

IE-0217 Estructuras abstractas de datos y algoritmos para ingeniería.

Informe final proyecto 0: Mosaico de imágenes

Belinda Brown Ramírez - B61254

I-2019

Tabla de contenidos

1. Reseña del programa:	2
2. Funcionamiento del programa:	2
3. Experimentos realizados:	2
4. Resultados obtenidos:	6
5. Conclusiones:	8
Referencias	8
6. Anexos:	9
6.1. Código Original:	9
6.2. Código Alternativo	11
6.3. README:	15

1. Reseña del programa:

El objetivo de este proyecto es crear un programa que tome una serie de imágenes de un directorio y a partir de ellas recrear otras imágenes base, dando el efecto de una imagen mosaico.

El programa debe ser capaz de adaptar la serie de imágenes del directorio de manera óptima, para que la imagen mosaico final se parezca lo más posible a la imagen base original.

Además, se usa la biblioteca **Open Source Computer Vision Library**, mejor conocida como **OpenCV**, quien fue creada para proveer una base para las aplicaciones de visión por computador, mejorando la implementación de los productos relacionados a esta área y el manejo de imágenes en programación.

2. Funcionamiento del programa:

El proyecto fue desarrollado en **Python** con el uso de bibliotecas como **OpenCV**, **Numpy**, **Glob** y **Sys** y está compuesto por dos archivos, un main y un archivo que contiene el desarrollo de las funciones.

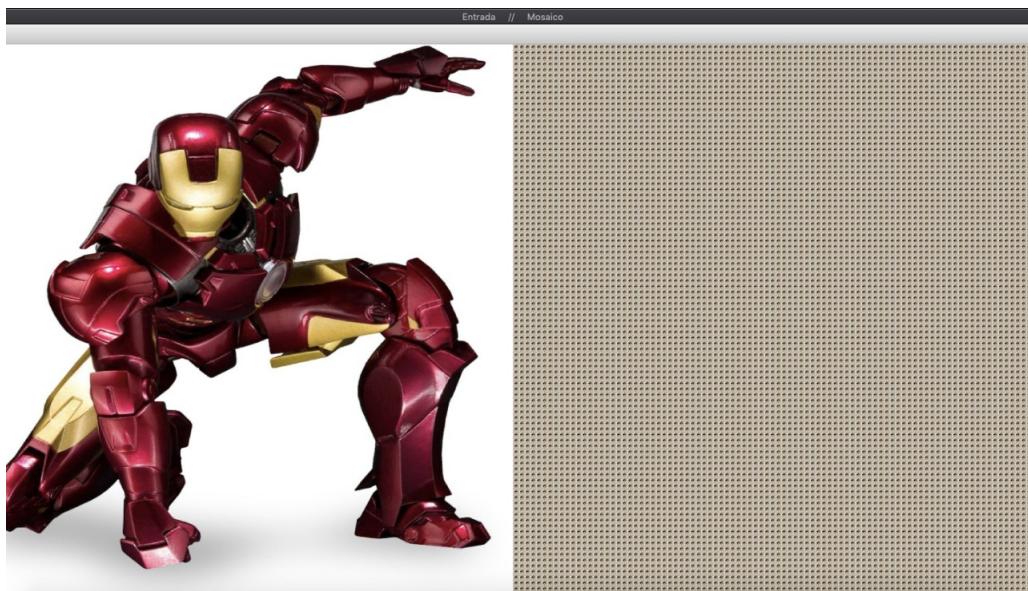
El algoritmo toma como fuente un directorio y un archivo, el directorio contiene la serie de imágenes que pueden ser utilizadas para crear el mosaico y el archivo será la imagen base por recrear.

Para recrear la imagen y crear el mosaico, se crea una función capaz de sacar el promedio de colores RGB de una imagen, promedio que después se utiliza para comparar cuál imagen del directorio es la mejor opción para recrear el pixel de la imagen base que se está analizando. Una vez realizado este proceso se utiliza un arreglo que contiene los valores promedios de las imágenes del directorio y la comparación. Con esto, se construye el mosaico por medio de las funciones np.vstack y np.hstack.

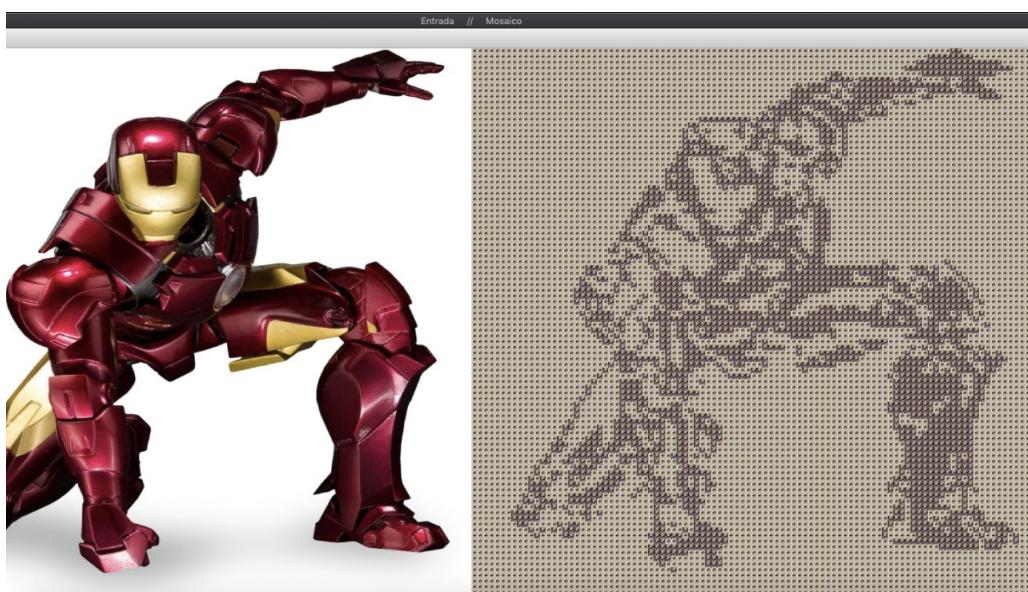
3. Experimentos realizados:

Se realizaron una serie de pruebas, donde se alteraba la cantidad de imágenes en el directorio para observar como cambiaba la definición de la imagen y la comparación por promedio.

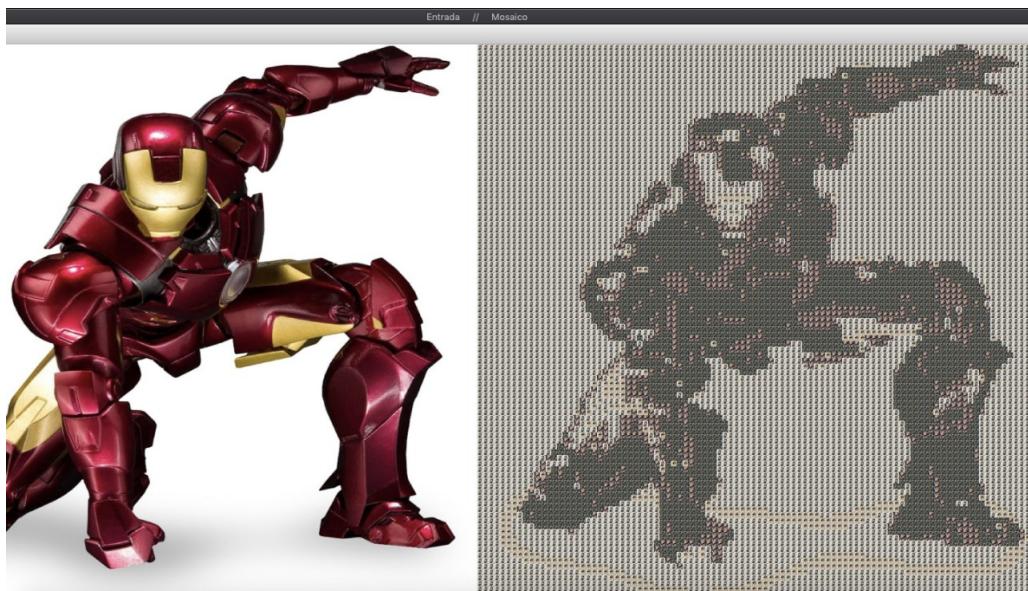
1. Prueba con solo una imagen dentro del directorio



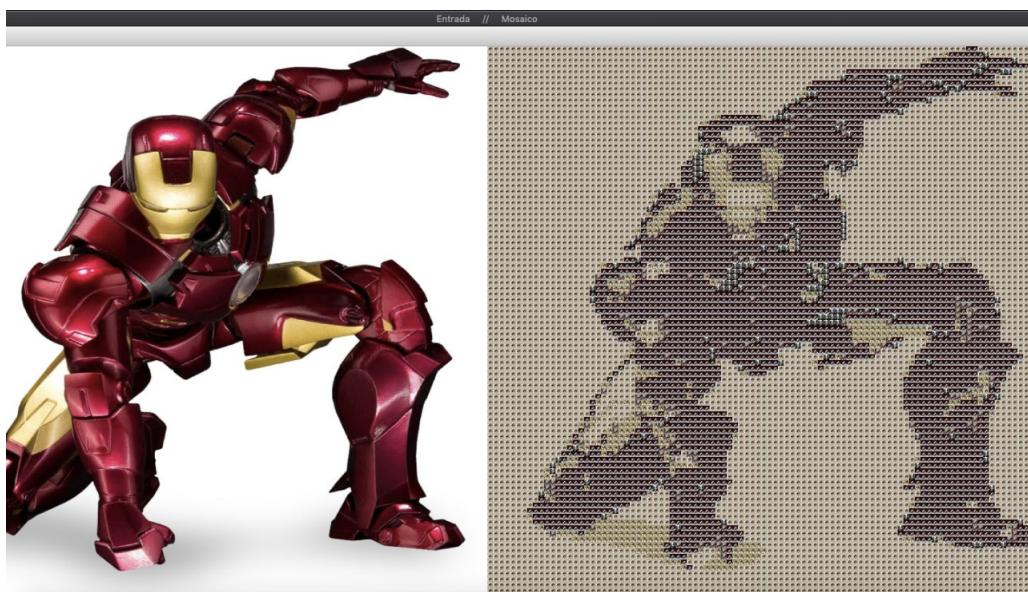
2. Dos imágenes dentro del directorio



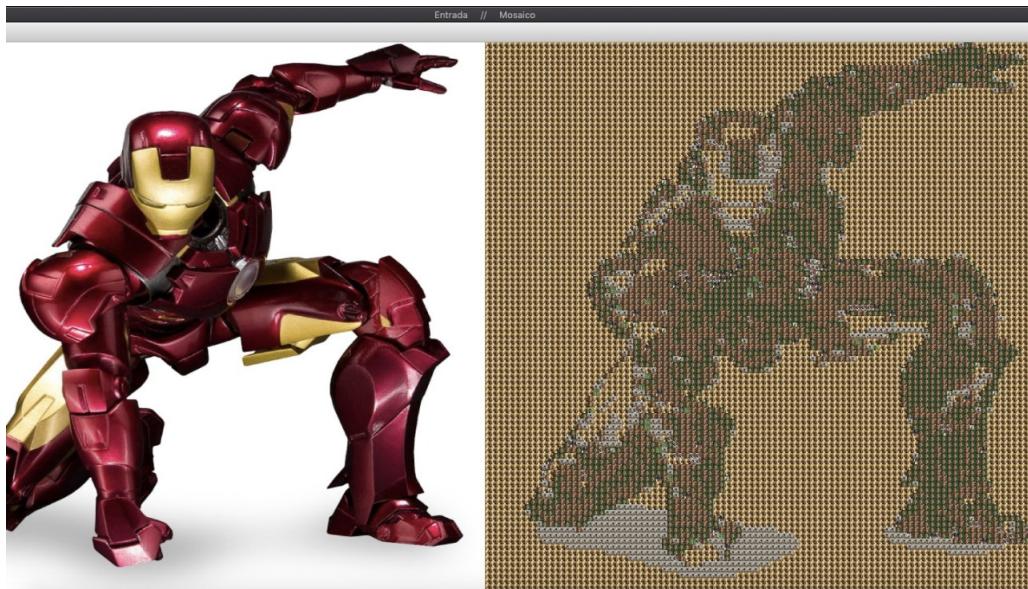
3. Cuatro imágenes dentro del directorio



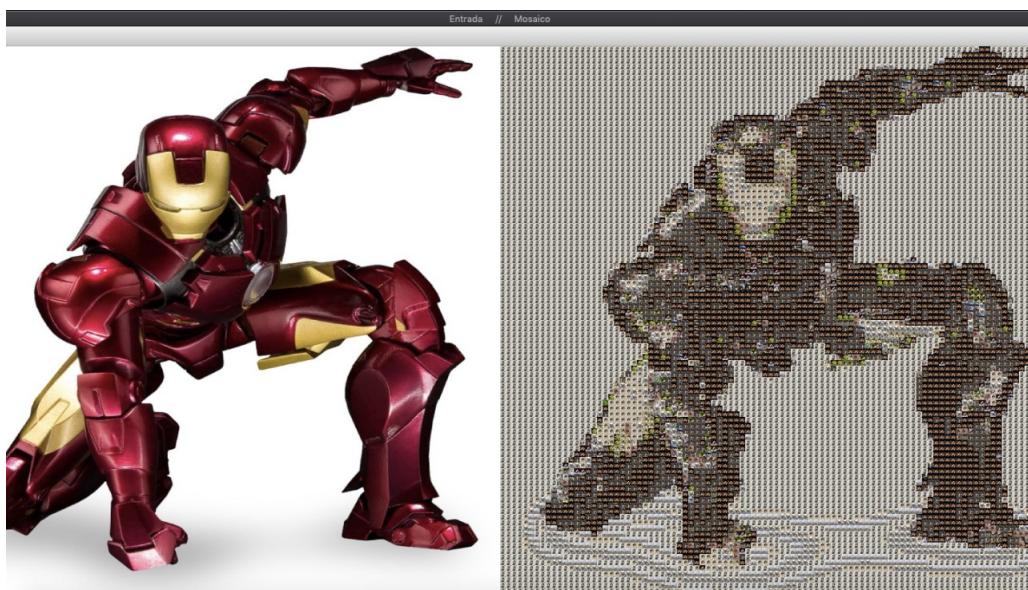
4. Cinco imágenes dentro del directorio



5. Diez imágenes dentro del directorio

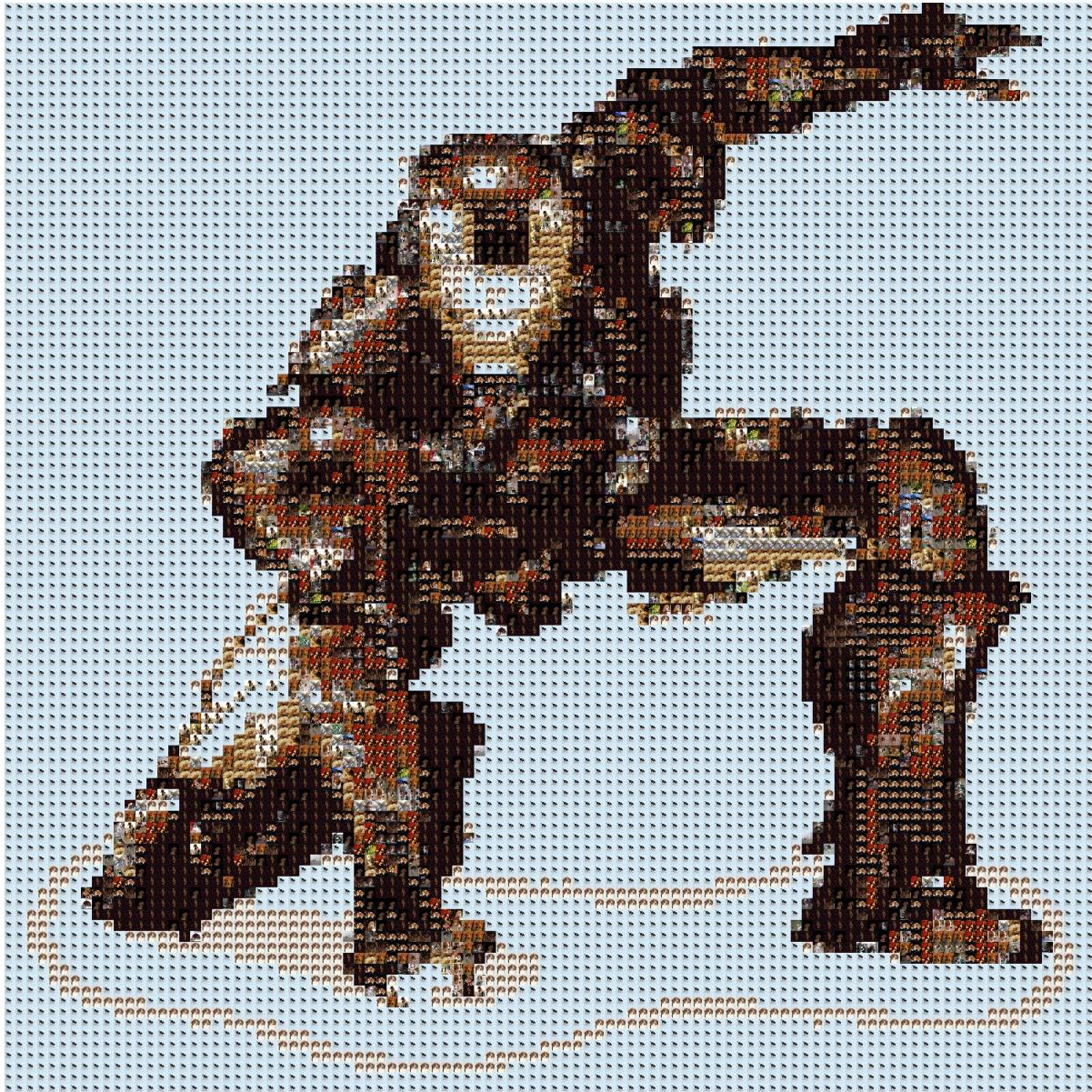


6. Cincuenta imágenes dentro del directorio

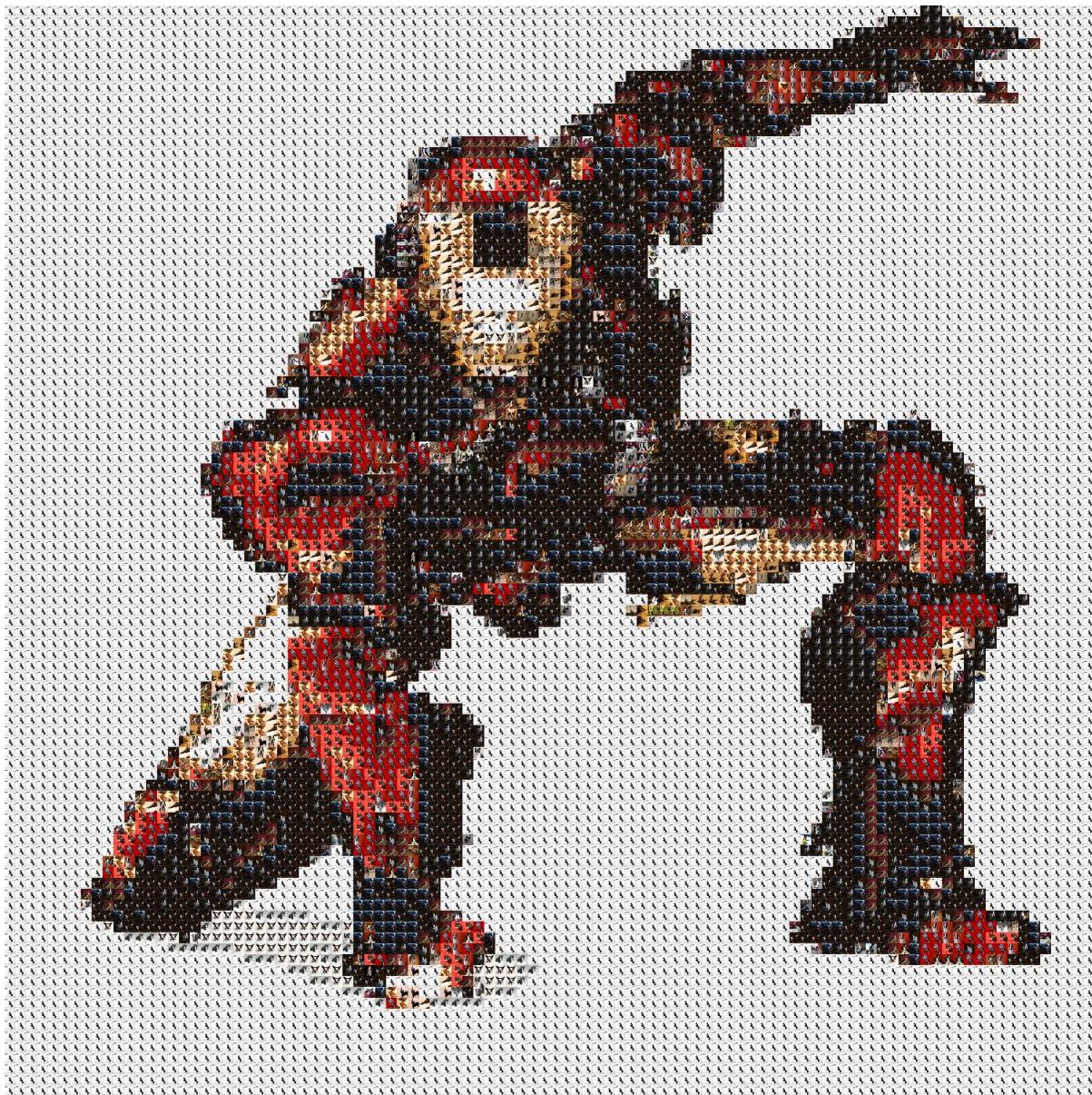


4. Resultados obtenidos:

Finalmente con el uso de 195 imágenes en el directorio y el código se obtiene una representación de la imagen base así:



Si se utilizan imágenes diferentes en el directorio, la imagen se ve de la siguiente manera:



5. Conclusiones:

El programa cumple los objetivos planteados, se practicó el uso del lenguaje Python, se aprendió sobre el manejo de imágenes con la biblioteca OpenCV, además, se puede decir que el resultado final del mosaico va a depender directamente de la cantidad de imágenes que pueden ser utilizadas y la gama de colores que las componen.

Referencias

- [1] OpenCV.(2019) *About OpenCV*
<https://opencv.org/about/>
- [2] Stack Overflow(2018) *Varios*
<https://es.stackoverflow.com/>
- [3] EcuRed(2018) *Lenguaje de programación Python*
<https://www.ecured.cu/>

6. Anexos:

6.1. Código Original:

Main.h:

```
1 import cv2
2 import numpy as np
3 import glob
4 import sys
5 from funciones import mosaico100x100
6
7
8 filenames = [img1 for img1 in glob.glob(str(sys.argv[2]) + "/*.jpg")] #Se accede a
9     todas los archivos del directorio con
10    filenames.sort()                                              #con
11        terminacion .jpg
12 img = []
13 for img1 in filenames: #Se guardan todas las imagenes del directorio en un array
14    n= cv2.imread(img1)
15    imagen = cv2.resize(n, (15,15)) #Se redimensionan las imagenes del directorio a
16        15x15 pixeles
17    img.append(imagen)
18
19
20 entrada = cv2.imread(str(sys.argv[1]))
21 entrada = cv2.resize(entrada, (800,800))
22 entrada_new = cv2.resize(entrada, (100,100))
23
24 mosaico = mosaico100x100(img, entrada_new)
25 cv2.imwrite("kokonete.jpg",mosaico)
26 mosaico = cv2.resize(mosaico, (800,800))
27
28 out = np.hstack([entrada, mosaico])
29
30 cv2.imshow("Entrada      //      Mosaico", out)
31 cv2.waitKey(0)
```

Funciones.h:

```
1 import numpy as np
2
3 def prom_rgb(image): #Esta funcion toma el promedio de rgb de una imagen (se define
4     par una de 15x15 pixeles)
5     r=0
6     g=0
7     b=0
8     for n in range (0,15):
9         for i in range (0, 15):
10             r=r+image[n,i,0]
11             g=g+image[n,i,1]
12             b=b+image[n,i,2]
13     r=r/225 #Cada imagen tiene una cantidad de 15x15=225 pixeles
14     g=g/225
15     b=b/225
16     rgb=[r,g,b]
17     return rgb
18
19 def comparacion (rgb, dir_rgb): #Esta funcion compara un valor de un pixel rgb con
20     el de todos los
```

```

19 #valores promedio de cada una de las imagenes del
20 directorio
21 imagen_num=0
22 cond0=False
23 cond1=False
24 cond2=False
25 cond3=False
26 cond4=False
27 cond5=False
28 cond6=False
29
30 for n in range(0, len(dir_rgb)):
31     if (cond0==False):
32         if (abs(dir_rgb[n][0]-rgb[0])<=200 and abs(dir_rgb[n][1]-rgb[1])<=200 and
33             abs(dir_rgb[n][2]-rgb[2])<=200 ):
34             imagen_num = n      #Si encuentra un valor con una diferencia de los
35             valores rgb menor a 200
36             cond0=True          #guarda la posicion de esa imagen en el
37             directorio , lo mismo para los siguientes casos ,
38             if (cond1==False):      #pero con valores de diferencia mas peque os
39                 if (abs(dir_rgb[n][0]-rgb[0])<=100 and abs(dir_rgb[n][1]-rgb[1])<=100 and
40                     abs(dir_rgb[n][2]-rgb[2])<=100 ):
41                     imagen_num = n
42                     cond1=True
43
44             elif (cond2==False):
45                 if (abs(dir_rgb[n][0]-rgb[0])<=80 and abs(dir_rgb[n][1]-rgb[1])<=80 and
46                     abs(dir_rgb[n][2]-rgb[2])<=80 ):
47                     imagen_num = n
48                     cond2=True
49
50             elif (cond3==False):
51                 if (abs(dir_rgb[n][0]-rgb[0])<=50 and abs(dir_rgb[n][1]-rgb[1])<=50 and
52                     abs(dir_rgb[n][2]-rgb[2])<=50 ):
53                     imagen_num = n
54                     cond3=True
55
56             elif (cond4==False):
57                 if (abs(dir_rgb[n][0]-rgb[0])<=30 and abs(dir_rgb[n][1]-rgb[1])<=30 and
58                     abs(dir_rgb[n][2]-rgb[2])<=30 ):
59                     imagen_num = n
60                     cond4=True
61
62             elif (cond5==False):
63                 if (abs(dir_rgb[n][0]-rgb[0])<=15 and abs(dir_rgb[n][1]-rgb[1])<=15 and
64                     abs(dir_rgb[n][2]-rgb[2])<=15 ):
65                     imagen_num = n
66                     cond5=True
67
68             elif (cond6==False):
69                 if (abs(dir_rgb[n][0]-rgb[0])<=5 and abs(dir_rgb[n][1]-rgb[1])<=5 and abs
70                     (dir_rgb[n][2]-rgb[2])<=5 ):
71                     imagen_num = n
72                     cond6=True
73
74 return imagen_num #retorna la posicion de la imagen en el directorio que tenga
75 los valores mas parecidos al pixel
76           #que se esta analizando de la imagen de entrada
77
78
79 def mosaico100x100(imagenes, entrada ): #Crea un mosaico de una imagen con 100x100
80     imagenes de un directorio
81     dir_rgb = []
82     for n in range(0, len(imagenes)):#Se crea un array que contiene los valores

```

```

66 promedios_rgb de las imagenes en el directorio
67     dir_rgb.append(prom_rgb(imagenes[n]))
68
69 colum = []
70 o = []
71 for n in range(0,100): #Se compara el valor rgb de cada pixel de la imagen con
72     #el de todos los valores rgb del directorio
73     for k in range (0,100):
74         imagen_num = comparacion (entrada[k,n], dir_rgb)
75         colum.append(imagen_num)
76     o.append(colum)
77     colum = []
78
79 vs1=[]
80 vs2=[]
81 vs3=[]
82 vs4=[]
83 hs1=[]
84 hs2=[]
85 hs3=[]
86 for n in range(0,10): #Se crea el mosaico con las funciones np.vstack y np.
87     hstack, que combinan los ndarray de
88     c1=10*n           #cada imagen del directorio en forma vertical y horizontal
89     respectivamente
90     for m in range(0,10):
91         for i in range(0,10):
92             c2=10*i
93             for j in range(0,10):
94                 vs1.append(imagenes[o[(c1+m)][(c2+j)]])
95                 vs2=np.vstack([vs1[0], vs1[1], vs1[2], vs1[3], vs1[4], vs1[5], vs1
96 [6], vs1[7], vs1[8], vs1[9]])
97                 vs3.append(vs2)
98                 vs1=[]
99                 vs2=[]
100                vs4=np.vstack([vs3[0], vs3[1], vs3[2], vs3[3], vs3[4], vs3[5], vs3[6],
101 vs3[7], vs3[8], vs3[9]])
102                hs1.append(vs4)
103                vs3=[]
104                vs4=[]
105                hs2=np.hstack([hs1[0], hs1[1], hs1[2], hs1[3], hs1[4], hs1[5], hs1[6], hs1
106 [7], hs1[8], hs1[9]])
107                hs3.append(hs2)
108                hs1=[]
109                hs2=[]
110 mosaico=np.hstack([hs3[0], hs3[1], hs3[2], hs3[3], hs3[4], hs3[5], hs3[6], hs3
111 [7], hs3[8], hs3[9]])
112 return mosaico

```

6.2. Código Alternativo

Main.h:

```

1 from funciones2 import *
2
3
4 if __name__ == '__main__':
5     mosaico_creado(sys.argv[1], sys.argv[2])

```

Funciones.h:

```
1 import sys
2 import os
3 from PIL import Image
4 from multiprocessing import Process, Queue, cpu_count
5 #se pueden modificar
6 TILE_SIZE = 50 # alto y ancho
7 TILE_MATCH_RES = 5 # encaja con la resolucion original
8 ENLARGEMENT = 8
9
10 TILE_BLOCK_SIZE = TILE_SIZE / max(min(TILE_MATCH_RES, TILE_SIZE), 1)
11 WORKER_COUNT = max(cpu_count() - 1, 1)
12 OUT_FILE = 'mosaico_creado.jpg'
13 EOQ_VALUE = None
14
15
16
17 class Procesador_img:
18     def __init__(self, directorio_img):
19         self.directorio_img = directorio_img
20
21     def __img_proceso(self, ruta_img):
22         try:
23             img = Image.open(ruta_img)
24             w = img.size[0]
25             h = img.size[1]
26             dimension_minima = min(w, h)
27             ancho_recortado = (w - dimension_minima) / 2
28             alto_recortado = (h - dimension_minima) / 2
29             img = img.crop((ancho_recortado, alto_recortado, w - ancho_recortado, h - alto_recortado))
30
31             img_larga = img.resize((TILE_SIZE, TILE_SIZE), Image.ANTIALIAS)
32             img_corta = img.resize((TILE_SIZE/TILE_BLOCK_SIZE, TILE_SIZE/TILE_BLOCK_SIZE), Image.ANTIALIAS)
33
34             return (img_larga.convert('RGB'), img_corta.convert('RGB'))
35         except:
36             return (None, None)
37
38     def obt_imgnes(self):
39         imagenes_largo_array = []
40         imagenes_corto_array = []
41
42
43     # buscando en el directorio
44     for root, subFolders, archivos in os.walk(self.directorio_img):
45         for img_nombre in archivos:
46             ruta_img = os.path.join(root, img_nombre)
47             large_tile, small_tile = self.__img_proceso(ruta_img)
48             if large_tile:
49                 imagenes_largo_array.append(large_tile)
50                 imagenes_corto_array.append(small_tile)
51
52
53     return (imagenes_largo_array, imagenes_corto_array)
54
55 class Objeto_Image:
```

```

56     def __init__(self, ruta_imagen):
57         self.ruta_imagen = ruta_imagen
58
59     def get_data(self):
60         img = Image.open(self.ruta_imagen)
61         w = img.size[0] * ENLARGEMENT
62         h = img.size[1] * ENLARGEMENT
63         imagen_largo_objeto = img.resize((w, h), Image.ANTIALIAS)
64         diferencia_ancho_objeto = (w % TILE_SIZE)/2
65         diferencia_alto_objeto = (h % TILE_SIZE)/2
66
67         if diferencia_ancho_objeto or diferencia_alto_objeto:
68             imagen_largo_objeto = imagen_largo_objeto.crop((diferencia_ancho_objeto,
69                     diferencia_alto_objeto, w - diferencia_ancho_objeto, h - diferencia_alto_objeto))
70
71         img_peque = imagen_largo_objeto.resize((w/TILE_BLOCK_SIZE, h/TILE_BLOCK_SIZE),
72             Image.ANTIALIAS)
73
74
75     return img_datos_obj
76
77 class Encaje_img:
78     def __init__(self, pic_datos):
79         self.pic_datos = pic_datos
80
81     def __diferencia_obteniendo_imagen(self, tras1, tras2, se_sale_del_valor):
82         diferencia = 0
83         for i in range(len(tras1)):
84             diferencia += ((tras1[i][0] - tras2[i][0])**2 + (tras1[i][1] - tras2[i][1])**2
85             + (tras1[i][2] - tras2[i][2])**2)
86             if diferencia > se_sale_del_valor:
87                 return diferencia
88         return diferencia
89
90     def obte_mejor_encaje_img(self, img_datos):
91         obte_mejor_encaje_img_indexadaando = None
92         diferencia_minima = sys.maxint
93         img_indexada = 0
94
95     #Buscando el mejor encaje
96         for dato_imagen_encaje in self.pic_datos:
97             diferencia = self.__diferencia_obteniendo_imagen(img_datos, dato_imagen_encaje
98             , diferencia_minima)
99             if diferencia < diferencia_minima:
100                 diferencia_minima = diferencia
101                 obte_mejor_encaje_img_indexadaando = img_indexada
102                 img_indexada += 1
103
104     return obte_mejor_encaje_img_indexadaando
105
106     def img_encaja(trabajo_img, resultado_trazo, pic_datos):
107         tile_fitter = Encaje_img(pic_datos)
108
109         while True:
110             try:
111                 img_datos, direcciones_img = trabajo_img.get(True)
112                 if img_datos == EOQ_VALUE:

```

```

111     break
112     img_indexada = tile_fitter.obte_mejor_encaje_img(img_datos)
113     resultado_trazo.put((direcciones_img, img_indexada))
114 except KeyboardInterrupt:
115     pass
116
117 resultado_trazo.put((EOQ_VALUE, EOQ_VALUE))
118
119 class Proceso_trazo:
120     def __init__(self, total):
121         self.total = total
122         self.counter = 0
123
124     def update(self):
125         self.counter += 1
126         sys.stdout.flush()
127
128 class mosaico_creadoImage:
129     def __init__(self, original_img):
130         self.image = Image.new(original_img.mode, original_img.size)
131         self.x_tile_count = original_img.size[0] / TILE_SIZE
132         self.y_tile_count = original_img.size[1] / TILE_SIZE
133         self.total_tiles = self.x_tile_count * self.y_tile_count
134
135     def img_agreg(self, dato_imagen_encaje, direcciones):
136         img = Image.new('RGB', (TILE_SIZE, TILE_SIZE))
137         img.putdata(dato_imagen_encaje)
138         self.image.paste(img, direcciones)
139
140     def save(self, path):
141         self.image.save(path)
142
143     def build_mosaico_creado(resultado_trazo, todos_img_dat, original_img_large):
144         mosaico_creado = mosaico_creadoImage(original_img_large)
145
146         active_workers = WORKER_COUNT
147         while True:
148             try:
149                 direcciones_img, obte_mejor_encaje_img_indexadaando = resultado_trazo.get()
150
151                 if direcciones_img == EOQ_VALUE:
152                     active_workers -= 1
153                     if not active_workers:
154                         break
155                 else:
156                     dato_imagen_encaje = todos_img_dat[obte_mejor_encaje_img_indexadaando]
157                     mosaico_creado.img_agreg(dato_imagen_encaje, direcciones_img)
158
159             except KeyboardInterrupt:
160                 pass
161
162         mosaico_creado.save(OUT_FILE)
163
164     def creando(original_img, tiles):
165         original_img_large, original_img_small = original_img
166         tiles_large, tiles_small = tiles
167
168         mosaico_creado = mosaico_creadoImage(original_img_large)
169

```

```

170 todos_img_dat = map(lambda tile : list(tile.getdata()), tiles_large)
171 all_tile_data_small = map(lambda tile : list(tile.getdata()), tiles_small)
172
173 trabajo_img = Queue(WORKER_COUNT)
174 resultado_trazo = Queue()
175
176 try:
177     Process(target=build_mosaico_creado, args=(resultado_trazo, todos_img_dat,
178         original_img_large)).start()
179
180     for n in range(WORKER_COUNT):
181         Process(target=img_encaja, args=(trabajo_img, resultado_trazo,
182             all_tile_data_small)).start()
183
184     progress = Proceso_trazo(mosaico_creado.x_tile_count * mosaico_creado.
185         y_tile_count)
186     for x in range(mosaico_creado.x_tile_count):
187         for y in range(mosaico_creado.y_tile_count):
188             large_box = (x * TILE_SIZE, y * TILE_SIZE, (x + 1) * TILE_SIZE, (y + 1) *
189             TILE_SIZE)
190             small_box = (x * TILE_SIZE/TILE_BLOCK_SIZE, y * TILE_SIZE/TILE_BLOCK_SIZE, (x +
191             1) * TILE_SIZE/TILE_BLOCK_SIZE, (y + 1) * TILE_SIZE/TILE_BLOCK_SIZE)
192             trabajo_img.put((list(original_img_small.crop(small_box).getdata()), large_box))
193             progress.update()
194
195     finally:
196         for n in range(WORKER_COUNT):
197             trabajo_img.put((EOQ_VALUE, EOQ_VALUE))
198
199 def mosaico_creado(img_path, tiles_path):
200     pic_datos = Procesador_img(tiles_path).obt_imgeanes()
201     img_datos_obj = Objeto_Image(img_path).get_data()
202     creando(img_datos_obj, pic_datos)

```

6.3. README:

README.txt

Mosaico de imágenes en Python
Junio 2019

Belinda Brown Ramírez - timna.brown@ucr.ac.cr
Carolina Urrutia Núñez - ana.urrutia@ucr.ac.cr
Rubén Venegas Zúñiga - ruben.venegaszuniga@ucr.ac.cr
License WTFPL4.

Deberá usar Python 3, por lo que se recomienda actualizarlo o ya sea descargarlo.

Este programa crea un mosaico de una imagen de entrada con imágenes de un directorio, si desea descargar imágenes diríjase al siguiente link:
<https://www.kaggle.com/jessicali9530/stanford-dogs-dataset>

Para ejecutar el programa deberá tener las siguientes librerías de Python
-opencv

```
-matplotlib  
-numpy
```

Se realiza de la siguiente manera:

-Si su computadora trabaja con sistema operativo de Windows,
ejecute lo siguiente en la línea de comandos o consola:

```
python get-pip.py to install pip  
pip3 install opencv-python  
pip3 install matplotlib  
pip3 install numpy-python
```

-Si su computadora trabaja con sistema operativo de Mac o Linux,
ejecute lo siguiente en la línea de comandos o consola:

```
curl -O https://bootstrap.pypa.io/get-pip.py  
sudo python get-pip.py  
sudo pip3 install opencv-python  
sudo pip3 install matplotlib  
sudo pip3 install numpy
```

Para correr el programa deberá ejecutar el siguiente comando:

```
python3 main.py imagen.jpg directorio
```

Donde imagen.jpg es la imagen a convertir si la imagen tiene otra extensión, pónerselo, por ejemplo si es png poner

Nota: Se adjunta un segundo código, donde la calidad de imagen es mejor, sin embargo, el primer código es el que se utilizó para el desarrollo de la presentación e informe del proyecto.