Comp Photography
Final Project

Yuanyuan Weng
Fall 2017
candyweng.cas@gmail.com

Advanced Seam Carving for Image Resizing

This project illustrates the content-aware image resizing via improved seam carving method with a various types of energy map and seam-carving algorithm, allowing more natural-looked resized image with object of interest protected or removed.

The Goal of Your Project

Old project scope from proposal:

In the final project, I plan to work on the content aware resize of video. Different from 2D images, video is represented by 3D space-time volumes and in terms of resize, we need to remove 2D seams. According to the reference paper, seam carving with graph cuts will be implemented to achieve this goal.

Current project scope:

To customize the size and aspect-ratio of image in a content-awared manner. Several aspects of seam carving have been investigated:

1. Object removal can remove a target object from the image while minimize the visual artifects.
2. Rescrutinize energy map: besides the gradient energy map, the Sobel energy map and the saliency map are considered
3. Forward energy map method can take into consideration the impact of the change in accumulated energy induced by removing the seam.

What motivated you to do this project?

Customize the image to better adapt to hardwares with different screen size and aspect-ratio, for example, display/smart phone. Also minimize the visual artifects.

Scope Changes

Retargeted our goal from implementing video resize to rescrutinizing map energy functions and object removal for the reasons below:

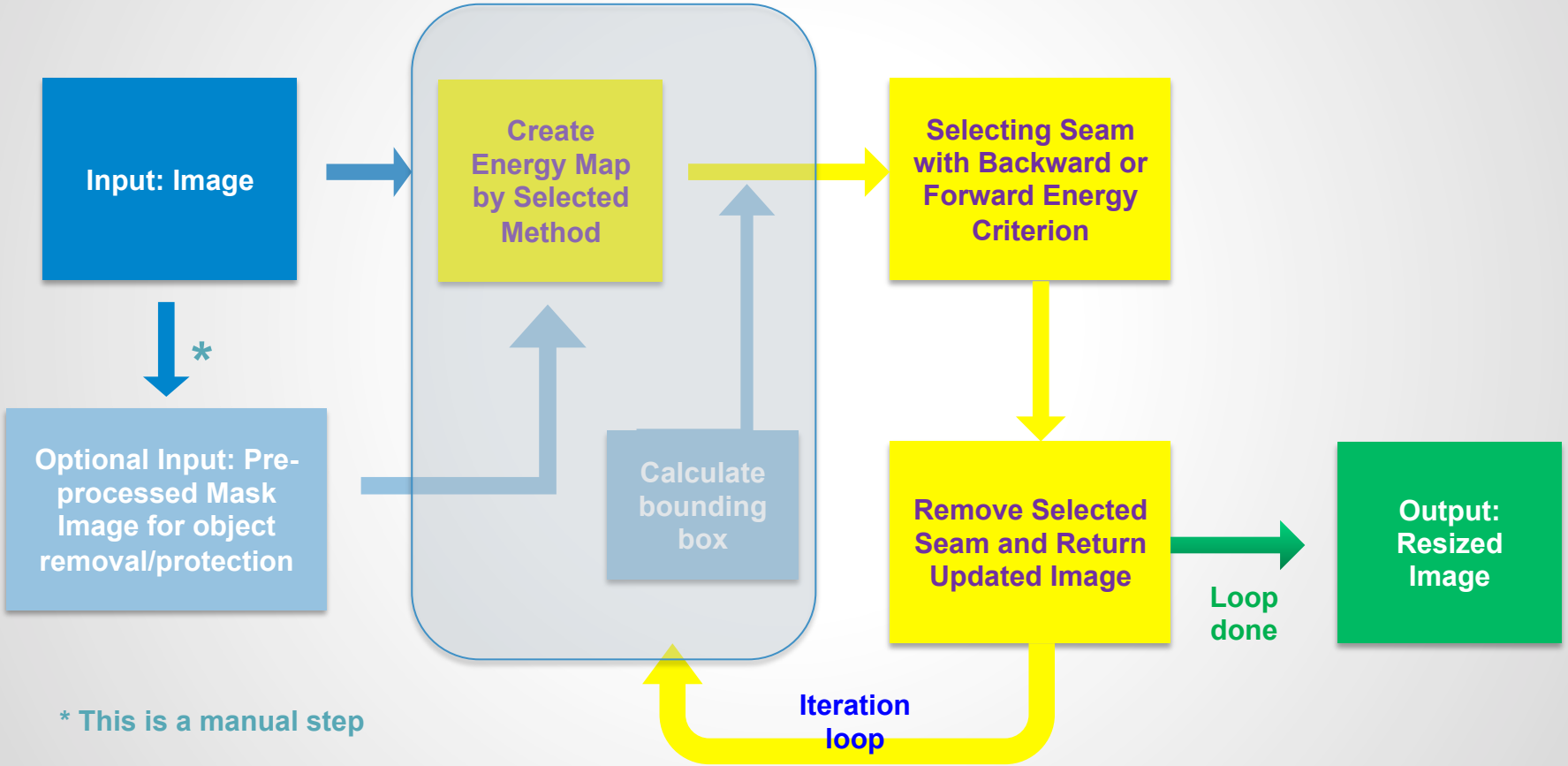
- Energy map is the crucial factor in seam carving for both image and videos.
- Object removal is good for presenting.
- From literature study I noticed that video resizing and graph cut requires relatively large amount of computation resources than available.

Showcase

Object protection

Object Removal

Project Pipeline: Overview



Project Pipeline: Input

Optional Input: Pre-processed Mask Image

This is an illustration for creating mask.
I have create my mask by drawing based on the input image. I have also tried to create my mask by my own code: I have used openCV canny method to detect the edges of black image. But this method doesn't work very well. Then I tried to draw the mask by using hand drawing on MAC, which is shown as the right mask. This mask works very well.

Calculate bounding box

If an mask is used for object removal, bounding box of that object will be calculated to get the index (location) of the object. The step calculates the minimum and maximum locations of both width and height. So when removing seams during the object removal process, the dimension of bounding box is a criterion for selecting a vertical seam or removing horizontal seam.

Project Pipeline: Energy Map

Create Energy Map by Selected Method

- Energy map can be created in various different methods. The most widely implemented energy map is gradient energy method.
- This method estimates the energy of each pixel based based on their gray scale, i.e., a value between [0, 255].
- Usually gradient energy map highlights the boundary of foreground region and background region in an image.
- Typical types includes simple gradient, improved gradient, Sobel, etc.
- Here simple gradient method and Sobel method are implemented to obtain energy map.

Project Pipeline: Energy Map

- The saliency map estimates the energy of pixels based on rgb information.
- Compared to gray-scale image based method, rgb-image based method

Create Energy Map by Selected Method

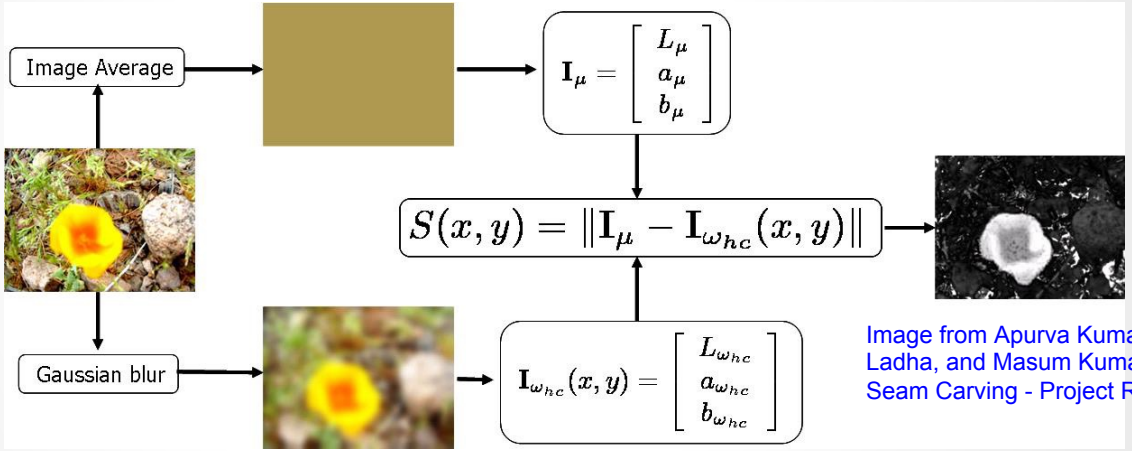


Image from Apurva Kumar, Arushi Ladha, and Masum Kumar Lodha, Seam Carving - Project Report

- The saliency map estimates the energy of each pixel based on their rgb information.
- Compared to gray-scale image based method, rgb-image based method
- Here frequent tuned method, rbd method, and mbd method are implemented to obtain energy map.

Project Pipeline: Seam Process

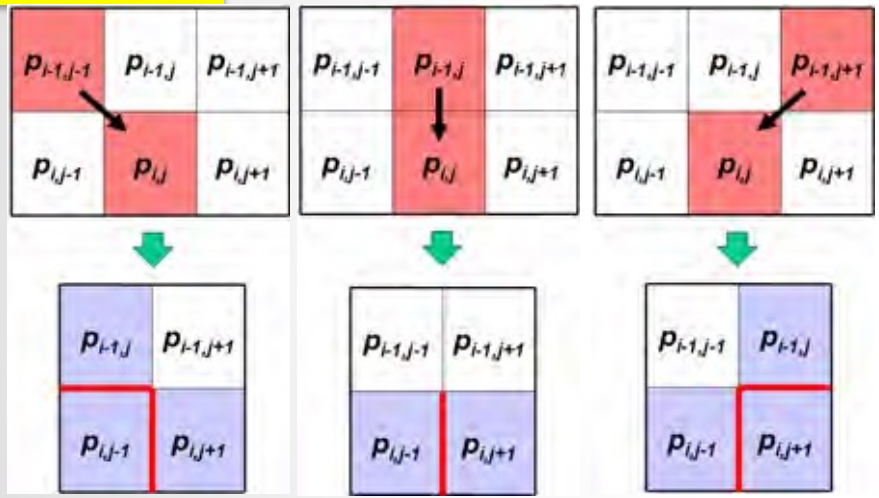
Selecting Seam with Backward or Forward Energy Criterion

- This part is to find the seam with minimum energy.
- A vertical seam is a connected series of pixels from top to bottom in the image. That is, if one pixel is in a seam is located at (height, width) = (h, w), its top neighbor must be one among (h + 1, w + 1), (h + 1, w), or (h + 1, w - 1); and its bottom neighbor must be (h - 1, w + 1), (h - 1, w), or (h - 1, w - 1).
- The same rule applies to a horizontal seam except switching top/bottom to left/right.

Project Pipeline: Seam Process

Selecting Seam with Backward or Forward Energy Criterion

- The forward energy criterion considers the newly formed boundaries after removing a pixel on the output image. In the figure below, these boundaries are marked as red.
- When the precedent removed pixel is at different locations (upper left/right or above), the induced energy change is C_L, C_R, or C_U.



$$M(i,j) = P(i,j) + \min \begin{cases} M(i-1,j-1) + C_L(i,j) \\ M(i-1,j) + C_U(i,j), \\ M(i-1,j+1) + C_R(i,j) \end{cases}$$

$$C_L(i,j) = |I(i,j+1) - I(i,j-1)| + |I(i-1,j) - I(i,j-1)|$$
$$C_U(i,j) = |I(i,j+1) - I(i,j-1)|$$
$$C_R(i,j) = |I(i,j+1) - I(i,j-1)| + |I(i-1,j) - I(i,j+1)|$$

Image from Shai Avid, Implementing Seam Carving for Image Resizing


```
Diff1 = abs(img[i, j+1, 0] - img[i, j-1, 0]) + abs(img[i, j+1, 1] - img[i, j-1, 1]) + abs(img[i, j+1, 2] - img[i, j-1, 2])
Diff2 = abs(img[i-1, j, 0] - img[i, j-1, 0]) + abs(img[i-1, j, 1] - img[i, j-1, 1]) + abs(img[i-1, j, 2] - img[i, j-1, 2])
Diff3 = abs(img[i-1, j, 0] - img[i, j+1, 0]) + abs(img[i-1, j, 1] - img[i, j+1, 1]) + abs(img[i-1, j, 2] - img[i, j+1, 2])

FEM[i, j] = min(FEM[i-1, j-1] + Diff1 + Diff2, FEM[i-1, j] + Diff1, FEM[i-1, j+1] + Diff1 + Diff3)
```

Project Pipeline: Seam Process

Remove Selected Seam and Return Updated Image

This part is to remove a selected seam from the image
Update the image with one seam removed.
Then the pipeline enter the loop on recalculating the energy map with the updated image, selecting next seam and remove next seam until set criterion: either remove an object or remove certain number of seams.

Demonstration set 1: Object removal with protection

Input image

Input showing interested area

Removal mask

Removing one balloon

Demonstration set 1: Object removal with protection

Protection mask

Removing one balloon with protection on other balloons

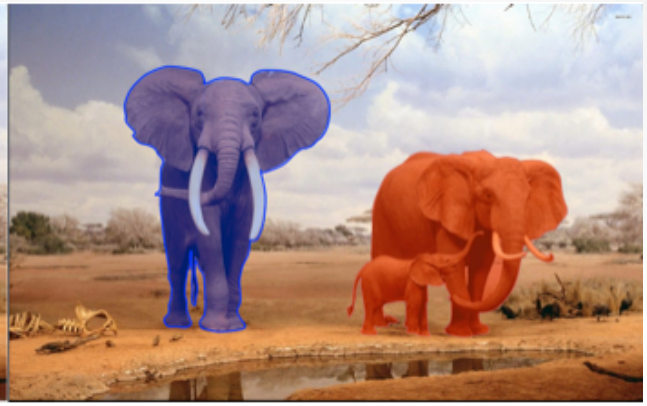
Input image

Removing one balloon
Though the second balloon has been removed, but there is distortion on the other balloons

Removing one balloon with protection on other balloons. Image shows the improved content preserving on other balloons.

Demonstration set 2: Object removal/protection

Input image

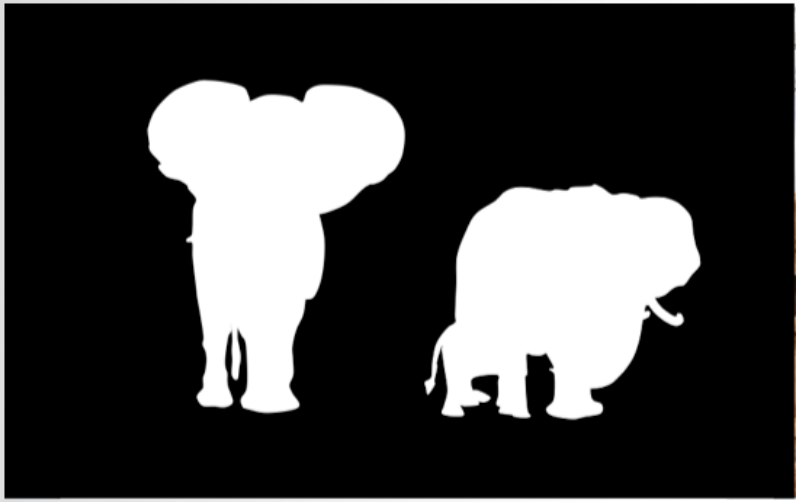


Input showing interested area

Original seam carving method

Output image showing bad results (see the elephant's leg looses the original shape)

Demonstration set 2: Object removal/protection



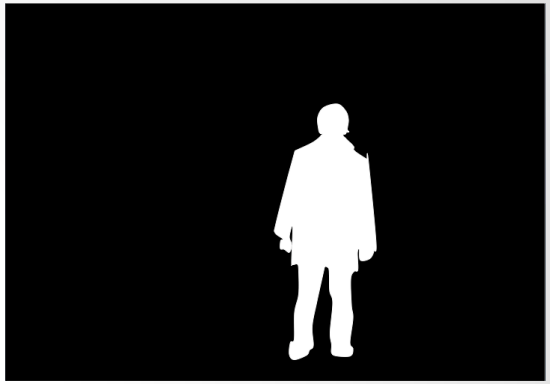
Protection mask

Resizing the image with protection on the interested area. (see the elephant's leg keep the original shape)

Demonstration set 3: Object removal/protection

Input image

Removal mask



Protection mask

Removing one person without protecting another

Removing one person with protection on another person

More Demonstration: Backward vs. Forward seam removal

The forward energy criterion, on top of the original forward energy criterion, can look forward at the image after each seam carving operation, taking into account the change of energy induced by the newly formed boundary between previously non adjacent pixels that become neighbours.

In this example, the width of bird accounts for 62% of the width of input image. From the normal backward seam carving method when the width is about 62% of original image, there is obvious distortion on the bird. However, forward algorithm shows better results.

Original

50% width reduction 62% of original width 80% of original width
(all sobel energy function with backward algorithm, indicates the distortion on the tail)

sobel energy function with forward algorithm

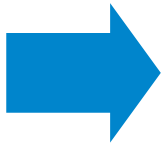
More Demonstration: : Energy map

The gradient energy map highlights the boundary between foreground (main object) and background, while the saliency map highlights the body of the main objects. This different results in different preference in seam finding and seam carving.

More Demonstration: Object Removal

(1) Image

(2) Mask for removal



(1) + (2)

- Protection
- Normal seam carving

Project Development

Literature Study

- The first step is literature study. I went over a various types of resources, including academic papers, websites, videos, etc, to understand the fundamental concepts, the scope of application, and the limitation of seam carving.

- Links of selected resources are listed in the motivation slide above.
- In ref [1], a general investigation of the common seam carving applications are reviewed and summarized.
- In ref [2], the concept of forward energy and its application are discussed.
- In ref [3], the implementation of object removal and object protection is demonstrated.

Project Development (cont'd)

Basic Framework I:

- After literature study, I decided to implement forward energy criterion and object removal/protection technique to get better results for image resizing based on basic seam carving method, which is the topic of my mid-term project.
- The forward energy criterion, on top of the original forward energy criterion, can look forward at the image after each seam carving operation, taking into account the change of energy induced by the newly formed boundary between previously non adjacent pixels that become neighbours.
- To implement the forward energy criterion, I created an new accumulated energy map function. This accumulated energy map function recorded the forward energy map (i.e., the accumulated gradient energy) of each pixel in regard to the top-left corner of the image, plus the change of energy induced by removing this pixel. This accumulated energy is then used to find out the next seam to carve.

Project Development (cont'd)

Basic Framework II:

- The second implementation of seam carving is to apply mask layers to remove and/or protect certain object of interests in the image. This is a fundamental and useful application available in most commercial image processing softwares in the market, however, this application is still of great interest because of the high demands in real life and the space for algorithm improvement.

- Mask layers used to define regions to remove or to protect are manually created based on the input images. Due to the time limit, the GUI for real-time mask layer creation is not provided in the current function. Based on my research, real-time mask layer creation or even automatic mask layer creation can be easily realized in Python via selectROI function in OpenCV package.

- The mask layer is loaded and converted into a 2D matrix, where all pixels are 0 except in the targeted region. This matrix is then multiplied by a large factor and added(or substracted) to(from) the energy map, whereby giving the pixels of interest very large value (positive for protection, or negative for removal). This new energy map is then used for seam finding and seam carving functions.

Project Development (cont’d)

Challenges I:

- Understanding the concept of forward energy is non-trivial
- In the original seam carving function, I used linked list to record the seam path of minimum energy, this is having problem because the recorded location need to be updated after adding forward energy terms. To fix this problem, I used a 2D array to record the location of the seam path in forward energy based seam carving function.
- In many cases, the result is not as expected.

Project Development (cont'd)

Challenges II:

- The creation of useful mask is non-trivial: First the input mask image should be preprocessed since the mask was created by powerpoint based on the original input image and it is likely the edges contains one or two line of white pixels, which leads to the incorrect (very big) boundind box. I have preprocessed the mask image by estimating the location of highlighted area, and make all other pixels equal to 0. For the calculation of coordinates of bounding box, find the minimum coordinates for both width and height, then flip the mask image and find the minimum coordinates for both width and height, which is the maximum coordinates for the non-flipped mask

- This approach failed when the mask for object removing contains multiple regions with large distance, because the current function uses a single rectangular bounding box to estimate the number of pixels need to remove. When two small target objects are far from each other, the algorithm tried to remove much more than enough seams and maked the size of the output image very small.

- Also when the object is large, the masking method doesn't work very well.

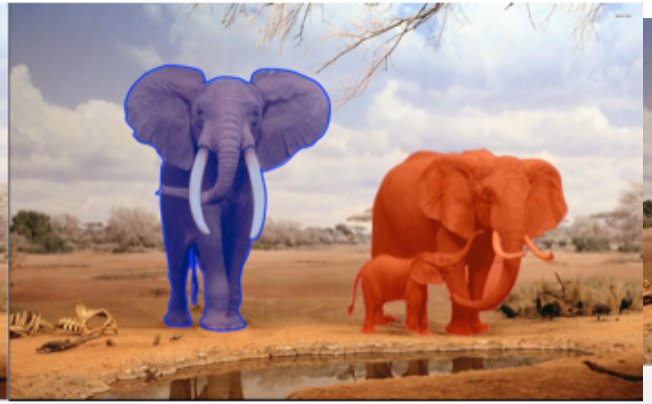
- The following results (next slide) show a case that when trying to remove the blue area and protecting the red area, the output image is not as expected.

Project Development (cont'd)

Limitations and Future Work I:

- For object removal, if the target object to remove is large, the size of the output image will be sharply reduced. An ideal input image for the current version of object removal function is suitable for removing small objects, for example, some unwanted feature shows up in the background of a selfie.
- To improve this function, a complementary image expansion function based on seam carving method can be added to enlarge the size of the output image back to its original size. This function could be easily incorporated given more time based on the current seam finding function and an new seam insertion function modified from the current seam deduction function.

results (next slide) show a case that when trying to remove the blue area and protecting the red area, the output image is not as expected.


Project Development (cont'd)

Limitations and Future Work II:

- The forward energy can protect the structure of image. However, maintaining the structure can sometime come at the expense of content. For example, important objects that can be resized without noticeable artifacts may be jeopardized during resizing.

- To protect the foreground objects while implementing forward energy criterion, a combination of the forward energy and object protection can be implemented.

Computation: Code Functional Description

```
def findRect(mask):
    '''
    mask is a numpy array with n rows a
    find the smallest n and m, biggist
    '''
    #find w_min
    index = mask.argmax(axis=1)
    w_min = min(index[index!=0])

    #find h_min
    index = mask.argmax(axis=0)
    h_min = min(index[index!=0])

    #flip mask totally
    mask = mask[::-1, ::-1]

    #find w_max
    index = mask.argmax(axis=1)
    w_max = min(index[index!=0])
    w_max = mask.shape[1] - w_max

    #find h_max
    index = mask.argmax(axis=0)
    h_max = min(index[index!=0])
    h_max = mask.shape[0] - h_max

    return h_min, w_min, h_max, w_max
```

- **Function: findRect (from boundingbox.py)**
- This function creates the minimal rectangular box region that covers the whole region that needs to be removed. The length and heights of this region determines the number of seam carving iterations needed in vertical direction and horizontal direction for completely removing the target object. The package used for this function is numpy (also used in all functions in my code)
- input: mask image
- output: the location and the size of the bounding box

First the input mask image should be preprocessed since the mask was created by powerpoint based on the original input image and it is likely the edges contains one or two line of white pixels, which leads to the incorrect (very big) boundind box. I have preprocessed the mask image by estimating the location of highlighted area, and make all other pixels equal to 0. For the calculation of coordinates of bounding box, find the minimum coordinates for both width and height, then flip the mask image and find the minimum coordinates for both width and height, which is the maximum coordinates for the non-flipped mask

Computation: Code Functional Description

```
def RemoveObject(img, mask_remove, mask_protect):
    # mask_remove and mask_protect are [0, 255] binary map
    # convert mask_remove into [0 ,1] binary map, and remove the overlap region from mask_remove


    # generate bounding box
    bb_w0, bb_h0, bb_ww, bb_hh = boundingbox.findRect(mask_remove)
    dh = bb_hh - bb_h0
    dw = bb_ww - bb_w0
    step0 = dh + dw
    step = dh + dw


    while step > 0:
        # remove vertical seam
        if dh <= dw:

            energy = sop.GradientEnergyMap(img)
            energy = energy + (10 * mask_protect)
            energy = energy + (-10 * mask_remove)

            seam = sop.findVerSeam(img, energy)

            # update mask_protect and mask_remove
            new_img = sop.removeVerSeam(img, seam)
            mask_remove = sop.removeVerSeam(mask_remove, seam)
            mask_protect = sop.removeVerSeam(mask_protect, seam)

            img = new_img
            dw = dw - 1

        # remove horizontal seam if dw > dh
        else:
            energy = sop.GradientEnergyMap(img)
            energy = energy + (100 * mask_protect)
            energy = energy + (-100 * mask_remove)

            seam = sop.findHorSeam(img, energy)

            # update mask_protect and mask_remove
            new_img = sop.removeHorSeam(img, seam)
            mask_remove = sop.removeHorSeam(mask_remove, seam)
            mask_protect = sop.removeHorSeam(mask_protect, seam)

            img = new_img
            dh = dh - 1

        #step = dh + dw
        step -=1
        print step
        if step == step0:

            cv2.imwrite('remove_initial_energy.jpg', mask_remove)

    return img
```

- **Function: RemoveObject (from removal.py)**
- This function removes the target region define by the pre-processed mask image from the original image. Seams in both horizontal direction and vertical direction are removed based on the updated masked energy map obtained by substracting 10000 (or 10/100/1000, this parameter has been studied) from the target pixels defined by the mask map, whereby these pixels have large negative energy in energy map and are preferred to be removed.This function also can be used to protect some area while removing targeted object by input a protection mask.
- input: original image and pre-preocessed mask image.
- output: an image with target object removed with reduced size. This size is determined by the size of the original image and that of the bounding box.

This function remove the desired seam one by one, however, the selection on a vertical seam or a horizontal seam depends on the shape of current bonding box. After every seam has been removed, the image, protection mask and removal mask all need to be updated for next round of seam removal.

Computation: Code Functional Description

```
def ProtectObject(img, hreduce, wreduce, mask_remove, mask_protect):
    # mask_remove and mask_protect are [0, 255] binary map

    dh = hreduce
    dw = wreduce
    step = hreduce + wreduce

    while step > 0:
        # remove vertical seam if dh > dw
        if dh < dw:

            energy = sop.GradientEnergyMap(img)
            energy = energy + (10 * mask_protect)
            energy = energy + (-10 * mask_remove)

            seam = sop.findVerSeam(img, energy)

            # update mask_protect and mask_remove
            new_img = sop.removeVerSeam(img, seam)
            mask_remove = sop.removeVerSeam(mask_remove, seam)
            mask_protect = sop.removeVerSeam(mask_protect, seam)

            img = new_img
            dw = dw - 1

        # remove horizontal seam if dw > dh
        else:
            energy = sop.GradientEnergyMap(img)
            print energy.shape
            energy = energy + (10 * mask_protect)
            energy = energy + (-10 * mask_remove)


            seam = sop.findHorSeam(img, energy)

            # update mask_protect and mask_remove
            new_img = sop.removeHorSeam(img, seam)
            mask_remove = sop.removeHorSeam(mask_remove, seam)
            mask_protect = sop.removeHorSeam(mask_protect, seam)

            img = new_img
            dh = dh - 1

        step = dh + dw

    return img
```

- **Function: ProtectObject (from protection.py)**
- This function resize the image while protecting target region define by the pre-processed mask image from the original image. Seams in both horizontal direction and vertical direction are removed based on the updated masked energy map obtained by adding 10000 (or 10/100/1000, this parameter has been studied) from the target pixels defined by the mask map, whereby these pixels have large possitive energy in energy map and are less preferred to be removed.
- input: original image and pre-preocessed protection mask image.
- output: an image with target object protected with reduced size. This size is determined by the input resolution.

This function remove the desired seams one by one. After every seam has been removed, the image, protection mask and removal mask (all zeros, a dummy mask) all need to be updated for next round of seam removal.

Computation: Code Functional Description

```python
def findVerSeam(image, energy_Map):

    # construct cumulative energy map and back pointers
    energy_cum = np.zeros((energy_Map.shape[0], energy_Map.shape[1] + 2), dtype=energy_Map.dtype)
    energy_cum[:, 0] = np.inf
    energy_cum[:, -1] = np.inf
    energy_cum[:, 1:-1] = energy_Map
    pointer = np.zeros(energy_Map.shape, dtype=np.uint8)

    # slow method
        # n represents row, and m represents column
    for n in range(1, energy_Map.shape[0]):
        for m in range(energy_Map.shape[1]):
            choices = energy_cum[n - 1, m:m + 3]
            smallest_m = choices.argmin()

            energy_cum[n, m + 1] = energy_Map[n, m] + choices[smallest_m]
            pointer[n, m] = smallest_m

    # extract n_seam seams
    extracted_seam = []
    column = energy_cum[-1, 1:-1].argmin()
    width = energy_Map.shape[0]
    for w in range(width - 1, 0, -1):
        extracted_seam.append(column)
        column += pointer[w, column] - 1
    extracted_seam.append(column)

    return np.array(extracted_seam[::-1], dtype=INDEX_DTYPE)
```

- **Function: findVerSeam (from seamOperation.py)**
- This function find a single vertical seam by constructing This function calculate the seam path in vertical direction with minimum energy loss using dynamica programming approach. The energy of each pixel is based on the gradient energy map. When calculating the accumulated energy for each pixel, the increase of energy induced by the removal of this pixel is considered.
- Input: image and corresponding energy map
- output: single vertical seam
- This function construct cumulative energy map and pointers matrix firstly. And then using dynamic programming to select pixels from last row to the first row to construct a single seam (reverse for return the list of pixels)
- (find horizontal seam is similar to this function)

Computation: Code Functional Description

```python
def removeVerSeam(image, seam):
    """
        input image is in color, not gray
        return an image with one vertical seam removed
    """

    shape = list(image.shape)
    shape[1] -= 1
    image_seamRemove = np.zeros(shape, dtype=image.dtype)

    for i in range(image.shape[0]):
        image_seamRemove[i, 0:seam[i]] = image[i, 0:seam[i]]
        image_seamRemove[i, seam[i]:] = image[i, seam[i] + 1:]

    return image_seamRemove
```

```python
def removeHorSeam(image, seam):
    """
        input image is in color, not gray
        return an image with one vertical seam removed
    """

    shape = list(image.shape)
    shape[0] -= 1
    image_seamRemove = np.zeros(shape, dtype=image.dtype)

    for i in range(image.shape[1]):
        image_seamRemove[0:seam[i], i] = image[0:seam[i], i]
        image_seamRemove[seam[i]:, i] = image[seam[i] + 1:, i]

    return image_seamRemove
```

- **Function: removeVer(Hor)Seam (from seamOperation.py)**
- This function remove one line from the input image in horizontal or vertical direction a seam of pixels selected by the finding seam function.
- input: a m*n image and the target seam
- output: an m*(n-1) or (m-1)*n image

Computation: Code Functional Description

```
def findSingleSeam(energyMap,img):
    """
        rewrite done except seam=[]
        input image: energy map derived from gray color image
        return a single vertical seam with lowest cost
    """

    FEM = np.zeros((energyMap.shape[0],energyMap.shape[1]),dtype=energyMap.dtype)

    FEM[0, :] = energyMap[0, :]
    pointer = np.zeros(energyMap.shape, dtype=np.uint8)

    # i represents row, and j represents column
    for i in range(1, energyMap.shape[0]):
        for j in range(0, energyMap.shape[1]):
            if j == 0:
                FEM[i, j] =energyMap[i, j] + min(FEM[i-1, j], FEM[i-1, j+1])

                # update pointer: which pixel was select in previous row
                choices = np.asarray([FEM[i-1, j], FEM[i-1, j+1]])
                pointer[i, j] = choices.argmin()+1

            elif j == energyMap.shape[1]-1:
                FEM[i, j] = energyMap[i, j] + min(FEM[i-1, j-1], FEM[i-1, j])

                # update pointer: which pixel was select in previous row
                choices = np.asarray([FEM[i-1, j-1], FEM[i-1, j]])
                pointer[i, j] = choices.argmin()

            else:
                Diff1 = abs(img[i, j+1, 0] - img[i, j-1, 0]) + abs(img[i, j+1, 1] - img[i, j-1, 1]) + abs(img[i, j+1, 2] - img[i, j-1, 2])
                Diff2 = abs(img[i-1, j, 0] - img[i, j-1, 0]) + abs(img[i-1, j, 1] - img[i, j-1, 1]) + abs(img[i-1, j, 2] - img[i, j-1, 2])
                Diff3 = abs(img[i-1, j, 0] - img[i, j+1, 0]) + abs(img[i-1, j, 1] - img[i, j+1, 1]) + abs(img[i-1, j, 2] - img[i, j+1, 2])

                FEM[i, j] = min(FEM[i-1, j-1] + Diff1 + Diff2, FEM[i-1, j] + Diff1, FEM[i-1, j+1] + Diff1 + Diff3)

                # update pointer: which pixel was select in previous row
                choices = np.asarray([FEM[i-1, j-1] + Diff1 + Diff2, FEM[i-1, j] + Diff1, FEM[i-1, j+1] + Diff1 + Diff3])
                pointer[i, j] = choices.argmin()


    # extract nth_seam seams
    extracted_seam = []
    column = FEM[-1,:].argmin()
    height = energyMap.shape[0]
    for h in range(height-1, 0, -1):
        extracted_seam.append(column)

        column += pointer[h,column] - 1
    extracted_seam.append(column)
    #print extracted_seam
    #print np.array(extracted_seam[::-1], dtype=INDEX_DTYPE)
```

Forward algorithm (this function is also for finding single seam, but different algorithm from the traditional backward algorithm)
The forward energy criterion, on top of the original forward energy criterion, can look forward at the image after each seam carving operation, taking into account the change of energy induced by the newly formed boundary between previously non adjacent pixels that become neighbours.

To implement the forward energy criterion, I created an new accumulated energy map function. This accumulated energy map function recorded the forward energy map (i.e., the accumulated gradient energy) of each pixel in regard to the top-left corner of the image, plus the change of energy induced by removing this pixel. This accumulated energy is then used to find out the next seam to carve.

```
for i in range(1, energyMap.shape[0]):
    for j in range(0, energyMap.shape[1]):
        if j == 0:
            FEM[i, j] =energyMap[i, j] + min(FEM[i-1, j], FEM[i-1, j+1])

            # update pointer: which pixel was select in previous row
            choices = np.asarray([FEM[i-1, j], FEM[i-1, j+1]])
            pointer[i, j] = choices.argmin()+1

        elif j == energyMap.shape[1]-1:
            FEM[i, j] = energyMap[i, j] + min(FEM[i-1, j-1], FEM[i-1, j])

            # update pointer: which pixel was select in previous row
            choices = np.asarray([FEM[i-1, j-1], FEM[i-1, j]])
            pointer[i, j] = choices.argmin()

        else:
            Diff1 = abs(img[i, j+1, 0] - img[i, j-1, 0]) + abs(img[i, j+1, 1] - img[i, j-1, 1]) + abs(img[i, j+1, 2] - img[i, j-1, 2])
            Diff2 = abs(img[i-1, j, 0] - img[i, j-1, 0]) + abs(img[i-1, j, 1] - img[i, j-1, 1]) + abs(img[i-1, j, 2] - img[i, j-1, 2])
            Diff3 = abs(img[i-1, j, 0] - img[i, j+1, 0]) + abs(img[i-1, j, 1] - img[i, j+1, 1]) + abs(img[i-1, j, 2] - img[i, j+1, 2])

            FEM[i, j] = min(FEM[i-1, j-1] + Diff1 + Diff2, FEM[i-1, j] + Diff1, FEM[i-1, j+1] + Diff1 + Diff3)

            # update pointer: which pixel was select in previous row
            choices = np.asarray([FEM[i-1, j-1] + Diff1 + Diff2, FEM[i-1, j] + Diff1, FEM[i-1, j+1] + Diff1 + Diff3])
            pointer[i, j] = choices.argmin()
```

- Input: image and corresponding energy map
- output: single vertical seam

Computation: Code Functional Description

```
def resize(img, new_resolution, mark=False):
    """
        parameters:
            img:original image to work with
            new_resolution: target resolution in list [w,h]
        output the resized image
    """
    # calculate the time, start
    start = timeit.default_timer()

    # print out the message to start seam carving
    current = 0
    orig = sys.stdout
    sys.stdout = sys.stderr

    # determine resize parameters
    n = new_resolution[0]
    m = new_resolution[1]
    m0 = img.shape[1]
    n0 = img.shape[0]
    Snumbers = (n - m0, m - n0)
    total_seam_number = abs(Snumbers[0]) + abs(Snumbers[1])

    # mark the seams by creating a copy of original images
    if mark:
        marked_image = img.copy()

    # this process go through two iteration to process horizontal resize
    # and then vertical resizing, but here for fig 5 and 8, there is only horiz
    # for imaging retargeting, using other file ordering.py to have a smarter o
    # iteration 1: horizontal seams (vertical resize)
    for seam_number in Snumbers:
        remember_seams = mark or seam_number > 0 #if need to marker image or do
        #need to remember calculated seams
        image_original = img

        # maintain seams and an "original index" 2D array, if needed
        if remember_seams:
            horizontal_range = np.arange(n0)
            seams = np.zeros((0, n0), dtype=INDEX_DTYPE)
            image_range = np.arange(m0, dtype=INDEX_DTYPE)
            image_idx = np.tile(image_range, (n0, 1))

        # delete seam one by one
        for s in range(abs(seam_number)):
            #find one optimal vertical seam and delete it

            energyMap = GradientMap(img)
```

Function resize is a supporting function to take the image and target resolution and remove seams to return the image with target dimension. This function calls the seam maker function and can be used to mark the selected seams on original image. This function also provide the image enlarge function.
This function is included in both seam carving and forward_carving.

Reference

1) Shai Avidan, Implementing Seam Carving for Image Resizing (forward energy)
2) Shai Avidan and Ariel Shamir, Seam Carving for Content-Aware Image Resizing (seam carving)
3) Michael Rubinstein, Ariel Shamir, Shai Avidan, Improved Seam Carving for Video Retargeting (forward energy)
4) Apurva Kumar, Arushi Ladha, and Masum Kumar Lodha, Seam Carving - Project Report (review)
5) Radhakrishna Achanta, Sheila Hemami, Francisco Estrada, and Sabine Su sstrunk, Frequency-tuned Salient Region Detection (FT saliency map)
6） http://pages.cs.wisc.edu/~moayad/cs766/ Moayad Alnammi (code structure and forward carving)

Teamwork *(required for teams only)*

This page should be different for each team member. The rest of the report should be the same for both of you.

- If you worked in a Team, describe your original division of labor, and how that worked out.
- Do you feel that the actual division of labor to develop your project, code, and report was equitable?
  - Explain your answer. This will not be shared with your teammate.

Appendix: Your Code

Submit your code in a folder in the resources.zip folder. Your code will be inspected, and may be run.

Code file and other resources:

https://drive.google.com/open?id=17WyDxykYpph1kIvXrtotF5FA8pD02Aoi

- **Due the the larger report file, all image files and complete code package is on google drive.**

Code Language: Python

List of code files:

- List each file here by name
- boundingbox.py
- removal.py
- protection.py
- seam_carving.py
- seamOperation.py
- Forward_carving.py

Credits or Thanks

- Thank all the instructor and TAs☺