ImageContrastEnhancement

ImageContrastEnhancement is a demonstration of the capabilities of convolutional genetic algorithms in image processing, specifically in the area of contrast enhancement. This README provides an overview of the project and explains how the contrast of black and white images can be improved using the algorithm.

Intro

In this project, each pixel of a black and white graphic is represented as a value between 0 and 255, with 0 representing black and 255 representing white. Our focus is on images with a small dynamic range, where the difference between the brightest and darkest pixels is relatively small, and images with limited shade diversity, which are best suited for our purposes.

Contrast Enhancement

To enhance the contrast of an image, we use a mapping technique where existing shades are transformed into new ones. This mapping should produce a better contrast. We map each pixel with a specific shade value to exactly one new value (surjection). This process is performed for each level of shade. We also ensure that any two pixels with different shades remain different shades to achieve injection, and the transformation is a bijection. We ensure that if a pixel at a particular position before the transformation was brighter than another, then after the transformation, this relationship is preserved.

Chromosome

In this project, we utilize chromosomes to store new shade values for the image. However, we always map the lightest and darkest shades to 255 and 0, respectively, to achieve the largest possible dynamic range, which enables greater contrast.

To generate a chromosome-based graphic, we map the second-smallest shade value from the image to the smallest value from the chromosome, the third shade from the image to the second value from the chromosome, and so on. For ease of use, we sort the chromosome to make mapping more straightforward.

```
@cache
def _image_generated_by_chromosome(self, chromosome: tuple):
    mapper = {key: value for key, value in zip(self.levels[1:-1], chromosome)}

    mapper[self.levels[0]] = 0
    mapper[self.levels[-1]] = 255
```

```
    result_image = np.ndarray(self.image.shape)
    for k in mapper:
        result_image[self.image == k] = mapper[k]
    return result_image.astype(np.uint8)
```

Graphics evaluation

In image processing, high contrast is necessary to differentiate between objects in the image and distinguish them from the background. To achieve this, we need visible edges between objects. Therefore, we will want to determine how many edges an image has after transformation, and prefer those that have as many as possible - that is, by definition, those with the best contrast.

Convolution

We utilize convolution to compare the number and intensity of edges. We use the Prewitt Operator or Sobel Operator to generate a 'sketch' of the edges, and then sum the value of all the pixels. The higher the sum, the more white in the image, indicating more edges. In this way, we will be able to compare images after different transformations in terms of contrast (which is related to edge observability).

Prewitt operator implementation

```
vertical_mask = np.array([[1, 0, -1],
                          [1, 0, -1],
                          [1, 0, -1]])
horizontal_mask = np.array([[1, 1, 1],
                            [0, 0, 0],
                            [-1, -1, -1]])

def apply_vertical_mask(image):
    return cv2.filter2D(image, -1, vertical_mask).astype(np.uint8)

def apply_horizontal_mask(image):
    return cv2.filter2D(image, -1, horizontal_mask).astype(np.uint8)

def apply_both_masks(image):
    return apply_vertical_mask(image) + apply_horizontal_mask(image)

def calculate_sum(image):
    return np.sum(apply_both_masks(image))
```

Sobel operator implementation

It differs only in kernels from Prewitt operator

```
vertical_mask = np.array([[1, 0, -1],
                          [2, 0, -2],
                          [1, 0, -1]])
horizontal_mask = np.array([[1, 2, 1],
                            [0, 0, 0],
                            [-1, -2, -1]])
```

Fitness functions

```
def prewitt_fitness_function(self, ga_instance,
                             solution, solution_idx):
    solution.sort()
    chromosome = tuple(solution)
    image_generated_by_chromosome = self._image_generated_by_chromosome(chromosome)
    return PrewittOperator.calculate_sum(image_generated_by_chromosome)
```

`sobel_fitness_function()` differs only in last line, instead of `PrewittOperator` uses `SobelOperator`

Sources

- *An image contrast enhancement method based on genetic algorithm. (2009, December 11). An Image Contrast Enhancement Method Based on Genetic Algorithm - ScienceDirect.* *https://doi.org/10.1016/j.patrec.2009.12.006*

License