Algoritmo Seam Carving. Implementación y aplicaciones

Realizado por:
José Miguel Pérez Navarrete
David Lorca Campos

Introducción

El algoritmo estudiado conocido como Seam Carving [1] trata de lidiar con la necesidad de modificar el tamaño de una imagen de forma óptima. Existen otras técnicas como un escalado de la imagen estándar en el cual se eliminan filas o columnas de píxeles de forma uniforme o la técnica conocida como *Cropping* la cual elimina píxeles de la periferia de la imagen. Sin embargo, estas técnicas solamente utilizan restricciones geométricas y no tienen en cuenta el propio contenido de la imagen. Seam Carving realiza un escalado de la imagen acorde al contenido de la imagen.

El algoritmo utiliza una función de energía que define la importancia de cada uno de los píxeles de la imagen. Un *Seam* es un camino de píxeles de baja energía completamente conectado que cruza la imagen de arriba a abajo o de izquierda a derecha. Si sucesivamente se eliminan o añaden estos *seams* es posible reducir o aumentar el tamaño de la imagen en ambas direcciones.

Definición del algoritmo

El algoritmo comienza definiendo una función de energía basada en los gradientes de la imagen. En concreto la expresión es la siguiente:

$$e(I) = \left|\frac{\partial}{\partial x}I\right| + \left|\frac{\partial}{\partial y}I\right|$$

En el caso de reducir el tamaño de la imagen se pueden plantear diferentes estrategias. Una de ellas sería preservar la energía de la imagen mediante la eliminación de píxeles con menor energía. Si se realiza directamente se perdería la forma rectangular de la imagen ya que se eliminarían distinto número de píxeles por cada fila/columna. Otra forma sería eliminar por cada fila o columna el mismo número de píxeles. Esto mantendría la forma rectangular pero produciría un efecto de zig-zag en la imagen. Existen otras técnicas basadas en eliminación de columnas con menor energía, auto-cropping, que selecciona una sub-ventana de la imagen que contenga la mayor energía, etc. Por tanto, es necesario una manera menos restrictiva que las anteriores y que preserve el contenido de la imagen. El método definido por el algoritmo se basa en los *seams*. Se define como *seam* de la siguiente manera:

Dada una imagen I de tamaño n x m se define un seam vertical cómo:

$$s^x = \left\{ s_i^x \right\}_{i=1}^{n} = \{(x(i),\ i)\}_{i=1}^{n},\ tal\ que\ \forall i,\ |x(i) - x(i-1)| \leq 1$$

Dónde $x:\ [1,\ ...,\ n] \rightarrow [1,\ ...,\ m]$

Y un *seam* horizontal cómo:

$$s^y = \left\{ s_j^y \right\}_{j=1}^{m} = \{(x(j),\ j)\}_{j=1}^{m},\ tal\ que\ \forall j,\ |x(j) - x(j-1)| \leq 1$$

Dónde $x:\ [1,\ ...,\ m] \rightarrow [1,\ ...,\ n]$

Lo que estas definiciones quieren decir es que un seam es un camino de píxeles totalmente conectado, es por ello que se debe cumplir la condición $|x(i) - x(i-1)| \leq 1$ ó $|x(j) - x(j-1)| \leq 1$ para que no se produzcan discontinuidades. A continuación se define el coste de un seam cómo $E(s) = E(I_s) = \sum_{i=1}^{n} e(I(s_i))$, siendo $I_s = \{I(s_i)\}_{i=1}^{n}$ y $e(\cdot)$ el operador descrito al comienzo. El objetivo por tanto consiste en encontrar el seam óptimo, definido cómo:

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

Es decir, el seam que menor coste acumulado proporcione, o lo que es lo mismo, el camino con menor energía total. El seam óptimo puede ser calculado mediante programación dinámica, explorando posibles caminos y eligiendo el que menor coste proporcione. Para ello, se calcula, dada una imagen, la matriz M que proporciona información del coste mínimo acumulado para cada posible seam para cada pixel de la imagen. Esta matriz se calcula (para el caso de seams verticales):

$$M(i,j) = e(i,j) + min[M(i-1,j-1),\ M(i-1,\ j)\ ,\ M(i-1,\ j+1)]$$

Una vez obtenida la matriz M, se escoge el mínimo de la última fila, siendo este el fin del *seam* óptimo. Por tanto, recorriendo el camino inverso a partir de dicha posición se obtiene el *seam* deseado. Para el caso de seams horizontales el razonamiento es el mismo.

Implementación del algoritmo base

La implementación del algoritmo se ha realizado utilizando el lenguaje de programación Python. El algoritmo base consta del cálculo de la matriz M y de la obtención del camino óptimo.

La función que realiza todo este cálculo se llama ***SeamCarving(image, VoH)*** y tiene como entradas la imagen original y un parámetro que determina si se calcula un seam vertical u horizontal. Este código se encuentra en el fichero *SeamCarving.py*

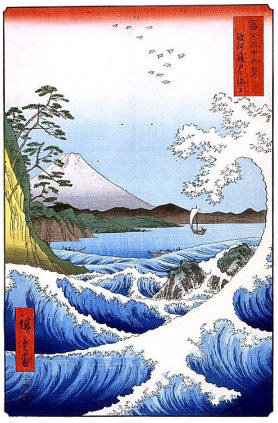
La primera parte del código se encarga de la aplicación del operador e(·) sobre toda la imagen a partir de los gradientes. Estos gradientes se han obtenido mediante los filtros de Sobel.

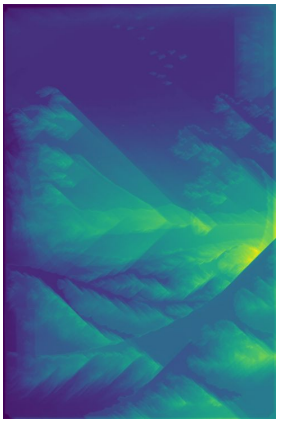
```
def SeamCarving(image, VoH): #If True - Vertical, if False - Horizontal

  image = image[:,:,0]

  #Sobel's filter
  Gx = np.array([[-1, 0, 1],[-2, 0, 2],[-1, 0, 1]])
  Gy = np.transpose(Gx)

  Ix = scipy.signal.convolve2d(image, Gx, mode='same')
  Iy = scipy.signal.convolve2d(image, Gy, mode='same')

  #Energy operator
  e = abs(Ix) + abs(Iy)
```

Para resumir, se mostrará el cálculo de la matriz M y del seam óptimo para el caso de un seam horizontal.

```
#Horizontal Seam
M = np.zeros(e.shape)

for j in range(e.shape[1]): #Columns
  for i in range(e.shape[0]): #Rows

    if (i-1)>=0 and (j-1)>=0:
      e1 = M[i-1,j-1]
    else:
      e1 = 1e20

    if (j-1)>=0:
      e2 = M[i,j-1]
    else:
      e2 = 1e20

    if (j-1)>=0 and (i+1)<=(e.shape[0]-1):
      e3 = M[i+1,j-1]
    else:
      e3 = 1e20

    if e1==1e20 and e2==1e20 and e3==1e20:
      M[i,j] = e[i,j]
    else:
      M[i,j] = e[i,j] + min(e1,e2,e3)

minCost = np.argmin(M[:, e.shape[1]-1])
```

Primero se obtiene la matriz M de la forma que se muestra en la imagen anterior. Los sucesivas sentencias If previenen al algoritmo de problemas en las fronteras de la imagen. En caso de intentar acceder a un píxel que se encuentre fuera de la imagen, se le asigna un coste muy alto para que así el algoritmo no lo tenga en cuenta.





En la imagen anterior derecha se muestra el valor de la matriz M para cada uno de los píxeles de la imagen original (izquierda).

El siguiente paso es calcular el camino con menor coste acumulado, para ello se ha utilizado el siguiente código:

```
#Removal pixel list
pixelListH = []
pixelListH.append((np.argmin(M[:,e.shape[1]-1]), e.shape[1]-1))

i = np.argmin(M[:,e.shape[1]-1])
j = e.shape[1]-1

while j!=0:
  e1 = M[i, j-1]

  if (i-1)<0:
    e2 = 1e20
  else:
    e2 = M[i-1, j-1]

  if (i+1)>e.shape[0]-1:
    e3 = 1e20
  else:
    e3 = M[i+1, j-1]

  min_pos = np.argmin([e1, e2, e3])

  if min_pos == 0:
    pixelListH.append((i, j-1))
  if min_pos == 1:
    pixelListH.append((i-1, j-1))
    i = i-1
  if min_pos == 2:
    pixelListH.append((i+1, j-1))
    i = i+1

  j -= 1
return pixelListH, minCost
```

Se comienza la búsqueda a partir del mínimo de la última columna de la imagen. A partir de ahí se recorre el camino cuyas posiciones tienen el menor coste acumulado (camino inverso al seguido para rellenar la matriz M). Al igual que antes, las sentencias If previenen de problemas al intentar acceder a una posición que no está dentro de la imagen. Como resultado devuelve una lista con los píxeles del *seam* óptimo. Algunos resultados para distintas imágenes son:

Para el caso de seams verticales sería totalmente análogo y los resultados son los siguientes:



Aplicaciones del algoritmo

- **Reducción del tamaño de la imagen**

  Utilizando el algoritmo base descrito anteriormente es posible reducir el tamaño de la imagen teniendo en cuenta el contenido de ésta. Para ello, lo primero que se ha realizado es una función, que dada una imagen y una lista de píxeles a eliminar, genera una nueva imagen con esos píxeles borrados. Esta función se encuentra en el archivo *removePixels.py* y permite tanto eliminar seams verticales como horizontales. Su funcionamiento se basa en la copia de cada píxel en una nueva imagen saltándose aquellos que pertenezcan a la lista de puntos a borrar. El código es el siguiente:

```
def removePixels(image, pixelList, VoH): # VoH: 1 if Vertical, 0 if Horizontal
  pixelList.reverse()

  if(VoH == 1):
    resized_image = np.zeros([image.shape[0], image.shape[1]-1, 3])

    for ind_fila in range(image.shape[0]):
      for ind_Colum in range(image.shape[1]-1):
        if(pixelList[ind_fila][1] > ind_Colum):
          resized_image[ind_fila, ind_Colum, 0] = image[ind_fila, ind_Colum, 0]
          resized_image[ind_fila, ind_Colum, 1] = image[ind_fila, ind_Colum, 1]
          resized_image[ind_fila, ind_Colum, 2] = image[ind_fila, ind_Colum, 2]
        else:
          resized_image[ind_fila, ind_Colum, 0] = image[ind_fila, ind_Colum+1, 0]
          resized_image[ind_fila, ind_Colum, 1] = image[ind_fila, ind_Colum+1, 1]
          resized_image[ind_fila, ind_Colum, 2] = image[ind_fila, ind_Colum+1, 2]
  else:
    resized_image = np.zeros([image.shape[0]-1, image.shape[1], 3])

    for ind_Colum in range(image.shape[1]):
      for ind_fila in range(image.shape[0]-1):
        if(pixelList[ind_Colum][0] > ind_fila):
          resized_image[ind_fila, ind_Colum, 0] = image[ind_fila, ind_Colum, 0]
          resized_image[ind_fila, ind_Colum, 1] = image[ind_fila, ind_Colum, 1]
          resized_image[ind_fila, ind_Colum, 2] = image[ind_fila, ind_Colum, 2]

        else:
          resized_image[ind_fila, ind_Colum, 0] = image[ind_fila+1, ind_Colum, 0]
          resized_image[ind_fila, ind_Colum, 1] = image[ind_fila+1, ind_Colum, 1]
          resized_image[ind_fila, ind_Colum, 2] = image[ind_fila+1, ind_Colum, 2]

  return resized_image
```

A continuación se ha creado una función llamada ***reduceSC(image, VoH)*** que se encuentra en el fichero *reduceSC.py* que se encarga de calcular sucesivamente seams, tanto verticales como horizontales (de forma alternativa) y los va eliminando de la imagen. El código usado es el siguiente:

```python
from SeamCarving import SeamCarving
from removePixels import removePixels

def SeamCarvingBase(image, VoH):
    pixelList, minCost = SeamCarving(image, VoH)
    image = removePixels(image, pixelList, VoH)
    return image


def reduceSC(image, final_resolution):
    row_final, col_final = final_resolution
    row_ini, col_ini = image.shape[0], image.shape[1]

    if (((row_ini - row_final) < 0) or ((col_ini - col_final) < 0)):
        print('Incorrect target resolution')
    else:
        step_row = row_ini - row_final
        step_col = col_ini - col_final

        if (step_row <= step_col):
            diff = step_col - step_row
            z = 1
        else:
            diff = step_row - step_col
            z = 0

        n_iter = step_row + step_col - diff

        VoH = True
        for i in range(n_iter):
            image = SeamCarvingBase(image, VoH)
            VoH = not VoH

        if (z == 1):
            for i in range(diff):
                image = SeamCarvingBase(image, True)
        else:
            for i in range(diff):
                image = SeamCarvingBase(image, False)

    return image
```

En la parte superior se ha creado una función que calcula el seam óptimo y lo elimina y en la parte inferior la función que lo realiza recursivamente. Ya que la reducción en cada dimensión puede ser distinta, se realiza una comprobación y se eliminan sems de forma alternativa hasta que el tamaño de la menor de las dimensiones se alcance. Posteriormente se realiza de forma recursiva lo mismo para una sola dimensión. Algunos resultados de este código se muestran a continuación. Para una reducción horizontal:

738 x 486

738 x 390

288 x 352

288 x 280

288 x 352

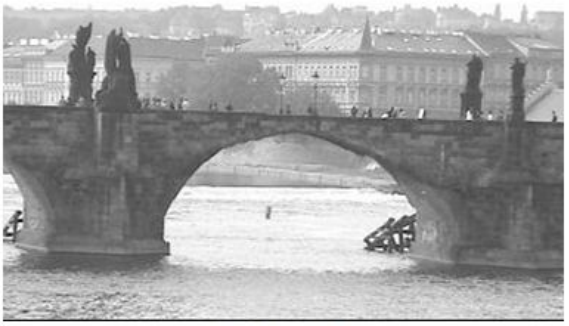
288 x 280

Para el caso de reducción vertical:

738 x 486

690 x 486

288 x 352



200 x 352

288 x 352

200 x 352

Y por último la combinación de ambas:

738 x 486

690 x 390

288 x 352

200 x 280

288 x 352

270 x 285

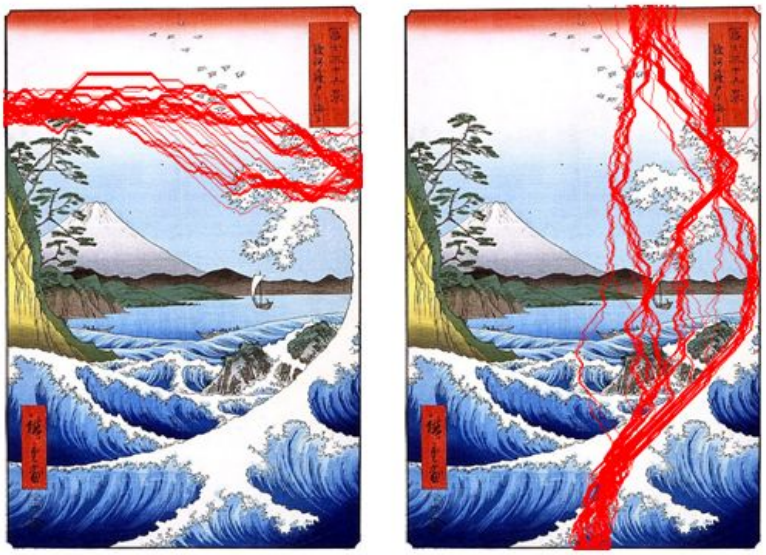
- **Alargamiento de la imagen**

A partir de la eliminación de seams horizontales y verticales, podemos invertir el proceso mediante la inserción de seams artificiales. Dicho seams artificiales pueden ser el mismo seam cuya función de energía es la menor de todas. Sin embargo, añadir el mismo seams reiteradas veces supone una artificialidad que no queremos en absoluto.

Nuestro objetivo es el aumento del tamaño mediante seams correspondiente de la imagen sin que pierda la apariencia natural que posee la imagen desde el principio.

La solución que hemos tomado consiste en manejarnos con una “matriz M”, que representa la función de costes de los píxeles de la imagen inicial, según si el alargamiento es en columnas o en filas.

Desde dicha matriz “M” obtenemos una cantidad determinada de de seams. Cada vez que se obtiene un seam óptimo, se guarda y se le establece a ese camino un coste muy alto en la matriz M de tal forma que no se vuelva a escoger. De esta forma, se obtiene el número deseado de seams con menor coste. Un ejemplo de lo que se está realizando muestra en la siguiente imagen:



Seam horizontal Seam vertical

Una vez obtenidos los seams, se duplican los píxeles pertenecientes a estos produciendo un alargamiento de la imagen.

Toda esta aplicación viene recogida en la librería “EnlargeSC.py”. A continuación se muestra un ejemplo de su uso. Algunos resultados son:

738 x 486 800 x 520

288 x 352

320 x 400

607 x 960

640 x 1100

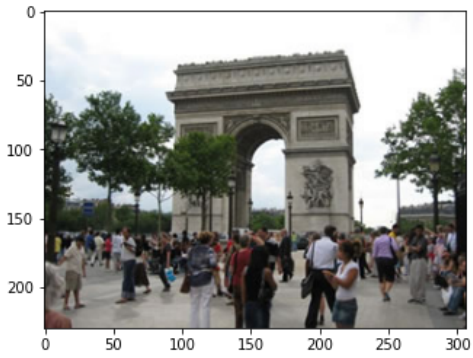
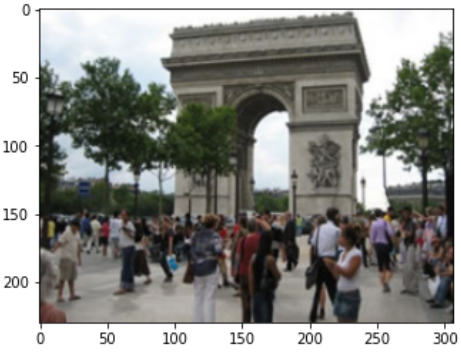
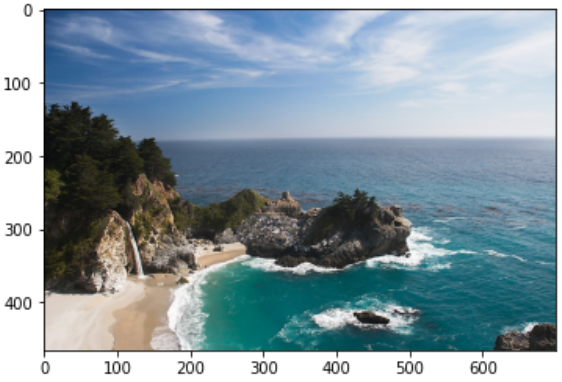
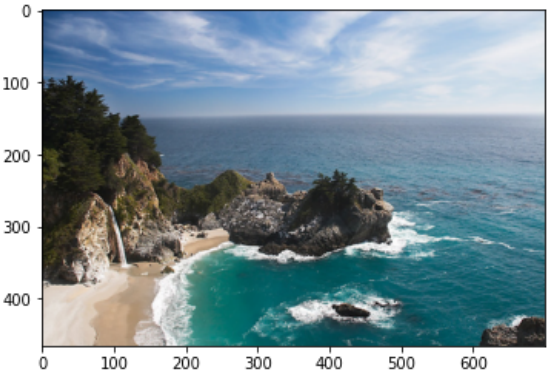
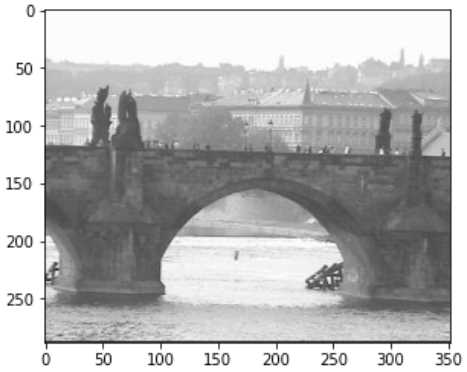
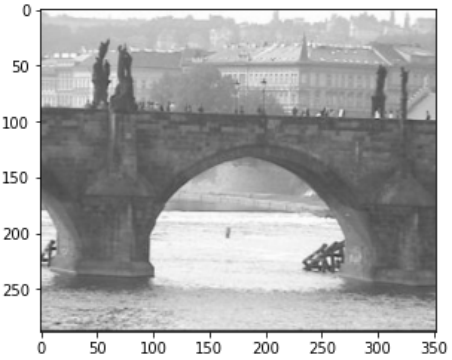
- **Amplificación del contenido**

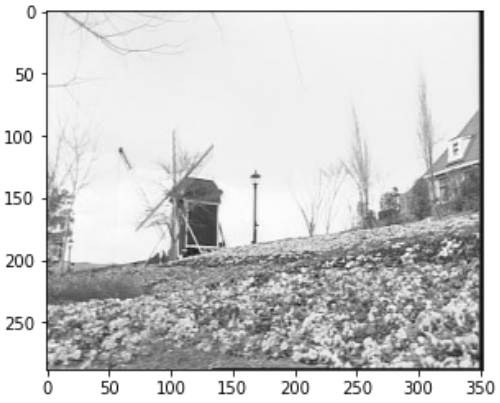
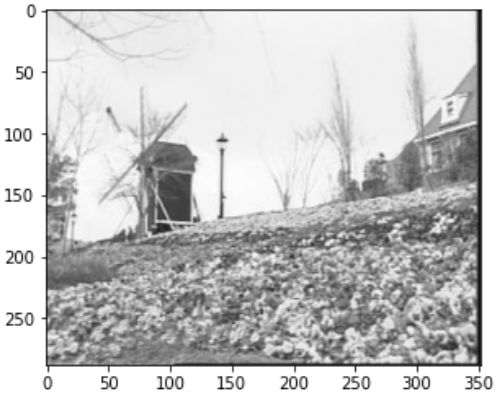
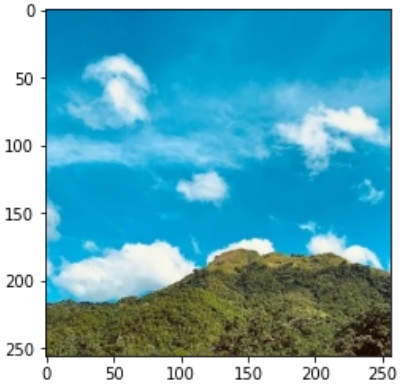
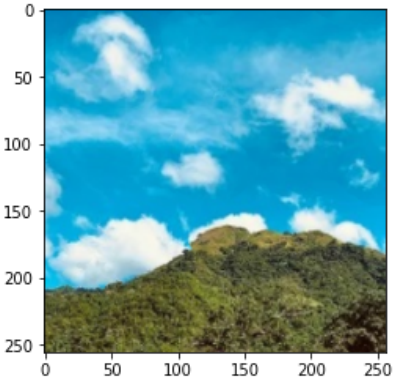
  No solo ***Seams Carving*** nos permite reducir y alargar la imagen, sino que también, con la misma resolución de la imagen, podemos ampliar el contenido de su imagen, mostrando mucho más los detalles de la foto.

  Dicha aplicación se ha implementado aumentando escalarmente la foto a un tamaño determinado y a continuación aplicando el algoritmo base ***SeamCarving.py*** hasta que logre tener la misma resolución del inicio. Simplemente el algoritmo base elimina los píxeles que son en realidad subpixeles de la imagen inicial.

  La aplicación viene recogida en la librería “ContentAmplification.py”. A continuación se muestra ejemplos de su uso. Las fotos de la izquierda representa la imagen inicial mientras que las imágenes de la derecha es el resultado de la amplificación del contenido.

  Cabe mencionar que la resolución final del programa es la misma que la resolución inicial de la imagen. Dentro de este documento puede ver casos donde no se vea esa igualdad de tamaños exacta por la colocación de estas imágenes dentro del documento.

- **Eliminación de objetos**

  A partir de ***Seams Carving***, podemos eliminar los píxeles de una región específica de la foto, que para ello forzaremos que los seams pase por la zona seleccionada.

  Una solución que planteamos para la eliminación de objetos fue la disminución al mínimo de los costes para dichos píxeles seleccionados, usando una matriz “M” auxiliar obtenida de la imagen inicial. Sin embargo, este planteamiento no es suficiente ya que debido a la distribución de energía de la foto no tiene por qué converger siempre a la región que minimizamos.

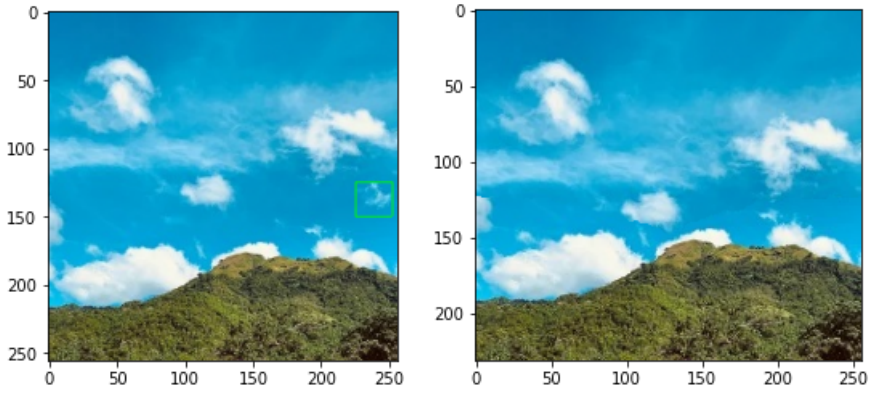
  Es por ello que no solo debemos de disminuir el coste en las zonas que queremos eliminar, sino aumentar el coste en aquellas regiones forzando que el seams pase por dicha región. La elección de la zona a maximizar su coste es fundamental ya que implica que esos pixeles nunca se eliminan ni pasara ningún seams por ello.

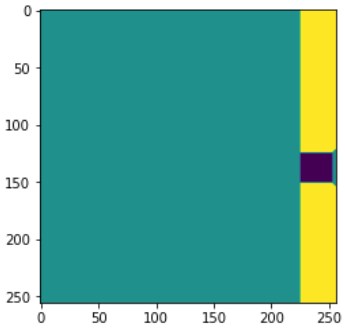
  Nuestra solución planteada para eliminación de objetos es marcar la zona mediante una ROI que, según si la dimensión de la ROI es menor en altura o en anchura, opta por realizar el ***Seams Carving*** mediante seams horizontales o seams verticales, reduciendo el tiempo de ejecución. Una vez decidido si se realiza mediante seams horizontales o verticales, se obtiene la matriz de coste que se modifica, minimizando

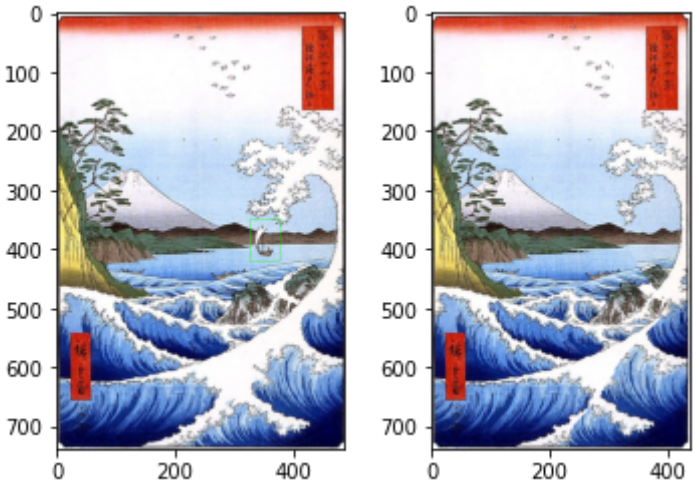
el coste en los píxeles seleccionados y maximizándolo en las regiones de fuera; que en el caso de tratarse de seams horizontales, situadas entre la misma posición de columnas y que son distintas de la región seleccionada; y en dos regiones que dibujan triángulos equiláteros, cuya hipotenusa va desde una de las esquinas de la ROI hasta el límite inferior o superior (si se trata de una esquina inferior o superior) de la foto. De esta manera forzamos que el seams siempre pasa por la zona de la ROi. Más adelante, para dar mayor claridad, se detalla con una fotografía dichas regiones mostrando la matriz M.

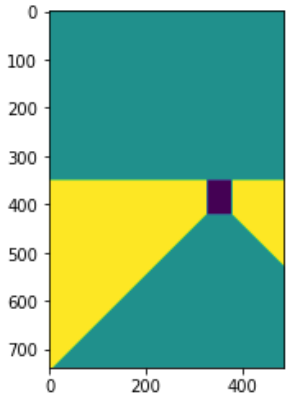
Finalmente, obtenida la matriz M se itera en conjunto el algoritmo base, siendo el cambio igual tanto en la matriz M como en la imagen.

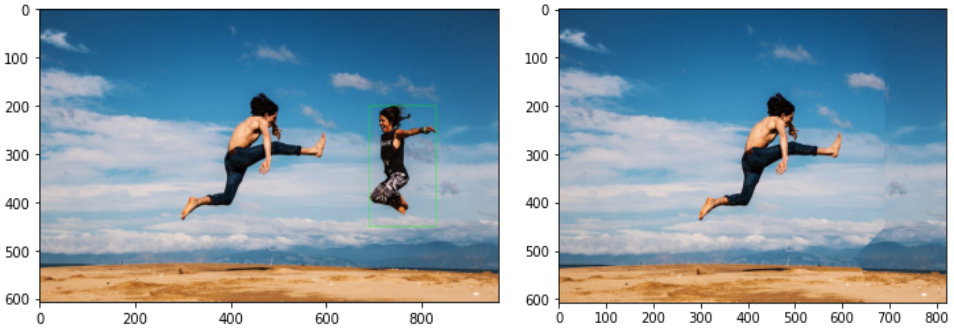
La aplicación viene recogida en la librería “objectRemoval.py”. A continuación se muestra ejemplos de su uso, junto con la matriz M modificada.

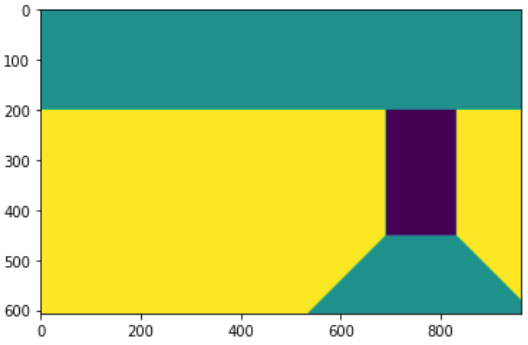





0
100
200
300
400
500
600
700
0
200
400
0
100
200
300
400
500
600
700
0
200
400


0
100
200
300
400
500
600
700
0
200
400


0
100
200
300
400
500
600
0
200
400
600
800
0
100
200
300
400
500
600
0
100
200
300
400
500
600
700
800


0
100
200
300
400
500
600
0
200
400
600
800

Referencia:

[1] - Shai Avidan, Mitsubishi Electric Research Labs, Ariel Shamir, The Interdisciplinary Center & MERL. *Seam Carving for Content-Aware Image Resizing*