INFORME TAREA 03 FUNDAMENTOS DE PROCESAMIENTO DE IMÁGENES

VICENTE MORENO

Introducción:

El programa fue desarrollado en Python 2.7 y se utilizaron las siguientes librerías: cv2, numpy, random y matplotlib. Se desarrollaron 2 algoritmos que son capaces de generar imágenes que parecen pinturas a partir de fotografías a color. Las imágenes del primer y segundo algoritmo se encuentran almacenadas en las carpetas FOTOS_ALGORITMO_1 y FOTOS_ALGORITMO_2 respectivamente.

Desarrollo:

Algoritmo 1:

*Todos los códigos insertados pertenecen a la clase Algoritmo_1, con excepción del código que define a la clase Color

Se parte leyendo la imagen por medio de la librería cv2 y se reordenan los valores de los pixeles de la imagen para evitar problemas al graficarla, puesto que una vez abierta una imagen con el método cv2.imread(), los valores de cada pixel están ordenados de la siguiente forma: azul, verde, rojo.

```
class Algoritmo_1:

    def __init__(self, filename):
        self.bgr_img = cv2.imread('./FOTOS_ALGORITMO_1/{}'.format(filename))
        self.b,self.g,self.r = cv2.split(self.bgr_img)# get b,g,r
        self.rgb_img = cv2.merge([self.r,self.g,self.b])# switch it to rgb
        self.lista_colores = []
```

Luego realizan las operaciones necesarias para obtener la imagen final (Los métodos utilizados aquí serán explicados posteriormente).

```
    def run(self):
        self.canny = self.auto_canny(self.rgb_img)
        self.rgb_img = cv2.fastNlMeansDenoisingColored(self.rgb_img,None,10,10,7,21)
        self.rgb_img = cv2.medianBlur(self.rgb_img,5)
        self.escoger_colores()
        self.cambiar_colores()
        plt.imshow(self.rgb_img)
        plt.show()
```

Para comprender lo que queda de código se debe saber que cuando se habla de distancia entre colores se refiere a lo siguiente:

```
    def distancia(self, color, r, g, b):
        return ((int(color.r) - int(r))**2 + (int(color.g) - int(g))**2 + (int(color.b) - int(b))**2)**0.5

class Color:

    def __init__(self, r, g, b):
        self.r = r
        self.g = g
        self.b = b
```

El parámetro color es una instancia de la clase Color y los parámetros r, g, b son los valores RGB del pixel con el cual se está calculando la distancia.

El método escoger_colores parte realizando un loop de 50 iteraciones con el objetivo de escoger los colores que conformaran la imagen final. Para esto en cada loop se escoge un pixel al azar, el que podría ser agregado eventualmente a la lista de colores. La condición necesaria para que este pixel escogido sea agregado a la lista es que tenga una distancia mayor a 30 con todos los colores que conforman la lista.

```
def escoger_colores(self):
    for number in range(50):
        x = randrange(self.rgb_img.shape[1])
        y = randrange(self.rgb_img.shape[0])
        r, g, b = self.rgb_img[y, x]
        if len(self.lista_colores) == 0:
            self.lista_colores.append(Color(r,g,b))
        else:
            bool = True
            for color in self.lista_colores:
                distancia = ((int(color.r) - int(r))**2 + (int(color.g) - int(g))**2 + (int(color.b) - int(b))**2)**0.5
                if distancia < 30:
                    bool = False
            if bool:
                self.lista_colores.append(Color(r,g,b))
```

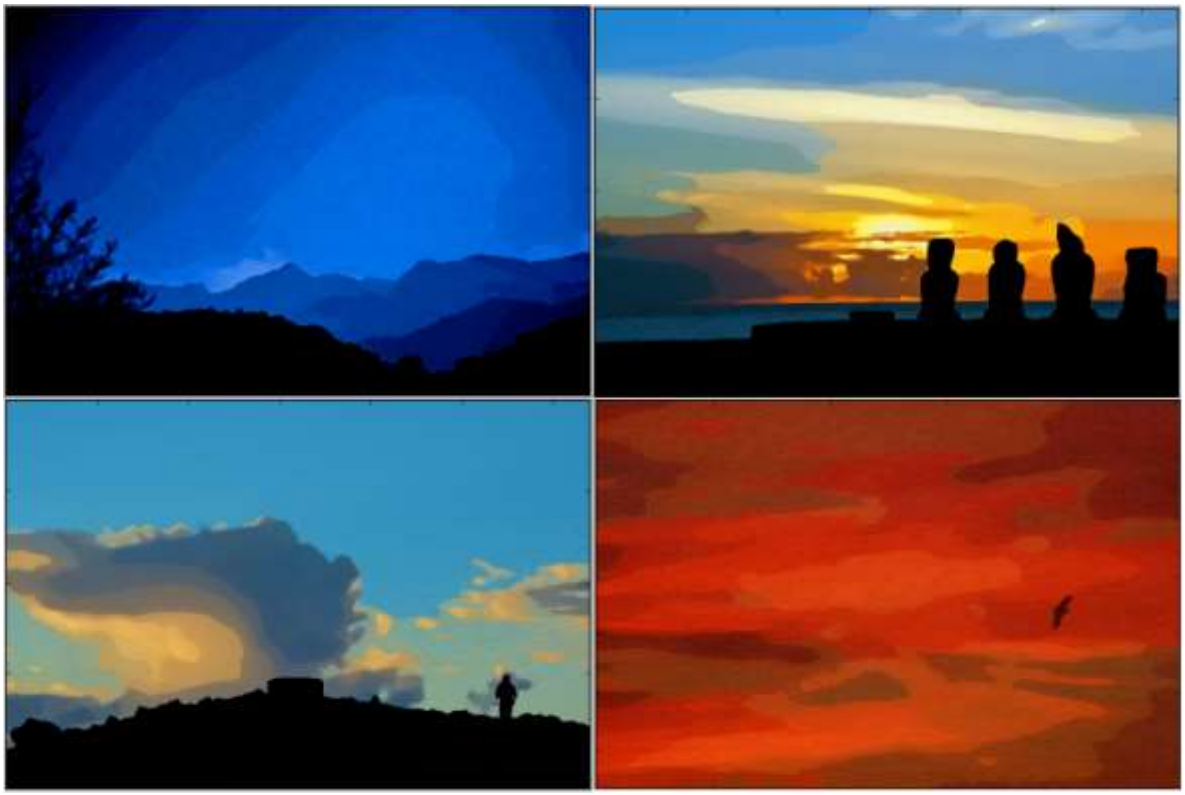
Se cambia el color de cada pixel por un promedio entre: los valores RGB del color de la lista_colores con el que tenga una menor distancia y sus propios valores.

```
def cambiar_colores(self):
    for x in range(self.rgb_img.shape[1]):
        for y in range(self.rgb_img.shape[0]):
            r, g, b = self.rgb_img[y, x]
            dict_distancia = {self.distancia(color, r, g, b): color for color in self.lista_colores}
            dist_min = min(dict_distancia.items(), key=lambda x: x[0])[0]
            color = dict_distancia[dist_min]
            if self.canny[y,x] == 255:
                self.rgb_img[y, x]= [max(int((int(color.r)+int(r))/2)-10, 0),max(int((int(color.g)+int(g)))/2-10, 0), max(int((int(color.b)+int(b))/2)-10, 0)]
            else:
                self.rgb_img[y, x]= [int((int(color.r)+int(r))/2),int((int(color.g)+int(g)))/2, int((int(color.b)+int(b))/2)]
```

Finalmente el método auto_canny entrega los bordes de la imagen. El código lo encontré AQUÍ y funciona bastante bien para encontrar los bordes principales de la imagen.

```
def auto_canny(self, image, sigma=0.33):
    v = np.median(image)
    lower = int(max(0, (1.0 - sigma) * v))
    upper = int(min(255, (1.0 + sigma) * v))
    edged = cv2.Canny(image, lower, upper)
    return edged
```

Los resultados obtenidos sobre las 4 imágenes adjuntadas son los siguientes:



URLs de imágenes:

- https://www.flickr.com/photos/dmery/838238121/in/album-72157629303295010/
- https://www.flickr.com/photos/dmery/9476679426/in/album-72157629303295010/
- https://www.flickr.com/photos/dmery/12825330394/in/album-72157629303295010/
- https://www.flickr.com/photos/dmery/13014286635/in/album-72157629303295010/

Algoritmo 2:

Se inicia leyendo la imagen de la misma forma que en el algoritmo 1.

```
class Algoritmo_2:

    def __init__(self, filename):
        self.bgr_img = cv2.imread('./FOTOS_ALGORITMO_2/{}'.format(filename))
        self.b,self.g,self.r = cv2.split(self.bgr_img)# get b,g,r
        self.rgb_img = cv2.merge([self.r,self.g,self.b])# switch it to rgb
        self.rgb_img = cv2.medianBlur(self.rgb_img,7)
```

Luego realizan las operaciones necesarias para obtener la imagen final en el método run.

```
    def run(self):
        self.canny = self.auto_canny(self.rgb_img)
        self.cambiar_colores()
        plt.imshow(self.rgb_img)
        plt.show()
```

El método auto_canny en este algoritmo tiene los thresholds inferior y superior contantes en los valores 0 y 10 respectivamente. Esto provoca que exista una gran cantidad de bordes y por lo tanto se cree una textura parecida a la de una pintura.

```
    def auto_canny(self, image):
        edged = cv2.Canny(image, 0, 10)
        return edged
```

El método cambiar_colores, cambia los colores de cada pixel restando una unidad de rojo, verde y azul a los pixeles que son blancos en la imagen canny obtenida. Este método además le “resta color” a los pixeles aledaños al que se está analizando.

Si el pixel al que se restará color es demasiado oscuro (tiene valores menores o iguales a 10 en alguno de sus colores) no se le restará color.

```
def cambiar_colores(self):
    for x in range(self.rgb_img.shape[1]):
        for y in range(self.rgb_img.shape[0]):
            r, g, b = self.rgb_img[y, x]
            if self.canny[y,x] == 255 and self.rgb_img[y,x][0]> 10 and self.rgb_img[y,x][1]> 10 and self.rgb_img[y,x][2]> 10:
                try:
                    self.rgb_img[y, x]= [max(r-1, 10),max(g-1, 10), max(b-1, 10)]
                    self.rgb_img[y, x+1]= [max(r-1, 10),max(g-1, 10), max(b-1, 10)]
                    self.rgb_img[y, x-1]= [max(r-1, 10),max(g-1, 10), max(b-1, 10)]
                    self.rgb_img[y+1, x]= [max(r-1, 10),max(g-1, 10), max(b-1, 10)]
                    self.rgb_img[y-1, x]= [max(r-1, 10),max(g-1, 10), max(b-1, 10)]
                    self.rgb_img[y+1, x-1]= [max(r-1, 10),max(g-1, 10), max(b-1, 10)]
                    self.rgb_img[y-1, x+1]= [max(r-1, 10),max(g-1, 10), max(b-1, 10)]
                    self.rgb_img[y-1, x-1]= [max(r-1, 10),max(g-1, 10), max(b-1, 10)]
                    self.rgb_img[y+1, x+1]= [max(r-1, 10),max(g-1, 10), max(b-1, 10)]
                except:
                    pass
```

Los resultados obtenidos sobre las 4 imágenes adjuntadas son los siguientes:

URLs de imágenes:

- https://www.flickr.com/photos/dmery/6888794062/in/album-72157629303295010/
- https://www.flickr.com/photos/dmery/8381766543/in/album-72157629303295010/
- https://www.flickr.com/photos/dmery/8796182308/in/album-72157629303295010/
- https://www.flickr.com/photos/dmery/12487273925/in/album-72157629303295010/