# Δ05 Computer Vision 2018-2019
# Assignment 2 – Content aware image resizing

Student: Christodoulos Asiminidis

Professor: Christophoros Nikou

Department of Computer Science and Engineering

University of Ioannina

Ioannina

November 2018

One Drive URL address: https://1drv.ms/f/s!Aj9og9Uwr00xgcUwk8mZ0BIrNWE_qg

Computer vision is a computing science field that its main purpose is to deal with how computers can be made to obtain a high level perception from image processing. Computer vision mostly focuses on acquiring, processing, analyzing and understanding digital images and videos. It is highly connected with the theory behind artificial systems that extract information from images. The image data can have many different forms. It can be video sequences, views from multiple cameras or even multi-dimensional data. Examples of applications of computer vision include systems for:

1. Automatic inspection
2. Assisting humans in identification tasks
3. Detecting events
4. Interaction
5. Modelling objects or environments
6. Navigation
7. Organizing information

The goal of this project to get a clearer view of the content-aware resizing technique called Seam carving published by S. Avidan et al. in 2007. According to [1] seam carving is an optimal 8-connected path of pixels on a single image from top to bottom or left to right where optimality is defined by an image energy function. Hence, by successively removing or inserting seams we can reduce as well as enlarge the size of an image in both directions. For instance, for image reduction, seam selection ensures that while preserving the image structure, we remove more of the low energy pixels and fewer of the high energy ones.

At this project, our approach to content-aware resizing is to remove pixels in a judicious manner. Therefore, what is the greatest way to choose pixels that will be removed? That question automatically leads to the energy function described as follows:

$$e_1(I) = \left|\frac{d}{dx}I\right| + \left|\frac{d}{dy}I\right| \qquad (1)$$

Before we proceed to the implementation of the function shown above we firstly need to load the image, transform it to an array and extract information. For the purpose of this project I used Python 3.7.0 on windows 10 operating system in a computer that runs in a dual core 1.40GHz processor and a 4GB memory ram. The operating system is based on an x64 bit processor. Libraries for Python programming language used are scipy, numpy, os, matplotlib, cv2 and PIL.

I imported the necessary modules into the script, created a function that uploads an image from the computer locally. I reduced the width of the Austin.jpg image by 100 pixels and the height of the Disney.jpg image by 100 pixels as well.

Thus, I created a function named arrayImage by which it transforms images into an array.

Before implementing the energy function we firstly need to check which one of the smoothing techniques we are going to use. In this approach, we are using the Gaussian Blurring in which Gaussian kernel is used. It is done with the function cv2.GaussianBlur(). At this function, we should specify the width and the height of kernel which should be positive and odd. We also should specify the standard deviation in the X and Y directions, sigmaX and sigmaY respectively. If only sigmaX is specified, sigmaY is taken as equal to sigmaX. If both are given as zeros, they are calculated from the kernel size. Gaussian filtering is highly effective in removing Gaussian noise from the image We also ahould specify the standard deviation inX and Y direction, sigmaX and sigmaY respectively. If only sigmaX is specified then sigmaY is taken as same as sigmaX. If both are given as zeros, they are calculated from kernel size. Gaussian blurring is highly effective in removing Gaussian noise from the image. The function is as follows:

```
def gaussian_blur(img):
return cv2.GaussianBlur(img, (5,5),0,0)
```

At this point, it is worth mentioning that we should be very careful with double and uint8 conversions as we compute and most importantly display them. Filtering should be done with doubles as has already been done in the gaussian_blur function above.

Now, we convert images in gray scale with the grayscale_converted_function() below:

```
def grayscale_converted_scale(img):
            return cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Grayscale  receives the original image and converts it from BGR color space to gray. In this case, austin image has been converted to gray scale. Figure 1 below shows the results.

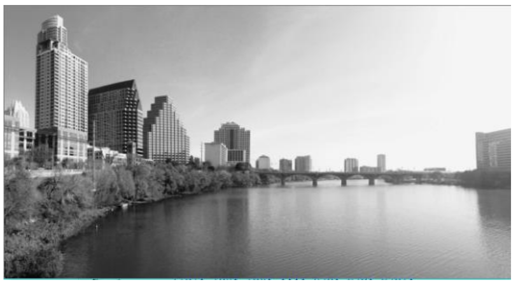

**Figure 1.** image converted into gray

As mentioned above, the energy function is described as:

$$e_1(I) = \left|\frac{d}{dx}I\right| + \left|\frac{d}{dy}I\right| \qquad (1)$$

The energy function has been implemented in Python as shown below:

```
def compute_energy_function(img):
    blur = gaussian_blur(img)
    gray = grayscale_converted_scale(blur)
        sy = compute_gradient_y(gray)
        sx = compute_gradient_x(gray)
    fabsx = np.absolute(sx)
        fabsy = np.absolute(sy)
        return cv2.add(fabsx,fabsy)
```

In that function we exactly calculate the equation (1). Specifically, we calculate the sum of the absolute values of the gradients in each direction. Particularly, the energy function outputs for the Austin.jpg and Disney.jpg images are the following ones:

Well, Given an energy function e, we can define the cost of a seam as $\mathrm{E(s)} = \mathrm{E(Is)} = \sum \mathrm{n\ i} = 1\ \mathrm{e(I(si))}$. We look for the optimal seam $\mathrm{s}*$ that minimizes this seam cost:

$$s^* = min_s \mathrm{E(s)} = min_s \sum_{i=1}^{n} e(I(s_i)) \qquad (2)$$

Now, we are going to calculate the cumulative minimum energy maps M(i,j) for each direction, vertically and horizontally, respectively. For all possible connected seams for every entry (I,j). The cumulative minimum energy is given by the equation:

$$\mathrm{M(i,j)} = \mathrm{e(i,j)} + \min(\mathrm{M(i-1,j-1)}, \mathrm{M(i-1,j)}, \mathrm{M(i-1,j+1)}) \qquad (3)$$

The next task is to calculate the first selected horizontal seam and the vertical one. Python Implementation has been made on Austin.jpg image as follows for both directions:

```
#Synarthsh pou ypologizei thn athroistikh synarthsh katakoryfa
def cumulative_energies_vertical(energy):
  height, width = energy.shape[:2]
  energies = np.zeros((height, width))

  for i in range(1, height):
    for j in range(width):
      left = energies[i - 1, j - 1] if j - 1 >= 0 else 1e6
      middle = energies[i - 1, j]
      right = energies[i - 1, j + 1] if j + 1 < width else 1e6
      energies[i, j] = energy[i, j] + min(left, middle, right)

  return energies

energies_vertical=cumulative_energies_vertical(energy)


#Synarthsh pou ypologizei thn atroistikh synarthsh orizontia
def cumulative_energies_horizontal(energy):
    height, width = energy.shape[:2]
    energies=np.zeros((height, width))
    for j in range(1, width):
        for i in range(height):
            top = energies[i-1,j-1] if i-1>=0 else 1e6
            middle = energies[i, j-1]
            bottom = energies[i+1, j-1] if i+1<height else 1e6
            energies[i,j] = energy[i,j]+min(top, middle, bottom)

    return energies
```

```
energies_horizontal=cumulative_energies_horizontal(energy)
```

Now, we compute the horizontal and the vertical seam. We find the optimal order using a transport map T that specifies, for each desired target image size, the cost of the optimal sequence of horizontal and vertical seam removal operations. That is, entry T(r,c) holds the minimal cost needed to obtain an image of any size . We compute T using dynamic programming.

Starting at T(0,0) = 0 we fill each entry (r,c) choosing the best of two options - either removing a horizontal seam from an image or removing a vertical seam from an image. Well, the two different functions implemented to find the horizontal and the vertical seams are the following ones:

```
#synarthsh pou vriskei ta orizontia seams

def find_horizontal_seam(energies_horizontal):

    height, width = energies.shape[:2]

    previous = 0

    seam = []



    for i in range(width - 1, -1, -1):

        col = energies[:, i]



        if i == width - 1:

            previous = np.argmin(col)


        else:
            top = col[previous - 1] if previous - 1 >= 0 else 1e6
            middle = col[previous]
            bottom = col[previous + 1] if previous + 1 < height else 1e6

            previous = previous + np.argmin([top, middle, bottom]) - 1

        seam.append([i, previous])

    return seam

horizontal_seam=find_horizontal_seam(energies_horizontal)

#synarthsh pou vriskei ta katakoryfa seams
def find_vertical_seam(energies_vertical):
        im = np.transpose(energies_vertical)
        u = find_horizontal_seam(im)
        for i in range(len(u)):
                temp = list(u[i])
                temp.reverse()
                u[i] = tuple(temp)
        return u

vertical_seam=find_vertical_seam(energies_vertical)
```

The results we get from finding the seams horizontally and vertically have been illustrated in the Figure 2 below:

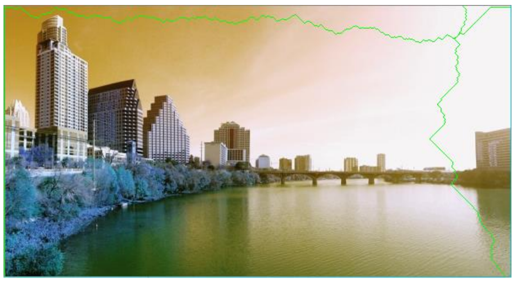

**Figure 2.** Horizontal and Vertical seam shown in austin image

Seams are removed either horizontally or vertically using the following two functions:

```
#Synarthsh pou afairei ta seams orizontia
def remove_horizontal_seam(imgarray,horizontal_seam):
    height, width, bands = imgarray.shape
    removed=np.zeros((height-1,width,bands))
    for x,y in reversed(horizontal_seam):
        removed[0:y,x]=imgarray[0:y,x]
        removed[y:height-1,x]=imgarray[y+1:height,x]
    return removed

#synarthsh pou afairei ta seams katakoryfa
def remove_vertical_seam(imgarray,vertical_seam):
    height, width, bands = imgarray.shape
    removed=np.zeros((height-1,width,bands))
    for x,y in reversed(vertical_seam):
        removed[y,0:x]=imgarray[y,0:x]
        removed[y,x:width-1]=imgarray[y,x+1:width]
    return removed
```

Results of removing the seams horizontally and vertically shown below:

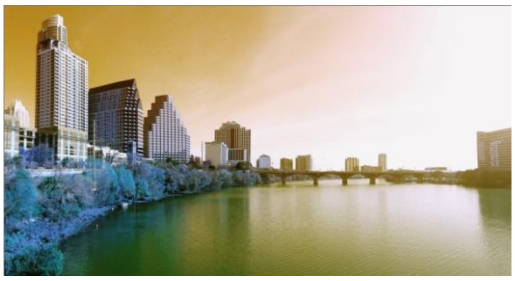

**Figure 3.** Austin image has been resized due to horizontal seam removal

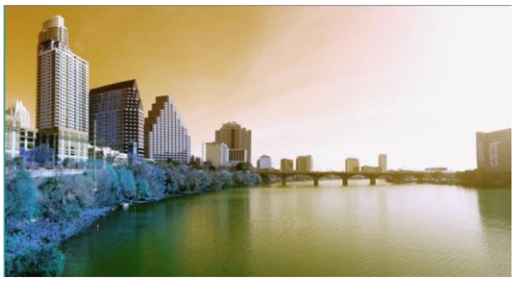

**Figure 4.** Austin image has been resized due to vertical seam removal

Now, the change I would like to make on the energy function will be on filtering. I am using Median Filtering for smoothing Austin.jpg image. Specifically, in Python programming language, the function cv2.medianBlur() computes the median of all the pixels under the kernel window and the central pixel is replaced with this median value. This is highly effective in removing salt-and-pepper noise. One interesting thing to note is that, in the Gaussian and box filters, the filtered value for the central element can be a value which may not exist in the original image. However this is not the case in median filtering, since the central element is always replaced by some pixel value in the image. This reduces the noise effectively. The kernel size must be a positive odd integer. Particularly, in this implementation I am adding 30% noise to my original image and then, use the median filter. The function that gives us a result the median filtering is the following one:

```
#Using Median filtering
#Vazw ws parameters ton pinaka ths eikonas kai ton akeraio arithmo poy thelw na ginei blurring
def median_filtering(imagearray,int):
    blur=cv2.medianBlur(imagearray,int)
    plt.subplot(121),plt.imshow(imgaustin),plt.title('Original')
    plt.xticks([]), plt.yticks([])
    plt.subplot(122),plt.imshow(blur),plt.title('Median Filtering')
    plt.xticks([]), plt.yticks([])
    plt.show()
```

Subsequently, I plot the original image and the median filtered image using matplot.lib module in Python programming language and I get the following as illustrated in Figure 3:

**Figure 5:** Original picture shown on the left and the median filtered picture shown on the right

Now, the real results. At this project, three different images have been performed the algorithm described above.

Eddy.jpg image has been performed in the first place. The size of the image is 20.4KB. Its dimensions are 256x256 and the resolution vertically and horizontally is 96dpi. Firstly, we examined the seam curvature and then we removed the horizontal and vertical seam.

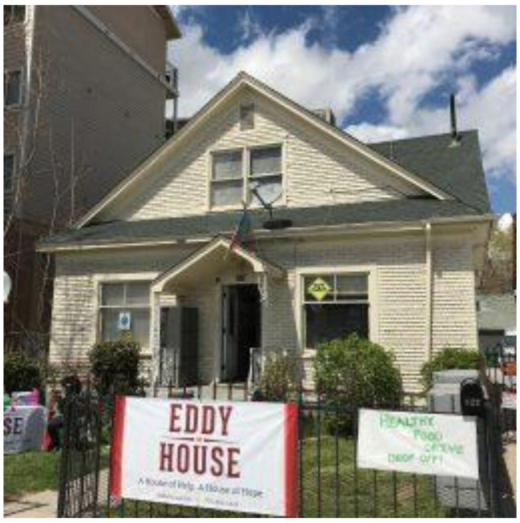
EDDY
HOUSE
A House of Help. A House of Hope
HEALTHY FOOD OFFERS DROP OFF

**Figure 6.** eddy.jpg original image

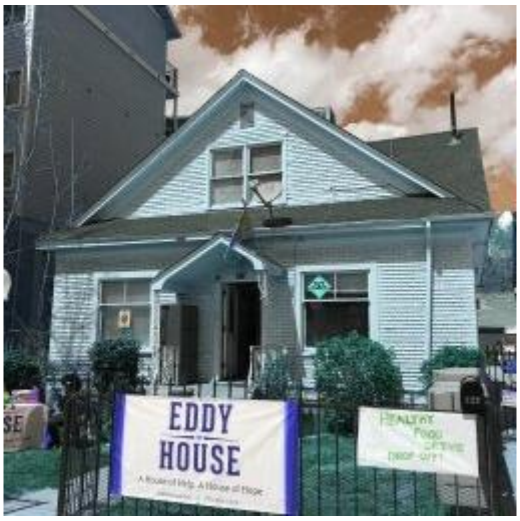

**Figure 7.** Horizontal seam of eddy.jpg image

The size of the image turned to 35.4KB. Its new dimensions are 256x255. The resolution of the horizontal seam has remained the same.

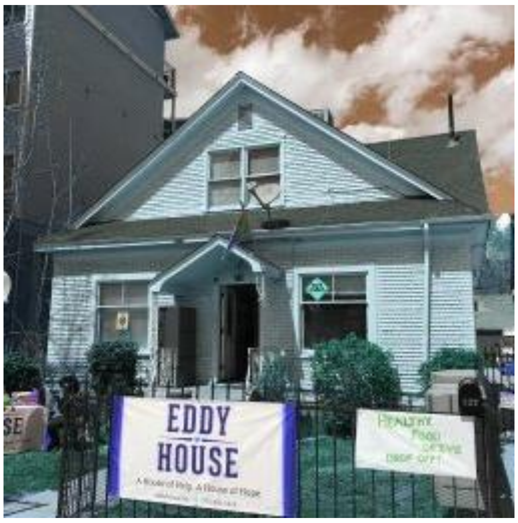

**Figure 8.** Vertical seam of eddy image

The size of the image turned to 35.6KB. Now, we see that the dimensions have switched to 255x256. The resolution has remained the same as the original one.

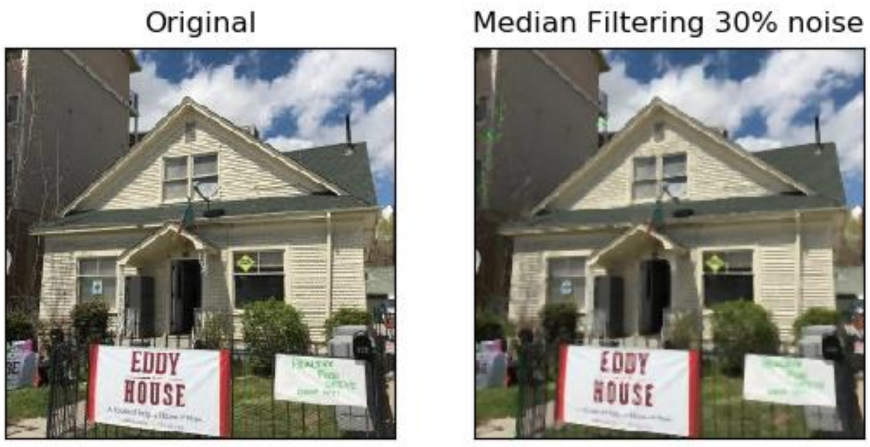

**Figure 9.** Original picture of eddy.jpg image shown on the left and the median filtered picture shown on the right with 30% of noise

The following image that has been examined is a car presented on the foreground. Initially, the car image has size of 13.4KB, 400x267 dimensionality and 72 dpi resolutions for each direction. Let’s see the results we get.

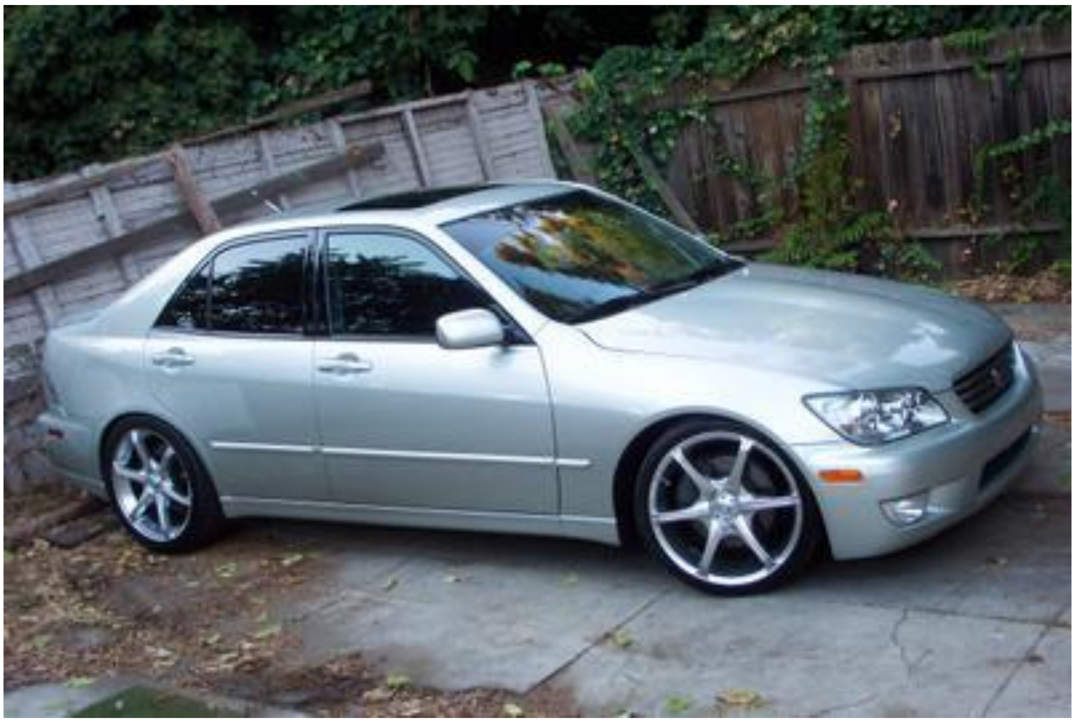

**Figure 10.** car.jpg original image

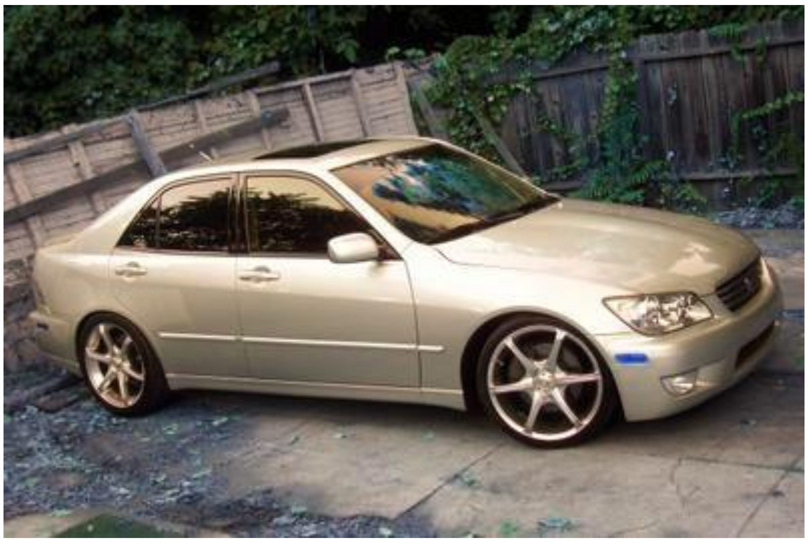

**Figure 11.** Horizontal seam of car.jpg image

The size of the image turned to 45.9KB. Its new dimensions are 400x266. The resolution of the horizontal seam has changed to 96 dpi for both horizontal and vertical resolution.

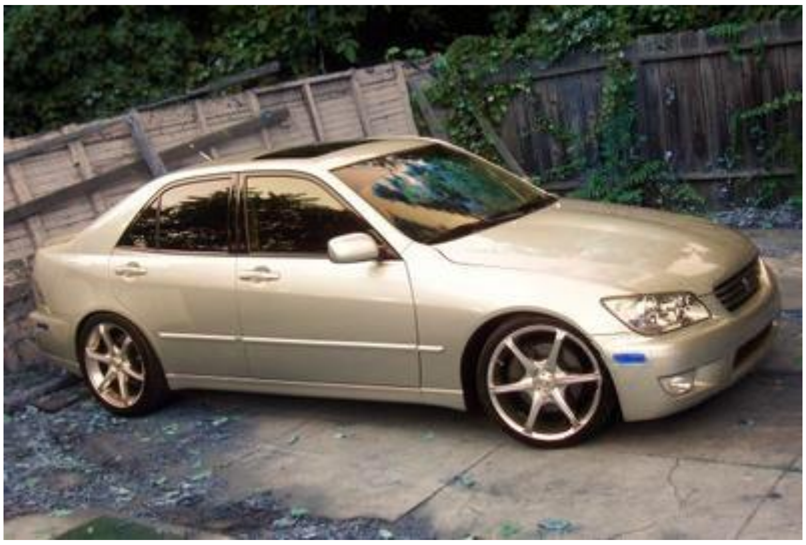

**Figure 12.** Vertical seam of car.jpg image

The size of the image turned to 46.9KB. Now, we see that the dimensions have switched to 399x267. The resolution has also changed to 96 dpi for each direction.

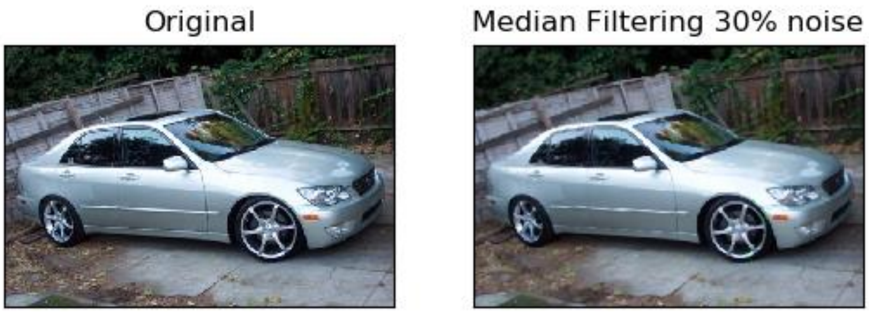

**Figure 13.** Original picture of car.jpg image shown on the left and the median filtered picture shown on the right with 30% of noise

Finally, the knit.jpg has been given as an input to the algorithm. The original image has size of 40KB, 350x241 dimensions and 96 dpi resolutions for each direction.

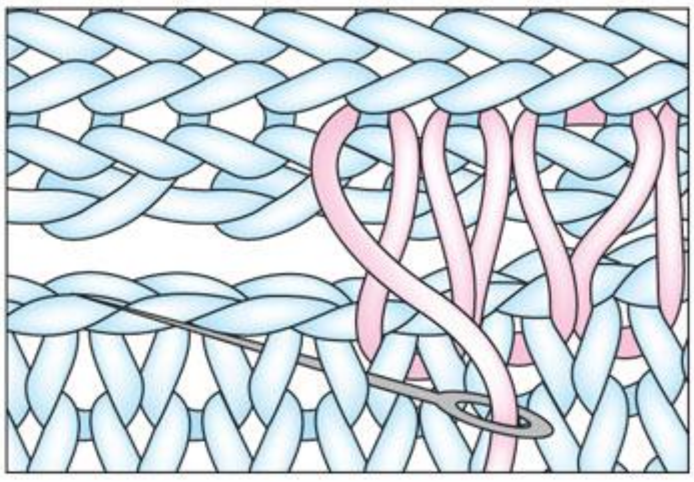

**Figure 14.** knit.jpg original image

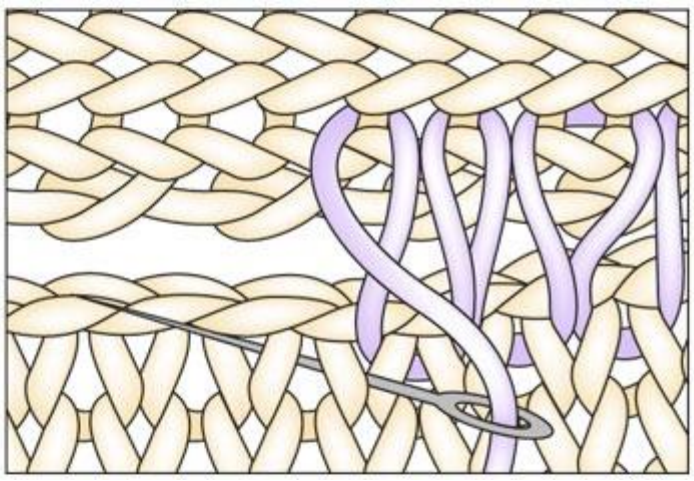

**Figure 15.** Horizontal seam of knit.jpg image

The size of the image turned to 56KB. Its new dimensions are 350x240. The resolution of the horizontal seam has 96 dpi resolutions for both horizontal and vertical resolution which has remained exactly the same as the original image.

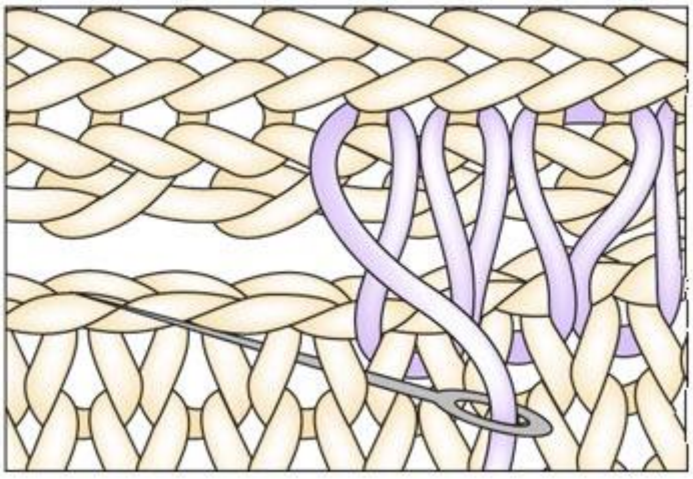

**Figure 16.** Vertical seam of knit.jpg image

The size of the image turned to 56.2KB. Now, we see that the dimensions have switched to 349x241. The resolution remains 96 dpi for each direction.

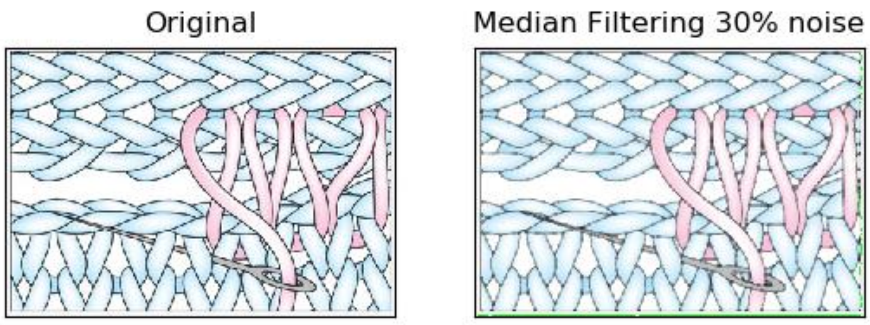

**Figure 17.** Original picture of knit.jpg image shown on the left and the median filtered picture shown on the right with 30% of noise