipy.ndimage.filters import convolve
from tqdm import trange


from scipy.ndimage import sobel, gaussian_filter1d, convolve1d, prewitt
from scipy.linalg import norm
import numpy as np

def energy_image(img):
    '''
    Compute the energy image of an image
    :param img: input image dimensions (MxNx3) of type uint8
    :return: energy image (MxN) of type double
    '''

    filter_dx = np.array([[0.25,0.5,0.25],
                  [0,0,0],
                  [-0.25,-0.5,-0.25]])

    filter_dx = np.stack([filter_dx] * 3, axis=2)
    # The folloing filter is used to compute the gradient in the y direction
    filter_dy = np.array([[0.25,0,-0.25],
                  [0.5,0,-0.5],
                  [0.25,0,-0.25]])

    filter_dy = np.stack([filter_dy] * 3, axis=2)

    img = img.astype('double')
    energy = np.absolute(convolve(img, filter_dx)) + np.absolute(convolve(img, filter_
dy))
    energy_map = np.sum(energy, axis=2)
    return energy_map.astype('double')
```

find_optimal_horizontal_seam.py

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import sys
import numpy as np
from imageio import imread, imwrite
from scipy.ndimage.filters import convolve
from tqdm import trange


def find_optimal_horizontal_seam(cumulativeEnergyMap):
    """
    Function to get the horizontal seam in a picture

    :param cumulativeEnergyMap: cumulative energy map of the image
    :return: optimal horizontal seam of the image
    """
```

```
    # backtracking to get the seam
    #print(cumulativeEnergyMap)
    row = len(cumulativeEnergyMap)
    col = len(cumulativeEnergyMap[0])
    seam = [0] * col
    seam[col-1] = np.argmin(cumulativeEnergyMap[:,col-1])
    for i in range(col-2, -1, -1):
        j = seam[i+1]
        #print(f'j {j}')
        if j == 0:
            seam[i] = np.argmin(cumulativeEnergyMap[j:j+2,i]) + j
        elif j == row - 1:
            #print(f'{cumulativeEnergyMap[i][j-1:j+1]}')
            seam[i] = np.argmin(cumulativeEnergyMap[j-1:j+1,i]) + j-1
        else:
            #print(f'{cumulativeEnergyMap[i][j-1:j+2]}')
            seam[i] = np.argmin(cumulativeEnergyMap[j-1:j+2,i]) + j-1
    return seam
```

find_optimal_vertical_seam.py

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import sys
import numpy as np
from imageio import imread, imwrite
from scipy.ndimage.filters import convolve
from tqdm import trange



def find_optimal_vertical_seam(cumulativeEnergyMap):
    """
    Function to get the horizontal seam in a picture

    :param cumulativeEnergyMap: cumulative energy map of the image
    :return: optimal vertical seam of the image

    """
    # backtracking to get the seam
    #print(cumulativeEnergyMap)
    row = len(cumulativeEnergyMap)
    col = len(cumulativeEnergyMap[0])
    seam = [0] * row
    seam[row-1] = np.argmin(cumulativeEnergyMap[row-1])
    for i in range(row-2, -1, -1):
        j = seam[i+1]
        #print(f'j {j}')
        if j == 0:
            seam[i] = np.argmin(cumulativeEnergyMap[i][j:j+2]) + j
        elif j == col - 1:
```

```
            #print(f'{cumulativeEnergyMap[i][j-1:j+1]}')
            seam[i] = np.argmin(cumulativeEnergyMap[i][j-1:j+1]) + j-1
        else:
            #print(f'{cumulativeEnergyMap[i][j-1:j+2]}')
            seam[i] = np.argmin(cumulativeEnergyMap[i][j-1:j+2]) + j-1
    return seam
```

cumulativeeMinimumEnergyMap.py

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import sys
import numpy as np
from imageio import imread, imwrite
from scipy.ndimage.filters import convolve
from tqdm import trange
from find_optimal_vertical_seam import *
from find_optimal_horizontal_seam import *


def cumulative_minimum_energy_map(energyImage, seamDirection):
    """
    Function to compute the cumulative minimum energy map of an image
    :param energyImage: energy image of the image
    :param seamDirection: string indicating the direction of seam to be removed

    :return: cumulative minimum energy map of the image
    """
    row = energyImage.shape[0]
    col = energyImage.shape[1]
    dp_energy = []

    if seamDirection == 'VERTICAL':
        dp_energy.append(energyImage[0].tolist())

        for i in range(1, row):
            temp = []
            for j in range(col):
                if j == 0:
                    temp.append(energyImage[i][j] + min(dp_energy[i-1][j], dp_energy[i
-1][j+1]))
                elif j == col - 1:
                    temp.append(energyImage[i][j] + min(dp_energy[i-1][j], dp_energy[i
-1][j-1]))
                else:
                    temp.append(energyImage[i][j] + min(dp_energy[i-1][j-1], dp_energy
[i-1][j], dp_energy[i-1][j+1]))
            dp_energy.append(temp)
        dp_return = np.asarray(dp_energy, dtype=np.double)
```

```
    if seamDirection == 'HORIZONTAL':
        dp_energy.append(energyImage[:,0].tolist())

        for i in range(1, col):
            temp = []
            for j in range(row):
                if j == 0:
                    temp.append(energyImage[j][i] + min(dp_energy[i-1][j], dp_energy[i
-1][j+1]))
                elif j == row - 1:
                    temp.append(energyImage[j][i] + min(dp_energy[i-1][j], dp_energy[i
-1][j-1]))
                else:
                    temp.append(energyImage[j][i] + min(dp_energy[i-1][j-1], dp_energy
[i-1][j], dp_energy[i-1][j+1]))
            dp_energy.append(temp)
        dp_return = np.asarray(dp_energy, dtype=np.double).T
    return dp_return
```