# Study of Digital Image Processing & Cleaning Documents using Neural Networks

*Report submitted in fulfillment of the requirements*
*for the Exploratory Project of*

**Second Year B.Tech.**

*by*

**Obilisetty Venkata Srikanth**
**Evuri Harish**

*Under the guidance of*

**Dr. Pratik Chattopadhyay**


**Department of Computer Science and Engineering**
**INDIAN INSTITUTE OF TECHNOLOGY (BHU) VARANASI**
**Varanasi 221005, India**
**May 2021**

# Declaration

I certify that

1. The work contained in this report is original and has been done by myself and the general supervision of my supervisor.
2. The work has not been submitted for any project.
3. Whenever I have used materials (data, theoretical analysis, results) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references.
4. Whenever I have quoted written materials from other sources, I have put them under quotation marks and given due credit to the sources by citing them and giving required details in the references.

Place: IIT (BHU) Varanasi
Date: 09-05-2021

**Obilisetty Venkata Srikanth** of B.Tech,
**Evuri Harish** of B.Tech
Department of Computer Science and Engineering,
Indian Institute of Technology (BHU) Varanasi,
Varanasi, INDIA 221005.

# Certificate

*This is to certify that the work contained in this report entitled "**Study of Digital Image Processing & Cleaning Documents using Neural Networks**" being submitted by **Obilisetty Venkata Srikanth** (**Roll No. 19075050**) and **Evuri Harish** (**Roll No. 19075090**), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology (BHU) Varanasi, is a bona fide work of our supervision.*

Place: IIT (BHU) Varanasi
Date: 09-05-2021

**Dr. Pratik Chattopadhyay**
Department of Computer Science and Engineering,
Indian Institute of Technology (BHU) Varanasi,
Varanasi, INDIA 221005.

# Acknowledgments

We would like to express our sincere gratitude to our supervisor Dr. Pratik Chattopopadhay for providing all the necessary support and guidance for the completion of this project.

Place: IIT (BHU) Varanasi
Date: 09-05-2021

**Obilisetty Venkata Srikanth,**
**Evuri Harish**

# Abstract

From E-books to online government records, the need for digitizing is exponentially increasing. However, some documents cannot be digitized easily as they may contain some strains, heavy foldings, etc. We used Autoencoders and conventional image enhancement techniques to clean the grimy documents and make them easy for digitization. At last, we stacked all our models (methods) and used CNN stacker to generate a better document than any individual model, with the aid of thinking about all models.

# Contents

# Chapter 1

# Introduction to Digital Image Processing

## 1.1 What Is Digital Image Processing ?

An Image is defined as a two-dimensional function, $f(x, y)$, where $x$ and $y$ are spatial coordinates, and the amplitude of $f$ at any pair of coordinates $(x, y)$ is called the *intensity* or *gray level* of the image at that point. When $x, y$, and the amplitude values of $f$ are all finite, discrete quantities, we call the image a *digital image.*[1] The field of *digital image processing* refers to processing digital images by means of a digital computer.[1] Digital images are made of picture elements called pixels, each pixel has its own location and value.

From image processing to computer vision, we consider three types of processes : low, mid and high-level processes. For low-level processes it is considered as both input and output are images and they involve in tasks such as noise removal, image sharpening. For mid-level processes it is considered that input is image and output is attributes extracted from the image and they involve in tasks such as segmentation, image classification and image recognition. Lastly for high-level process it is considered that input is attributes extracted from image and output is understanding and they involve in tasks such as scene understanding and autonomous navigation.

## 1.2 Origin of Digital Image Processing

One of the very first applications of digital images was *Bartlane cable picture transmission system* used in newspaper industry where pictures were sent via submarine cable between New York and London and this reduced the required time from more than a week to less than three hours. The picture below was transmitted in this way.

**Figure 1.1**: A digital picture transmitted by Bartlane system [1]

## 1.3 Components of an Image Processing System

Figure 1.2 shows the basic components of an digital image processing system. The description of each component is discussed below.

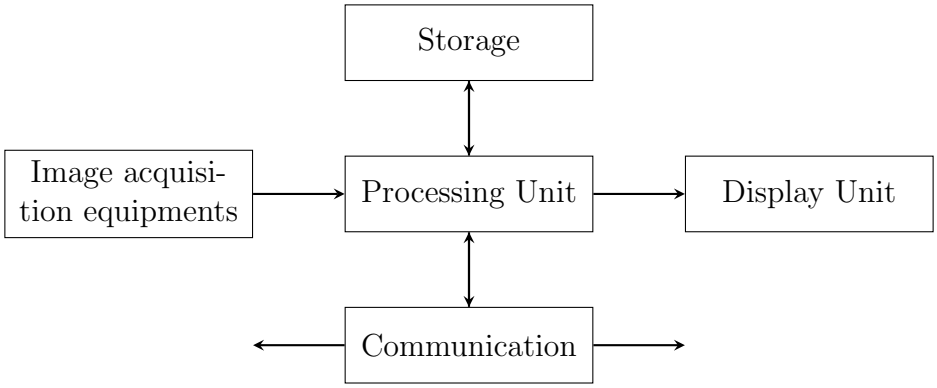



**Figure 1.2**: Basic Fundamental components of an image processing system

- *Image acquisition equipments* such as camera and scanner are input devices (through which image is taken as input) which consists of image sensors which senses the intensity, co-ordinates and other features of the images.
- *Processing Unit* such as computer is used for processing the images.
- *Display Unit* such as TV, monitors and printers are output devices that displays the processed images.
- *Storage* devices such as optical disks, tape are used for storing images.
- *Communication* is passing of images from one device to another device through cables or internet.

# Chapter 2

# Fundamentals of Digital Image

## 2.1 Image Sensing and Acquisition

We use imaging sensor to convert the incoming EM energy into electrical signals with the help of input electrical power. The sensing material respond to specific type of incident energy and the filter is used to improve selectivity.

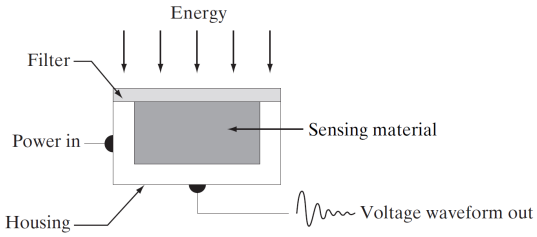



**Figure 2.1**: An Imaging sensor. [1]

To generate a digital image there are three types of sensor arrangements.

- Single imaging sensor
- Line sensor
- Array sensor

*Single imaging sensor* can generate one point of image at a time, so to generate a 2-D image there should be motion of sensor in both $x$ and $y$ directions. In this method high-resolution images can be obtained, but this is slow process.

*Line sensor (Sensor strip)* can generate one line of an image at a time, So 2-D image can be generated by moving in perpendicular direction to strip. This system is used in flat bed scanners.

*Sensor array* is the 2-D arrangement of individual sensors which is generally used in digital cameras. Since the arrangement is two dimensional, there is no need of motion. As shown in the Figure 2.2 the imaging system (sensor array) collects the reflected light rays from scene element and focuses it onto image plane. The outputs of sensor array is digitized. And finally digital image is generated.

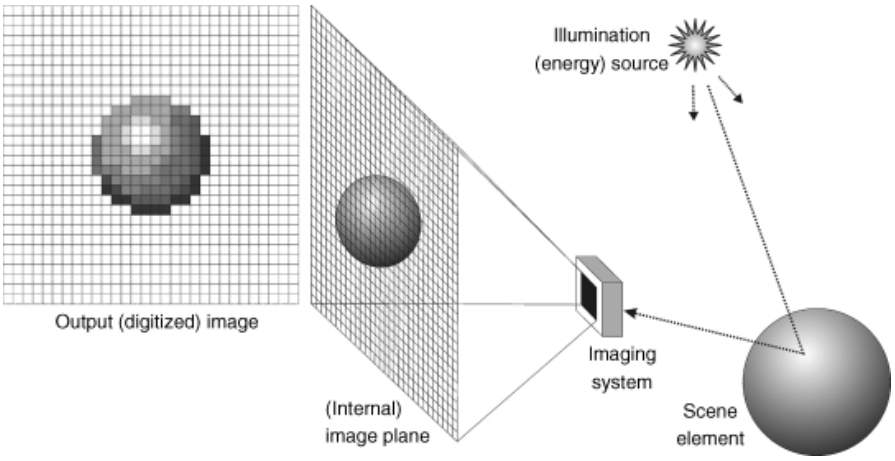



**Figure 2.2**: Image formation using sensor array. Redrawn from [1].

## 2.2 Digitization of Image

As the output of image sensors is in the form of analog signal. So, by converting continuous signal to digital form, digital image can be created. This process includes 2 steps :

- Sampling
- Quantization

### 2.2.1 Sampling and Quantization

*Sampling* refers to discretization of spatial coordinates. In this process we measure the value of function at discrete intervals along the $x$ and $y$ dimensions. It can be said that more samples we take, the quality of the image would be more better.

*Quantization* refers to discretization of amplitude (gray level values). In this we replace the continuously varying function $f(x, y)$ with a discrete set of quantization levels (gray levels). Generally, we consider $L$ quantization levels in digitizing an image where $L$ is usually an integral power of 2.

$$L = 2^k$$

The result of sampling and quantization is a digital image which can be represented in the form of matrix. And each element of the matrix is called *pixel*. Consider a digital image with $M$ rows and $N$ columns, then the digital image in matrix form is:

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & \dots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \dots & f(M-1,N-1) \end{bmatrix}$$

To digitize an image we require the values of $M, N$ and $L$ where $L = 2^k$, so to encode each pixel we need $k$ bits. So, that image is referred as k-bit image. And

to store that digitized image, the number of bits $b$ required is:

$$b = M * N * k$$

### 2.2.2 Spatial and Gray-Level Resolution

Spatial resolution of an image represents the number of pixels in that image, and Gray-level resolution of an image represents the number of gray levels in it. It is common to say that a digital image of size $M$ x $N$ and having $L$ gray levels, has spatial resolution of $M$ x $N$ pixels and gray-level resolution of $L$ levels.

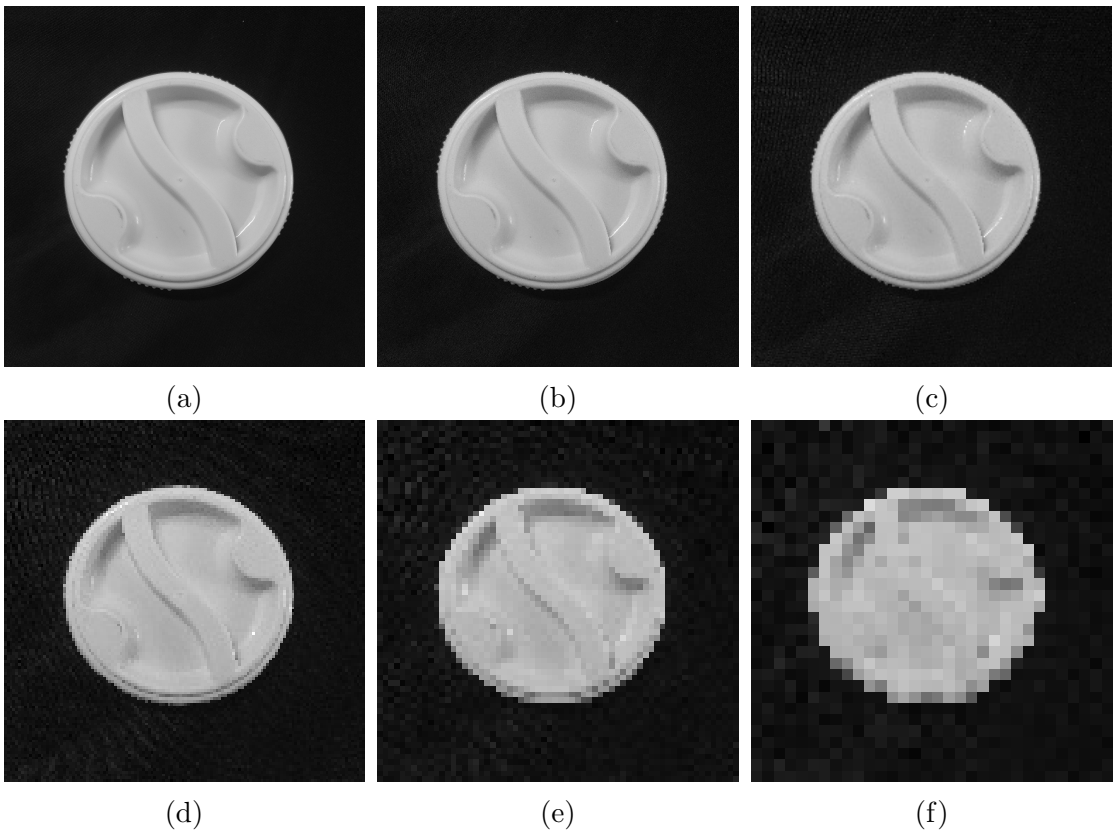



**Figure 2.3**: (a) 1024 x 1024, 8-bit image. (b) through (f) Re-sampling 512 x 512, 256 x 256, 128 x 128, 64 x 64 and 32 x 32 images to size of 1024 x 1024.

Figure 2.3(a) shows a 8-bit image of size 1024 x 1024 pixels. The other images are obtained by subsampling 1024 x 1024 image to their respective sizes as shown in the figure and then resampling into size of 1024 x 1024. The resampling was done by row and column duplication.

By observing Figure 2.3 on moving from (a) to (f) we can clearly observe that the level of detail lost is increasing and checkerboard pattern also getting increased.

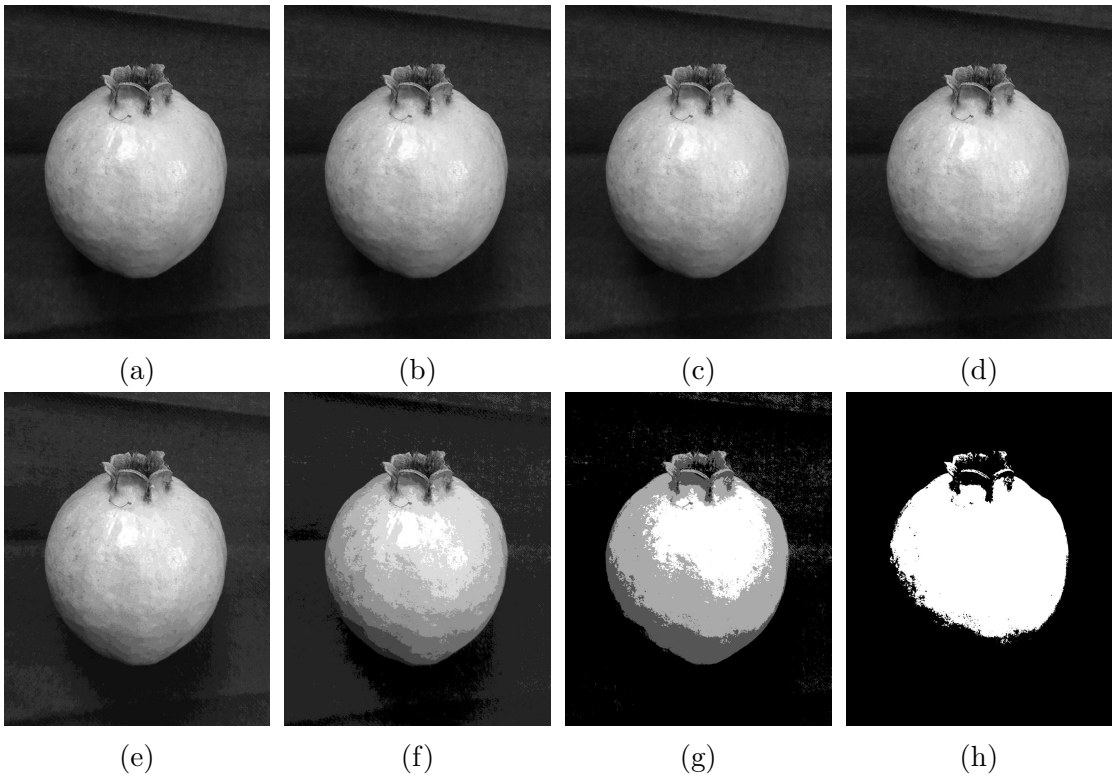


**Figure 2.4**: (a) 1000 x 800, 256-level image. (b) through (h) 128-level, 64-level, 32-level, 16-level, 8-level, 4-level and 2-level images with same spatial resolution.

In the Figure 2.4 we reduce the number of gray levels from 256 to 128, 64, 32, 16, 8, 4 and finally 2, by keeping the spatial resolution constant. So, the value of k (no.of bits) is reduced from 8 in (a) to 1 in (h). Figures 2.4 (a) through (d) looks similar, we can notice the effect of decreasing gray levels from Fig 2.4 (e). In Fig 2.4 (h) we can see only two gray levels (white and black).

### 2.2.3 Zooming and Shrinking

Zooming is considered as oversampling and shrinking is considered as undersampling. The main difference between zooming, shrinking and sampling, quantization is : sampling, quantization are applied on original continuous image to generate digital image, whereas zooming, shrinking are applied to digital image.

Zooming is a method to increase the size of a image. It requires creation of new pixel locations, and assigning gray levels to those locations. There are 2 techniques to zoom an image :

*Nearest neighbor interpolation* is a simple method in which we choose the nearest pixel in original image and assign its gray level to the new pixel.

*Bilinear interpolation* considers four nearest neighboring pixels surrounding the unknown pixel. Then the weighted average of these 4 pixels is assigned to the new pixel. The resulting image looks more smoother than image generated by nearest neighbor interpolation.

(a) (b) (c)

**Figure 2.5**: (a) 128 x 128, (b) 64 x 64, (c) 32 x 32, images zoomed to size of 1024 x 1024 using nearest neighbor interpolation.

(a) (b) (c)

**Figure 2.6**: (a) 128 x 128, (b) 64 x 64, (c) 32 x 32, images zoomed to size of 1024 x 1024 using bilinear interpolation.

In the Figure 2.5 (c) the image is very blurry since it was zoomed 32 times bigger. But the result of bilinear interpolation as shown in Fig 2.6 (c) is relatively smoother and better.

*Shrinking* the image is similar to zooming. In shrinking we delete appropriate number of pixels which results in smaller image.

# Chapter 3

# Image Enhancement Techniques

Image enhancement is a procedure of processing an image so that the resulting image is more suited/useful than original for a particular application. It falls into two categories: One is enhancement in spatial domain and the other is in frequency domain. The spatial domain techniques directly operate on the pixels of image, where as the frequency domain techniques deal with modifying the Fourier transform of image.

We use the following expression for denoting spatial domain methods :

$$g(x, y) = T[f(x, y)] \quad [1] \quad (3.1)$$

In the above expression, the input image is denoted by $f(x, y)$, output image is denoted by $g(x, y)$, and $T$ is an operator on the image. $T$ is defined on some neighborhood of $(x, y)$, the neighborhood is a rectangular subimage area which has center $(x, y)$. The simplest form of input is a single pixel neighborhood. Then $g$ at $(x, y)$ depends only on the value of $f$ at $(x, y)$. Then the above expression reduces to :

$$s = T(r) \quad [1] \quad (3.2)$$

where r and s denotes the intensity values of $f(x, y)$ and $g(x, y)$ at point $(x, y)$.

## 3.1 Basic Gray Level Transformations

### Image thresholding

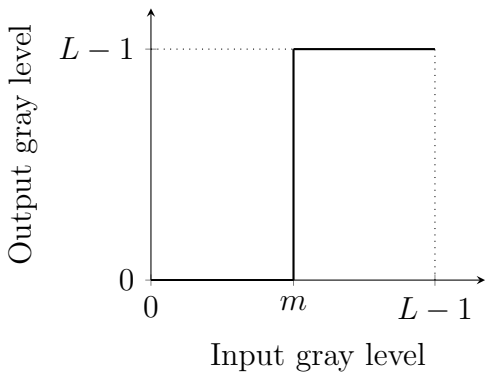



**Figure 3.1**: Thresholding function

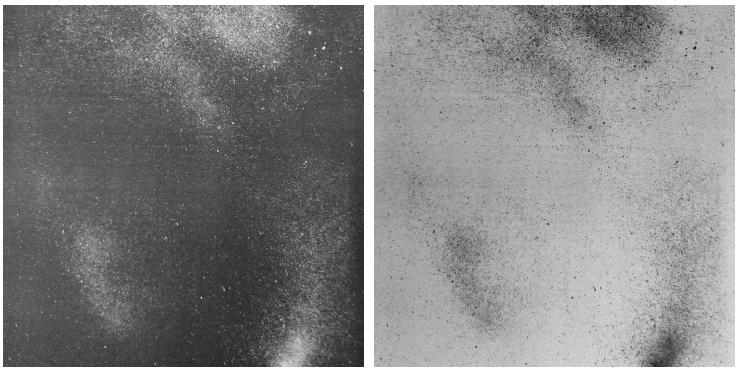
In the Figure 3.1 the function $T(r)$ is thresholding function. In this process the pixels with intensity greater than threshold value ($m$) in the input image are replaced by white pixels and remaining by black pixels. The result of thresholding is a binary image.

### Negative Transformation

The negative of an image is obtained by reversing the intensity levels of the image, The negative transformation of image with $L$ gray levels is given by:

$$s = L - 1 - r \qquad [1] \quad (3.3)$$

Negative transformation is suitable when there is minute gray or white details in dark regions.



(a) (b)

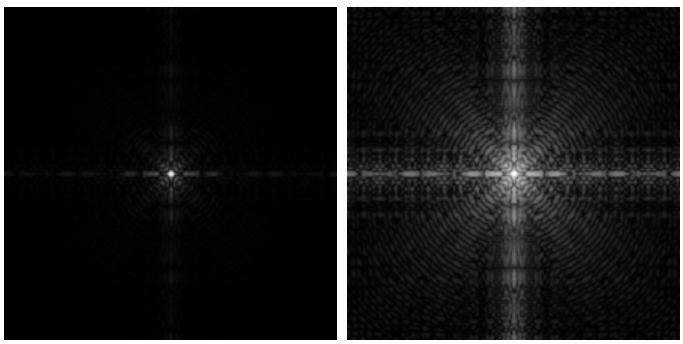
**Figure 3.2**: (a) Original Image. (b) Image Negative of (a).

### Log Transformations

The mathematical expression for log transformation is:

$$s = c\log(1 + r) \qquad [1] \quad (3.4)$$

where c is a constant. The log transform is used to expand the values of dark pixels and compress the higher-level pixels in an image.



(a) (b)

**Figure 3.3**: (a) Fourier Spectrum. (b) Result of log transformation with c = 1. Both the images are taken from [1].

The Figure 3.3 (a) shows the fourier spectrum with values in range 0 to $10^6$, when these values are scaled linearly from 0 to 255, most of the pixels get darker. So, if we apply log transform then the range becomes 0 to 6. Now, if we scale linearly, the result would be better, Figure 3.3 (b) shows the result.

Inverse log transform is opposite to log transform, it is used to expand the values of higher-level pixels and compress the dark pixels.

### Power Transformations

The mathematical expression for power transformation is:

$$s = c^{\gamma} \quad [1] \quad (3.5)$$

where $c$ and $\gamma$ are positive constants. We use $\gamma > 1$ to map smaller range of higher-level input values to larger range of output values and vice-versa.

(a) (b) (c) (d)

**Figure 3.4**: (a) Image having washed-out look. (b) through (d) Results of power-law transformation on (a) with $c = 1$, $\gamma = 3$, $\gamma = 6$ and $\gamma = 9$. (Original Image courtesy of Mr.Ansel Adams, The Tetons and the Snake River, Grand Teton National Park, Wyoming).

In the Figure 3.4, since the original image has white-washed look, so power-law transformation is applied with $\gamma > 1$ to get a darker image, and we can clearly observe that image gets more darker on increasing the value of $\gamma$.

Power-law transformation with $\gamma < 1$ is exactly opposite with $\gamma > 1$. It is used to enhance a dark image to generate a lighter one.

## 3.2 Histogram Techniques

The representation of a histogram of an image is given by:

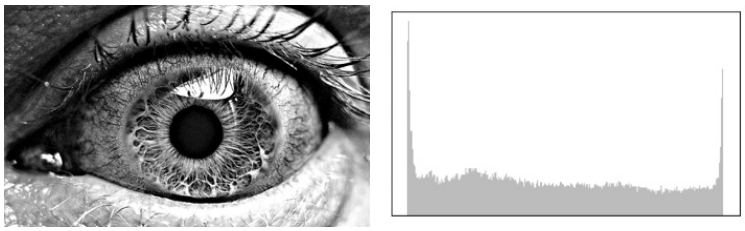
$$h(r_k) = n_k$$

where, the number of pixels with gray level $r_k$ is denoted by $n_k$. The range of $k$ is $[0, L-1]$, where $L$ is the number of gray levels in the image.

Normalised histogram is represented by:

$$p(r_k) = n_k/n$$

where n denotes the total number of pixels in the image.



(a) (b)

**Figure 3.5**: (a) Original Image. (b) Histogram of Image (a). (Original Image taken from clipart-library.com).

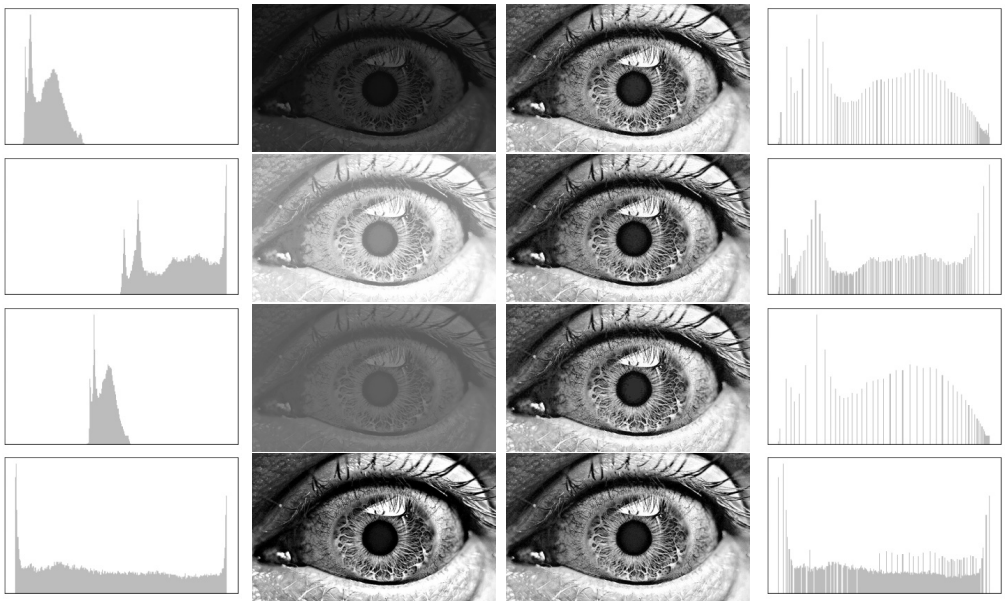
### 3.2.1 Histogram Equalization

The transformation function for histogram equalization is given by:

$$s_k = T(r_k) = \sum_{i=0}^{k} p(r_i) = \sum_{i=0}^{k} \frac{n_i}{n} \qquad [1] \quad (3.6)$$

where, $n_i$ denotes the number of pixels with $i^{th}$ gray-level, n denotes the total number of pixels. $k$ is in range $[0, L-1]$.

The histogram-equalized image has a uniform histogram which covers the entire range of gray scale, so the resultant image has a good contrast. And also this transformation does not require any other parameters.



(a) (b) (c) (d)

**Figure 3.6**: (b) consists of four images (dark-image, light-image, low-contrast image, high-contrast image). (a) is the corresponding histogram of (b). (c) is result of histogram equalization on (b). Finally, (d) is the corresponding histogram of (c).

Figure 3.6 (c) shows the results of histogram equalization on images in fig 3.6 (b), we can notice that all the resultant images have a good contrast, and their histograms shown in fig 3.6 (d) covers the entire gray scale. We can see that the fourth image does not have improvement because the histogram of that image already covers the whole gray scale.

### 3.2.2 Histogram Matching

Histogram Matching is a method to generate an output image that has specified histogram. In this method we have to provide the histogram along with the input image.

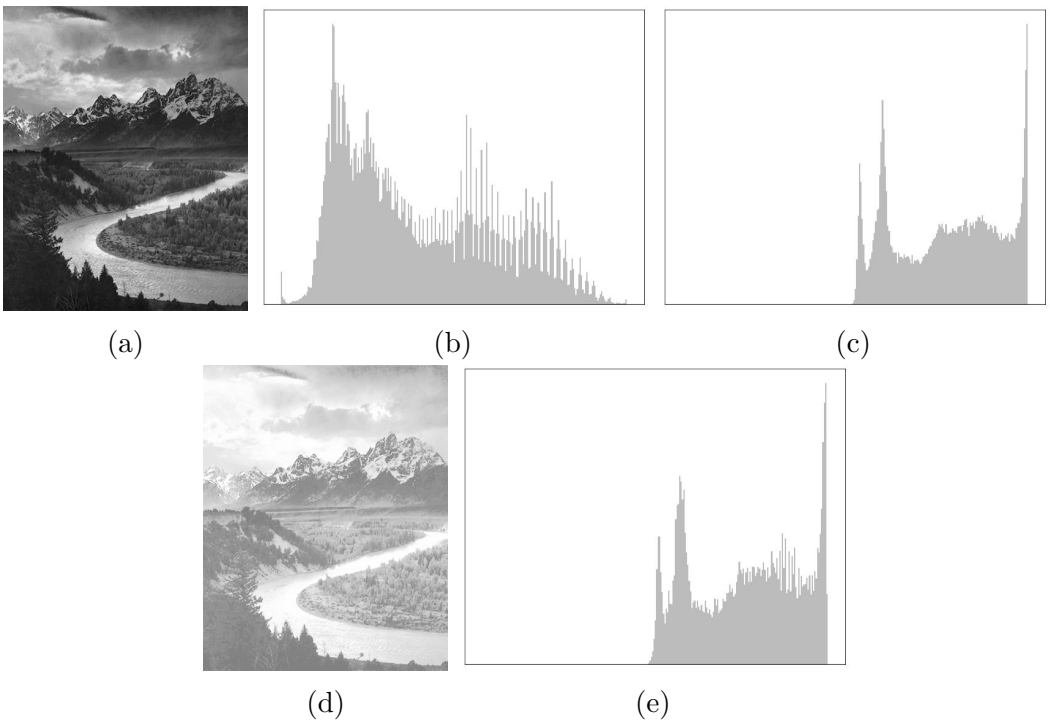



**Figure 3.7**: (a) Original Image. (b) Histogram of Image (a). (c) Specified Histogram. (d) Result of Histogram Matching of image (a) with histogram (c). (e) Histogram of Image (d).

Figure 3.7 (a) is the original image, and fig 3.7 (c) shows the provided histogram, and fig 3.7 (d) shows the result of Histogram Matching.

In this method we can enhance the image to our requirement by specifying the suitable histogram.

## 3.3 Arithmetic and Logic Operations

### Image Subtraction

In this method we compute the difference between two images $f(x, y)$ and $g(x, y)$ :

$$h(x, y) = f(x, y) - g(x, y) \tag{3.7}$$

This method is mainly used to get the differences between images.

(a) (b) (c)

**Figure 3.8**: (a) Image-1 [ $f(x, y)$ ], (b) Image-2 [ $g(x, y)$ ], (c) Result of subtraction of Image-2 from Image-1 [ $f(x, y) - g(x, y)$ ].

## 3.4 Spatial Filtering Basics

Spatial Filtering is a process which deals with values of pixels in neighborhood and the corresponding values in the subimage which has same dimensions as the neighborhood. That subimage is called as mask, kernel or filter. The values in the subimage are referred as coefficients. In the spatial filtering process we convolute the mask on the image and at each point we compute the response of the mask using predefined relationship.

| | | |
|---|---|---|
| $w_1$ | $w_2$ | $w_3$ |
| $w_4$ | $w_5$ | $w_6$ |
| $w_7$ | $w_8$ | $w_9$ |

**Figure 3.9**: A representation of 3 x 3 filter mask.

In linear filtering of an image with a mask of size $m$ x $n$, the response $R$ is given by the expression:

$$R = w_1 z_1 + w_2 z_2 + w_3 z_3 + \ldots + w_{mn} z_{mn} = \sum_{i=1}^{mn} w_i z_i$$

where $w$'s and $z$'s respectively denote the mask coefficients and the gray level values of pixels corresponding to those coefficients.

## 3.5 Smoothing spatial filters

We use smoothing filters for image blurring and reduction of noise.

### Smoothing using Linear Filters

The smoothing linear filters are also called as averaging filters, because the response $R$ of this filter is the average of all pixels in the neighborhood defined by the filter. Since the average of all neighboring pixels is computed at each point, so the resultant image do not have sharp transitions in gray-levels and it also has less noise which is an advantage. But the disadvantage is the edges in resultant image get blurred. There are two main types of smoothing linear filters : Standard Averaging filter ( Mean filter ), Weighted Averaging filter.

$$\frac{1}{9}\times\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix} \qquad \frac{1}{16}\times\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}$$

**Figure 3.10**: Two 3 x 3 smoothing filters, the one in left is a standard averaging filter and the one in right is a weighted averaging filter.

The response $R$ of standard averaging filter shown in Figure 3.10 at a point is the standard average of all pixels in the 3 x 3 neighborhood defined by the filter. In that filter all the coefficients have the same value 1, and at end of this filtering process, the whole image is divided by the value 9.

The other filter shown in Figure 3.10 is a weighted average filter, in which pixels are multiplied by coefficients with different values. In this mask, center is weighted with higher value and the other pixels are weighted according to the distance from the center of the mask i.e. the nearest pixel has more weight. So this filter has less blurring effect compared to the previous one.

(a) (b) (c) (d)

**Figure 3.11**: (a) Original Image. (b) through (d) Results of smoothing on (a) using mean filter with kernel sizes (3,3), (5,5) and (9,9).

Form the figure 3.11 we can see that the blurring effect increases on increasing the kernel size.

### Smoothing using Non-linear filters

These filters include order-statistics filters, in which the response $R$ of the filter depend on the order of the pixels in the neighborhood defined by the filter.

Types of order-statistics filters :

- Minimum filter : The center value is replaced by the minimum value in the neighborhood defined by filter. It is also called as 0th percentile filter.
- Maximum filter : The center value is replaced by the maximum value in the neighborhood defined by filter. It is also called as 100th percentile filter.
- Median filter : Each pixel is considered, and all the pixels in the neighborhood defined by filter are sorted and median is calculated, and the original value of pixel is replaced by the median.

*Median filter* is very useful for smoothing images. They they provide good noise reduction for images with impulse noise *(salt-and-pepper noise)*. This filter also produce less blurring than linear filters of same size.

Using a median filter of size $n$ x $n$, the clusters of pixels that are dark or light compared to neighbors having area less than $n^2/2$ can be removed.

(a) (b)

**Figure 3.12**: (a) Image with salt-and-pepper noise. (b) Result of Median filtering on (a).

From the figure 3.12 we can see that median filter works very effectively in removing salt-and-pepper noise.

## 3.6 Sharpening Spatial Filters

We use sharpening spatial filter to enhance the details that have been blurred and highlight the edges. Sharpening can be achieved with the aid of spatial differentiation.

Generally, sharpening filters use the second order operators because the second order operators are more sensitive to intensity variations than the first order operators. And also we focus on sharpening filters that are isotropic (rotation invariant).

### Sharpening using Laplacian filter

The Laplacian is a simple isotropic derivative operator (with respect to the principal directions), Laplacian operator on function $f(x, y)$ is defined as:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \quad [1] \quad (3.8)$$

In a digital image the second derivatives wrt. x and y are computed as:

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y) \quad [1] \quad (3.9)$$

$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y) \quad [1](3.10)$$

Hence the Laplacian results:

$$\nabla^2 f = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y) \quad (3.11)$$

To obtain a sharpened image $g(x, y)$ we subtract the laplacian image from the original image:

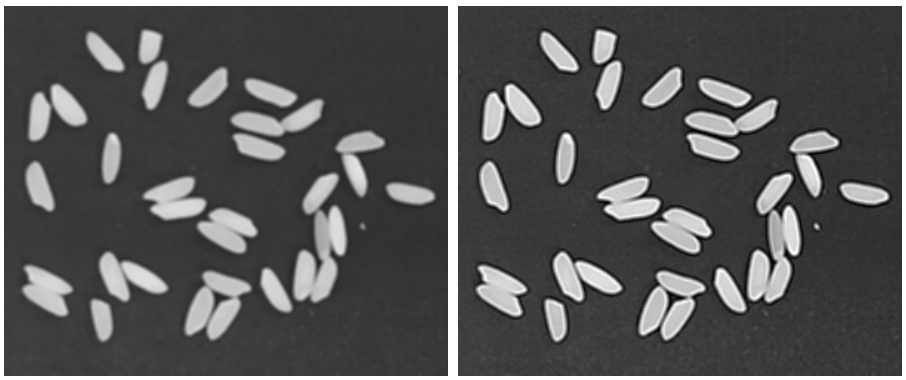
$$g(x, y) = f(x, y) - \nabla^2 f \quad [1](3.12)$$

$$g(x, y) = 5f(x, y) - f(x+1, y) - f(x-1, y) - f(x, y+1) - f(x, y-1) \quad (3.13)$$

The above equation could be implemented using the filter shown in Figure 3.13

| | | |
|---|---|---|
| 0 | −1 | 0 |
| −1 | 5 | −1 |
| 0 | −1 | 0 |

**Figure 3.13**: Filter for implementing above equation of $g(x, y)$



(a) (b)

**Figure 3.14**: (a) Original Image. (b) Sharpening on Image (a) using the above filter.

Figure 3.14 (b) shows the result of sharpening on fig 3.14 (a) using the filter in fig 3.13. We can see the resultant image has clearer and sharper edges than in orginal image.

### Unsharp Masking

In this process we subtract blurred version of image from the original image to obtain a sharpened image.

$$f_s(x,y) = f(x,y) - \bar{f}(x,y) \qquad [1](3.14)$$

where $\bar{f}(x,y)$ is the blurred version of $f(x,y)$ and $f_s(x,y)$ is the resulting sharpened image.

### High-boost filtering

This process is represented by the expression :

$$f_{hb}(x,y) = Af(x,y) - \nabla^2 f(x,y) \qquad [1](3.15)$$

where $A \geq 1$.

As the value of the $A$ increases, the sharpening effect decreases and the resulting image looks more like the original.

# Chapter 4

# Cleaning Documents using Traditional techniques

In the chapter we will discuss about cleaning the images of printed text which contain shadows, wrinkles(folds), stains and other noises using traditional techniques.

## 4.1 Dataset Used

For dataset preparation we used the denoising-dirty-documents dataset[2] from kaggle which was created by RM.J. Castro-Bleda, S. España-Boquera, J. Pastor-Pellicer, F. Zamora-Martinez. We have done some data agumentations like horizontal flip and vertical flip to increase the size of dataset. This dataset consists of different types of dirty documents (images) and their corresponding cleaned (target) images.

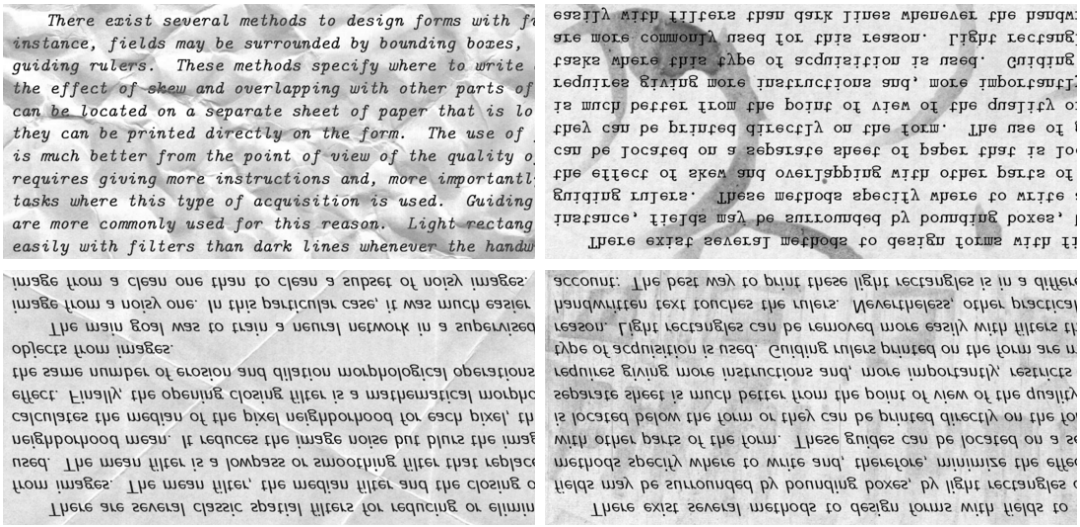
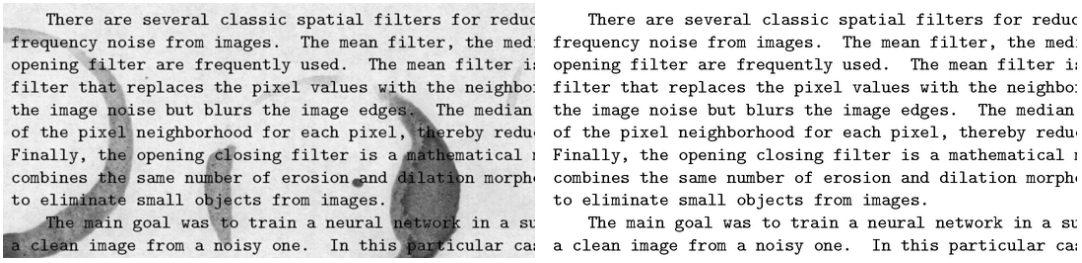


**Figure 4.1**: Different types of dirty images.




(a) (b)

**Figure 4.2**: (a) A dirty image (b) Cleaned Image (target) corresponding to (a).

## 4.2 Using Gaussian Blur and Adaptive Thresholding

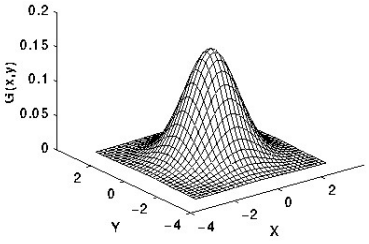
In this method first we use gaussian blur to smoothen the input image and to reduce the noise and then we use adaptive thresholding to generate the mask with background white and text with black. Finally we add this mask to the input original image to generate the desired output image.

### Gaussian Blurring(Filtering)

Gaussian blur is a method to blur the image using the Gaussian function. The two-dimensional Gaussian function is given by :

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where $\sigma$ is the standard deviation.




$$\frac{1}{273}\times\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

(a) (b)

**Figure 4.3**: (a) A 2D Gaussian distribution with mean (0,0) and $\sigma = 1$, (b) Discrete approximation of Gaussian function with $\sigma = 1$. [3]

Theoretically Gaussian distribution is non-zero everywhere, which would require an infinitely big kernel. But practically this distribution approaches very close to zero at about three standard deviations from the mean. So we can truncate the kernel at this point. The Figure 4.3 (b) shows a integer valued 5 by 5 kernel approximating Gaussian with $\sigma = 1$.

In Gaussian filtering near-by pixels have a bigger influence on smoothing rather than more distant ones. But in the mean filter, all the pixels which belong to the kernel are given equal weight. So gaussian blur produces pure smoothing effect without side effects.

### Adaptive Thresholding

In previous chapter, we have discussed about simple thresholding where we have used a fixed threshold value globally, but it may not be good when an image have some bright areas and some dark areas. So, in such cases we can use Adaptive Thresholding in which we calculate threshold for *small regions* in the image. Since every region has its own threshold value, the result will be much better than simple thresholding.

The threshold value can be decided in two ways :

- threshold value = mean of neighborhood values.
- threshold value = Gaussian weighted sum of neighborhood values.

It is obvious that result of adaptive thresholding is a binary image.

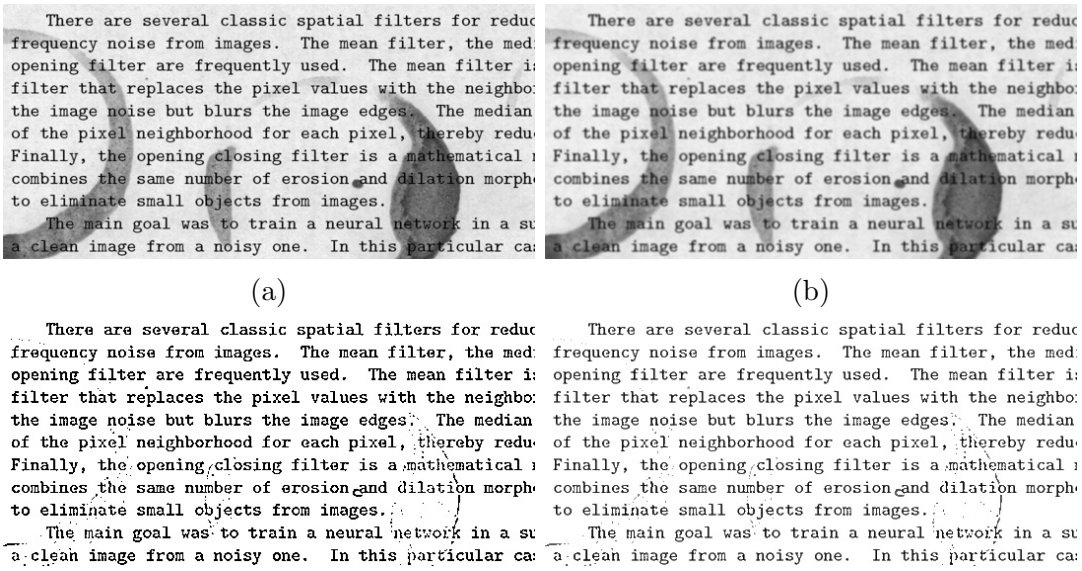



**Figure 4.4**: (a) Input image (source) of size 258 x 540. (b) Result of Gaussian filtering on (a) with kernel size (3,3). (c) Resultant mask by adaptive (gaussian) thresholding on (b) with block size (5,5). (d) Output image obtained by adding mask (c) to original input image (a).

Form the figure 4.4 we can clearly observe that the generated output image contains clear text with very minute noise.

Now to compare the generated output image with target image, we use RMSE (Root Mean Square Error) as the metric.

Using this we achieved a RMSE score of **0.0751**

## 4.3 Using Median filter

In previous chapter we have discussed how the median filter works. Here we use median filter to get the background of the input image and then we generate a

mask comparing the median filtered image with the original one. Finally we use this mask on the original image to generate the desired output image.

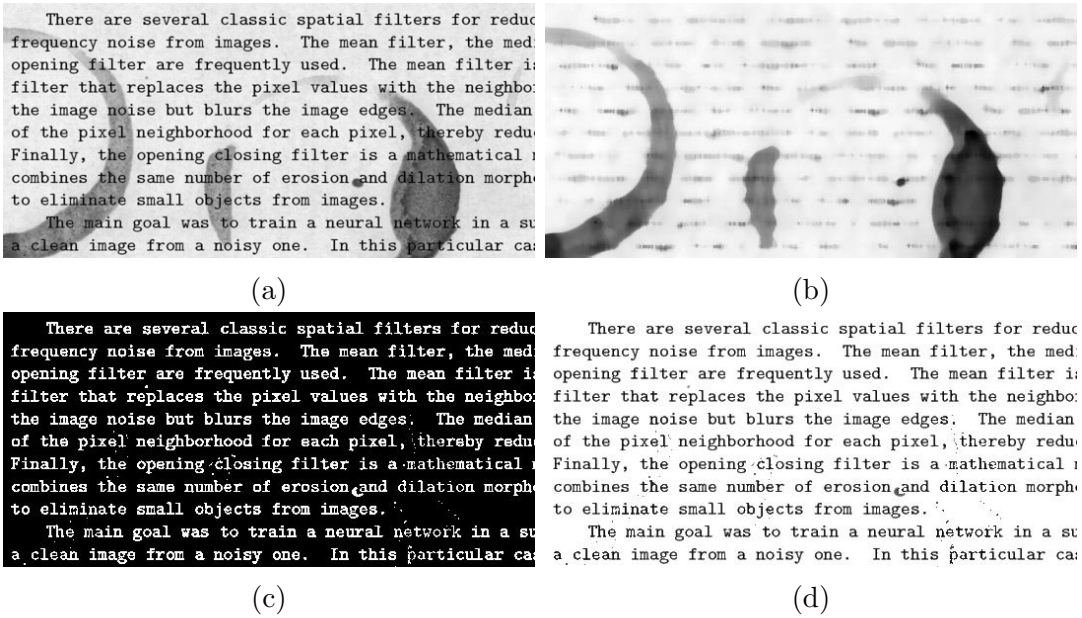


**Figure 4.5**: (a) Input image (source) of size 258 x 540. (b) Result of Median filtering on (a) with kernel size 11. (c) Resultant mask by comparing (b) with (a). (d) Output image by using mask (c) on original input image (a).

From the figure 4.5 we can see that the output image generated has negligible amount of noise with very clear text. This technique works quite better than the previous one.

Using this method we achieved a RMSE score of **0.0519**

# Chapter 5

# Cleaning Documents using Neural Networks

In the last chapter, we enhanced our image using traditional image enhancement techniques. In this chapter, we are going to solve our problem using Neural Networks. We specifically used Autoencoders to solve this.

## 5.1 Autoencoders

Autoencoders[4] are a unique sort of neural network architecture wherein the output is identical to the input in size and image itself. Autoencoders are skilled in an unsupervised way to research the extraordinarily low degree representations of the input data. These low-level features are then deformed back to project the input data. These networks have a tight bottleneck of some neurons in the middle.[5]

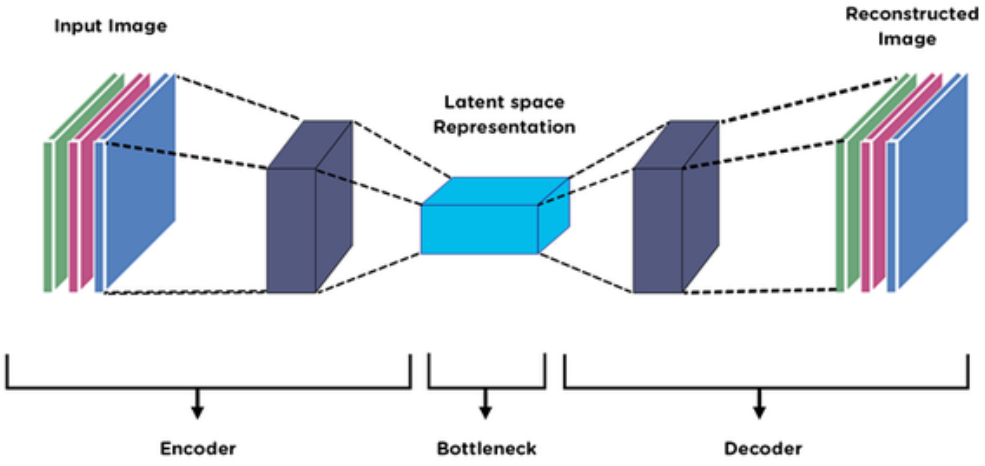



**Figure 5.1**: Autoencoder with fully connected hidden layers.[6]

An autoencoder architecture comprises of three main components:

- Encoding Architecture.
- Latent View Representation.
- Decoding Architecture.

The main use cases of Autoencoders are :

1. Image Denoising.
2. Dimensionality Reduction.
3. Image Compression.
4. Image Generation.

So, here for denoising and cleaning our tampered documents, autoencoders are the right choice.

## 5.2 Using Convolutional Autoencoders

### Architecture

We used **Convolutional Autoencoders** architecture for solving this task. Our model consists of 4 convolutional hidden layers (2 layers as encoder and 2 layers as decoder).We used optimizer as RMSprop with learning rate 0.001. We used Mean square error(mse) as our loss function. For activation function we used ReLU[7] to hidden layers and Sigmoid function to output layer. Batch Normalization[8] is used for some layers to stabilize the training process and reduce the number of iterations required to train the network.

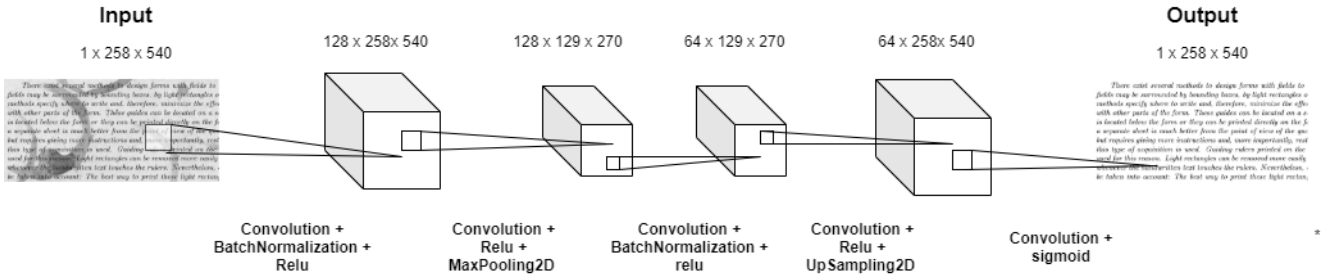



**Figure 5.2**: Schematic representation of implemented Autoencoder model.

| Layer Name | Type | Kernel Size | Output | Parameters |
|---|---|---|---|---|
| conv2D_1 | Convolutional | 3 x 3 | 258 x 540 x 128 | 1280 |
| batchNo_1 | Batch Normalization | - | 258 x 540 x 128 | 512 |
| activation_1 | ReLU | - | 258 x 540 x 128 | 0 |
| conv2D_2 | Convolutional + ReLU | 3 x 3 | 258 x 540 x 128 | 147584 |
| max_pooling_1 | Max Pooling | 2 x 2 | 129 x 270 x 128 | 0 |
| conv2D_3 | Convolutional | 3 x 3 | 129 x 270 x 64 | 73792 |
| batchNo_2 | Batch Normalization | - | 129 x 270 x 64 | 256 |
| activation_2 | ReLU | - | 129 x 270 x 64 | 0 |
| conv2D_4 | Convolutional + ReLU | 3 x 3 | 129 x 270 x 64 | 36928 |
| upsampling | Up Sampling | 2 x 2 | 258 x 540 x 64 | 0 |
| conv2D_5 | Convolutional + Sigmoid | 3 x 3 | 258 x 540 x 1 | 577 |

**Table 5.1**: Detailed Layer configuration of implemented Convolutional Autoencoder network

### Parameters and Hyperparameters

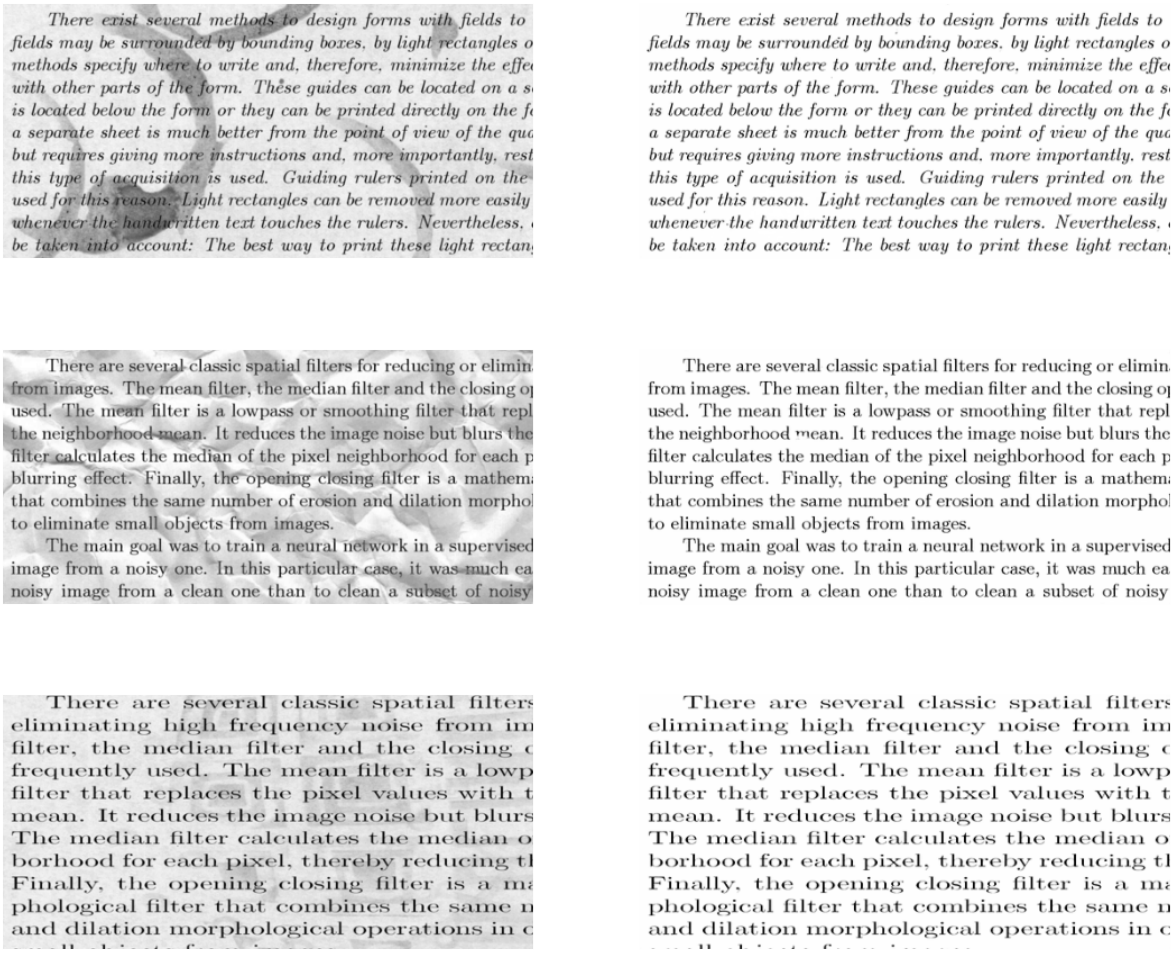
| Parameters and Hyperparameters | |
|---|---|
| Optimizer | RMSprop (lr=0.001) |
| Loss function | Mean square error(mse) |
| Activation function | Relu,Sigmoid |
| Kernal Size | 3x3 |
| Padding | same |
| Epochs | 100 |
| Batch size | 7 |

### Evaluation and Predictions

After training this convolutional autoencoder model for 100 epochs on train dataset we got Train RMSE as 0.0228 and Validation RMSE as 0.0226. On test dataset we got an RMSE as **0.0231**.

### Predictions :

Dirty images | Predicted Images



## 5.3 Using Stacked Convolutional Neural Network

Stacked generalization[9] is an ensemble technique in which a new model learns how to comprise the best predictions of more than one existing model. Stacked CNNs learn different features and image representations generated from models already developed in their unique way. So stacked CNN model will be generalized over all models, and it gives relatively better performance.

We are stacking three models which are all mentioned above and they are :

- Model generated using Gaussian Blur and Adaptive Thresholding .
- Model generated using Median filter.
- Model generated using Convolutional Autoencoders.

### Architecture

The Stacker we built is a specific type of convolutional neural network. This Stacker consists of 6 convolutional layers with batch normalization for each layer. We used Relu as activation for all hidden layers and sigmoid output layer. This CNN stacker inputs a 3d array which has 4 channels comprise of 4 images and they are :

1. Original image
2. Image generated using Gaussian Blur and Adaptive Thresholding .
3. Image generated using Median filter.
4. Image generated using Convolutional Autoencoders.

Thus this CNN stacker model makes use of all four data to provide the wiped clean output.

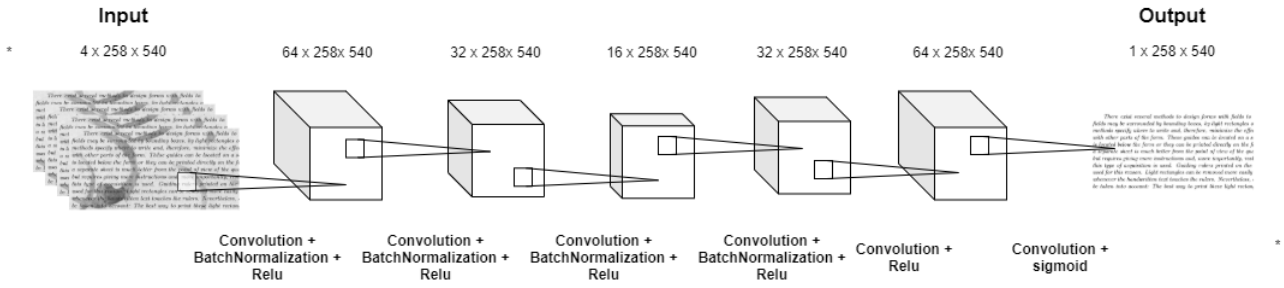



**Figure 5.3**: Schematic representation of implemented Stacked CNN model.

| Layer Name | Type | Kernel Size | Output | Parameters |
|---|---|---|---|---|
| conv2D_1 | Convolutional | 3 x 3 | 258 x 540 x 64 | 2368 |
| batchNo_1 | Batch Normalization | - | 258 x 540 x 64 | 256 |
| activation_1 | ReLU | - | 258 x 540 x 64 | 0 |
| conv2D_2 | Convolutional | 3 x 3 | 258 x 540 x 32 | 18464 |
| batchNo_2 | Batch Normalization | - | 258 x 540 x 32 | 128 |
| activation_2 | ReLU | - | 258 x 540 x 32 | 0 |
| conv2D_3 | Convolutional | 3 x 3 | 258 x 540 x 16 | 4624 |
| batchNo_3 | Batch Normalization | - | 258 x 540 x 16 | 64 |
| activation_3 | ReLU | - | 258 x 540 x 16 | 0 |
| conv2D_4 | Convolutional | 3 x 3 | 258 x 540 x 32 | 4640 |
| batchNo_4 | Batch Normalization | - | 258 x 540 x 32 | 128 |
| activation_4 | ReLU | - | 258 x 540 x 32 | 0 |
| conv2D_5 | Convolutional | 3 x 3 | 258 x 540 x 64 | 18496 |
| activation_5 | ReLU | - | 258 x 540 x 64 | 0 |
| conv2D_6 | Convolutional + Sigmoid | 3 x 3 | 258 x 540 x 1 | 577 |

**Table 5.2**: Detailed Layer configuration of implemented CNN Stacker network

### Evaluation

By stacking original image and images generated by median filter and convolutional autoencoder as input for CNN Stacker, we achived RMSE as **0.0195**

By stacking original image and images generated by all three models as input for CNN Stacker, we achived RMSE as **0.0194** .

# Results and Analysis

The following table shows the RMSE scores obtained in the various methods we have discussed:

| **Results** | |
|---|---|
| Model/Method | RMSE |
| Raw Data | 0.14974 |
| Adaptive Thresholding | 0.07516 |
| Median Filtering | 0.05197 |
| CNN Autoencoder | 0.02319 |
| CNN Stacker (using only Median and Autoencoder) | 0.01954 |
| CNN Stacker (using all) | **0.01948** |

Our first approach, Adaptive thresholding itself reduced rmse score by 49 percent. Adaptive thresholding used on thinner structures (like some strokes of alphabets) will not lead to a overall good image. So this is not a best method, and moving on to next method, median filtering. Median filtering really worked well in our context, as our scanned documents mainly consists of text which should result as text with background white.

Later we implemented Autoencoder model which gave a RMSE value of 0.023 which almost halved the best rmse we got using traditional image enhancement techniques. Then we have developed a CNN stacker model and it performed extraordinarily with RMSE score of 0.01948. This shows how simply ensemble improves overall performance by almost 16 percent.

Out best RMSE score is **0.01948** using CNN Stacker model.

# Conclusion

## What we have learnt

Firstly, we have learnt what is a digital image, and how an image is digitized into matrix using sampling and quantization. Later we learnt about various Image enhancement techniques to enhance the image to our requirement. Later we worked on cleaning the dirty documents, we first did this using traditional techniques ( Adaptive thresholding and Median filtering ), and then we have used Autoencoders to perform the same task. Further we tried to make a better model using CNN stacker by stacking all the previous methods, which performed very well.

## Future Work

- We can use Generative Adversarial Networks like Cycle-Consistent Adversarial Networks (CycleGAN).
- Our all methods works excellent for text documents, in future we will tune our models for documents which even consists of images.

# Bibliography


[1] Rafael Gonzalez and Zahraa Faisal. *Digital Image Processing Second Edition*. June 2019.

[2] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[3] Oleg Shipitko and Anton Grigoryev. "Gaussian filtering for FPGA based image processing with High-Level Synthesis tools". In: June 2018.

[4] Dor Bank, Noam Koenigstein, and Raja Giryes. *Autoencoders*. 2021. arXiv: 2003.05991 [cs.LG].

[5] Shivam Bansal. *How Autoencoders Work: Intro and UseCases*. URL: https://www.kaggle.com/shivamb/how-autoencoders-work-intro-and-usecases.

[6] Deepak Birla. *Basics of Autoencoders*. Mar. 2019. URL: https://medium.com/@birla.deepak26/autoencoders-76bb49ae6a8f.

[7] Abien Fred Agarap. *Deep Learning using Rectified Linear Units (ReLU)*. 2019. arXiv: 1803.08375 [cs.NE].

[8] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[9] David H. Wolpert. "Stacked generalization". In: *Neural Networks* 5.2 (1992), pp. 241–259. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(05)80023-1. URL: https://www.sciencedirect.com/science/article/pii/S0893608005800231.