

FORECASTING TSUNAMI USING CNN

Submitted in partial fulfillment of the requirements
for the award of
Bachelor of Engineering degree in Computer Science and Engineering by

KATAKAM LEELAMANO HAR (Reg. No: 37110331)

KOTHAKAAPU PRANEETHREDDY (Reg. No: 37110373)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY) Accredited with Grade "A"
by NAAC JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI – 600 119

MARCH – 2021



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with "A" grade by NAAC

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai – 600 119

www.sathyabama.ac.in



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this project report is the bonafide work of **KATAKAM LEELA MANOHAR (Reg.no:37110331 and KOTHAKAAPU PRANEETHREDDY (Reg.no:37110373)** who carried out the project entitled **"FORECASTING TSUNAMI USING CNN"** under my supervision from **August 2020 to March 2021**.

Internal Guide

Dr.Jeberson retna raj,M.E., Ph.D.,

Head of the Department

Dr. S. Vigneswari,M.E., Ph.D.,

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I **KATAKAM LEELAMANO HAR** and **KOTHAKAAPU PRANEETHREDDY** hereby declare that the Project Report entitled “**FORECASTING TSUNAMI USING CNN**” is done by us under the guidance of **Dr.Jeberson retnaraj.M.E.,Ph.d.**, Department of Computer Science and Engineering at Sathyabama Institute of Science and Technology is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE:

PLACE: CHENNAI

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management** of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala, M.E., Ph.D., Dean, School of Computing, Dr. S. Vigneswari, M.E., Ph.D., and Dr. L. Lakshmanan, M.E., Ph.D., Heads of the Department** of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. Jeberson Retnaraj, M.E., Ph.D.,** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

Tsunami early warning systems have provided to be the extreme importance after the tsunami that hit Japan in March 2011. This research article presents a case study based on the tsunami detection using Bottom Pressure Rate (BPR) measurement and the post the analysis using the SAR datasets. A final decision based system using BPR has been studied to carry out the measurements of tsunami wave parameters. SAR based study has also been carried out for the post tsunami studies. Wiener filters are utilized to remove the speckle noise presents in imagery. Future scope of this work has also been proposed.

Table content

Chapter 1	Introduction	
	1.1 Domain overview	1-2
	1.2 Deep learning	2-3
Chapter 2	Literature Survey	
	2.1 Literature	4-6
	2.2 Existing system	6
	2.3 Proposed system	7
Chapter 3	Aim and Scope Of Project	
	3.1 Aim of the project	8-9
	3.2 Scope of the project	9
	3.3 Advantages of the proposed system	9-10
Chapter 4	Materials,Methods and Algorithm	
	4.1 Image Prosscening	11
	4.1.1 Purpose of Image processing	12-13
	4.1.2 Image Processing Concepts	13-14
	4.1.3 Color Images	14
	4.1.4 8-bit Color Images	14-16
	4.1.5 24-bit Color Images	16-17
	4.1.6 Color Quantization	17-18
	4.1.7 Convolution	19-20

4.1.8 Block Diagram	21
4.1.9 Grayscale Images	22
4.1.10 Image Editing Software	22-23
4.2 Distance Metrics	
4.2.1 Euclidean Distance	23
4.2.2 City Block Distance	23
4.2.3 Chessboard Distance	24
4.2.4 Dithering	24
4.2.5 Edge Detectors	25-27
4.2.6 Frequency Domain	28
4.3 Idempotence	29
4.4 Isotropic Operators	29
4.5 Kernel	29-30
4.6 Logical Operators	30-32
4.6.1 Look-up Tables and Colour maps	32-33
4.7 Masking	33-34
4.8 Mathematical Morphology	34-36
4.9 Multi-spectral filtering	36-37
4.9.1 Non-linear Filtering	37-38
4.10 Pixels	38-39
4.10.1 Pixel Values	39-40

4.11 Primary Colors	40-42
4.11.1 RGB and Colorspaces	42
4.12 Spatial Domain	42-43
4.13 Structuring Elements	43-45
4.14 Wrapping and Saturation	45-46
4.15 EDGE DETECTION METHODS	47-55
4.16 RATIO OF AVERAGES (ROA) OPERATOR	55-58
4.17 Ratio Of Exponentially Weighted	59-61
Averages operator	
4.18 Python(introduction)	61-63
Chapter 5 Result and Discussion	
Result & output	64
Chapter 6 Conclusion and Futurework	
6.1 conclusion	65
6.2 Futurework	66
REFERENCES	67
APPENDICES	
Sample Code	68-71

LIST OF FIGURES

FIG-1.1	Image processing network.	13
FIG-1.2	2×2 pixel area displaying one composite color.	18
FIG-1.3	The labels within each grid square are used to identify each square.	19
FIG-1.4	Block diagram.	21
FIG-1.5	1st and 2nd derivative of an edge illustrated in one dimension.	25
FIG-1.6	Convolution kernel for a mean filter with 3×3 neighborhood.	29
FIG-1.7	Truth-tables for AND and NAND.	31
FIG-1.8	Simple graylevel image and the corresponding surface in image space.	35
FIG-1.9	Two connected components based on 4-connectivity.	40
FIG-1.10	Some example structuring elements.	44
FIG-1.11	Mapping function for wrapping the pixel values of an 8-bit image.	45
FIG-1.12	Mapping function for saturating an 8-bit image.	46
FIG-1.13	Unidirectional mono-edge model.	56
FIG-1.14	Conditional probability of ROA for different	58

window sizes and reflectivity.

FIG-1.15 Unidirectional multi-edge model.

60

FIG-1.16 The impulse response of the exponential filter
in horizontal direction ($\alpha = 0.25$).

61

LIST OF ABBREVIATIONS

CNN- Convolutional neural network.

SAR- synthetic aperture radar.

MVM- Minimum variance method.

ASR- Adaptive sidelobe
reduction. SVA- Space variant
apodization.

HR- High-resolution.

LOG- Laplacian of Gaussian.

SGL- Simple graylevel image.

PC- Pixel Connectivity.

MSF- Multi-spectral filtering.

WS- Wrapping and Saturation.

ROEWAO- Ratio Of Exponentially Weighted Averages
operator.

CWI- Centrum Wiskunde & Informatica.

BDFL- Benevolent dictator for life.

CHAPTER 1

INTRODUCTION

1.1 Domain Overview:

The basic principle of SAR imaging is transmitting the electromagnetic waves from moving platform and then receiving the reflected signals (echo) from the target surface using single beam forming antenna. Thus an image of the target surface is reconstructed by processing this echo. The remote sensing devices use their own signals to detect any object. Even the radar can be referred as one of the remote sensing device. The imaging by means of radar is not effected by any environmental conditions such as clouds, rain, day or night. Image can be degraded while acquiring it, or during enhancement or while transmitting it. Coherence problems arise when echo signals are captured by any moving platform. As a result of this problem in synchronization noises will be added to the received images. Different types of noise that might affect the image are, Speckle noise, additive noise, Gaussian noise etc. Most important aspect that has to be considered while removing noise from the images is that, the useful information or edges of the images should not be tampered or degraded. The denoising technique should be in such a way, that it has to remove only noisy information, and thus retain necessary information of the image.

Revised Manuscript Received on February 28, 2019. S.Ranjitha, Research scholar, VTU, Dept. Of ECE, East west institute of Technology, Bengaluru-560091, India Dr. S G Hiremath, Professor and Head, Dept. Of ECE, East west institute of Technology, Bengaluru-560091, Karnataka, India We require noise free original quality image for enhanced performance and appropriate analysis of the image. Thus the concept of de-noising comes into the picture.

De-speckling or de-noising of SAR images has been performed using various methodologies which include both filters and transforms.

1.2 DEEP LEARNING

Discusses the use of modern 2D spectral estimation algorithms for synthetic aperture radar (SAR) imaging. The motivation for applying power spectrum estimation methods to SAR imaging is to improve resolution, remove sidelobe artifacts, and reduce speckle compared to what is possible with conventional Fourier transform SAR imaging techniques. This paper makes two principal contributions to the field of adaptive SAR imaging. First, it is a comprehensive comparison of 2D spectral estimation methods for SAR imaging.

It provides a synopsis of the algorithms available, discusses their relative merits for SAR imaging, and illustrates their performance on simulated and collected SAR imagery.

Some of the algorithms presented or their derivations are new, as are some of the insights into or analyses of the algorithms.

Second, this work develops multichannel variants of four related algorithms, Minimum variance method (MVM), reduced-rank MVM (RRMVM), adaptive sidelobe reduction (ASR) and space variant apodization (SVA) to estimate both reflectivity intensity and interferometric height from polarimetric displaced-aperture interferometric data.

All of these interferometric variants are new. In the interferometric contest, adaptive spectral estimation can improve the height estimates through a combination of adaptive nulling and averaging.

Examples illustrate that MVM, ASR, and SVA offer significant advantages over Fourier methods for estimating both scattering intensity and interferometric height, and allow empirical comparison of the accuracies of Fourier, MVM, ASR, and SVA interferometric height estimates.

CHAPTER 2

LITERATURE SURVEY

2.1 Literature :-

1. Title: High Density Impulse Noise Removal and Edge Detection in SAR Images based on Frequency and Spatial Domain Filtering

Author: S.Ranjitha, S G Hiremath, February 2019

Development in the science and technology, has been a great improvement in the field of space technology. Now a day's we can even capture the image from the different layers of the earth. This can be referred as Synthetic Aperture Radar (SAR) imaging.

2. Title: Edge Enhancement Algorithm Based on the Wavelet Transform for Automatic Edge Detection in SAR Images

Author: Mariví Tello Alonso, Jordi J. Mallorquí, January 2011

The second part of this paper suggests the extraction of the coastline in SAR images as a particular case of edge detection. Hence, after highlighting its practical interest, the technique that is theoretically presented in the first part of this paper is applied to real scenarios

3. Title: WAVELET SPECKLE REDUCTION FOR SAR IMAGERY BASED ON EDGE DETECTION

Author: Yingdan Wu a, Xiuxiao Yuan a, **04 May 2015**

The multi-scale and edge fusion strategy enables to detect only edge crossings and ignore the local oscillations. Then local wavelet

soft-threshold filter is applied to the area that there is no edge crossing point

4. Title: DESIGN OF AN EMBEDDED SYSTEM FOR THE DETECTION OF TSUNAMI

Author: Sidharth Das May, 2012

The embedded system is required for the Deep-ocean Assessment and Reporting of Tsunamis (DART). Each Deep-ocean Assessment and Reporting of Tsunamis (DART) gage is designed to detect and report tsunamis on its own, without instructions from land. The function is achieved with the help of a tsunami detection algorithm. The tsunami detection algorithm in the gage's software works by first estimating the amplitudes of the pressure fluctuations within the tsunami frequency band and then testing these amplitudes against a threshold value. This algorithm is based on bottom pressure recording (BPR). There are other algorithms based on tidal gauges (TG) and Wind wave gauges (WWG).

5. Title: This research is dedicated to victims and survivors of tsunamis in Indonesia and in other countries

Author: Palu-Donggala, September 2018

In the Indonesian tsunami early warning system grand design, the main focus was on development of the structural or upstream aspect. In the GITEWS program, the downstream aspect, also referred to as the 'last mile', was considered at the end phase of the project. The term 'last mile' also signifies that the focus of the system was on technocratic features, i.e. once the structural aspects had been developed successfully then work on the other end, the community response and local government capacity,

would begin. To ensure the community would take on the system, a top-down social engineering approach was applied to make sure the design of the upstream/structural components would be responded to properly at the downstream level.

2.2 Existing system:

- In the past, several multistate approaches, relying on the analysis of the information contained in the wavelet domain, have been proposed. The method proposed in this paper is based on a different way of managing multiscale data. Moreover, it does not assume any statistical distribution of the input data nor any particular type of edge, and it works exclusively in the transformed domain. Further differences of operation between these methods and the proposed technique will be highlighted throughout the paper.
- In the scope of SAR images, the study in proposes an interesting review of existing edge detection algorithms. A group of techniques is based on the evaluation of the ratio of averages over a sliding window. These methods present a low computational load, but they are highly dependent on the dimensions of the window and are not robust in noisy scenes. Several algorithms are based on the wavelet theory. For instance, the study in or proposes an edge detector based on a threshold operation of wavelet coefficients
- In The Previous Method They Are Using Only The Some Technic To Image Enhancement.

2.3 Proposed system:

In this article, we carried out a case study of existing tsunami system for the detection and monitoring. A decision based matrix has been prepared to provide the early warning issues based upon the bottom pressure rate measurements. The model was then followed up by the enhancing SAR image processing techniques with the removal of speckle noise using wiener filters.

The filters are more suitable for the post tsunami analysis.

For the future work, we could propose the improving communication network using the fiber optics microwave cables in the ocean for the selected region. The realtime monitoring can be observed based on the data acquition techniques to provide BPR to the coastal based radars.

CHAPTER 3

AIM AND SCOPE OF THE PRESENT INVESTIGATION

3.1 Aim Of The Project

Earth observation technologies, such as optical imaging and synthetic aperture radar (SAR), provide excellent means to continuously monitor ever-growing urban environments. Notably, in the case of large-scale disasters (e.g., tsunamis and earthquakes), in which a response is highly time-critical, images from both data modalities can complement each other to accurately convey the full damage condition in the disaster aftermath. However, due to several factors, such as weather and satellite coverage, which data modality will be the first available for rapid disaster response efforts is often uncertain. Hence, novel methodologies that can utilize all accessible EO datasets are essential for disaster management. In this study, we developed a global multimodal and multitemporal dataset for building damage mapping. We included building damage characteristics from three disaster types, namely, earthquakes, tsunamis, and typhoons, and considered three building damage categories. The global dataset contains high-resolution (HR) optical imagery and high-to-moderate-resolution SAR data acquired before and after each disaster.

Using this comprehensive dataset, we analyzed five data modality scenarios for damage mapping: single-mode (optical and SAR datasets), cross-modal (pre-disaster optical and post-disaster SAR datasets), and mode fusion scenarios. We defined

a damage mapping framework for semantic segmentation of damaged buildings based on a deep convolutional neural network (CNN) algorithm.

3.2 Scope Of The Project:

The scope of the project is to identify whether the tsunami's (or) earthquakes will occur or not.

We can easily identify whether the tsunami will occur or not with the help of video processing .

We can quickly identify whether the tsunami will occur or not.

3.3 Advantages Of Proposed System:

We are proposing recognition framework based on the structured three dimensional convolutional neural networks (CNNs) type of forecasting tsunami using cnn and improve the accuracy of workflow.

The proposed method for this project is to train a Deep Learning algorithm capable of classifying forecasting tsunami images and data preprocessing and visualizing the image then feature extracting to build video CNN forecasting tsunami 15 image dataset CNN model.

It is predicted that the success of the obtained results will increase if the CNN method is supported by adding extra feature extraction methods and classify successful forecasting tsunami's using CNN.

We have demonstrated the efficacy and potential of using deep CNN to images.

CHAPTER 4

MATERIALS,METHODS, AND ALGORITHM:

4.1 IMAGE PROCESSING

What is Image Processing?

Image processing is a method to convert an image into digital form and perform some operations on it, in order to get an enhanced image or to extract some useful information from it. It is a type of signal dispensation in which input is image, like video frame or photograph and output may be image or characteristics associated with that image. Usually **Image Processing** system includes treating images as two dimensional signals while applying already set signal processing methods to them.

It is among rapidly growing technologies today, with its applications in various aspects of a business. Image Processing forms core research area within engineering and computer science disciplines too.

Image processing basically includes the following three steps.

- Importing the image with optical scanner or by digital photography.
- Analyzing and manipulating the image which includes data compression and image enhancement and spotting patterns that are not to human eyes like satellite photographs.
- Output is the last stage in which result can be altered image or report that is based on image analysis.

4.1.1 Purpose of Image processing

The purpose of image processing is divided into 5 groups. They are:

1. Visualization - Observe the objects that are not visible.
2. Image sharpening and restoration - To create a better image.
3. Image retrieval - Seek for the image of interest.
4. Measurement of pattern – Measures various objects in an image.
5. Image Recognition – Distinguish the objects in an image.

Types

The two types of **methods used for Image Processing** are **Analog and Digital** Image Processing. Analog or visual techniques of image processing can be used for the hard copies like printouts and photographs. Image analysts use various fundamentals of interpretation while using these visual techniques. The image processing is not just confined to area that has to be studied but on knowledge of analyst. Association is another important tool in image processing through visual techniques. So analysts apply a combination of personal knowledge and collateral data to image processing.

Digital Processing techniques help in manipulation of the digital images by using computers. As raw data from imaging sensors from satellite platform contains deficiencies. To get over such flaws and to get originality of information, it has to undergo various phases of processing. The three general phases that all types of data have to undergo while using digital technique are Pre- processing, enhancement and display, information extraction.

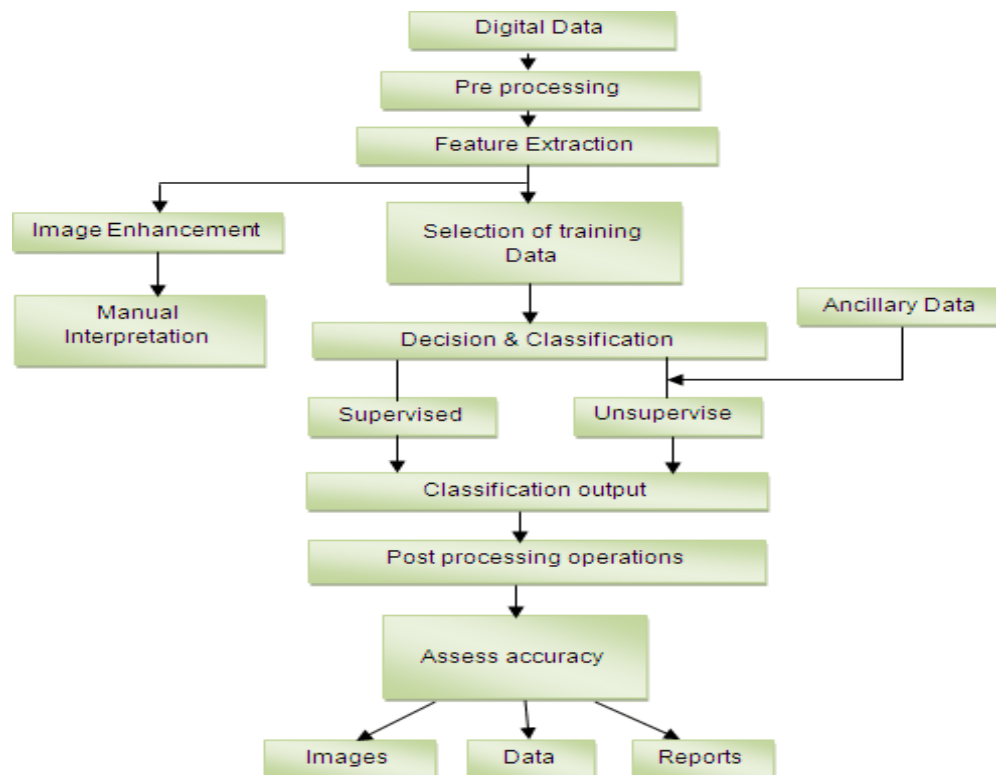


FIG 1.1 Image processing network

4.1.2 IMAGE PROCESSING CONCEPTS

Binary Images

Binary images are images whose pixels have only two possible intensity values. They are normally displayed as black and white. Numerically, the two values are often 0 for black, and either 1 or 255 for white.

Binary images are often produced by thresholding a grayscale or color image, in order to separate an object in the image from the background. The color of the object (usually white) is referred to as the foreground color. The rest (usually black) is referred to as the background color. However, depending on the image which is to be thresholded, this polarity might be inverted, in which case the object is displayed with 0 and the background is with a non-zero value.

Some morphological operators assume a certain polarity of the binary input image so that if we process an image with inverse polarity the operator will have the opposite effect. For example, if we apply a closing operator to a black text on white background, the text will be opened.

4.1.3 Color Images

It is possible to construct (almost) all visible colors by combining the three primary colors red, green and blue, because the human eye has only three different color receptors, each of them sensible to one of the three colors. Different combinations in the stimulation of the receptors enable the human eye to distinguish approximately 350000 colors. A RGB color image is a multi-spectral image with one band for each color red, green and blue, thus producing a weighted combination of the three primary colors for each pixel.

A full 24-bit color image contains one 8-bit value for each color, thus being able to display $2^{24} = 16777216$ different colors.

However, it is computationally expensive and often not necessary to use the full 24-bit image to store the color for each pixel. Therefore, the color for each pixel is often encoded in a single byte, resulting in an 8-bit color image. The process of reducing the color representation from 24-bits to 8-bits, known as color quantization, restricts the number of possible colors to 256. However, there is normally no visible difference between a 24-color image and the same image displayed with 8 bits. An 8-bit color images are based on colormaps, which are look-up tables taking the 8-bit pixel value as index and providing an output value for each color.

4.1.4 8-bit Color Images

Full RGB color requires that the intensities of three color components be specified for each and every pixel. It is common for each component intensity to be stored as an 8-bit integer, and so

each pixel requires 24 bits to completely and accurately specify its color. If this is done, then the image is known as a 24-bit color image. However there are two problems with this approach:

- Storing 24 bits for every pixel leads to very large image files that with current technology are cumbersome to store and manipulate. For instance a 24-bit 512×512 image takes up 750KB in uncompressed form.
- Many monitor displays use colormaps with 8-bit index numbers, meaning that they can only display 256 different colors at any one time. Thus it is often wasteful to store more than 256 different colors in an image anyway, since it will not be possible to display them all on screen.

Because of this, many image formats (e.g. 8-bit GIF and TIFF) use 8-bit colormaps to restrict the maximum number of different colors to 256. Using this method, it is only necessary to store an 8-bit index into the colormap for each pixel, rather than the full 24-bit color value. Thus 8-bit image formats consist of two parts: a colormap describing what colors are present in the image, and the array of index values for each pixel in the image.

When a 24-bit full color image is turned into an 8-bit image, it is usually necessary to throw away some of the colors, a process known as color quantization. This leads to some degradation in image quality, but in practice the observable effect can be quite small, and in any case, such degradation is inevitable if the image output device (e.g. screen or printer) is only capable of displaying 256 colors or less.

The use of 8-bit images with colormaps does lead to some problems in image processing. First of all, each image has to have its own colormap, and there is usually no guarantee that each image will have exactly the same colormap. Thus on 8-bit displays it is frequently impossible to correctly display two different color images that have different colormaps at the same time. Note that in practice

8-bit images often use reduced size colormaps with less than 256 colors in order to avoid this problem.

Another problem occurs when the output image from an image processing operation contains different colors to the input image or images. This can occur very easily, as for instance when two color images are added together pixel-by-pixel. Since the output image contains different colors from the input images, it ideally needs a new colormap, different from those of the input images, and this involves further color quantization which will degrade the image quality. Hence the resulting output is usually only an approximation of the desired output. Repeated image processing operations will continually degrade the image colors. And of course we still have the problem that it is not possible to display the images simultaneously with each other on the same 8-bit display.

Because of these problems it is to be expected that as computer storage and processing power become cheaper, there will be a shift away from 8-bit images and towards full 24-bit image processing.

4.1.5 24-bit Color Images

Full RGB color requires that the intensities of three color components be specified for each and every pixel. It is common for each component intensity to be stored as an 8-bit integer, and so each pixel requires 24 bits to completely and accurately specify its color. Image formats that store a full 24 bits to describe the color of each and every pixel are therefore known as 24-bit color images.

Using 24 bits to encode color information allows $2^{24} = 16777216$ different colors to be represented, and this is sufficient to cover the full range of human color perception fairly well.

The term 24-bit is also used to describe monitor displays that use 24 bits per pixel in their display memories, and which are hence capable of displaying a full range of colors.

There are also some disadvantages to using 24-bit images. Perhaps the main one is that it requires three times as much memory, disk space and processing time to store and manipulate 24-bit color images as compared to 8-bit color images. In addition, there is often not much point in being able to store all those different colors if the final output device (e.g. screen or printer) can only actually produce a fraction of them. Since it is possible to use colour maps to produce 8-bit color images that look almost as good, at the time of writing 24-bit displays are relatively little used. However it is to be expected that as the technology becomes

4.1.6 Color Quantization

Color quantization is applied when the color information of an image is to be reduced. The most common case is when a 24-bit color image is transformed into an 8-bit color image.

Two decisions have to be made:

1. which colors of the larger color set remain in the new image, and
2. how are the discarded colors mapped to the remaining ones.

The simplest way to transform a 24-bit color image into 8 bits is to assign 3 bits to red and green and 2 bits to blue (blue has only 2 bits, because of the eye's lower sensitivity to this color). This enables us to display 8 different shades of red and green and 4 of blue. However, this method can yield only poor results. For example, an image might contain different shades of blue which are all clustered around a certain value such that only one shade of blue is used in the 8-bit image and the remaining three blues are not used.

Alternatively, since 8-bit color images are displayed using a colormap, we can assign any arbitrary color to each of the 256 8-bit values and we can define a separate colormap for each image. This enables us perform a color quantization adjusted to the data contained in the image. One common approach is the popularity algorithm, which creates a histogram of all colors and retains the 256 most frequent ones. Another approach, known as the median-cut algorithm, yields even better results but also needs more computation time. This technique recursively fits a box around all colors used in the RGB colorspace which it splits at the median value of its longest side. The algorithm stops after 255 recursions. All colors in one box are mapped to the centroid of this box.

All above techniques restrict the number of displayed colors to 256. A technique of achieving additional colors is to apply a variation of half-toning used for gray scale images, thus increasing the color resolution at the cost of spatial resolution. The 256 values of the colormap are divided into four sections containing 64 different values of red, green, blue and white. As can be seen in Figure 1, a 2×2 pixel area is grouped together to represent one composite color, each of the four pixels displays either one of the primary colors or white. In this way, the number of possible colors is increased from 256 to 64^4 .

LUT

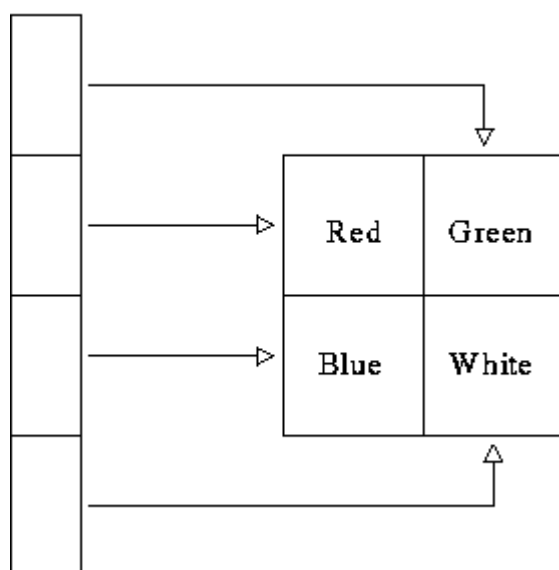


fig 1.2 2×2 pixel area displaying one composite color.

4.1.7 Convolution

Convolution is a simple mathematical operation which is fundamental to many common image processing operators. Convolution provides a way of 'multiplying together' two arrays of numbers, generally of different sizes, but of the same dimensionality, to produce a third array of numbers of the same dimensionality. This can be used in image processing to implement operators whose output pixel values are simple linear combinations of certain input pixel values.

In an image processing context, one of the input arrays is normally just a graylevel image. The second array is usually much smaller, and is also two-dimensional (although it may be just a single pixel thick), and is known as the kernel. Figure 1 shows an example image and kernel that we will use to illustrate convolution.

I₁₁	I₁₂	I₁₃	I₁₄	I₁₅	I₁₆	I₁₇	I₁₈	I₁₉
I₂₁	I₂₂	I₂₃	I₂₄	I₂₅	I₂₆	I₂₇	I₂₈	I₂₉
I₃₁	I₃₂	I₃₃	I₃₄	I₃₅	I₃₆	I₃₇	I₃₈	I₃₉
I₄₁	I₄₂	I₄₃	I₄₄	I₄₅	I₄₆	I₄₇	I₄₈	I₄₉
I₅₁	I₅₂	I₅₃	I₅₄	I₅₅	I₅₆	I₅₇	I₅₈	I₅₉
I₆₁	I₆₂	I₆₃	I₆₄	I₆₅	I₆₆	I₆₇	I₆₈	I₆₉

K₁₁	K₁₂	K₁₃
K₂₁	K₂₂	K₂₃

FIG 1.3 An example small image (left) and kernel (right) to illustrate convolution. The labels within each grid square are used to identify each square.

The convolution is performed by sliding the kernel over the image, generally starting at the top left corner, so as to move the kernel through all the positions where the kernel fits entirely within the boundaries of the image. (Note that implementations differ in what they do at the edges of images, as explained below.) Each kernel position corresponds to a single output pixel, the value of which is calculated by multiplying together the kernel value and the underlying image pixel value for each of the cells in the kernel, and then adding all these numbers together.

So, in our example, the value of the bottom right pixel in the output image will be given by:

$$O_{57} = I_{57}K_{11} + I_{58}K_{12} + I_{59}K_{13} + I_{67}K_{21} + I_{68}K_{22} + I_{69}K_{23}$$

If the image has M rows and N columns, and the kernel has m rows and n columns, then the size of the output image will have M - m + 1 rows, and N - n + 1 columns.

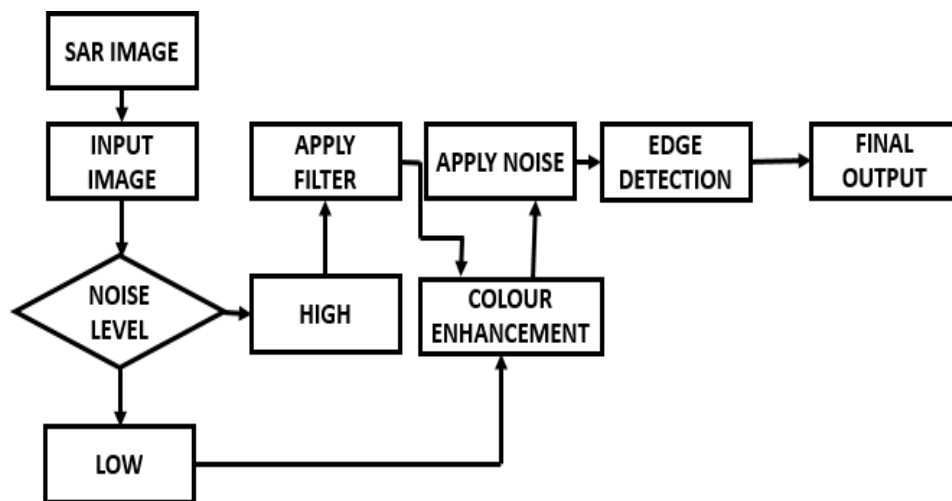
Mathematically we can write the convolution as:

$$O(i, j) = \sum_{k=1}^m \sum_{l=1}^n I(i + k - 1, j + l - 1) K(k, l)$$

where i runs from 1 to M - m + 1 and j runs from 1 to N - n + 1. Note that many implementations of convolution produce a larger output image than this because they relax the constraint that the kernel can only be moved to positions where it fits entirely within the image. Instead, these implementations typically slide the kernel to all positions where just the top left corner of the kernel is within the image. Therefore the kernel 'overlaps' the image on the bottom and right edges. One advantage of this approach is that the output image is the same size as the input image. Unfortunately, in order to calculate the output pixel values for the bottom and right edges of the image, it is necessary to invent input pixel values for places where the kernel extends off the end of the image. Typically pixel values of zero are chosen for regions outside the true image, but this can often distort the output image at these places. Therefore in

general if you are using a convolution implementation that does this, it is better to clip the image to remove these spurious regions. Removing $n - 1$ pixels from the right hand side and $m - 1$ pixels from the bottom will fix things. Convolution can be used to implement many different operators, particularly spatial filters and feature detectors. Examples include Gaussian smoothing and the Sobel edge detector.

4.1.8 Block diagram:



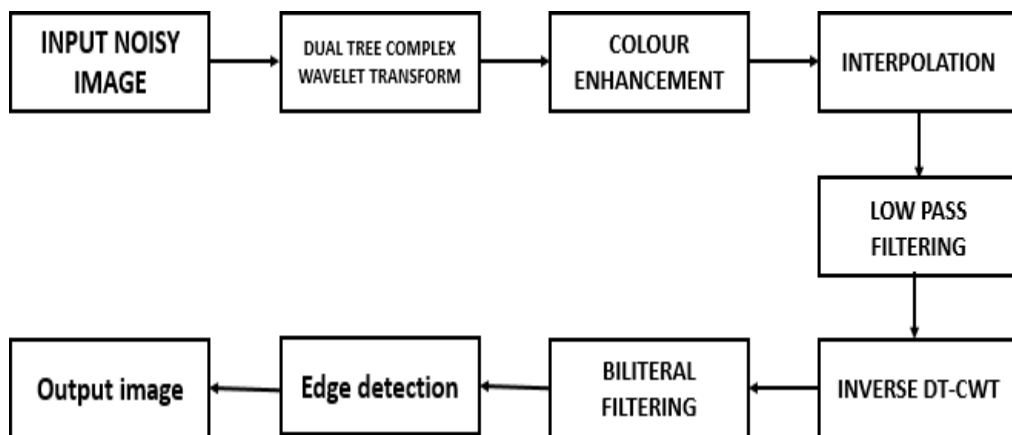


FIG 1.4 Block diagram

4.1.9 Grayscale Images

A grayscale (or graylevel) image is simply one in which the only colors are shades of gray. The reason for differentiating such images from any other sort of color image is that less information needs to be provided for each pixel. In fact a 'gray' color is one in which the red, green and blue components all have equal intensity in RGB space, and so it is only necessary to specify a single intensity value for each pixel, as opposed to the three intensities needed to specify each pixel in a full color image.

Often, the grayscale intensity is stored as an 8-bit integer giving 256 possible different shades of gray from black to white. If the levels are evenly spaced then the difference between successive graylevels is significantly better than the graylevel resolving power of the human eye.

Grayscale images are very common, in part because much of today's display and image capture hardware can only support 8-bit images. In addition, grayscale images are entirely sufficient for many tasks and so there is no need to use more complicated and harder-to-process color images.

4.1.10 Image Editing Software

There is a huge variety of software for manipulating images in various ways. Much of this software can be grouped under the heading image processing software, and the bulk of this reference is concerned with that group.

Another very important category is what we call image editing software. This group includes painting programs, graphic art packages and so on. They are often useful in conjunction with image processing software packages, in situations where direct immediate interaction with an image is the easiest way of achieving something. For instance, if a region of an image is to be masked out for subsequent image processing, it may be easiest to create the mask using an art package by directly drawing on top of the original image. The mask used in the description of the AND operator was created this way for instance. Art packages also often allow the user to move sections of the images around and brighten or darken selected regions interactively. Few dedicated image processing packages offer the same flexibility and ease of use in this respect.

4.2 Distance Metrics

It is often useful in image processing to be able to calculate the distance between two pixels in an image, but this is not as straightforward as it seems. The presence of the pixel grid makes several so-called distance metrics possible which often give different answers to each other for the distance between the same pair of points. We consider the three most important ones.

4.2.1 Euclidean Distance

This is the familiar straight line distance that most people are familiar with. If the two pixels that we are considering have coordinates (x_1, y_1) and (x_2, y_2) , then the Euclidean distance is given by:

$$D_{Euclid} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

4.2.2 City Block Distance

Also known as the Manhattan distance. This metric assumes that in going from one pixel to the other it is only possible to travel directly along pixel grid lines. Diagonal moves are not allowed. Therefore the 'city block' distance is given by:

$$D_{City} = |x_2 - x_1| + |y_2 - y_1|$$

4.2.3 Chessboard Distance

This metric assumes that you can make moves on the pixel grid as if you were a King making moves in chess, *i.e.* a diagonal move counts the same as a horizontal move. This means that the metric is given by:

$$D_{Chess} = \max(|x_2 - x_1|, |y_2 - y_1|)$$

Note that the last two metrics are usually much faster to compute than the Euclidean metric and so are sometimes used where speed is critical but accuracy is not too important.

4.2.4 Dithering

Dithering is an image display technique that is useful for overcoming limited display resources. The word dither refers to a random or semi-random perturbation of the pixel values.

Two applications of this techniques are particularly useful:

- Low quantization display: When images are quantized to a few bits (e.g. 3) then only a limited number of gray levels are used in the display of the image. If the scene is smoothly shaded, then the image display will generate rather distinct boundaries around the edges of image regions when the original scene intensity moves from one quantization level to the next. To eliminate this effect, one dithering technique adds random noise (with a small range of values) to the input signal before quantization into the output range. This randomizes the quantization of the pixels at the original quantization boundary, and thus pixels make a more gradual transition from neighborhoods containing 100% of the first quantization level to neighborhoods containing 100% of the second quantization level.

- Limited color display: When fewer colors are able to be displayed (e.g. 256) than are present in the input image (e.g. 24 bit color), then patterns of adjacent pixels are used to simulate the appearance of the unrepresented colors.

4.2.5 Edge Detectors

Edges are places in the image with strong intensity contrast. Since edges often occur at image locations representing object boundaries, edge detection is extensively used in image segmentation when we want to divide the image into areas corresponding to different objects. Representing an image by its edges has the further advantage that the amount of data is reduced significantly while retaining most of the image information.

Since edges consist of mainly high frequencies, we can, in theory, detect edges by applying a highpass frequency filter in the Fourier domain or by convolving the image with an appropriate kernel in the

spatial domain. In practice, edge detection is performed in the spatial domain, because it is computationally less expensive and often yields better results.

Since edges correspond to strong illumination gradients, we can highlight them by calculating the derivatives of the image. This is illustrated for the one-dimensional case in Figure 1.

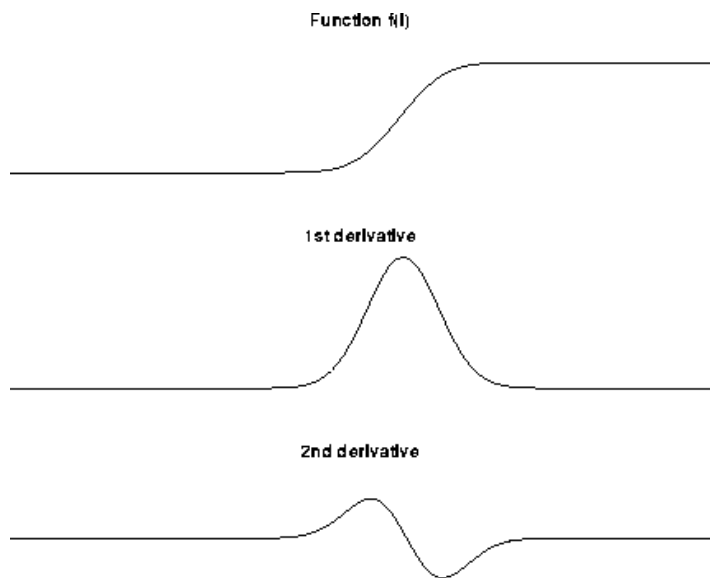


FIG-1.5 1st and 2nd derivative of an edge illustrated in one dimension.

We can see that the position of the edge can be estimated with the maximum of the 1st derivative or with the zero-crossing of the 2nd derivative. Therefore we want to find a technique to calculate the derivative of a two-dimensional image. For a discrete one-dimensional function $f(i)$, the first derivative can be approximated by

$$\frac{df(i)}{d(i)} = f(i+1) - f(i)$$

Calculating this formula is equivalent to convolving the function with $[-1 \ 1]$. Similarly the 2nd derivative can be estimated by convolving $f(i)$ with $[1 \ -2 \ 1]$.

Different edge detection kernels which are based on the above formula enable us to calculate either the 1st or the 2nd derivative of a two-dimensional image. There are two common approaches to estimate the 1st derivative in a two-dimensional image, Prewitt compass edge detection and gradient edge detection.

Prewitt compass edge detection involves convolving the image with a set of (usually 8) kernels, each of which is sensitive to a different edge orientation. The kernel producing the maximum response at a pixel location determines the edge magnitude and orientation. Different sets of kernels might be used: examples include Prewitt, Sobel, Kirsch and Robinson kernels.

Gradient edge detection is the second and more widely used technique. Here, the image is convolved with only two kernels, one estimating the gradient in the x-direction, G_x , the other the gradient in the y-direction, G_y . The absolute gradient magnitude is then given by

$$|G| = \sqrt{G_x^2 + G_y^2}$$

and is often approximated with

$$|G| = |G_x| + |G_y|$$

In many implementations, the gradient magnitude is the only output of a gradient edge detector, however the edge orientation might be calculated with

The most common kernels used for the gradient edge detector are the Sobel, Roberts Cross and Prewitt operators.

After having calculated the magnitude of the 1st derivative, we now have to identify those pixels corresponding to an edge. The easiest

way is to threshold the gradient image, assuming that all pixels having a local gradient above the threshold must represent an edge. An alternative technique is to look for local maxima in the gradient image, thus producing one pixel wide edges. A more sophisticated technique is used by the Canny edge detector. It first applies a gradient edge detector to the image and then finds the edge pixels using non-maximal suppression and hysteresis tracking.

An operator based on the 2nd derivative of an image is the Marr edge detector, also known as zero crossing detector. Here, the 2nd derivative is calculated using a Laplacian of Gaussian (LoG) filter. The Laplacian has the advantage that it is an isotropic measure of the 2nd derivative of an image, i.e. the edge magnitude is obtained independently from the edge orientation by convolving the image with only one kernel. The edge positions are then given by the zero-crossings in the LoG image. The scale of the edges which are to be detected can be controlled by changing the variance of the Gaussian.

A general problem for edge detection is its sensitivity to noise, the reason being that calculating the derivative in the spatial domain corresponds to accentuating high frequencies and hence magnifying noise. This problem is addressed in the Canny and Marr operators by convolving the image with a smoothing operator (Gaussian) before calculating the derivative.

4.2.6 Frequency Domain

For simplicity, assume that the image I being considered is formed by projection from scene S (which might be a two- or three-dimensional scene, etc.).

The frequency domain is a space in which each image value at image position F represents the amount that the intensity values in image I vary over a specific distance related to F . In the frequency domain, changes in image position correspond to changes in the spatial

frequency, (or the rate at which image intensity values) are changing in the spatial domain image I.

For example, suppose that there is the value 20 at the point that represents the frequency 0.1 (or 1 period every 10 pixels). This means that in the corresponding spatial domain image I the intensity values vary from dark to light and back to dark over a distance of 10 pixels, and that the contrast between the lightest and darkest is 40 gray levels (2 times 20).

The spatial frequency domain is interesting because: 1) it may make explicit periodic relationships in the spatial domain, and 2) some image processing operators are more efficient or indeed only practical when applied in the frequency domain.

In most cases, the Fourier Transform is used to convert images from the spatial domain into the frequency domain and vice-versa.

A related term used in this context is spatial frequency, which refers to the (inverse of the) periodicity with which the image intensity values change. Image features with high spatial frequency (such as edges) are those that change greatly in intensity over short image distances.

4.3 Idempotence

Some operators have the special property that applying them more than once to the same image produces no further change after the first application. Such operators are said to be idempotent. Examples include the morphological operators opening and closing.

4.4 Isotropic Operators

An isotropic operator in an image processing context is one which applies equally well in all directions in an image, with no particular sensitivity or bias towards one particular set of directions (e.g. compass directions). A typical example is the zero crossing edge detector which responds equally well to edges in any orientation. Another example is Gaussian smoothing. It should be borne in mind that although an operator might be isotropic in theory, the actual implementation of it for use on a discrete pixel grid may not be perfectly isotropic. An example of this is a Gaussian smoothing filter with very small standard deviation on a square grid.

4.5 Kernel

A kernel is a (usually) smallish matrix of numbers that is used in image convolutions. Differently sized kernels containing different patterns of numbers give rise to different results under convolution. For instance, Figure 1 shows a 3×3 kernel that implements a mean filter.

1	1	1
1	1	1
1	1	1

Set of coordinate points =

{ (-1, -1), (0, -1), (1, -1),
 (-1, 0), (0, 0), (1, 0),
 (-1, 1), (0, 1), (1, 1) }

FIG-1.6 Convolution kernel for a mean filter with 3×3 neighborhood.

The word 'kernel' is also commonly used as a synonym for 'structuring element', which is a similar object used in mathematical morphology. A structuring element differs from a

kernel in that it also has a specified origin. This sense of the word 'kernel' is not used in HIPR.

4.6 Logical Operators

Logical operators are generally derived from Boolean algebra, which is a mathematical way of manipulating the truth values of concepts in an abstract way without bothering about what the concepts actually mean. The truth value of a concept in Boolean value can have just one of two possible values: true or false. Boolean algebra allows you to represent things like:

The block is both red and large

by something like:

$A \text{ AND } B$

where A represents 'The block is red', and B represents 'The block is large'. Now each of these sub-phrases has its own truth value in any given situation: each sub-phrase is either true or false. Moreover, the entire composite phrase also has a truth value: it is true if both of the sub-phrases are true, and false in any other case. We can write this AND combination rule (and its dual operation NAND) using a truth-table as shown in Figure 1, in which we conventionally represent true by 1, and false by zero.

A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

AND

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

NAND

FIG-1.7 Truth-tables for AND and NAND

The left hand table shows each of the possible combinations of truth values of A and B, and the the resulting truth value of A AND B. Similar truth-tables can be set up for the other logical operators: NAND, OR, NOR, XOR, XNOR and NOT.

Turning now to an image processing context, the pixel values in a binary image, which are either 0 or 1, can be interpreted as truth values as above. Using this convention we can carry out logical operations on images simply by applying the truth-table combination rules to the pixel values from a pair of input images (or a single input image in the case of NOT). Normally, corresponding pixels from each of two identically sized binary input images are compared to produce the output image, which is another binary image of the same size. As with other image arithmetic operations, it is also possible to logically combine a single input image with a constant logical value, in which case each pixel in the input image is compared to the same constant in order to produce the corresponding output pixel. See the individual logical operator descriptions for examples of these operations.

Logical operations can also be carried out on images with integer pixel values. In this extension the logical operations are normally carried out in bitwise fashion on binary representations of those integers, comparing corresponding bits with corresponding bits to

produce the output pixel value. For instance, suppose that we wish to XOR the integers 47 and 255 together using 8-bit integers. 47 is 00101111 in binary and 255 is 11111111. XORing these together in bitwise fashion, we have 11010000 in binary or 208 in decimal.

Note that not all implementations of logical operators work in such bitwise fashion. For instance some will treat zero as false and any non-zero value as true and will then apply the conventional 1-bit logical functions to derive the output image. The output may be a simple binary image itself, or it may be a graylevel image formed perhaps by multiplying what would be the binary output image (containing 0's and 1's) with one of the input images.

4.6.1 Look-up Tables and Colour maps

Look-Up Tables or LUTs are fundamental to many aspects of image processing. An LUT is simply a table of cross-references linking index numbers to output values. The most common use is to determine the colors and intensity values with which a particular image will be displayed, and in this context the LUT is often called simply a colormap.

The idea behind the colormap is that instead of storing a definite color for each pixel in an image, for instance in 24-bit RGB format, each pixel's value is instead treated as an index number into the colormap. When the image is to be displayed or otherwise processed, the colormap is used to look up the actual colors corresponding to each index number. Typically, the output values stored in the LUT would be RGB color values.

There are two main advantages to doing things this way. Firstly, the index number can be made to use fewer bits than the output value in order to save storage space. For instance an 8-bit index number can be used to look up a 24-bit RGB color value in the LUT. Since only the 8-bit index number needs to be stored for each pixel, such 8-bit color images take up less space than a full 24-bit image of the same size. Of course the image can only contain 256 different colors (the

number of entries in an 8-bit LUT), but this is sufficient for many applications and usually the observable image degradation is small.

Secondly the use of a color table allows the user to experiment easily with different color labeling schemes for an image.

One disadvantage of using a colormap is that it introduces additional complexity into an image format. It is usually necessary for each image to carry around its own colormap, and this LUT must be continually consulted whenever the image is displayed or processed.

Another problem is that in order to convert from a full color image to (say) an 8-bit color image using a color image, it is usually necessary to throw away many of the original colors, a process known as color quantization. This process is lossy, and hence the image quality is degraded during the quantization process. Additionally, when performing further image processing on such images, it is frequently necessary to generate a new colormap for the new images, which involves further color quantization, and hence further image degradation.

As well as their use in colormaps, LUTs are often used to remap the pixel values within an image. This is the basis of many common image processing point operations such as thresholding, gamma correction and contrast stretching. The process is often referred to as anamorphosis.

4.7 Masking

A mask is a binary image consisting of zero- and non-zero values. If a mask is applied to another binary or to a grayscale image of the same size, all pixels which are zero in the mask are set to zero in the output image. All others remain unchanged.

Masking can be implemented either using pixel multiplication or logical AND, the latter in general being faster.

Masking is often used to restrict a point or arithmetic operator to an area defined by the mask. We can, for example, accomplish this by first masking the desired area in the input image and processing it with the operator, then masking the original input image with the inverted mask to obtain the unprocessed area of the image and finally recombining the two partial images using image addition. An example can be seen in the worksheet on the logical AND operator. In some image processing packages, a mask can directly be defined as an optional input to a point operator, so that automatically the operator is only applied to the pixels defined by the mask .

4.8 Mathematical Morphology

The field of mathematical morphology contributes a wide range of operators to image processing, all based around a few simple mathematical concepts from set theory. The operators are particularly useful for the analysis of binary images and common usages include edge detection, noise removal, image enhancement and image segmentation.

The two most basic operations in mathematical morphology are erosion and dilation. Both of these operators take two pieces of data as input: an image to be eroded or dilated, and a structuring element (also known as a kernel). The two pieces of input data are each treated as representing sets of coordinates in a way that is slightly different for binary and grayscale images.

For a binary image, white pixels are normally taken to represent foreground regions, while black pixels denote background. (Note that in some implementations this convention is reversed, and so it is very important to set up input images with the correct polarity for the implementation being used). Then the set of coordinates corresponding to that image is simply the set of two-dimensional Euclidean coordinates of all the foreground pixels in the image, with

an origin normally taken in one of the corners so that all coordinates have positive elements.

For a grayscale image, the intensity value is taken to represent height above a base plane, so that the grayscale image represents a surface in three-dimensional Euclidean space. Figure 1 shows such a surface. Then the set of coordinates associated with this image surface is simply the set of three-dimensional Euclidean coordinates of all the points within this surface and also all points below the surface, down to the base plane. Note that even when we are only considering points with integer coordinates, this is a lot of points, so usually algorithms are employed that do not need to consider all the points.

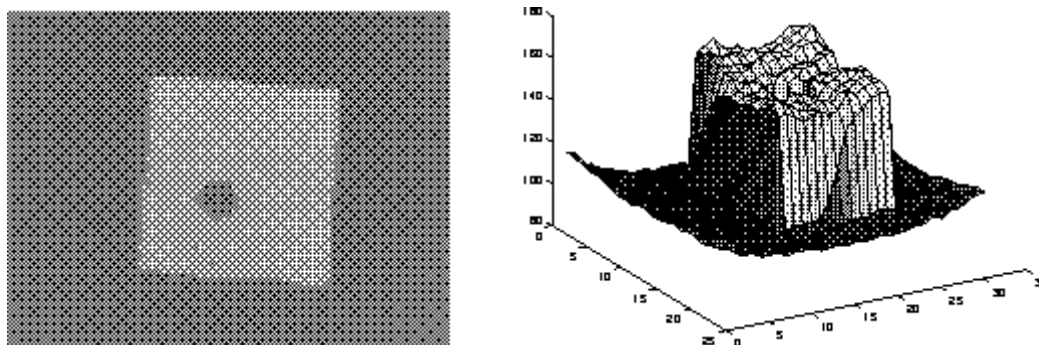


FIG-1.8 Simple graylevel image and the corresponding surface in image space

The structuring element is already just a set of point coordinates (although it is often represented as a binary image). It differs from the input image coordinate set in that it is normally much smaller, and its coordinate origin is often not in a corner, so that some coordinate elements will have negative values. Note that in many implementations of morphological operators, the structuring element is assumed to be a particular shape (e.g. a 3×3 square) and so is hardwired into the algorithm.

Binary morphology can be seen as a special case of graylevel morphology in which the input image has only two graylevels at values 0 and 1.

Erosion and dilation work (at least conceptually) by translating the structuring element to various points in the input image, and examining the intersection between the translated kernel coordinates and the input image coordinates. For instance, in the case of erosion, the output coordinate set consists of just those points to which the origin of the structuring element can be translated, while the element still remains entirely 'within' the input image.

Virtually all other mathematical morphology operators can be defined in terms of combinations of erosion and dilation along with set operators such as intersection and union. Some of the more important are opening, closing and skeletonization.

4.9 Multi-spectral filtering

A multi-spectral image is a collection of several monochrome images of the same scene, each of them taken with a different sensor. Each image is referred to as a band. A well known multi-spectral (or multi-band image) is a RGB color image, consisting of a red, a green and a blue image, each of them taken with a sensor sensitive to a different wavelength. In image processing, multi-spectral images are most commonly used for Remote Sensing applications. Satellites usually take several images from frequency bands in the visual and non-visual range. Landsat 5, for example, produces 7 band images with the wavelength of the bands being between 450 and 1250 nm.

All the standard single-band image processing operators can also be applied to multi-spectral images by processing each band separately. For example, a multi-spectral image can be edge detected by finding the edges in each band and then ORing the three edge images together. However, we would obtain more reliable edges, if we

associate a pixel with an edge based on its properties in all three bands and not only in one.

To fully exploit the additional information which is contained in the multiple bands, we should consider the images as one multi-spectral image rather than as a set of monochrome graylevel images. For an image with k bands, we can then describe the brightness of each pixel as a point in a k -dimensional space represented by a vector of length k .

Special techniques exist to process multi-spectral images. For example, to classify a pixel as belonging to one particular region, its intensities in the different bands are said to form a feature vector describing its location in the k -dimensional feature space. The simplest way to define a class is to choose a upper and lower threshold for each band, thus producing a k -dimensional 'hyper-cube' in the feature space. Only if the feature vector of a pixel points to a location within this cube, is the pixel classified as belonging to this class. A more sophisticated classification method is described in the corresponding worksheet.

The disadvantage of multi-spectral images is that, since we have to process additional data, the required computation time and memory increase significantly. However, since the speed of the hardware will increase and the costs for memory will decrease in the future, it can be expected that multi-spectral images will become more important in many fields of computer vision.

4.9.1 Non-linear Filtering

Suppose that an image processing operator F acting on two input images A and B produces output images C and D respectively. If the operator F is linear, then

$$F(a \times A + b \times B) = a \times C + b \times D$$

where a and b are constants. In practice, this means that each pixel in the output of a linear operator is the weighted sum of a set of pixels in the input image.

By contrast, non-linear operators are all the other operators. For example, the threshold operator is non-linear, because individually, corresponding pixels in the two images A and B may be below the threshold, whereas the pixel obtained by adding A and B may be above threshold. Similarly, an absolute value operation is non-linear:

$$|-1 + 1| \neq |-1| + |1|$$

as is the exponential operator:

$$\exp(1 + 1) \neq \exp(1) + \exp(1)$$

4.10 Pixels

In order for any digital computer processing to be carried out on an image, it must first be stored within the computer in a suitable form that can be manipulated by a computer program. The most practical way of doing this is to divide the image up into a collection of discrete (and usually small) cells, which are known as pixels. Most commonly, the image is divided up into a rectangular grid of pixels, so that each pixel is itself a small rectangle. Once this has been done, each pixel is given a pixel value that represents the color of that pixel. It is assumed that the whole pixel is the same color, and so any color variation that did exist within the area of the pixel before the image was discretized is lost. However, if the area of each pixel is very small, then the discrete nature of the image is often not visible to the human eye.

Other pixel shapes and formations can be used, most notably the hexagonal grid, in which each pixel is a small hexagon. This has some advantages in image processing, including the fact that pixel connectivity is less ambiguously defined than with a square grid, but hexagonal grids are not widely used. Part of the reason is that many

image capture systems (e.g. most CCD cameras and scanners) intrinsically discretize the captured image into a rectangular grid in the first instance.

4.10.1 Pixel Connectivity

The notation of pixel connectivity describes a relation between two or more pixels. For two pixels to be connected they have to fulfill certain conditions on the pixel brightness and spatial adjacency.

First, in order for two pixels to be considered connected, their pixel values must both be from the same set of values V . For a grayscale image, V might be any range of graylevels, e.g. $V=\{22,23,\dots,40\}$, for a binary image we simply have $V=\{1\}$.

To formulate the adjacency criterion for connectivity, we first introduce the notation of neighborhood. For a pixel p with the coordinates (x,y) the set of pixels given by:

$$N_4(p) = \{(x+1, y), (x-1, y), (x, y+1), (x, y-1)\}$$

is called its 4-neighbors. Its 8-neighbors are defined as

$$N_8(p) = N_4 \cup \{(x+1, y+1), (x+1, y-1), (x-1, y+1), (x-1, y-1)\}$$

From this we can infer the definition for 4- and 8-connectivity:

Two pixels p and q , both having values from a set V are 4-connected if q is from the set $N_4(p)$ and 8-connected if q is from $N_8(p)$.

General connectivity can either be based on 4- or 8-connectivity; for the following discussion we use 4-connectivity.

A pixel p is connected to a pixel q if p is 4-connected to q or if p is 4-connected to a third pixel which itself is connected to q . Or, in other words, two pixels q and p are connected if there is a path from p and q on which each pixel is 4-connected to the next one.

A set of pixels in an image which are all connected to each other is called a connected component. Finding all connected components in an image and marking each of them with a distinctive label is called connected component labeling.

An example of a binary image with two connected components which are based on 4-connectivity can be seen in Figure 1. If the connectivity were based on 8-neighbors, the two connected components would merge into one.

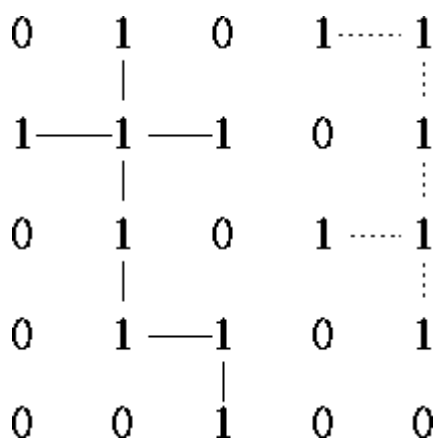


FIG-1.9 Two connected components based on 4-connectivity.

4.10.1 Pixel Values

Each of the pixels that represents an image stored inside a computer has a pixel value which describes how bright that pixel is, and/or what color it should be. In the simplest case of binary images, the pixel value is a 1-bit number indicating either foreground or background. For a grayscale images, the pixel value is a single number that represents the brightness of the pixel. The most common pixel format is the byte image, where this number is stored as an 8-bit integer giving a range of possible values from 0 to 255. Typically zero is taken to be black, and 255 is taken to be white. Values in between make up the different shades of gray.

To represent color images, separate red, green and blue components must be specified for each pixel (assuming an RGB colorspace), and so the pixel 'value' is actually a vector of three numbers. Often the three different components are stored as three separate 'grayscale' images known as color planes (one for each of red, green and blue), which have to be recombined when displaying or processing.

Multi-spectral images can contain even more than three components for each pixel, and by extension these are stored in the same kind of way, as a vector pixel value, or as separate color planes.

The actual grayscale or color component intensities for each pixel may not actually be stored explicitly. Often, all that is stored for each pixel is an index into a colormap in which the actual intensity or colors can be looked up.

Although simple 8-bit integers or vectors of 8-bit integers are the most common sorts of pixel values used, some image formats support different types of value, for instance 32-bit signed integers or floating point values. Such values are extremely useful in image processing as they allow processing to be carried out on the image where the resulting pixel values are not necessarily 8-bit integers. If this approach is used then it is usually necessary to set up a colormap which relates particular ranges of pixel values to particular displayed colors.

4.11 Primary Colors

It is a useful fact that the huge variety of colors that can be perceived by humans can all be produced simply by adding together appropriate amounts of red, blue and green colors. These colors are known as the primary colors. Thus in most image processing applications, colors are represented by specifying separate intensity values for red, green and blue components. This representation is commonly referred to as RGB.

The primary color phenomenon results from the fact that humans have three different sorts of color receptors in their retinas which are each most sensitive to different visible light wavelengths.

The primary colors used in painting (red, yellow and blue) are different. When paints are mixed, the 'addition' of a new color paint actually subtracts wavelengths from the reflected visible light.

4.11.1 RGB and Colorspaces

A color perceived by the human eye can be defined by a linear combination of the three primary colors red, green and blue. These three colors form the basis for the RGB-colorspace. Hence, each perceivable color can be defined by a vector in the three-dimensional colorspace. The intensity is given by the length of the vector, and the actual color by the two angles describing the orientation of the vector in the colorspace.

The RGB-space can also be transformed into other coordinate systems, which might be more useful for some applications. One common basis for the color space is IHS. In this coordinate system, a color is described by its intensity, hue (average wavelength) and saturation (the amount of white in the color). This color space makes it easier to directly derive the intensity and color of perceived light and is therefore more likely to be used by human beings.

4.12 Spatial Domain

For simplicity, assume that the image I being considered is formed by projection from scene S (which might be a two- or three-dimensional scene, etc.).

The spatial domain is the normal image space, in which a change in position in I directly projects to a change in position in S . Distances in I (in pixels) correspond to real distances (e.g. in meters) in S .

This concept is used most often when discussing the frequency with which image values change, that is, over how many pixels does a cycle of periodically repeating intensity variations occur. One would refer to the number of pixels over which a pattern repeats (its periodicity) in the spatial domain.

In most cases, the Fourier Transform will be used to convert images from the spatial domain into the frequency domain.

A related term used in this context is spatial frequency, which refers to the (inverse of the) periodicity with which the image intensity values change. Image features with high spatial frequency (such as edges) are those that change greatly in intensity over short image distances.

Another term used in this context is spatial derivative, which refers to how much the image intensity values change per change in image position.

4.13 Structuring Elements

The field of mathematical morphology provides a number of important image processing operations, including erosion, dilation, opening and closing. All these morphological operators take two pieces of data as input. One is the input image, which may be either binary or grayscale for most of the operators. The other is the structuring element. It is this that determines the precise details of the effect of the operator on the image.

The structuring element is sometimes called the kernel, but we reserve that term for the similar objects used in convolutions.

The structuring element consists of a pattern specified as the coordinates of a number of discrete points relative to some origin. Normally cartesian coordinates are used and so a convenient way of representing the element is as a small image on a rectangular grid. Figure 1 shows a number of different structuring elements of various sizes. In each case the origin is marked by a ring around that point. The origin does not have to be in the center of the structuring element, but often it is. As suggested by the figure, structuring elements that fit into a 3×3 grid with its origin at the center are the most commonly seen type.

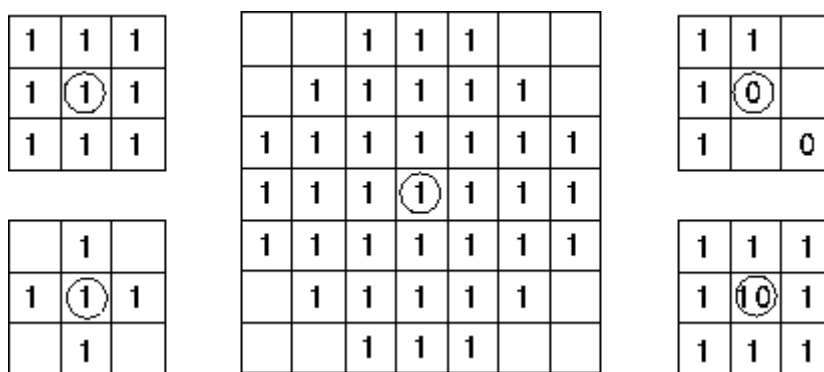


FIG-1.10 Some example structuring elements.

Note that each point in the structuring element may have a value. In the simplest structuring elements used with binary images for operations such as erosion, the elements only have one value, conveniently represented as a one. More complicated elements, such as those used with thinning or grayscale morphological operations, may have other pixel values.

An important point to note is that although a rectangular grid is used to represent the structuring element, not every point in that grid is part of the structuring element in general. Hence the elements shown in Figure 1 contain some blanks. In many texts, these blanks are represented as zeros, but this can be confusing and so we avoid it here.

When a morphological operation is carried out, the origin of the structuring element is typically translated to each pixel position in the image in turn, and then the points within the translated structuring element are compared with the underlying image pixel values. The details of this comparison, and the effect of the outcome depend on which morphological operator is being used.

4.14 Wrapping and Saturation

If an image is represented in a byte or integer pixel format, the maximum pixel value is limited by the number of bits used for the representation, e.g. the pixel values of a 8-bit image are limited to 255.

However, many image processing operations produce output values which are likely to exceed the given maximum value. In such cases, we have to decide how to handle this pixel overflow.

One possibility is to wrap around the overflowing pixel values. This means that if a value is greater than the possible maximum, we subtract the pixel value range so that the value starts again from the possible minimum value. Figure 1 shows the mapping function for wrapping the output values of some operation into an 8-bit format.

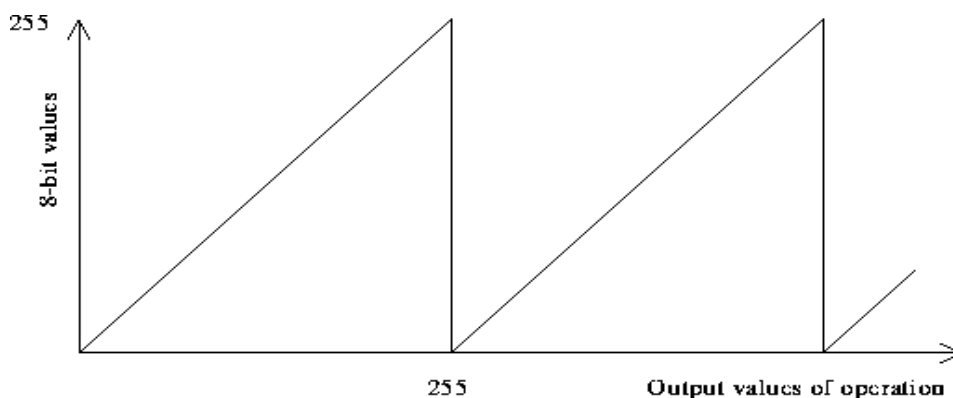


FIG-1.11 Mapping function for wrapping the pixel values of an 8-bit image.

Another possibility is to set all overflowing pixels to the maximum possible values --- an effect known as saturation. The corresponding mapping function for an 8-bit image can be seen in Figure 2.

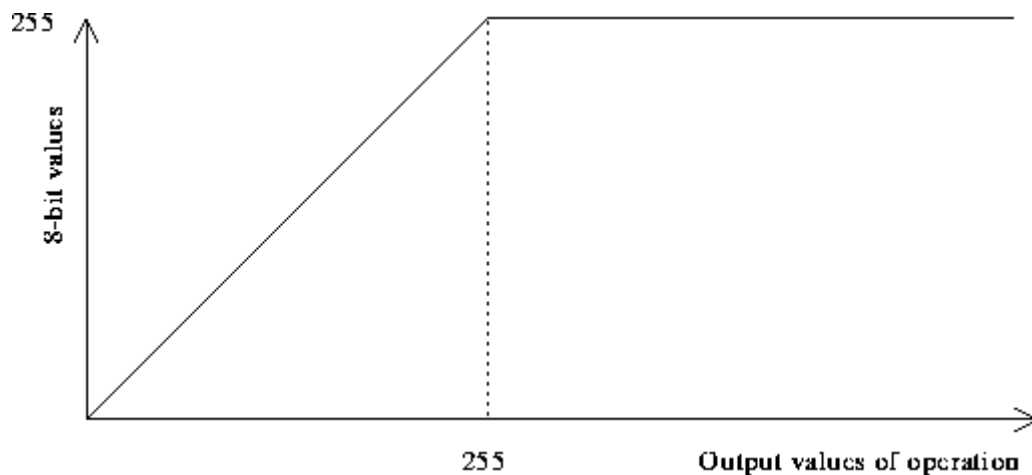


FIG-1.12 Mapping function for saturating an 8-bit image.

If only a few pixels in the image exceed the maximum value it is often better to apply the latter technique, especially if we use the image for display purposes. However, by setting all overflowing pixels to the same value we lose an essential amount of information. In the worst case, when all pixels exceed the maximum value, this would lead to an image of constant pixel values. Wrapping around overflowing pixel retains the differences between values. On the other hand, it might cause the problem that pixel values passing the maximum 'jump' from the maximum to the minimum value. Examples for both techniques can be seen in the worksheets of various point operators.

If possible, it is easiest to change the image format, for example to float format, so that all pixel values can be represented. However,

we should keep in mind that this implies an increase in processing time and memory.

PROCESS DETAILS:

4.15 EDGE DETECTION METHODS

The estimation of the boundary location of an object is an important subject in image processing and edge detection is a fundamental issue in SAR image analysis. Since edge detection has been studied, several techniques for edge enhancement are proposed. Robust edge detection techniques are essentially based on the following two steps: edge enhancement and decision. Unlike optical images, in SAR data, which is highly heterogeneous, a robust edge enhancement phase is critical in providing acceptable detection rates. This phase is usually performed through techniques that are related to derivation, namely, simple differences, Sobel filter, Roberts filter, Prewitt filter, and morphological gradients etc., possibly combined with smoothing. However, there are three problems by the use of derivative as edge detector: i) false edges produced by noise are enhanced; ii) ramp edges produce lower derivative magnitude; iii) the location of the ramp edges are not accurate. Several edge detection methods as proposed by Marr-Hildreth, Canny and Perona-Malik provide tools to overcome the edge detection problems by applying the derivative to edge detection. Most of these methods apply a smooth gaussian operator in order to suppress the noisy edges and then estimate the location of the edges by detection of the zero-crossing on the second derivative of an image. Edge detectors proposed by Petrou-Kitter and Zang et. al. are based on ramp edges model. Petrou-Kitter propose a ramp edge

model to an optimal edge detector. D. Wang proposed a multiscale gradient algorithm that enhances the morphological gradient of ramp edges however the edge of thin objects can be lost. Due to existence of speckle, edge detection in SAR images is a very challenging task and the above methods provide limited efficiency in SAR applications due to the presence of a speckle which is a multiplicative noiselike pattern. This paper aims at presenting a novel method for edge enhancement in SAR images based on the exploitation of the information provided by the wavelet coefficients. The usual edge detectors, successful with incoherent images, yield poor results when applied to radar images, especially those with a small number of looks. Their performance degrades seriously because they are very sensitive to the speckle noise.

Edge detection plays a main role in various areas of image processing, and it is one of the most commonly used operations in image analysis. Edges have been defined as pixel intensity discontinuities within an image which form the outline of an object and the boundary between an object and the background. Also, edge indicates the boundary between overlapping objects. This means all of the objects can be located if the edges in an image can be identified accurately. There are many ways to perform edge detection. The majority of different methodological approach of the edge detection may be grouped into two categories, gradient and Laplacian. The gradient method detects the edges by looking for the maximum and minimum in the first derivative of the image. Examples of gradient-based edge detectors are Roberts, Prewitt, and Sobel operators. Mathematically, each operator consists of a pair of convolution masks. The masks can be applied to the input image separately, to produce separate measurements of the gradient component (call these G_x and G_y). These can be then combined

together to find the absolute magnitude of the gradient at each pixel and the orientation of that gradient. Gradient operation is an effective detector for sharp edges where the pixel gray levels change over space very rapidly. But when the gray levels change slowly from dark to bright, the gradient operation will produce a very wide edge. It is helpful in this case to consider using the Laplace operation. The Laplacian method searches for zero crossings in the second derivative of the image to find edges. Because when the first derivative is at a maximum, the second derivative is zero. The Laplacian is often applied to an image that has first been smoothed in order to reduce its sensitivity to noise, but it still behaves poorly at corners. Canny later proposed a new detector. His attention was to enhance the many edge detectors already out at the time he started his work. He followed a list of criteria to improve current methods of edge detection and he was very successful in achieving his goal and his ideas and methods can be found in his paper.

The edge representation of an image significantly reduces the quantity of data to be processed, yet it retains essential information regarding the shapes of objects in the scene. This explanation of an image is easy to incorporate into a large amount of object recognition algorithms used in computer vision along with other image processing applications. The major property of the edge detection technique is its ability to extract the exact edge line with good orientation as well as more literature about edge detection has been available in the past three decades. On the other hand, there is not yet any common performance directory to judge the performance of the edge detection techniques. The performance of an edge detection techniques are always judged personally and separately dependent to its application.

Edge detection is a fundamental tool for image segmentation. Edge detection methods transform original images into edge images benefits from the changes of grey tones in the image. In image processing especially in computer vision, the edge detection treats the localization of important variations of a gray level image and the detection of the physical and geometrical properties of objects of the scene. It is a fundamental process detects and outlines of an object and boundaries among objects and the background in the image. Edge detection is the most familiar approach for detecting significant discontinuities in intensity values.

Edges are local changes in the image intensity. Edges typically occur on the boundary between two regions. The main features can be extracted from the edges of an image. Edge detection has major feature for image analysis. These features are used by advanced computer vision algorithms. Edge detection is used for object detection which serves various applications like medical image processing, biometrics etc. Edge detection is an active area of research as it facilitates higher level image analysis. There are three different types of discontinuities in the grey level like point, line and edges. Spatial masks can be used to detect all the three types of discontinuities in an image.

There are many edge detection techniques in the literature for image segmentation. The most commonly used discontinuity based edge detection techniques are reviewed in this section. Those techniques are Roberts edge detection, Sobel Edge Detection, Prewitt edge detection, Kirsh edge detection, Robinson edge detection, Marr-Hildreth edge detection, LoG edge detection and Canny Edge Detection.

Roberts Edge Detection

The Roberts edge detection is introduced by Lawrence Roberts (1965). It performs a simple, quick to compute, 2-D spatial gradient measurement on an image. This method emphasizes regions of high spatial frequency which often correspond to edges. The input to the operator is a grayscale image the same as to the output is the most common usage for this technique. Pixel values in every point in the output represent the estimated complete magnitude of the spatial gradient of the input image at that point.

-1	0
0	+1

G_x

0	-1
+1	0

G_y

Sobel Edge Detection

The Sobel edge detection method is introduced by Sobel in 1970 (Rafael C.Gonzalez (2004)). The Sobel method of edge detection for image segmentation finds edges using the Sobel approximation to the derivative. It precedes the edges at those points where the gradient is highest. The Sobel technique performs a 2-D spatial gradient quantity on an image and so highlights regions of high spatial frequency that correspond to edges. In general it is used to find the estimated absolute gradient magnitude at each point in an input grayscale image. In conjecture at least the operator consists of a pair of 3x3 convolution kernels as given away in under table. One kernel is simply the other rotated by 90°. This is very alike to the Roberts Cross operator.

-1	-2	-1
----	----	----

-1	0	-1
----	---	----

0	0	0
+1	+2	+1

G_x

-2	0	+2
-1	0	+1

G_y

Prewitt Edge Detection

The Prewitt edge detection is proposed by Prewitt in 1970 (Rafael C. Gonzalez [1]). To estimate the magnitude and orientation of an edge Prewitt is a correct way. Even though different gradient edge detection wants a quite time consuming calculation to estimate the direction from the magnitudes in the x and y-directions, the compass edge detection obtains the direction directly from the kernel with the highest response. It is limited to 8 possible directions; however knowledge shows that most direct direction estimates are not much more perfect. This gradient based edge detector is estimated in the 3x3 neighborhood for eight directions. All the eight convolution masks are calculated. One complication mask is then selected, namely with the purpose of the largest module.

-1	-1	-1
0	0	0
+1	+1	+1

G_x

-1	0	+1
-1	0	+1
-1	0	+1

G_y

Prewitt detection is slightly simpler to implement computationally than the Sobel detection, but it to produce somewhat noisier results.

Kirsch Edge detection

Kirsch edge detection is introduced by Kirsch (1971). The masks of this Kirsch technique are defined by considering a single mask and rotating it to eight main compass directions: North, The edge magnitude is defined as the maximum value found by convolution of each mask with the image. The direction is defined by mask that produces the maximum magnitude. Example, mask k_0 corresponds to a vertical edge, while mask k_5 corresponds to a diagonal edge. Notice that the last four masks are actually the same as the first four, but flipped about a central axis.

Robinson Edge detection

The Robinson method (Robinson 1977) is similar to Kirsch masks but is easier to implement because they rely only on coefficients of 0, 1 and 2. The masks are symmetrical about their directional axis, the axis with the zeros. One need only to compute the result on four masks and the result from other four can be obtained by negating the result from the first four.

The magnitude of the gradient is the maximum value gained from applying all eight masks to the pixel neighborhood, and the angle of the gradient can be approximated as the angle of the line of zeroes in the mask yielding the maximum response.

Marr-Hildreth Edge Detection

The Marr-Hildreth (1980) technique is a method of detecting edges in digital images that is continuous curves wherever there are well-built and fast variations in image brightness. It is a easy and it operates by convolving the image with the LoG function, or, as a quick approximation by DoGs. Subsequently the zero-crossings are discovered in the filtered result to find the edges. The LoG method is sometimes as well referred to as the Mexican hat wavelet due to its

image shape while turned up-side-down. Algorithm for the Marr-Hildreth edge detector is:

- Smooth the image using a Gaussian
- Apply a two-dimensional Laplacian to the smoothed image (often the first two steps are combined into a single operation)
- Loop through the result and look for sign changes. If there is a sign change plus the slope across the sign change is greater than some threshold, mark as an edge.
- To get better results it is possible to run the result of the Laplacian through a hysteresis like to Canny's edge detection although this is not how the edge detector was firstly implemented.

LoG edge detection

The Laplacian of Gaussian (LoG) was proposed by Marr(1982). The LoG of an image $f(x,y)$ is a second order derivative defined as,

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

It has two effects, it smoothes the image and it computes the Laplacian, which yields a double edge image. Locating edges then consists of finding the zero crossings between the double edges. The digital implementation of the Laplacian function is usually made through the mask below,

0	-1	0
-1	4	-1
0	-1	0

-1	-1	-1
-1	8	-1
-1	-1	-1

$$G_x$$

$$G_y$$

The Laplacian is generally used to found whether a pixel is on the dark or light side of an edge.

Canny Edge Detection

In industry, the Canny edge detection technique is one of the standard edge detection techniques. It was first created by John Canny for his Master's thesis at MIT in 1983, and still outperforms many of the newer algorithms that have been developed. To find edges by separating noise from the image before find edges of image the Canny is a very important method. Canny method is a better method without disturbing the features of the edges in the image afterwards it applying the tendency to find the edges and the serious value for threshold. The algorithmic steps are as follows:

- Convolve image $f(r, c)$ with a Gaussian function to get smooth image $f^{\wedge}(r, c)$.

$$f^{\wedge}(r, c) = f(r, c) * G(r, c, 6)$$

- Apply first difference gradient operator to compute edge strength then edge magnitude and direction are obtain as before.
- Apply non-maximal or critical suppression to the gradient magnitude.
- Apply threshold to the non-maximal suppression image.

Unlike Roberts and Sobel, the Canny operation is not very susceptible to noise. If the Canny detector worked well it would be superior.

4.16 RATIO OF AVERAGES (ROA) OPERATOR

The Ratio of averages (ROA) is an edge detector with Constant False Alarm Rate (CFAR), developed specifically for SAR images. In

this operator, an analyzing window of size $N \times N$ is considered. This analyzing window is divided into two zones relative to central pixel. It calculates the ratio of arithmetic means $\bar{I}_1, \bar{I}_2$ of intensity on opposite halves of the analyzing window

On the analyzing window, we consider the presence of one edge only. Figure 2 illustrates the idealized monoedge model in the horizontal direction. To make the operator independent of the scanning direction the ratio can be normalized. Both cases are present. The first one is based on the choice of the maximum ratio, then we obtain a ratio of intensity r greater than one. But, in this case the variation interval of the normalized ratio is not defined. The second one is based on the choice of the minimum ratio, this gives a limited interval. In this paper we consider the second case, and the ratio is normalized to lie between zero and one

$$r = \min \left\{ \frac{\bar{I}_1}{\bar{I}_2}, \frac{\bar{I}_2}{\bar{I}_1} \right\}$$

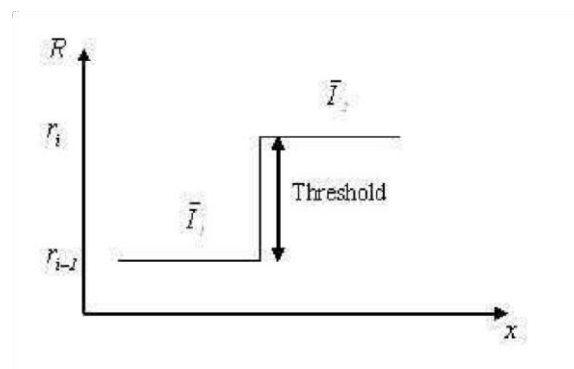


FIG-1.13 : Unidirectional mono-edge model.

For each pixel, the ratio is calculated in different directions (horizontal, vertical, diagonals), and the minimum value of ratio is considered. A low and high ratio indicates respectively the presence of an edge and homogenous region. A pixel is affected to an edge if $r \leq T$ (T is decision threshold of detection). Else, the pixel is affected to

homogenous region. Then, the performance of the operator ROA depends on the choice of decision threshold T .

The performance of the operator ROA depends on the choice of decision threshold T . It is very important to know the conditional probability distribution of ROA $P_r(r/R_1, R_2, N_1, N_2)$ to fix the false alarm rate. The probability density of the ratio for Single Look image is given by

$$Pr(r/R_1, R_2, N_1, N_2) = \frac{\Gamma(N_1 + N_2) \frac{1}{r} \left(\frac{N_1 r}{N_2 C} \right)^{N_1}}{\Gamma(N_1) \Gamma(N_2) \left(1 + \frac{N_1 r}{N_2 C} \right)^{N_1 + N_2}}$$

Where N_1, N_2 are the numbers of pixels in the two halves of the analyzing window. Figure 2 shows the conditional probability of ROA. In SAR image, the speckle is generally modeled as a strong multiplicative, Gammadistributed random noise, with unity mean and variance equal to the inverse of the Equivalent Number of Independent Looks (ENIL) [3].

For a given decision threshold T and ratio contrast C , the probability of detecting an edge is:

$$P(T, C) = \int_T^{\infty} Prob(r > T/C) = P(r/C) dr \quad (4)$$

Where $P(T, C)$ is the conditional probability of the normalized ratio. The probability of false alarm (PFA) is the probability of detecting an edge in homogeneous region (the probability of detecting an edge in the case of the ratio contrast $C = 1$)

$$PFA(T) = P(T, 1)$$

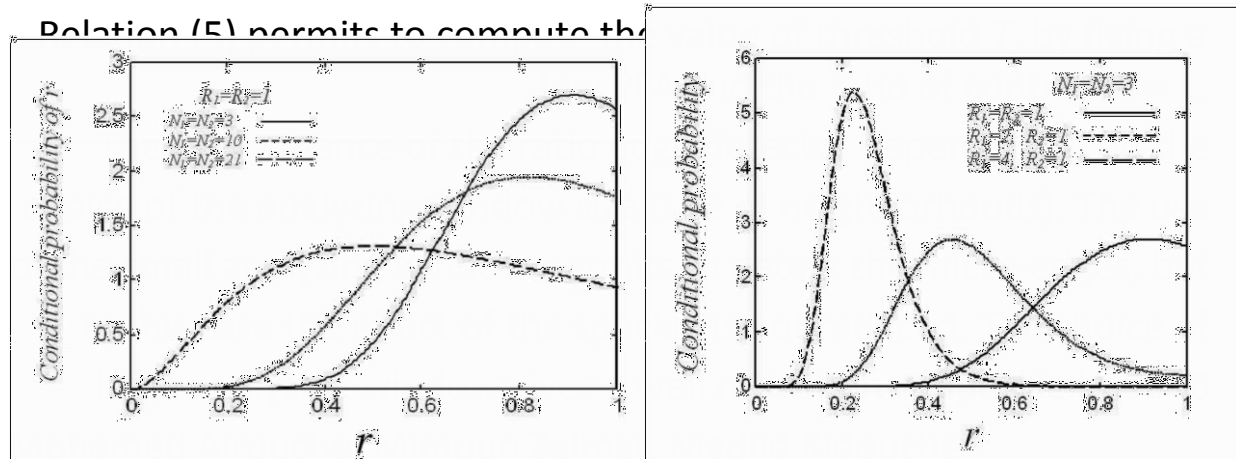


FIG-1.14 Conditional probability of ROA for different window sizes and reflectivity.

the micro-edges are not detected. In order to detect most edges and reduce the speckle, the ratio must be applied for different sizes of analyzing window. In our algorithm, we calculate the ratio on windows of increasing size from 3×3 to 9×9 . The processes begin with computation of the ratio for the analyzing window of size 3×3 . The computed ratio is compared to decision threshold and if an edge is detected we pass to the next pixel, else the ratio is computed for an analyzing window of size 5×5 . We test the ratio and if an edge is not detected we pass to a window of size 7×7 . The same operation is applied for each analyzing window. The process is stopped for a window size of 9×9 , and if an edge is not detected for this window

the considered pixel is assigned to homogenous region. To detect more edges the ROA detector is applied for all possible directions (the four usual directions are: horizontal, vertical, diagonal 1, diagonal 2). The ratio is computed for each direction and the minimum value corresponds to the direction of the most probable edge. The PFA for the ratio computed in four directions is given by

$$PFA_4 = 1 - (1 - PFA_1)^3$$

Where PFA_1 is the PFA of the ratio computed in one direction.

4.17 Ratio Of Exponentially Weighted Averages operator

To reduce the influence of the speckle sufficiently, the analyzing window size must be important. A large window can contain several edges simultaneously. There is a conflict between speckle reduction and high spatial resolution. An edge detector based on multi-edge model, which is optimal in the Minimum Mean Square Error (MMSE) sense, is applied to SAR images to improve the speckle suppression and edge detection properties. Basically, it consists on the means ratio computation on opposite sides of the central pixel, with non-uniform weighting. The pixels coefficients weighting are considered as a function of the distance to the central pixel. In the case of multi-edge detector, several edges are considered in the analyzing window, Figure 3 presents the multi-edge model in one direction.

We suppose that SAR image is composed by zones of constant reflectivity. In the horizontal direction or in the vertical direction, the reflectivity R is a random process composed by segments of reflectivity, with mean μ_r and deviation σ_r . The change of reflectivity follows a Poisson distribution. That, the probability to have n jumps in the interval Δx is given by

$$(\lambda \Delta x)^n e^{-\lambda \Delta x}$$

$$P(n/\Delta x)= \frac{e^{-\lambda_x \Delta x} (\lambda_x \Delta x)^n}{n!}$$

Where λ_x is the mean jump frequency.

The optimum impulse response in horizontal direction is obtained by minimizing the mean square error of

reflectivity

$$f(x)=Ce^{-\alpha|x|}$$

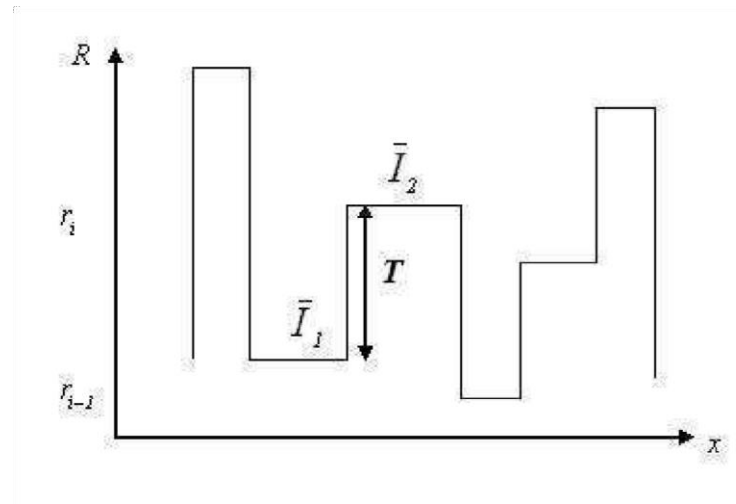


FIG-1.15
Unidirectional
multi-edge model.

where

$$2L\lambda^x$$

$$\alpha =$$

$$\frac{1}{1+(\mu./\sigma.)}$$

Figure 4 illustrates the optimum impulse response in the horizontal direction.

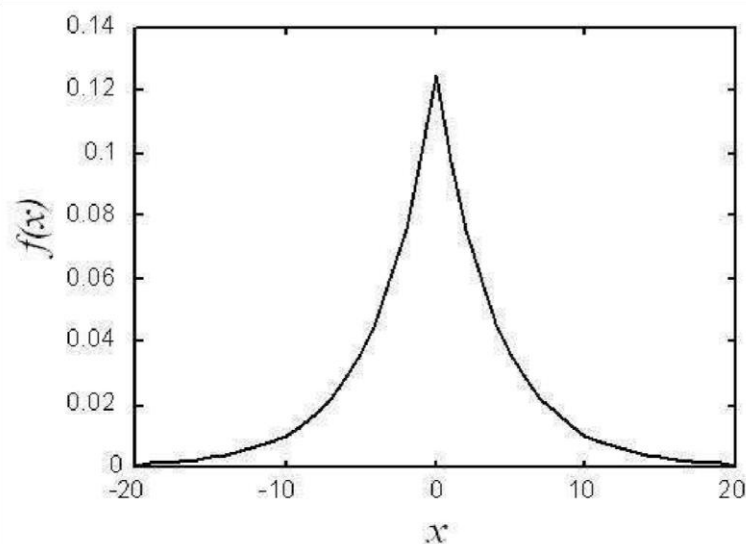


FIG-1.16 The impulse response of the exponential filter in horizontal direction ($\alpha = 0.25$).

In the discrete case, the filter $f(x)$ can be implemented very efficiently by pair of filters (causal and anti-causal). To detect the horizontal edge, the image is first smoothed column by column using the filter $f(x)$. Next, the means on the two opposite sides of each pixel are computed line by line independently, using the two filters causal and anti-causal. The means ratio obtained is calculated for each pixel. The ratio is normalized to lie between zero and one by taking the minimum of the ratio computed and it's inverse. The ratio in the vertical direction is computed by the same manner. The performance of this operator depends on the parameter choice α of the exponential function. The ROEWA is computed only for two directions (horizontal, vertical), the minimum of ratio indicates the presence of an edge. The ratio is compared to decision threshold, but in this case it is difficult to calculate the PFA correspond to the decision threshold.

4.18 Python (programming language)

Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991. An interpreted language, Python has a design philosophy which emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax which allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java. The language provides constructs intended to enable writing clear programs on both a small and large scale. Python features a dynamic type system and automatic memory management and supports multiple programming paradigms, including object-oriented, imperative, functional programming, and procedural styles. It has a large and comprehensive standard library. Python interpreters are available for many operating systems, allowing Python code to run on a wide variety of systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation

History

Python was conceived in the late 1980s, and its implementation began in December 1989 [28] by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL) capable of exception handling and interfacing with the operating system Amoeba. Van Rossum is Python's principal author, and his continuing central role in deciding the direction of Python is reflected in the title given to him by the Python community, Benevolent dictator for life. (BDFL). About the origin of Python, Van Rossum wrote in 1996: Over six years ago, in December 1989, I was looking for a "hobby" programming

project that would keep me occupied during the week around Christmas. My office ... would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus). Python 2.0 was released on 16 October 2000 and had many major new features, including a cycle-detecting garbage collector and support for Unicode. With this release the development process was changed and became more transparent and community-backed. Python 3.0 (which early in its development was commonly referred to as Python 3000 or py3k), a major, backwards-incompatible release, was released on 3 December 2008 after a long period of testing. Many of its major features have been backported to the backwards-compatible Python 2.6.x and 2.7.x version series. The End Of Life date (EOL, sunset date) for Python 2.7 was initially set at 2015, then postponed to 2020 out of concern that a large body of existing code cannot easily be forward-ported to Python 3. In January 2017, Google announced work on a Python 2.7 to Go transcompiler, which The Register speculated was in response to Python 2.7's planned end-of-life but Google cited performance under concurrent workloads as their only motivation. Features and philosophy Python is a multi-paradigm programming language: object-oriented programming and structured programming are fully supported, and many language features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)). Many other 6/13/2017 Python (programming language) -

Wikipedia

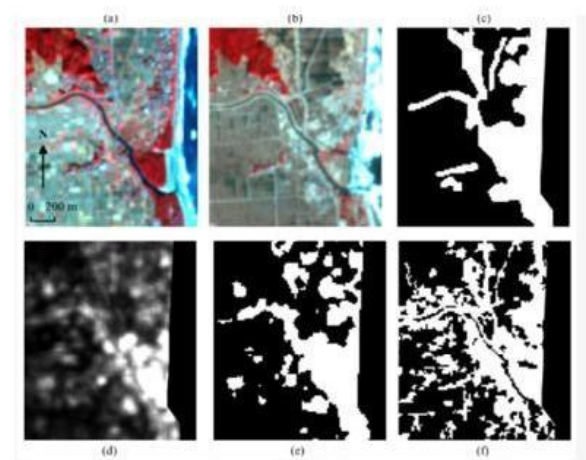
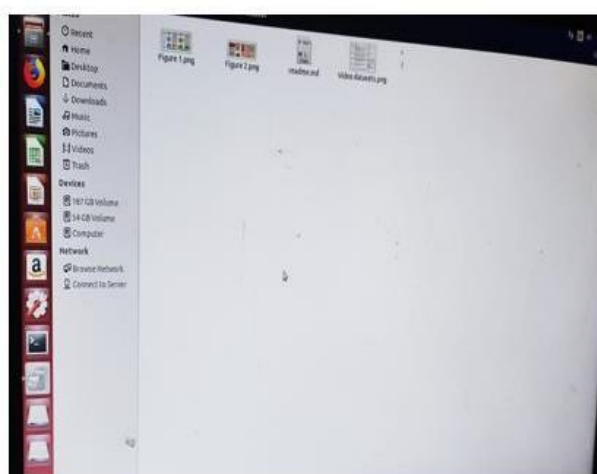
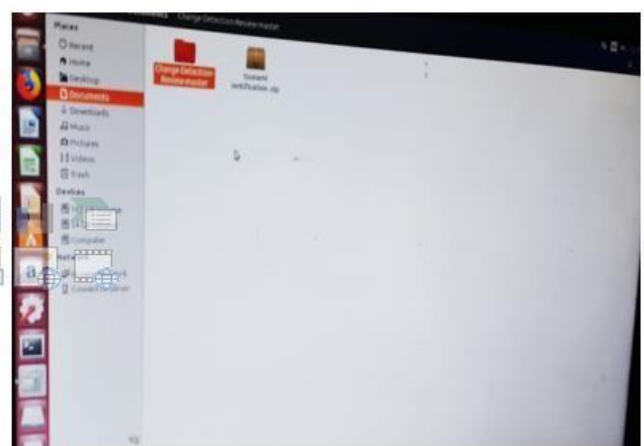
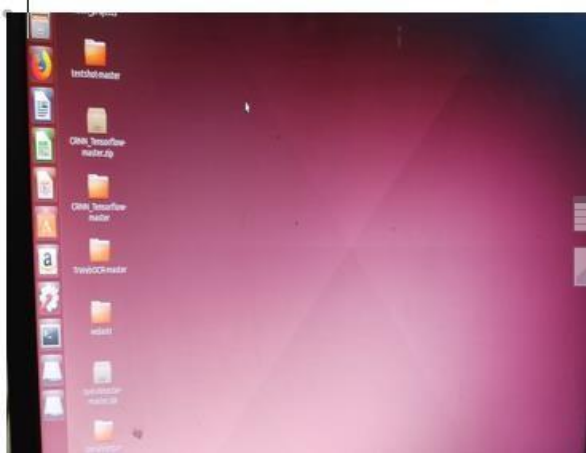
[https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language)) 3/19

paradigms are supported via extensions, including design by contract and logic programming.

CHAPTER 5

RESULT AND DISCUSSION:

RESULTS:



CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 CONCLUSION:

The state-of-the-art image processing algorithm is often used in high-resolution remote-sensing image-based damage-mapping approaches, but it has seldom been improved from the perspective of operational practice.

In this work, a framework aiming to enhance the operational damage-mapping practice was proposed. The framework is implemented using a deep learning algorithm under the CNTK framework through an end-to-end improved U-net convolutional network.

The 2011 Tohoku Earthquake-Tsunami was selected as the case study to develop and validate the framework. The design of the framework considers the availability of data sources, the feasibility of model implementation, time cost and accuracy of the method immediately after the disaster.

6.2 FUTURE WORK

To deployment real time this process by show the prediction result in web application or desktop application.

To optimize the work to implement in Artificial Intelligence environment. To deploy this model to AI on web application.

To Develop the code, so that it runs much faster and with utmost accuracy

CHAPTER 7

REFERENCE

S

[1] Hydraulic Structures, Equipment and Water Data Acquisition Systems – Tsunamis and Tsunami.

[2] Bolt, Bruce (W. H. Freeman and Company), "Earthquakes: 2016 Centennial Update — The 1906 Big One (Fifth ed.)", August 2015.

[3] Warning Systems - William Mansfield Adams, Sr., Jan Malan Jordaan, Jr.

[4] Tsunami wave seismicity, 2012 (Online), Available: <https://en.wikipedia.org/wiki/Tsunam>

[5] University of Washington, Dept. of Earth Sciences and Space Sciences. "Tsunami!" last update: March 29, 2015.

*6+ Adam Pascale, "Using Micro-Electro Mechanical Systems (MEMS) accelerometers for earthquake

monitoring”, Environmental systems and services (ES&S), June 2019.

*7+ Baron J. Richardson, “Measuring Acceleration Caused by Seismic Shaking using a High-Speed Camera”, NSF.

*8+ Japan Meteorological Agency, “What is an Earthquake Early Warning?”, Retrieved on 2018-06-29.

APPENDICES

Sample Code:-

```
from __future__ import
```

```
division import math
```

```
import cv2
```

```
from mwt_objects import Section
```

```
# Boolean flag to filter blobs by area:
```

```
FLAGS_FILTER_BY_AREA = True
```

```
# Boolean flag to filter blobs by inertia (shape):
```

```
FLAGS_FILTER_BY_INERTIA = True
```

```
# Minimum area threshold for contour:
```

```
MINIMUM_AREA = 100
```

```
# Maximum area threshold for contour:
```

```
MAXIMUM_AREA = 1e7
```

```
# Minimum inertia threshold for contour:
```

```
MINIMUM_INERTIA_RATIO = 0.0
```

```
# Maximum inertia threshold for contour:
```

```
MAXIMUM_INERTIA_RATIO = 0.1
```

```
def find_contours(frame):
```

```
    contours, hierarchy = cv2.findContours(  
        image=frame,  
        mode=cv2.RETR_EXTERNAL,  
        method=cv2.CHAIN_APPROX_NONE,  
        hierarchy=None,  
        offset=None)
```

```
    return contours
```

```
def keep_contour(contour,
```

```
    area=FLAGS_FILTER_BY_AREA,
```

```
    inertia=FLAGS_FILTER_BY_INERTIA,
```

```
min_area=MINIMUM_AREA,  
max_area=MAXIMUM_AREA,  
min_inertia_ratio=MINIMUM_INERTIA_RATIO,  
max_inertia_ratio=MAXIMUM_INERTIA_RATIO):
```

```
# Initialize the return value.
```

```
ret = True
```

```
# Obtain contour moments.
```

```
moments = cv2.moments(contour)
```

```
# Filter Contours By Area.
```

```
if area is True and ret is True:
```

```
    area = cv2.contourArea(contour)
```

```
    if area < min_area or area >= max_area:
```

```
        ret = False
```

```
# Filter contours by inertia.
```

```
if inertia is True and ret is True:
```

```
    denominator = math.sqrt((2*moments['m11'])**2  
                             + (moments['m20']-moments['m02'])**2)
```

```
    epsilon = 0.01
```

```
    ratio = 0.0
```

```
if denominator > epsilon:
```

```
    cosmin = (moments['m20']-moments['m02']) / denominator
```

```
    sinmin = 2*moments['m11'] / denominator
```

```
    cosmax = -cosmin
```

```
    sinmax = -sinmin
```

```
    imin = (0.5*(moments['m20']+moments['m02'])
```

```
            - 0.5*(moments['m20']-moments['m02'])*cosmin
```

```
            - moments['m11']*sinmin)
```

```
    imax = (0.5*(moments['m20']+moments['m02'])
```

```
            - 0.5*(moments['m20']-moments['m02'])*cosmax
```

```
            - moments['m11']*sinm
```

```
    ax) ratio = imin / imax
```

```
else:
```

```
    ratio = 1
```

```
if ratio < min_inertia_ratio or ratio >= max_inertia_ratio:
```

```
    ret = False
```

```
    #center.confidence = ratio * ratio;
```

```
return ret
```

##

=====

```
def detect_sections(frame, frame_number):
```

```
    # Convert to single channel for blob detection if necessary.
```

```
    if len(frame.shape) > 2:
```

```
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

```
    # Initiate an empty list of sections.
```

```
    sections = []
```

```
    # 1. Find the contours.
```

```
    contours = find_contours(frame)
```

```
    # 2. Filter the contours.
```

```
    for contour in contours:
```

```
        if keep_contour(contour) is False:
```

```
            continue
```

```
    # If contour passes thresholds, convert it to a Section.
```

```
    section = Section(points=contour, birth=frame_number)
```

3. Add the section to sections list.

```
sections.append(section)
```

```
return sections
```