Image Processing Using Artificial Intelligence Algorithms in Python

Shivam Sahu¹, Mohit Mittal²
¹,²Department of Mechanical Engineering, College of Engineering Roorkee, Roorkee.
E-mail addresses: 1. shivamrenu@gmail.com , 2. Mohit.engg.iitr@gmail.com

Abstract-- This work consists of study and implementation of “Digital Image Processing” using “Python” language and its numerical packages. Python is a “very high level”, object oriented, modern and highly efficient language. It is suitable for rapid application development (RAD). Many Image processing activities can be done by python, such as: Converting image from RGB to Gray scale, image blurring, image filtration using Gaussian filter, median filter, Bilateral filter and many others, drawing a histogram of an image by using multidimensional mathematical processing (such as calculating cumulative distribution function), image normalization, image equalization, image segmentation, edge detection etc. Digital image processing is done by using Algorithms of Artificial Intelligence.

Keywords: Digital Image Processing, Python, Gaussian filter, Median filter, Bilateral filter, Equalization, Normalization, Artificial Intelligence, Edge Detection.

I. INTRODUCTION
Artificial Intelligence is playing an important role in automation. This work is focused on one such field of Artificial Intelligence, which is “Digital Image Processing”. Digital Image Processing is the analysis and manipulation of a digitized image, especially in order to improve its quality. Digital image processing is a part of imaging science. It processes the data using mathematical operations by using signal processing on digital images. These images are processed in either two dimensional or three dimensional signals. It avoids problems such as the build-up of noise and signal distortion during processing. To be more precise this paper is more focused on normalizing, equalizing and filtering of an image. The normalization is done by plotting a histogram and cumulative distribution function (CDF) curve of an image and the equalizing the Image’s contrast to obtain a better histogram and CDF curve. Filtration process includes converting image into b&w image and then processing it with image averaging, image blurring, image filtration using Gaussian filter, median filter, and bilateral filter. Edge detection is done by “Harris Corner Detection” algorithm.

II. LOADING AN IMAGE
Before we can explore how to process and analyze images we need to understand how a computer sees an image. As a digital image this is actually represented as an array of numbers that indicate pixel intensities between 0 and 255. A colored image is represented as a three – dimensional array with each dimension corresponding to red, green and blue hues in an image. The specifics depends on the format of image and how you want to work on them. Here we have converted this image to Grayscale and now it can be represented as a two – dimensional array. Working with python we import several libraries to do so, such as importing “image” from PIL (pillow). PIL is a “Python Imaging Library” used for image manipulation.

PIL/Pillow offers several standard procedures for image manipulation. These include:
- Per-pixel manipulations,
- Masking and transparency handling,
- Image filtering, such as blurring, contouring, smoothing, or edge finding,
- Image enhancing, such as sharpening, adjusting brightness, contrast or color,
- Adding text to images and much more.

A. Image is loaded by using:

```
  Image.open() method
```

B. Initialize the method into a variable where we put the name of the image with extension in the parenthesis. The image gets loaded like:

![Image 1: Loaded Image](http://www.ij-tejas.com)

To perform further tasks we convert the image into b&w image i.e. a two-dimensional signals. To convert the image into b&w we import “cv2 library”.
In cv2 all OpenCV data types are preserved as such. For example, when loaded, images are of format cvMat. In cv2 everything is returned as NumPy objects like ndarray and native Python objects like lists, tuples, dictionary, etc. So due to this NumPy support, you can do any numpay operation here. NumPy is a highly stable and fast array processing library, and so we also have to import “numpay library as np”.

- Plotting the histogram of a gray scale Image

For normalizing the image we need to first plot the histogram graph. We import a library “pyplot” from “matplotlib” as “plt”.

A. The graph is plotted by using:

```python
cv2.calcHist()
```

In parenthesis “images, channels, mask, histSize, ranges[0 , hist[accumulate]]”

1. images: it is the source image of type uint8 or float32. It should be given in square brackets, i.e., “[im]”.
2. channels: It is also given in square brackets. It is the index of channel for which we calculate histogram. For example, if input is grayscale image, its value is [0]. For color image, you can pass [0], [1] or [2] to calculate histogram of blue, green or red channel respectively.
3. mask: mask image. To find histogram of full image, it is given as "None". But if you want to find histogram of particular region of image, you have to create a mask image for that and give it as mask. (I will show an example later.)
4. histSize: this represents our BIN count. Need to be given in square brackets. For full scale, we pass [256].
5. ranges: this is our “Range”. Normally, it is [0,256].

B. To plot the histogram we use:

```python
plt.hist()
```

In parenthesis “im.ravel(), 256, range”

- ravel(): It is used to flatten out the image array before computing the histogram. If you pass a 2D array to plt.hist(), it will treat each row as a separate data series, which is not what we want here.
- range is from [0,256] same as above.

After the image gets converted into b&w it looks like:

![B&W image of loaded image](image1.png)

Figure 2: B&W image of loaded image

We get a histogram for gray scale image like:

![Histogram for gray scale picture](image2.png)

Figure 3: Histogram for gray scale picture

III. NORMALIZATION OF THE GRAY SCALE IMAGE

After plotting a histogram we require a normalized curve. For obtaining a normalized cumulative distribution function (CDF) curve we start with obtaining hist, bins of that image.

A. To obtain that we start with:

```python
hist,bins = np.histogram(im.flatten(), 256, range)
```

1. hist is same as we used above.
2. bins: will have 257 elements, because Numpy calculates bins as 0-0.99, 1-1.99, 2-2.99 etc. So final range would be 255-255.99. To represent that, they also add 256 at end of bins. But we don't need that 256. Up to 255 is sufficient.
3. flatten(): converts any array to a one-dimensional array with values taken row-wise because hist() takes a one – dimensional array as input.

B. Now we calculate cdf which is cumulative sum of hist(). The cumulative sum of hist is calculated by hist.cumsum() method.

C. From the cdf we calculate normalized cdf by:

```python
Normalized_cdf = cdf * hist.max()/cdf.max()
```

- hist.max(): Maximum value of hist
- cdf.max(): Maximum value of cdf function

D. Now plot the histogram curve by same method as we obtained the hist, bins. And plot the cdf function by:

```python
plt.plot(cdf_normalized, color = 'b')
```

- b is for color blue

We get a graph, plotted like this:

![Histogram (red region) and cdf (blue line) graph](image3.png)

Fig.4. Histogram (red region) and cdf (blue line) graph
IV. EQUALIZATION ON THE IMAGE

Because an image is actually an array of numeric values the distribution of these values can tell you something about the range of colors or shades the image contains or the objects that are in it. You can plot these as a histogram to see the distribution of values and you can also use a cumulative distribution function to see how the pixel values accumulate each point on the y axis. It shows the frequency of all the values up to and including the value on the x axis. Now for digital image processing, it is usually good to have a relatively even distribution of pixel values. A markedly uneven distribution can indicate there are extremes of contrast in the image that need to be equalized so you can normalize an image to equalize pixel distribution and the CDF for a perfectly equalized image is basically a “straight diagonal line”.

A. For equalizing the image we mask the cdf and store the value in another variable cdf_m, by:

```python
np.ma.masked_equal(cdf,0)
```

1. The masked constant is a special case of masked array, with a float data type and a null shape. It is used to test whether a specific entry of a masked array is masked, or to mask one or several entries of a masked array.
2. masked_equal is sub part of masked_where.
3. All operations are performed on non-masked elements.

B. Now cdf_m is calculated by:

```python
cdf = (cdf_m - cdf_m.min()) * 255 / (cdf_m.max() - cdf_m.min())
```

1. cdf_m.min() is the minimum value of cdf_m.
2. cdf_m.max() is the maximum value of cdf_m.

C. np.ma.filled(cdf_m,0).astype('uint8')

1. The masked places in cdf_m are 0 now, and then the cdf values are assigned to the flattened array.

Now, the graph looks like:

We can observe that the cdf function (the blue curve) is now a straight line and the histogram (the red region) is more concentrated in the center.

We can observe the changes in the two images:

Fig. 6: Original image (Left) and Equalized Image (Right)

V. FILTERING/SMOOTHING OF AN IMAGE

Filtering is a technique for modifying or enhancing an image. For example, you can filter an image to emphasize certain features or remove other features. Image processing operations implemented with filtering include smoothing, sharpening, and edge enhancement. Filtering is a neighborhood operation, in which the value of any given pixel in the output image is determined by applying some algorithm to the values of the pixels in the neighborhood of the corresponding input pixel. Here filtering of the image is done by various methods, such as:

- Gaussian blur Filter
- 2D Convolution (Image Filter)
- Averaging Filter
- Gaussian Filter
- Median Filter
- Bilateral Filter

A. Gaussian Blur

In image processing, a Gaussian blur (also known as Gaussian smoothing) is the result of blurring an image by a Gaussian function. It is a widely used effect in graphics software, typically to reduce image noise and reduce detail. The visual effect of this blurring technique is a smooth blur.

1. We can apply Gaussian Blur by:

```python
filter(ImageFilter.GaussianBlur(5))
```

To use Gaussian Blur we need to Import “Image Filter” from “PIL”. The value 5 is the amount of blur the original image will convert into.

2. By changing the value in the parenthesis we can change the amount of blur in the image.
The image will look like
Fig. 7. Original Image (Left) and Gaussian Blur Image (Right)

B. 2D Convolution

As for one-dimensional signals, images also can be filtered with various low-pass filters (LPF), high-pass filters (HPF), etc. A LPF helps in removing noise, or blurring the image. A HPF filters helps in finding edges in an image.

OpenCV provides a function, cv2.filter2D(), to convolve a kernel with an image. As an example, we will try an averaging filter on an image. A 5x5 averaging filter kernel can be defined as follows:

$$K = \frac{1}{25} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Filtering with the above kernel results in the following being performed: for each pixel, a 5x5 window is centered on this pixel, all pixels falling within this window are summed up, and the result is then divided by 25. This equates to computing the average of the pixel values inside that window. This operation is performed for all the pixels in the image to produce the output filtered image.

1. 2D Convolution can be proceeded by making a kernel area:

```python
kernel = np.ones((15, 15), np.float32)/225
cv2.filter2D(im,-5,kernel)
```

- im is the image
- kernel is the value of kernel
- -5 is the value of noise

This creates a normalized box filter of 15x15 which is divided by 225 for obtaining a kernel.

The new image would be:
Fig. 8: Original Image (Left) and 2D Convolution Image (Right)

C. Averaging

This is done by convolving the image with a normalized box filter. It simply takes the average of all the pixels under kernel area and replaces the central element with this average. This is done by the function cv2.blur() or cv2.boxFilter(). The width and height of kernel is specified. A 3x3 normalized box filter would look like this:

1. Averaging can be done by just using blur() method

```python
cv2.blur(im,(10,10))
```

The processed image would look like:
Figure 9: Original Image (Left) and Averaging Filtered Image (Right)

D. Gaussian Filter

In this a, instead of a box filter consisting approach of equal filter coefficients, a Gaussian kernel is used. It is done with the function, cv2.GaussianBlur(). We should specify the width and height of the kernel which should be positive and odd. We also should specify the standard deviation in the X and Y directions, sigmaX and sigmaY respectively. If only sigmaX is specified, sigmaY is taken as equal to sigmaX. If both are given as zeros, they are calculated from the kernel size. Gaussian filtering is highly effective in removing Gaussian noise from the image.

If you want, you can create a Gaussian kernel with the function “cv2.getGaussianKernel()”.

1. Gaussian Filter can be proceeded by making a kernel area:

```python
K = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}
```

The processed image would look like:
1. Gaussian Filtering is done by using “Gaussian blur”:
   
   \[
   \text{cv2.GaussianBlur(blur,(5,5),0)}
   \]

2. Gaussian filtering is applied on a blurred image here.

The images would look like:

![Original Image (Left) and Gaussian Filtered Image (Right)](image1)

**E. Median Filter**

Here, the function cv2.medianBlur() computes the median of all the pixels under the kernel window and the central pixel is replaced with this median value. This is highly effective in removing salt-and-pepper noise. One interesting thing to note is that, in the Gaussian and box filters, the filtered value for the central element can be a value which may not exist in the original image. However this is not the case in median filtering, since the central element is always replaced by some pixel value in the image. This reduces the noise effectively. The kernel size must be a positive odd integer.

1. Median Filtering is done by:
   
   \[
   \text{cv2.medianBlur(blur,5)}
   \]

2. Here also we have taken a disturbed image.

The image looks like:

![Original Image (Left) and Median Filtered Image (Right)](image2)

**F. Bilateral Filter**

As we noted, the filters we presented earlier tend to blur edges. This is not the case for the bilateral filter, cv2.bilateralFilter(), which was defined for, and is highly effective at noise removal while preserving edges. But the operation is slower compared to other filters. We already saw that a Gaussian filter takes the neighborhood around the pixel and finds its Gaussian weighted average. This Gaussian filter is a function of space alone, that is, nearby pixels are considered while filtering. It does not consider whether pixels have almost the same intensity value and does not consider whether the pixel lies on an edge or not. The resulting effect is that Gaussian filters tend to blur edges, which is undesirable.

The bilateral filter also uses a Gaussian filter in the space domain, but it also uses one more (multiplicative) Gaussian filter component which is a function of pixel intensity differences. The Gaussian function of space makes sure that only pixels are ‘spatial neighbors’ are considered for filtering, while the Gaussian component applied in the intensity domain (a Gaussian function of intensity differences) ensures that only those pixels with intensities similar to that of the central pixel (‘intensity neighbors’) are included to compute the blurred intensity value. As a result, this method preserves edges, since for pixels lying near edges, neighboring pixels placed on the other side of the edge, and therefore exhibiting large intensity variations when compared to the central pixel, will not be included for blurring.

1. Bilateral Filter is applied by:

   \[
   \text{cv2.bilateralFilter(blur, 9, 75, 75)}
   \]

   By changing the value = 9. We get a smoother image as we gradually increase the value. (Here the value is 90)

The image looks like:

![Original Image (Left) and Bilateral Filtered Image (Right)](image3)

**VI. CONCLUSION**

Artificial Intelligence is a very wide branch of research and development. This work is more focused on image processing. We can plot a normalization curve which is a histogram and a cdf function which was the blue zig-zag line in the graph. And then we can equalize the image’s contrast and plot the equalized graph which would be of better contrast than the previous image. The newly obtained graph would have a quite center concentrated red colored histogram region and a straight cumulative distribution function blue colored line. And on
comparing the original b&w image and the newly obtained image, we can see the clarity in the newly obtained image. This work also has some of the many image filtering/smoothening techniques; like Gaussian Blur, 2D convolution, Averaging, Gaussian, Median and Bilateral Filter. These all techniques help us removing noise from the images. A Gaussian blur is blurring an image by a Gaussian function. Averaging is done by convolving the image with a normalized box filter. In Gaussian Filter, instead of a box filter consisting approach of equal filter coefficients, a Gaussian kernel is used. The function computes the median of all the pixels under the kernel window and the central pixel is replaced with this median value and is highly effective in removing salt-and-pepper noise. Bilateral Filtering is highly effective at noise removal while preserving edges.

References


[8]. Filtering in the Time and Frequency Domains by Herman J. Blinchikoff, Anatol I. Zverev.


[16]. https://samarthbhargav.wordpress.com/2014/05/05/image-processing-with-python-rbg-to-grayscale-conversion/