КУРЕЦ Любовь ИСУ

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

Бинаризация полутоновых изображений

Преобразование цветного изображения в полутоновое происходит с помощью использования *python* модуля **PIL.** Процесс бинаризации это перевод цветного изображения в двухцветное черно-белое. Главным параметром такого преобразования является порог t – значение, с которым сравнивается яркость каждого пикселя. По результатам сравнения, пикселю присваивается значение 0 или 1.

Получим бинарное изображение для данного изображения:

РЕЗУЛЬТАТ:

Листинг программы

```
import argparse
from PIL import Image, ImageDraw
def binarization(image_height, image_width, image_pixels, threshold,
image, draw, file_name):
  for i in range(image_height):
    for j in range(image_width):
      if image_pixels[i, j][0] <= threshold:
        draw.point((i, j), (0, 0, 0))
      else:
        draw.point((i, j), (255, 255, 255))
  image.save(file_name, "JPEG")
def find_threshold(image_height, image_width, image_pixels):
  size = 256
  intensity_histogram = [0] * size  # Image intensity histogram
  for i in range(image_height):
    for j in range(image_width):
      intensity_histogram[image_pixels[i, j][0]] += 1
  pixel_count = image_width * image_height  # The number of pixels in
the image
  intensity_sum = sum(index * value for index, value in
enumerate(intensity_histogram))  # Image intensity
  best_threshold = 0  # Best threshold
  max_sigma = 0.0  # Max interclass variance
  first_class_pixel_count = 0  # The number of pixels in the first group
  first_class_intensity_sum = 0  # First group intensity
  for threshold in range(size - 1):
    first_class_pixel_count += intensity_histogram[threshold]
    first_class_intensity_sum += threshold *
intensity_histogram[threshold]
    if pixel_count - first_class_pixel_count == 0 or first_class_pixel_count
== 0:
      continue
    first_class_prob = first_class_pixel_count / pixel_count
    second_class_prob = 1.0 - first_class_prob
```

```
        first_class_mean = first_class_intensity_sum /
first_class_pixel_count
        second_class_mean = (intensity_sum - first_class_intensity_sum) /
(pixel_count - first_class_pixel_count)
        mean_delta = first_class_mean - second_class_mean
        sigma = first_class_prob * second_class_prob * mean_delta *
mean_delta
        if sigma > max_sigma:
            max_sigma = sigma
            best_threshold = threshold
    return best_threshold
def start_processing(file_name):
    image = Image.open(file_name)
    image_height, image_width, = image.size
    image_pixels = image.load()
    draw = ImageDraw.Draw(image)
    return image, image_height, image_width, image_pixels, draw
def end_processing(draw):
    del draw
def parse():
    parser = argparse.ArgumentParser()
    parser.add_argument('-name')
    parser.add_argument('-path')
    return parser.parse_args()
def main():
    args = parse()
    if args.name and args.path:
        image, image_height, image_width, image_pixels, draw =
start_processing(file_name=args.name)
        threshold = find_threshold(image_height=image_height,
image_width=image_width, image_pixels=image_pixels)
        binarization(image_height=image_height, image_width=image_width,
image_pixels=image_pixels,
                     threshold=threshold, image=image, draw=draw,
file_name=args.path + "binary.jpg")
        end_processing(draw=draw)
```

```
    else:
        raise AttributeError("Incorrect number of argument")
if __name__ == '__main__':
    main()
```