

STONE INSCRIPTION RECOGNITION AND IDENTIFICATION SYSTEM

A PROJECT REPORT

Submitted by

TRIYAN SUBRAMANYAM [Reg No:RA2011003010870]

KALLURI SHANMUKHA SAI [Reg No:RA2011003010841]

Under the Guidance of

Dr. SINDHU. C

Assistant Professor, Department of Computing Technologies

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING



DEPARTMENT OF COMPUTING TECHNOLOGIES

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR– 603 203

NOVEMBER 2023



SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR–603 203
BONAFIDE CERTIFICATE

Certified that 18CSP107L project report titled “**STONE INSCRIPTION RECOGNITION AND IDENTIFICATION SYSTEM**” is the bonafide work of **TRIYAN SUBRAMANYAM [RegNo:RA2011003010870]** and **KALLURI SHANMUKHA SAI [RegNo:RA2011003010841]** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported here in does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

Dr. SINDHU C
SUPERVISOR
Assistant Professor
Department of Computing Technologies

Dr. KOWSIGAN M
PANEL HEAD
Associate Professor
Department of Computing Technologies

Dr. M. PUSHPALATHA
PROFESSOR AND HEAD
Department of Computing Technologies



Department of Computing Technologies
SRM Institute of Science and Technology
Own Work Declaration Form

Degree/Course : B.Tech in Computer Science and Engineering

Student Names : TRIYAN SUBRAMANYAM, KALLURI
SHANMUKHA SAI

Registration Number: RA2011003010870, RA2011003010841

Title of Work : **STONE INSCRIPTION RECOGNITION AND IDENTIFICATION SYSTEM**

I/We here by certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism, as listed in the University Website, Regulations, and the Education Committee guidelines.

I / We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions:

- Clearly references / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc.)
- Given the sources of all pictures, data etc that are not my own.
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course hand book / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:

I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.

Student 1 Signature:

Student 2 Signature:

Date: 6th November 2023

If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

ACKNOWLEDGEMENT

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to Dean-CET, SRM Institute of Science and Technology, **Dr.T. V. Gopal**, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor and Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We are incredibly grateful to our Head of the Department, **Dr. M. Pushpalatha**, Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We register our immeasurable thanks to our Faculty Advisor, **Dr. P. Murali**, Associate Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for leading and helping us to complete our course.

We want to convey our sincere thanks to our Project Coordinators, **S. Godfrey Winster**, **Dr. M. Baskar**, **Dr. P. Murali**, **Dr. J. Selvin Paul Peter**, **Dr. C. Pretty Diana Cyril** and **Dr.G.Padmapriya**, Panel Head, **Dr.Kowsigan M**, Associate Professor and Panel Members, **Dr. R. Yamini**, **Dr. R. Thamizhamuthu**, **Mrs. R. Madhura** Assistant Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for their inputs during the project reviews and support.

Our inexpressible respect and thanks to our guide, **Dr. Sindhu C**, Assistant Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for providing us with an opportunity to pursue our project under her mentorship. She provided us with the freedom and support to explore the research topics of our interest. Her passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank all the staff and students of Computing Technologies Department, School of Computing, S.R.M Institute of Science and Technology, for their help during our project. Finally, we would like to thank our parents, family members, and friends for their unconditional love, constant support and encouragement.

TRIYAN SUBRAMANYAM [Reg. No: RA2011003010870]

KALLURI SHANMUKHA SAI [Reg. No: RA2011003010841]

ABSTRACT

Ancient stone inscriptions are a valuable source of information about the past, but they can be difficult to read due to erosion, damage, and other factors. Character recognition systems can help to overcome these challenges by automatically extracting and recognizing the characters in stone inscriptions. Character recognition of ancient stone inscriptions using the Histogram of Oriented Gradients approach is a cutting-edge application that holds immense promise for the fields of archaeology and historical preservation. These inscriptions, often bearing vital information about past civilizations and cultures, are susceptible to degradation over time due to weathering and erosion. The Histogram of Oriented Gradients approach, renowned for its ability to capture detailed texture and shape information in images, proves to be a valuable tool for tackling the unique challenges posed by ancient stone inscriptions. By analyzing the distribution of local gradient orientations, Histogram of Oriented Gradients extracts distinctive features from the inscriptions, allowing for accurate character recognition, even in the presence of erosion and varying lighting conditions. This innovative technique not only aids in the transcription and documentation of invaluable historical content but also facilitates the broader study of ancient civilizations and their languages, shedding light on their stories, beliefs, and practices, while contributing to the preservation and understanding of our shared human heritage.

TABLE OF CONTENTS

Chapter No	Chapter Title	Pg. No
	ABSTRACT	vi
	LIST OF FIGURES	ix
	LIST OF ABBREVIATIONS	x
1.	INTRODUCTION	1
1.1	Canny Edge Detection Algorithm	3
1.2	Otsu Threshold Method	5
1.3	Morphology	7
1.4	Watershed Algorithm	9
1.5	HOG Features	10
2	LITERATURE SURVEY	11
3	SYSTEM DESIGN	14
3.1	SIRIS System Architecture	14
3.1.1	Input Image for Training	16
3.1.2	Pre-Processing	16
3.1.3	Feature Extraction	17
3.2	Steps in the SIRIS System	18
3.3	SIRIS Modules Description	15
3.3.1	Gray Scale Conversion	19
3.3.2	Filtration	20
3.4	Gray Scale Conversion	22
3.5	Smoothing using Gaussian Filter	23
3.6	Canny Edge Detection	24

3.7	Otsu Binary Threshold	25
3.8	Watershed Algorithm	27
4	SYSTEM IMPLEMENTATION	28
4.1	Code Snapshots	28
4.2	Datasets	30
4.3	Outputs for different datasets	33
4.4	Results	40
4.5	Discussion	40
5	CONCLUSION	41
5.1	Conclusion	41
5.2	Future Enhancements	41
	REFERENCES	43
	APPENDIX 1	46
	APPENDIX 2	52
	PLAGIARISM REPORT	58

LIST OF FIGURES

Fig. No	Title of the Figure	Pg.No
2.1	Datasets of Similar papers and Algorithm used	8
3.1	Identifying inscriptions using proposed method	9
3.2	SIRIS text conversion system	10
3.3	SIRIS architecture diagram	12
3.4	SIRIS modules and filters	15
4.1	Gray Scale Conversion Code	24
4.2	Edge Detection Code	24
4.3	Otsu's binary threshold Code	25
4.4	Inscription from Brihadeeswarar Temple	30
4.5	Inscription from Airavatesvara Temple	30
4.6	Inscription from Meenakshi Amman Temple	30
4.7	Inscription from Kailasanathar Temple	30
4.8	Inscription from Chidambaram Nataraja Temple	31
4.9	Inscription from Thillai Nataraja Temple	31
4.10	Inscription from Shore Temple	32
4.11	Inscription from Jambukeswarar Temple	32

LIST OF ABBREVIATIONS

ACR	Ancient Character Recognition
CNN	Convolutional Neural Network
CV	Computer Vision
CR	Character Recognition
HOG	Histogram of Oriented Gradients
ICR	Incidental Character Recognition
ML	Machine Learning
MLP	Multilayer Perceptron
NN	Neural Network
OCR	Optical Character Recognition
RNN	Recurrent Neural Network
ROI	Region of Interest
SVM	Support Vector Machine

CHAPTER 1

INTRODUCTION

Ancient stone inscriptions are a valuable source of information about the past, but they can be difficult to read due to erosion, damage, and other factors. Character recognition (CR) systems can help to overcome these challenges by automatically extracting and recognizing the characters in stone inscriptions.

Epigraphy is the study of inscriptions, which are often carved or engraved into durable materials such as stone, metal, or wood. These inscriptions can provide valuable insights into various aspects of human history, culture, and language.

Epigraphy serves as a vital discipline for historians, archaeologists, and linguists. It encompasses the examination and interpretation of inscriptions to decipher their content and context. Epigraphers, experts in this field, use a combination of linguistic, archaeological, and historical methods to unravel the meaning and significance of these inscriptions.

Paleography is the scholarly discipline that focuses on the study and deciphering of ancient and historical scripts, particularly handwriting and written documents. It is a field of study that plays a crucial role in understanding the evolution of writing systems, languages, and historical manuscripts.

The primary objective of paleography is to analyze, interpret, and date ancient manuscripts and documents by examining the style, form, and content of the written text. Paleographers, specialists in this field, meticulously examine handwritten materials, taking into account the shape and structure of individual characters, the use of abbreviations and ligatures, and the overall layout of the text.

One approach to CR for ancient stone inscriptions is to use a combination of the following methods: Canny edge detection algorithm, Otsu method, Morphology, Water shed, HOG features.

1.1 Canny edge detection algorithm

The Canny edge detection algorithm is a widely used algorithm for detecting edges in images. It works by first applying a Gaussian filter to the image to smooth out noise. Then, the image is convolved with two Sobel operators to calculate the gradient magnitude and direction at each pixel. Finally, a non-maximum suppression step is applied to identify edges that are connected and have strong gradients.

The first step is to reduce noise in the image by applying a Gaussian filter. This blurs the image, reducing the impact of small, high-frequency details that may result from noise. Next, the algorithm calculates the image's gradients, determining the derivatives in both the horizontal (x) and vertical (y) directions. Using specific formulas, it computes the gradient magnitude and direction at each pixel. Non-maximum suppression follows, where potential edge pixels are identified by analyzing the gradient magnitude and direction. Only the pixels with the highest magnitude among their neighbors in the direction of the gradient are considered candidate edge pixels; others are suppressed.

To address the challenge of weak and strong edge pixels, the algorithm classifies edge pixels as either "strong" (above a high threshold) or "weak" (above a low threshold but not strong). Weak edges adjacent to strong ones are considered part of the edge, thus connecting continuous edge lines and bridging gaps.

The algorithm employs thresholding, utilizing a high threshold for strong edges and a low threshold for weak edges. Pixels with gradient magnitudes above the high threshold are marked as strong edges, while those below the low threshold are discarded.

Finally, edge tracing connects the strong and weak edges, resulting in a final edge map that highlights continuous and prominent edges within the image.

Canny edge detection is highly regarded for its accuracy in edge detection and its ability to mitigate the impact of noise. This technique finds applications in various areas of computer vision, including object detection, image segmentation, and feature extraction. Adjusting the threshold values and the size of the Gaussian filter allows for fine-tuning of edge detection

results for specific applications.

1.2 Otsu method

Otsu thresholding is a method for binarizing an image. It works by finding the threshold that maximizes the variance between the foreground and background pixels. This method is particularly effective for images with low contrast, such as edge images of stone inscriptions. The essence of the Otsu method lies in maximizing the separation between two classes of pixels in an image. One class represents the background, while the other class represents the foreground or the objects of interest. The key steps involved in implementing the Otsu method are as follows:

The algorithm begins by creating a histogram of pixel intensity values from the grayscale image. This histogram illustrates the distribution of pixel values, highlighting peaks that correspond to the background and foreground intensities.

1.3 Morphology

Morphology is a set of image processing operations that can be used to remove noise, smooth edges, and fill inholes in images. Some common morphological operations include dilation, erosion, opening, and closing. Morphology, in the context of image processing, is a fundamental technique that deals with the analysis and manipulation of the shape and structure of objects within images. It draws inspiration from mathematical set theory and relies on binary operations to transform and extract information from images. Morphological operations are particularly valuable in tasks like image segmentation, noise reduction, and feature extraction.

The basic building blocks of morphology are binary images, where objects of interest are represented in white (foreground), while the background is in black. Two primary binary operations used in morphology are dilation and erosion. Dilation enlarges the white regions in an image, effectively expanding objects, while erosion shrinks them.

These operations are guided by structuring elements, which are small binary patterns or masks that define the shape and size of the operation. When a structuring element is applied to an image, it slides through the image, and its relationship with the pixels determines how dilation and erosion are performed. This interaction helps in modifying the shape of objects in the image, emphasizing certain features, and filtering out noise or unwanted elements.

1.4 Watershed

Watershed is an image segmentation algorithm that can be used to identify and separate individual characters in a stone inscription image. It works by finding the pixels in the image that correspond to the highest ridges in the gradient image. These pixels are then used to create seed points for the watershed algorithm. The watershed algorithm then propagates the seed points into the image, and the pixels that are assigned to the same seed point are considered to be part of the same character. The Watershed algorithm is a powerful image processing technique used for image segmentation, particularly in cases where objects are touching or overlapping. It derives its name from the metaphor of treating an image as a topographic landscape, where intensity values are interpreted as heights. The algorithm effectively "floods" this landscape to delineate and separate objects, creating watershed lines that mark the boundaries between them.

The fundamental concept of the Watershed algorithm lies in the notion of catchment basins, where each local minimum in the image represents a basin. When water is poured onto the landscape, it flows downhill and accumulates in these basins. Watershed lines form where the water from different basins meet, effectively separating objects.

The process of applying the Watershed algorithm involves several key steps. Initially, the algorithm considers the image as a grayscale topographic map. It identifies regional minima and labels them as markers for different catchment basins. These markers guide the flooding process.

1.5 HOG features

HOG features are a type of feature extraction technique that is commonly used for object recognition and classification tasks. HOG features are computed by dividing the image into a grid of cells and calculating the gradient magnitude and direction at each pixel in each cell. The gradient magnitudes are binned into different histogram bins according to their orientations. The HOG features for each cell are then normalized to account for variations in lighting.

Histogram of Oriented Gradients features, or HOG features, are an essential component of image processing and computer vision. These characteristics are made to record how gradient orientations vary in specific areas of an image. HOG features provide important details about the edges and textures in an image by segmenting it into tiny, overlapping cells and creating histograms of gradient orientations in each cell. Their ability to describe shape and texture while being resistant to changes in lighting and object size makes them an essential tool in the computer vision toolkit, and they are widely used in tasks like object detection and tracking. The HOG feature extraction process involves several key steps. Initially, the image is divided into small cells, typically square or rectangular. Within each cell, gradient information is computed, representing the local edge orientations and strengths. This is done by calculating the gradient magnitude and orientation for each pixel using methods like the Sobel or Scharr operators.

Once gradient information is computed, the image is further divided into blocks, which are formed by grouping multiple cells. The purpose of this block formation is to provide some level of local normalization and invariance to lighting and contrast variations. Within each block, the gradient information from the constituent cells is combined, typically by histogramming the gradient orientations. The histograms within each block are then normalized, often using a process called block normalization. This step helps to make the HOG features robust to variations in lighting and contrast.

CHAPTER 2

LITERATURE SURVEY

According to G. Siromoney et al [1] , every symbol should be taken out as a string and compared to the strings in the dictionary. When there is consensus, the letters are identified and printed using Roman letters using a unique transliteration technique. Numbers placed above each letter reflect the hardness of consonants and the lengthening of vowels.

Suresh et al [3] in 1999 advocated a fuzzy set theory that offers a rough yet useful way to characterize the behavior of ill-defined systems. Handwritten characters are examples of human-derived patterns that are rather ambiguous in nature. This work actually aims to identify handwritten numerals and Tamil characters as one among the prototype unknown, where prototype characters are preprocessed and taken into consideration for recognition using the fuzzy idea. The algorithm's success percentage ranges from 76% to 94% after testing it on roughly 250 samples of numerals and seven selected Tamil characters.

Using Seethalakshmi's et al [10] technique of translating printed Tamil text documents into Unicode Tamil text translated by software is known as optical character recognition, or OCR. A skewing check is performed on the image file during the preprocessing stage. The collected characteristics are fed into a Support Vector Machine (SVM), where a Supervised Learning Algorithm classifies the characters. For recognition, these classes are mapped onto Unicode. After that, Unicode typefaces are used to rebuild the text.

N. Joshi et al [9] , in an online handwriting recognition comparison of elastic matching algorithms for writer-dependent isolated Tamil letters. Preprocessed x-y coordinates, quantized slope values, and dominating point coordinates are the three features that are taken into consideration. We compare seven approaches in terms of number of training templates, identification accuracy, and recognition speed, based on these three aspects and dynamic time warping distance measure.


S.No	Algorithm	Dataset/Representation Sample
Et al [4]	Self adaptive Lion Optimization Algorithm and Otsu's algorithm	
Et al [7]	BRISK algorithm	
Et al [8]	HOG Algorithm	
Et al [10]	SVM	
Et al [11]	Self adaptive Lion Optimization Algorithm and Otsu's algorithm	
Et al [12]	BRISK algorithm	

Fig. 2.1 Datasets of Similar papers and Algorithm used

Using the recently developed models of Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN), Uma Maheswari's et al [14] research suggests a revolutionary method for identifying inscription characters from old Tamil stone inscriptions. recorded by a camera on stone inscriptions Utilizing image correction techniques like filtering, lighting, erosion, dilation, and blurring, script photographs are improved for clarity. The process of extracting unique characters from scripts involves character segmentation based on project profile photos.

In terms of accuracy and speed, Devi R's et al [15] hybrid model, the Self-Adaptive Lion Optimization Algorithm with Transfer Learning (SLOA- TL), performs better than other current algorithms. It is a useful design for maintaining Tamil cultural heritage and identifying Tamil scripts in stone writing.

The goal of our initiative, according to Pranav Rajnish et al [16], is to digitize old Brahmi stone inscriptions and documents in order to preserve them and improve their quality and accessibility. Once the stone engravings are in digital format, they can be read, studied, and preserved with ease. The conservation of ancient inscriptions found all over India and the rest of the world depends on this effort.

CHAPTER 3

SYSTEM DESIGN

Designing a system for character recognition of ancient stone inscriptions using Histogram of Oriented Gradients involves several key components. Initially, the stone inscription images need to be preprocessed to enhance their quality and reduce noise. This may include tasks like contrast adjustment, noise removal, and resizing to standard dimensions. Next, the images are divided into smaller, non-overlapping blocks or cells, and gradient orientations are computed within each cell using the HOG method. These orientations are used to build histograms that capture the local texture and edge information. For character recognition, a machine learning model, such as a Support Vector Machine or a Convolutional Neural Network, is trained on the HOG feature vectors extracted from a labeled dataset of ancient characters. The trained model can then be employed to classify and recognize characters in new stone inscriptions. Additionally, post-processing techniques like character segmentation and context analysis can be integrated into the system to improve recognition accuracy, especially in cases where characters are closely connected or exhibit unusual variations. This system design not only leverages the power of HOG features but also combines it with robust machine learning techniques to tackle the unique challenges posed by ancient stone inscriptions, preserving and interpreting valuable historical artifacts.

Let us discuss the System Architecture, what all module descriptions are used by the SIRIS model and a brief explanation how SIRIS uses various algorithms for achieving it's intended purpose.

3.1 System Architecture

Proposed Method for Recognition of Ancient Stone Inscription: This proposed method outlines a systematic approach to recognize and translate ancient stone inscription characters into modern characters using the Histogram of Oriented Gradients (HOG) feature extraction technique as shown in the figure below.

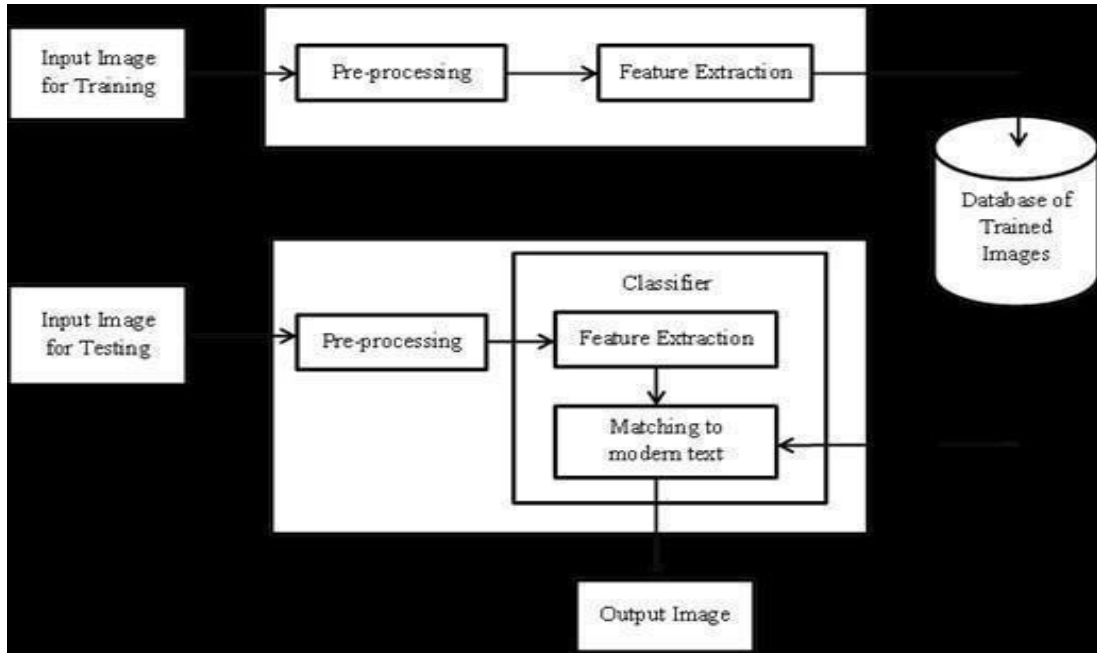


Fig. 3.1 Identifying ancient stone inscriptions using the proposed method

The existing system for transliterating ancient stone inscription characters into modern characters primarily focuses on the area of epigraphy, which is the study of inscriptions and their historical significance. In this system, a framework is employed to extract structural features from the ancient characters and map them onto Unicode for recognition and interpretation. While the existing system provides a method for transliterating ancient stone inscription characters, it's important to note that it mentions the extraction of relatively basic structural features.

The proposed system shown on Figure1 for recognizing ancient characters and converting them into modern characters utilizes a feature known as the Histogram of Oriented Gradients (HOG). This system is designed to work with stone inscribed images and involves several preprocessing steps to enhance the recognition accuracy.

3.1.1 Input Image for Training

The input image for training plays a pivotal role in the development of a robust and accurate character recognition system. The quality, content, and diversity of the training data directly influence the performance and effectiveness of the recognition model. In this specific application, the input images for training are a critical component of the pipeline, shaping the model's ability to decipher characters engraved on ancient stone inscriptions.

These input images for training consist of a collection of photographs or scans of ancient stone inscriptions. These inscriptions are often weathered, aged, and subject to various environmental factors, making the character recognition task challenging. Therefore, the training dataset should be carefully curated to encompass a wide range of inscriptions, including variations in stone type, font styles, text sizes, and degradation levels. This diversity is essential for training a model that can adapt to the intricacies of different historical inscriptions.

Preprocessing techniques are commonly applied to the input images before training the character recognition model. This may involve tasks like image enhancement, noise reduction, and contrast adjustment to improve the legibility of the inscriptions. Moreover, considering the possible presence of unwanted elements in the images, such as scratches, stains, or other artifacts, cleaning and segmentation steps may be performed to isolate the inscriptions from the background.

The process of character recognition using HOG features begins by extracting the HOG features from these preprocessed training images. The HOG feature extraction involves dividing the images into cells and blocks, computing gradient information, and generating histograms of oriented gradients. This results in a rich set of features that encode the local shape and texture characteristics of the inscriptions.

3.1.2 Pre-Processing

Pre-processing has several critical objectives in this workflow. First and foremost, it aims to reduce noise and artifacts that may be present in the image. Ancient stone inscriptions can accumulate noise over time, affecting the clarity of characters. Techniques like median filtering

or Gaussian smoothing are applied to clean up the image and enhance its overall quality.

Enhancing the contrast of the inscription is another essential aspect of pre-processing. Adjusting the contrast can make the characters more discernible, bringing out fine details carved into the stone surface. Techniques such as histogram equalization or contrast stretching are employed for this purpose.

Additionally, the pre-processing stage involves image thresholding to convert the grayscale image into a binary format. This step is crucial for distinguishing the inscribed characters from the stone background. Adaptive thresholding methods are often used to create a clear distinction between characters and the surface.

Furthermore, ancient stone inscriptions may exhibit geometric distortions, variations in scale, and perspective due to their historical context and the method of creation. To address this, image registration techniques are applied to correct these distortions and ensure consistency in character size and orientation.

3.1.3 Feature Extraction

The HOG technique excels at capturing shape and texture details within an image by examining localized gradient distributions. The feature extraction process involves several crucial stages. Initially, the image is partitioned into smaller cells, serving as the foundation for feature computation. Within these cells, gradients are calculated to describe intensity variations and edge orientations. These gradients are derived using established operators like the Sobel or Scharr filters.

Next, histograms of gradient orientations are generated within each cell. These histograms provide insights into the dominant edge directions in the local area. To further enhance the resilience of the system to variations in lighting and contrast, multiple neighboring cells' histograms are amalgamated within blocks. The normalization of these histograms, typically involving scaling to a constant Euclidean norm, is a vital step to ensure stability in the face of intensity and contrast fluctuations.

The ultimate output of this feature extraction process is a comprehensive feature vector that encapsulates shape and texture characteristics present in the image. This vector is an efficient representation of the inscription's relevant features, making it amenable to machine learning.

These techniques can capture finer details and contextual information in the inscriptions, enhancing the accuracy of the transliteration and interpretation process.

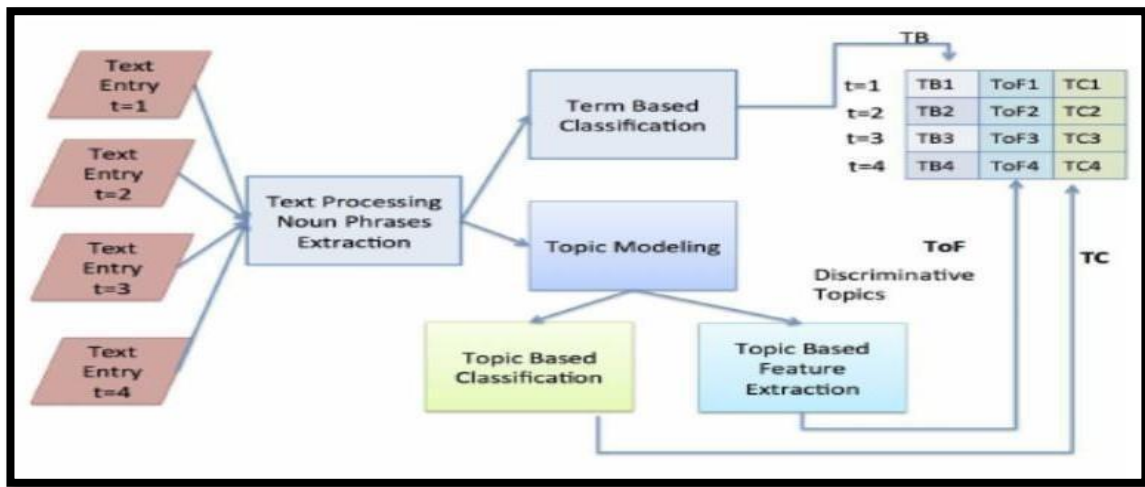


Fig.3.2 SIRIS text conversion system

3.2 Steps in the Systems

In the system of "SIRIS" the key steps are thoughtfully orchestrated to effectively decipher and recognize characters carved into weathered and ancient stone surfaces. These steps are instrumental in achieving accurate character recognition, even when dealing with challenging and deteriorated inscriptions.

The initial step in this process involves the pre-processing of the image data. Ancient stone inscriptions often come with imperfections, noise, and varying contrast levels due to their age and exposure to environmental conditions. Pre-processing techniques are employed to mitigate these challenges. Noise reduction methods, contrast enhancement, thresholding, and even image registration are applied to clean the image, improve contrast, and correct distortions.

Additionally, segmentation is utilized to isolate individual characters within the inscription.

Following the pre-processing stage, the core of the system involves feature extraction using the Histogram of Oriented Gradients (HOG) technique. HOG excels in capturing shape and texture details, making it particularly well-suited for recognizing characters in stone inscriptions. The image is divided into cells, and gradients are computed within each cell to describe local edge orientations and strengths. Histograms of gradient orientations are generated for each cell, and these histograms are further combined within blocks. Normalization of the histograms enhances robustness to lighting and contrast variations. The result is a feature vector that effectively represents the image's distinctive characteristics.

Machine learning models, such as Support Vector Machines (SVMs) or neural networks, are then employed in the system. These models are trained on the feature vectors extracted from the stone inscriptions to learn and recognize individual characters. The training process involves feeding the models with labeled data, where characters are associated with their respective feature vectors.

Once the machine learning models are trained, they are ready for character recognition. The feature vectors of characters within the stone inscription are extracted using HOG and are then fed into the trained models. The models compare the extracted features with the learned patterns to make character predictions.

The final step of this system is character classification. The models provide predictions for each character, and these predictions are further validated and refined. The system assesses the predictions, considering factors like confidence levels, and verifies them against a predefined character set. Post-processing techniques may be applied to enhance the accuracy of character recognition.

3.3 Modules Description

In the context of "SIRIS" the system's modules play a pivotal role in orchestrating the character recognition process. These modules are carefully designed to address specific tasks

and challenges in deciphering characters from ancient stone inscriptions.

The first module encompasses image pre-processing techniques, which are instrumental in preparing the input data. This module is responsible for tasks such as noise reduction, contrast enhancement, thresholding, and image registration. By cleansing the image and improving its quality, it sets the stage for effective character recognition.

The core module of the system involves HOG-based feature extraction. This module dissects the image into cells and computes gradients to capture local edge orientations and strengths. It constructs histograms of gradient orientations within these cells and further aggregates them into blocks. The normalization of these histograms ensures invariance to lighting and contrast variations. The outcome is a comprehensive feature vector that encapsulates the image's distinctive texture and shape details, making it a robust foundation for character recognition.

3.3.1 Gray scale Conversion

To obtain single intensity information, the first step is to convert RGB color images of stone inscriptions to grayscale color images. Scale of gray All that makes up an image is a range of grayscale, from black at the lowest intensity to white at the highest. The weighted sum of the R, G, and B components is formed in the following manner to convert RGB values to gray scale values.

$$\text{Intensity} = 0.2989 * R + 0.5870 * G + 0.1140 * B \quad (i)$$

3.3.2 Filtration

The image can be made noise-free by applying a 5X5 median filter. The median of the neighboring pixel values is used as a replacement for the pixel value in the median filter. To find the median, first arrange all of the neighboring neighborhood's pixel values in numerical order. Next, replace the pixel under consideration with the value of the middle pixel.

3.3.3 Normalization

In size normalization, the images of different sizes are normalized to constant dimensions 40 X 40 by keeping the aspect ratio of the images.

A bilinear interpolation algorithm preserves the original pattern's height to width ratio.

The block diagram below outlines the various stages of the proposed model. It begins with preprocessing steps to clean and enhance the input stone inscription image. The characters are then segmented, and HOG features are extracted. The model is trained on a labeled dataset of ancient characters and recognizes the characters in the input image. Post- processing and translation steps can be applied optionally to improve accuracy and providemodern character translations. The result is the recognized modern text, which can be usedfor historical interpretation and research.

One of the primary objectives of normalization is to eliminate the influence of different units and scales that may exist within a dataset. When working with data that has varied measurement units, normalization ensures that each variable contributes fairly to the analysis. This is particularly important in machine learning and statistical modeling, where features with different scales can lead to biased results or a lack of convergence in algorithms.

Normalization can help improve the performance of machine learning models and statistical analyses by ensuring that all features or variables are on a level playing field. It prevents features with larger numerical values from dominating the learning process and facilitates the convergence of optimization algorithms.

In summary, normalization is a fundamental data preprocessing technique that ensures data is scaled to a common range, allowing for fair comparisons and consistent modeling. Whether in the context of machine learning, statistics, or any domain where data analysis is involved, normalization plays a vital role in promoting accurate and reliable results.

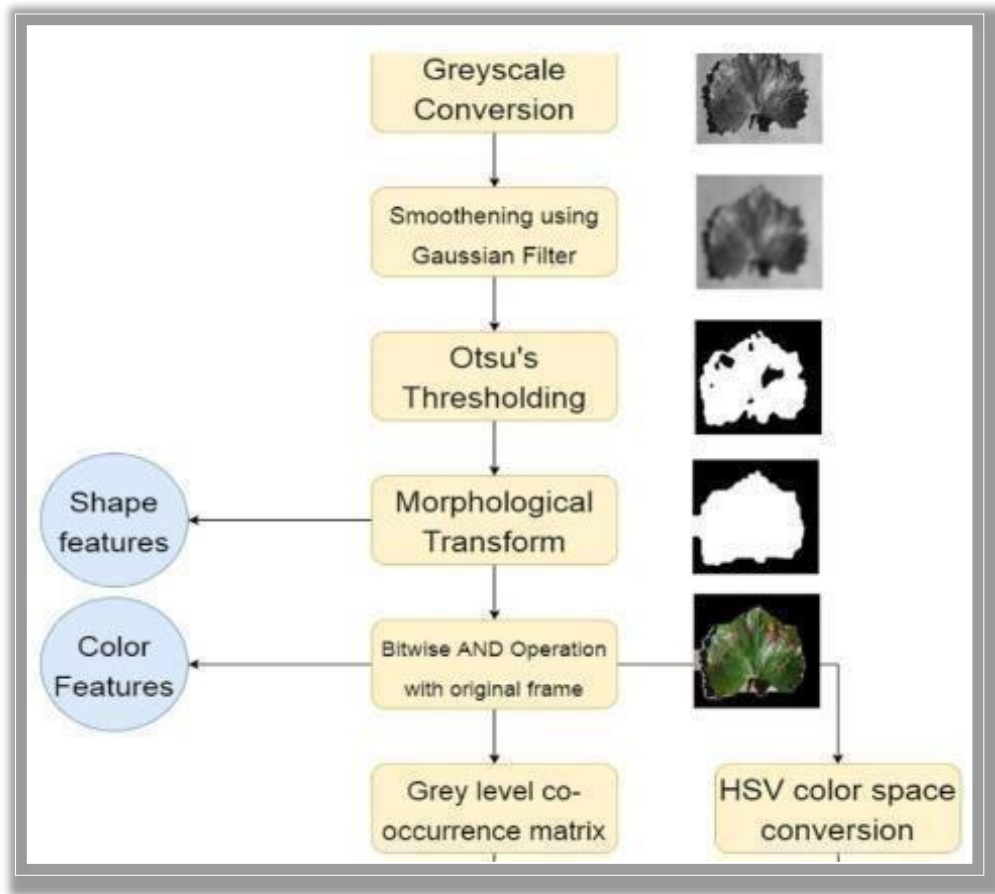


Fig. 3.3 SIRIS Architecture diagram

3.4 Gray Scale Conversion

Grayscale conversion is the process of converting an image from color to shades of gray. This can be done for a variety of reasons, such as to save space, to improve contrast, or to create a more artistic effect.

While there are numerous ways to convert an image to grayscale, the most popular approach is to use the red, green, and blue channels' weighted average. The formula for determining the grayscale value of a given pixel is as follows:

$$\text{grayscale} = (\text{red} * 0.299) + (\text{green} * 0.587) + (\text{blue} * 0.114)$$

The coefficients in this equation are based on the human visual system's sensitivity to

different colors. Red is the least sensitive color, followed by green, and then blue. Therefore, the red channel is given the lowest weight, and the blue channel is given the highest weight.

Another common method for grayscale conversion is to use a luminance-preserving algorithm. These algorithms attempt to preserve the brightness of the image while converting it to grayscale. This can be done by using a variety of different techniques, such as calculating the average luminance of each pixel or using a weighted average of the red, green, and blue channels.

Grayscale conversion can be used for a variety of purposes. One common use is to save space. Grayscale images require only one byte per pixel, while color images require three bytes per pixel. Therefore, converting a color image to grayscale can reduce the file size by up to two-thirds.

Another common use of grayscale conversion is to improve contrast. Grayscale images often have better contrast than color images, because the different shades of gray are more easily distinguishable. This can be helpful for tasks like optical character recognition (OCR) and image processing.

A more artistic effect can also be achieved by converting to grayscale. Compared to color images, grayscale images can seem more traditional and timeless. Grayscale graphics can also be used to convey a more somber or serious atmosphere.

3.5 Smoothing using Gaussian Filter

A Gaussian filter is a type of linear filter that is used to smooth images. It works by convolving the image with a Gaussian kernel, which is a bell-shaped function. The Gaussian kernel has a standard deviation parameter, which controls the amount of smoothing that is applied to the image.

Gaussian filters are effective at smoothing images because they suppress high-frequency noise while preserving low-frequency components of the image. This is because the Gaussian kernel is weighted towards the centre, so pixels that are close to each other have a greater influence on the filtered image than pixels that are further away.

Gaussian filters are widely used in image processing for a variety of tasks, including:

Noise reduction: Gaussian filters can be used to reduce noise in images by blurring the image and suppressing high-frequency noise.

Edge detection: Gaussian filters are often used as a pre-processing step for edge detection algorithms. This is because Gaussian filters can help to reduce noise and enhance edges in the image.

Image segmentation: Gaussian filters can be used to segment images by blurring the image and then using a thresholding algorithm to identify different regions in the image.

Feature extraction: Gaussian filters can be used to extract features from images, such as the corners and edges of objects.

3.6 Canny Edge Detection

The Canny edge detection algorithm is designed to detect edges that are both strong and well-localized. A strong edge is one where the gradient magnitude is high. A well-localized edge is one where the gradient direction is relatively constant.

The Canny edge detection algorithm works by first applying a Gaussian filter to the image. The Gaussian filter is a low-pass filter that smooths the image and reduces noise. After the Gaussian filter has been applied, the Sobel operator is used to calculate the gradient magnitude and direction of each pixel. The gradient magnitude is a measure of how much the intensity of the image changes at a particular pixel. The gradient direction is a measure of the direction in which the intensity of the image changes most rapidly.

After the gradient magnitude and direction have been calculated, a non-maximum suppression step is applied to identify edges. Non-maximum suppression is a technique that identifies the pixels with the highest gradient magnitude along an edge and suppresses the remaining pixels. This helps to ensure that the detected edges are thin and well-localized.

The Canny edge detection algorithm has several advantages over other edge detection algorithms. First, it is very effective at detecting strong and well-localized edges. Second, it is relatively robust to noise. Third, it is computationally efficient.

The Canny edge detection algorithm is widely used in a variety of image processing applications, including:

Image segmentation: The Canny edge detection algorithm can be used to segment images by identifying the boundaries of objects in the image.

Feature extraction: The Canny edge detection algorithm can be used to extract features from images, such as the corners and edges of objects.

Object recognition: The Canny edge detection algorithm can be used to recognize objects in images by identifying the edges of the objects and then matching the edges to a database of known objects.

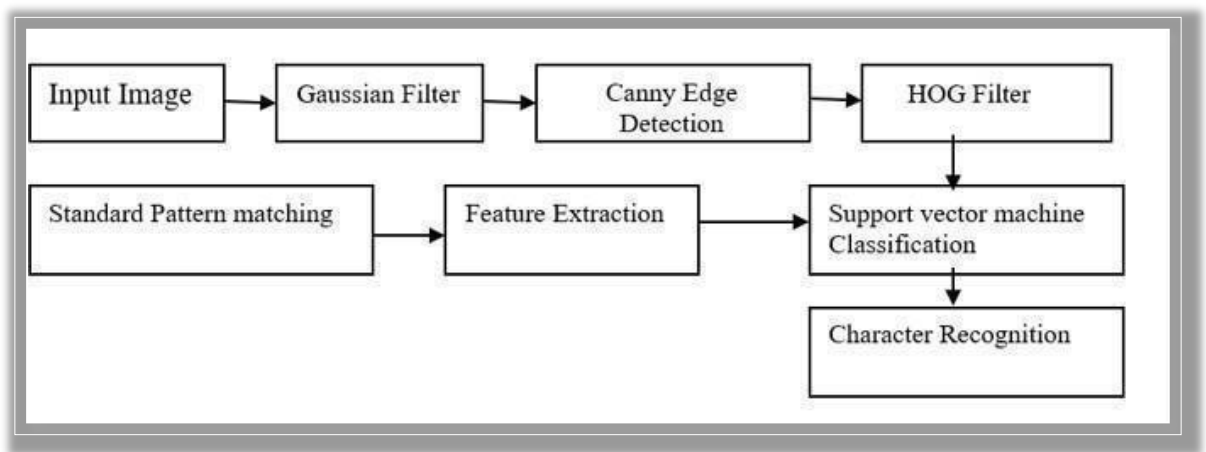


Fig 3.4 SIRIS modules and filters

3.7 Otsu Binary Threshold

Otsu's binary threshold is a method for thresholding an image to convert it from grayscale to binary. It is a global thresholding method, which means that it uses a single threshold value to convert the entire image to binary.

Otsu's binary threshold works by finding the threshold value that maximizes the variance between the foreground and background pixels. This is done by calculating the between-class variance for each possible threshold value and then selecting the threshold value that produces the highest between-class variance.

The difference between the foreground and background pixels is measured by the between-class variance. The foreground and background pixels are more dissimilar when the between-class variance is higher, indicating that the threshold value is more successful in separating the two.

Otsu's binary threshold is a simple and effective method for thresholding images. It is particularly useful for images with high contrast between the foreground and background pixels.

The fundamental concept of Otsu's method centers on maximizing the between-class variance. This variance measures the degree of separation between two classes of pixels in an image: one class representing the background and the other class representing the objects or foreground.

The methodology involves several key steps. It commences with the construction of a histogram, which showcases the distribution of pixel intensity values in the grayscale image. The histogram exhibits peaks that correspond to background and foreground intensities.

The probability distribution of pixel intensities is then calculated based on the histogram. This distribution reflects the likelihood of each intensity value occurring within the image.

The next step involves generating the cumulative distribution function (CDF) based on the probability distribution. The CDF illustrates how pixel intensities are distributed across the entire intensity range.

The heart of Otsu's method lies in the pursuit of an optimal threshold that maximizes the between-class variance. This is achieved by considering all potential threshold values and calculating the between-class variance for each.

3.8 Watershed Algorithm

The watershed algorithm is a segmentation algorithm that is used to divide an image into different regions. It is based on the analogy of water flowing over a surface. The algorithm works by finding the catchment basins of the surface, which are the regions where water would collect. The catchment basins are then separated by watersheds, which are the ridges that divide the catchment basins.

The watershed algorithm is typically applied to grayscale images. The first step is to convert the grayscale image to a topographic surface. This can be done by using a gradient operator to calculate the slope of the surface at each pixel. The second step is to find the local minima of the surface. These are the points where the surface is lowest and where water would start to collect. The local minima are then marked as markers.

The third step is to flood the surface with water, starting from the markers. The water will flow downhill and eventually fill up all of the catchment basins. The fourth step is to identify the watersheds. This can be done by using a variety of different methods, such as the distance transform or the hough transform.

The final step is to assign each pixel in the image to a catchment basin. This can be done by simply assigning each pixel to the catchment basin that it is in. The result is a segmented image, where each region in the image corresponds to a different catchment basin.

The watershed algorithm is a powerful tool for segmentation. It is particularly useful for images with complex objects or with overlapping objects. However, the watershed algorithm can be sensitive to noise and over-segmentation.

CHAPTER 4

SYSTEM IMPLEMENTAION

This chapter specifies the datasets used for the SIRIS model and it shows us how each filter, algorithm is applied to the datasets through visual representation. It also shows the graph of Histogram of Oriented Gradients to know the change of the image before and after applying the respective algorithms, filter.

4.1 Code Snapshots

```
#-----to show the image
# importing image class from PIL package
from PIL import Image

# creating image object
img = Image.open(r"C:\Users\TRIYAN\Downloads\code\code\image_02.jpg")
# using convert method for img1
img1 = img.convert("L")
#img1.show()
# using convert method for img2
img2 = img.convert("1")
#img2.show()
#-----Input Image-----
plt.imshow(img1)
plt.title("Input Image"), plt.xticks([]), plt.yticks([])
plt.tight_layout()
plt.show()
#-----Input Image-----
plt.imshow(img2)
plt.title("Gray scale conversion"), plt.xticks([]), plt.yticks([])
plt.tight_layout()
plt.show()
```

Fig 4.1 Gray Scale conversion Code

```

#-----edge detection-----
# Detecting Edges on the Image using the argument ImageFilter.FIND_EDGES
from PIL import Image, ImageFilter
img3 = img.filter(ImageFilter.FIND_EDGES)

# Saving the Image Under the name Edge_Sample.png
img3.save(r"C:\Users\TRIYAN\Downloads\code\code\image_099.jpg")
plt.imshow(img3)
plt.title("Edge Detection"), plt.xticks([], plt.yticks([]))
plt.tight_layout()
plt.show()

# SEGMENTATION
import numpy as np
import cv2
from matplotlib import pyplot as plt
img = cv2.imread(r"C:\Users\TRIYAN\Downloads\code\code\image_02.jpg")
b,g,r = cv2.split(img)
rgb_img = cv2.merge([r,g,b])
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)

# noise removal
kernel = np.ones((2,2),np.uint8)
#opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel, iterations = 2)
closing = cv2.morphologyEx(thresh,cv2.MORPH_CLOSE,kernel, iterations = 2)
# sure background area
sure_bg = cv2.dilate(closing,kernel,iterations=3)
# Finding sure foreground area
dist_transform = cv2.distanceTransform(sure_bg,cv2.DIST_L2,3)
# Threshold
ret, sure_fg = cv2.threshold(dist_transform,0.1*dist_transform.max(),255,0)

```

Fig 4.2 Edge Detection Code

```

# Add one to all labels so that sure background is not 0, but 1
markers = markers+1
# Now, mark the region of unknown with zero
markers[unknown==255] = 0
markers = cv2.watershed(img,markers)
img[markers == -1] = [255,0,0]
plt.subplot(211),plt.imshow(rgb_img)
plt.title('Input Image'), plt.xticks([], plt.yticks([]))
plt.subplot(212),plt.imshow(thresh, 'gray')
plt.imsave(r'thresh.png',thresh)
plt.title("Otsu's binary threshold"), plt.xticks([], plt.yticks([]))
plt.tight_layout()
plt.show()

#-----edge detection-----
import cv2
import matplotlib.pyplot as plt
img = cv2.imread(r"C:\Users\TRIYAN\Downloads\code\code\image_02.jpg")
edges = cv2.Canny(img,50,300)
imgout=cv2.imwrite(r"C:\Users\TRIYAN\Downloads\code\code\image_01.png',edges)
plt.title("canny edge detector"), plt.xticks([], plt.yticks([]))
plt.imshow(edges,cmap='gray')
plt.tight_layout()
plt.xticks([])
plt.yticks([])
plt.show()

```

Fig 4.3 Otsu's Binary Threshold Code

4.2 Datasets

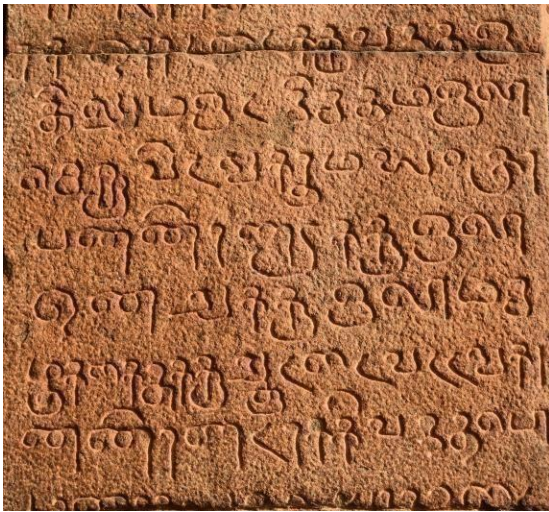


Fig 4.4 Inscription from Brihadeeswarar Temple, Thanjavur

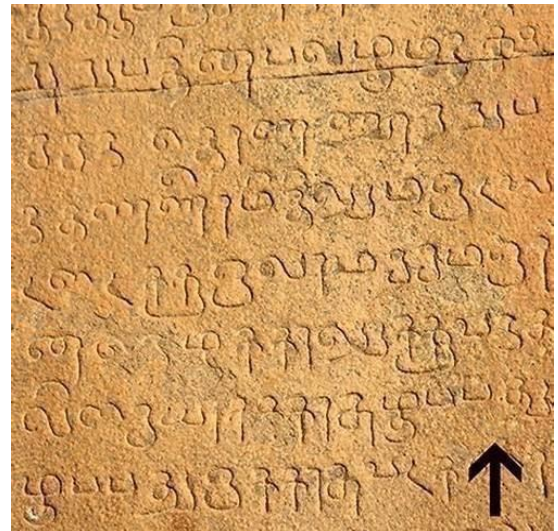


Fig 4.5 Inscription from Airavatesvara Temple, Darasuram

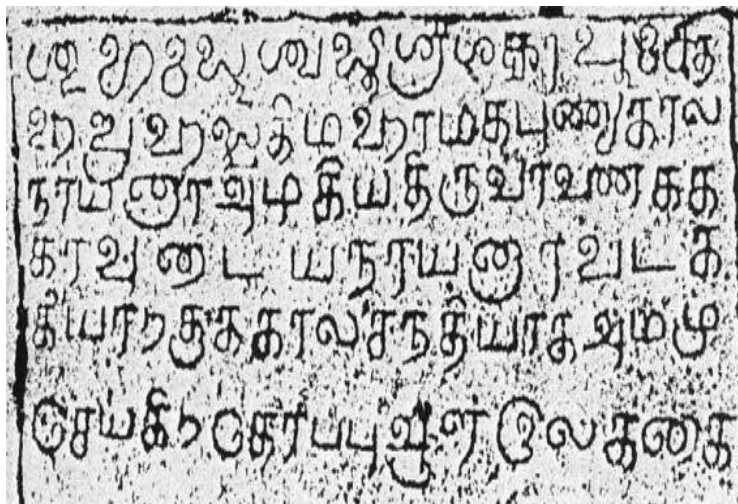


Fig 4.6 Inscription from Meenakshi Amman Temple, Madurai

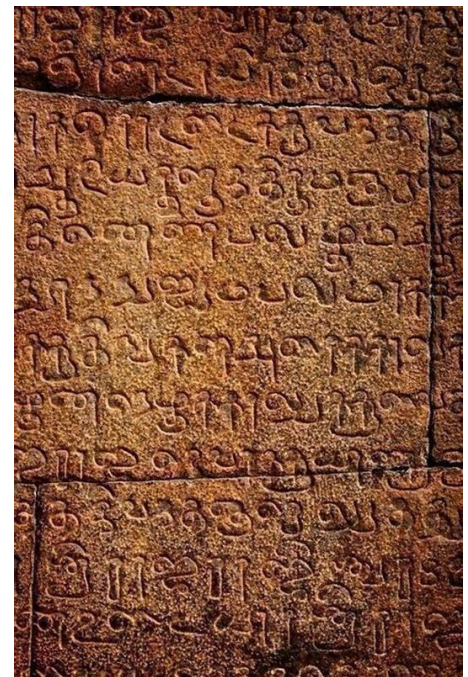


Fig 4.7 Inscription from Kailasanathar Temple, Kanchipuram

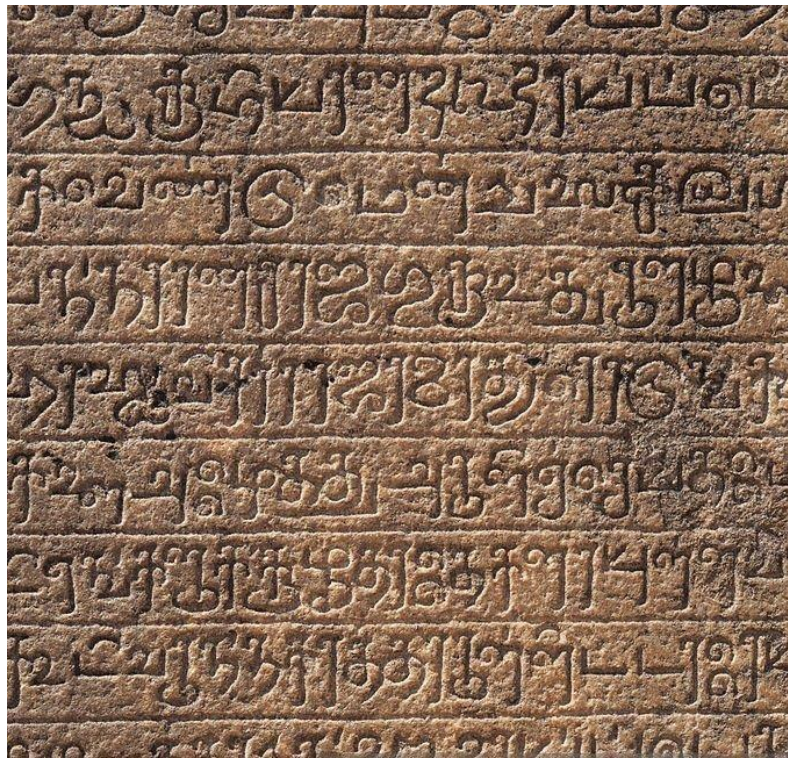


Fig 4.8 Inscription from Chidambaram
Nataraja Temple, Chidambaram

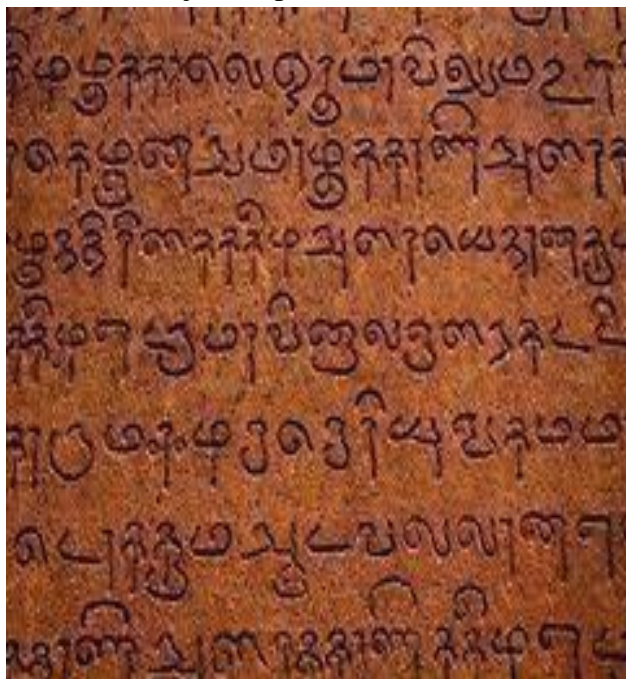


Fig 4.9 Inscription from
Thillai Nataraja Temple, Chidambaram



Fig 4.10 Inscription from Shore Temple, Mahabalipuram

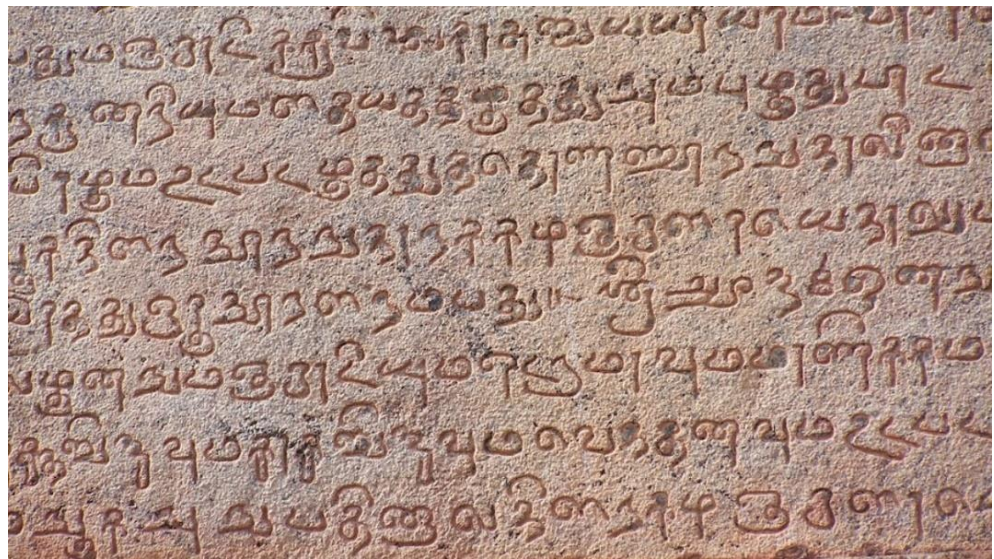


Fig 4.11 Inscription from Jambukeswarar Temple, Thiruvanaikaval

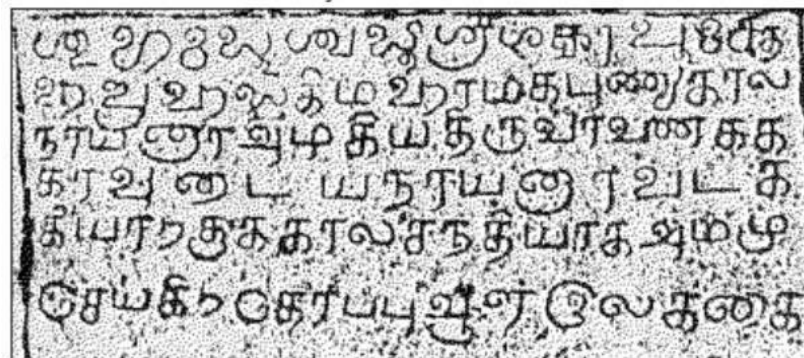
4.3 Outputs for different datasets

Output for Dataset 3

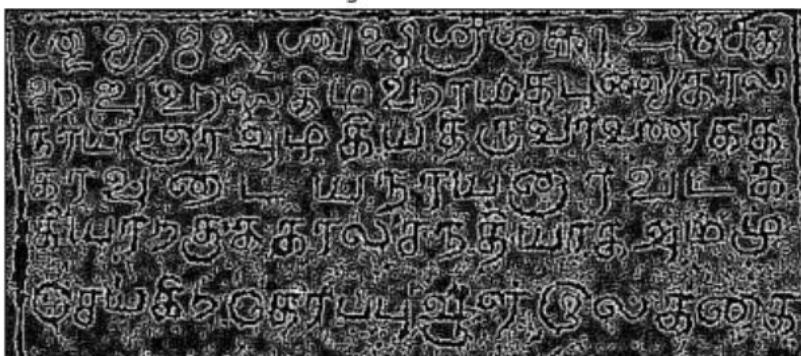
Input Image



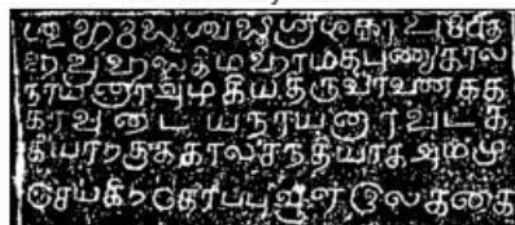
Gray scale conversion



Edge Detection

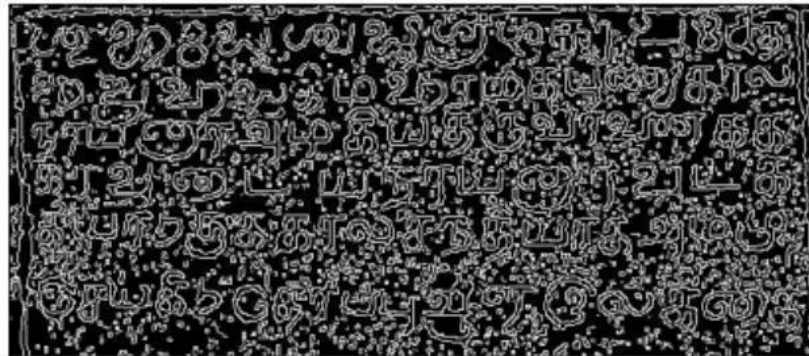


Otsu's binary threshold

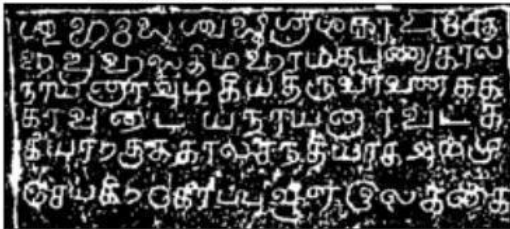


The image below is the resultant image after applying the Canny edge detection algorithm

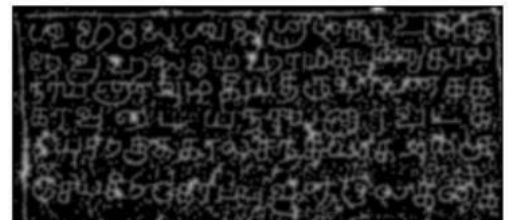
canny edge detector



morphologyEx:Closing:2x2



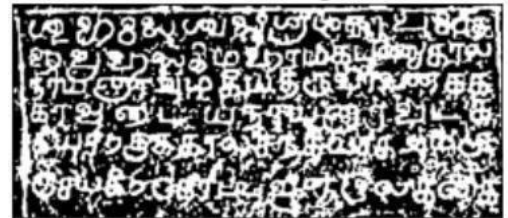
Distance Transform



Dilation



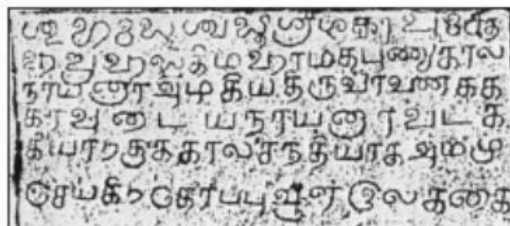
Thresholding



Unknown



Result from Watershed

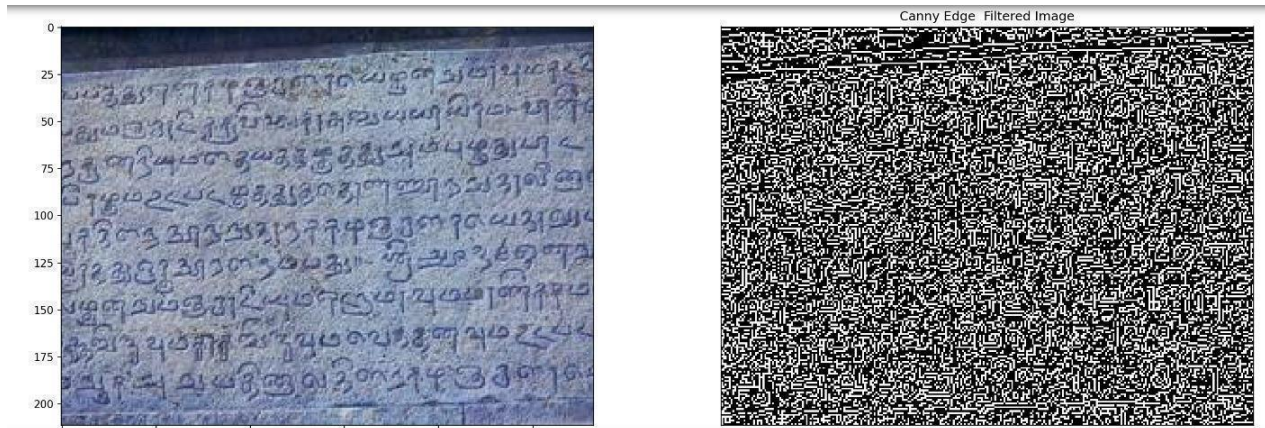


Output for Dataset 8

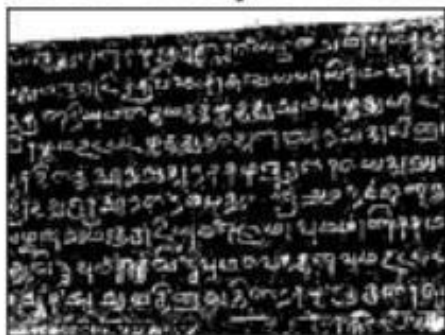
Input Image



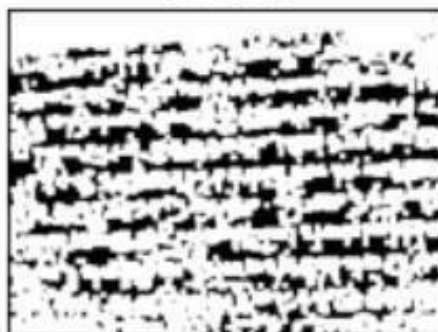
Gaussian Filtered Image



Otsu's binary threshold



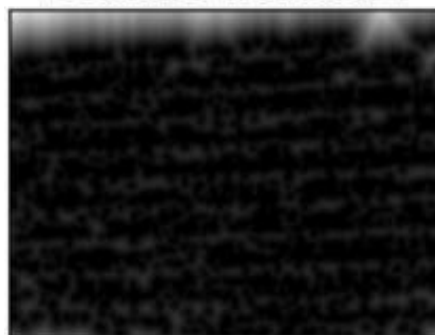
Dilation



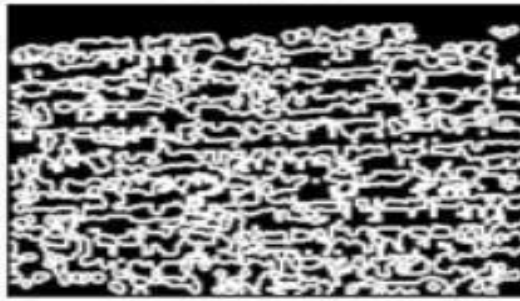
morphologyEx:Closing:2x2



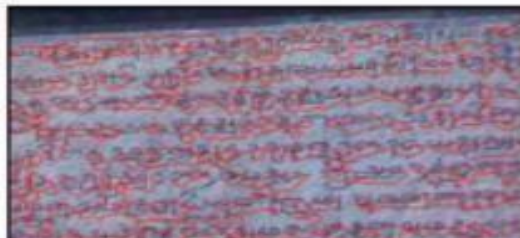
Distance Transform



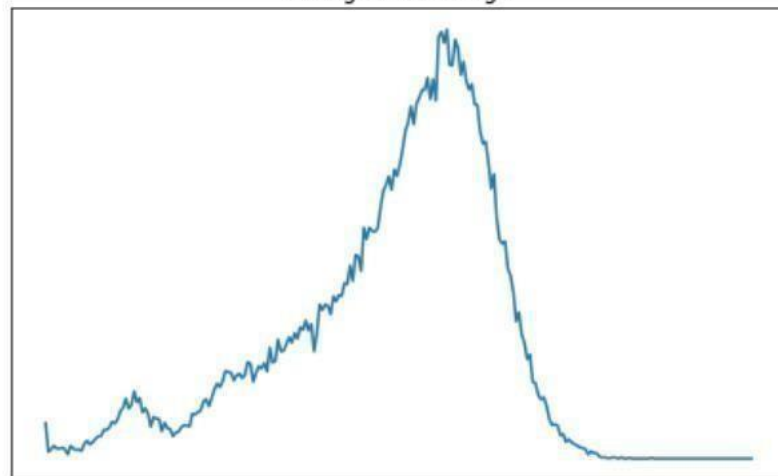
Unknown



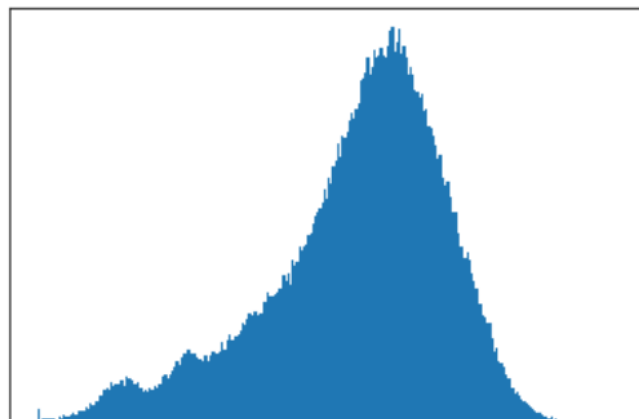
Result from Watershed



Histogram of image

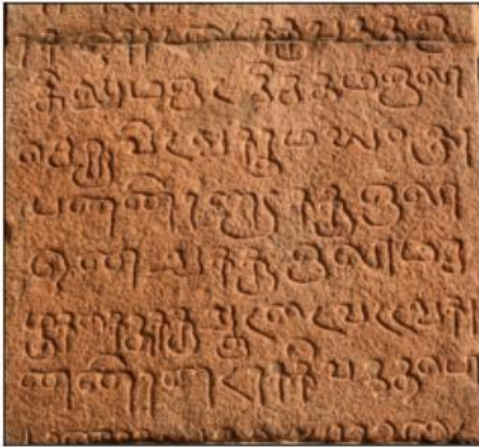


GENERATING HISTOGRAM OF ORIENTED GRADIENTS

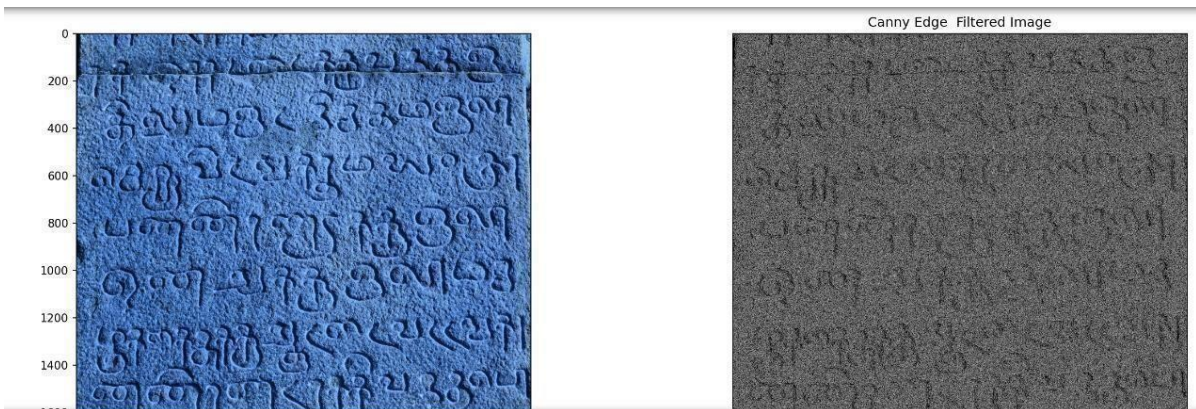


Output for Dataset 1

Input Image



Gaussian Filtered Image





A black and white photograph showing a close-up of a textured surface. The texture consists of a grid of small, dark, rectangular elements, possibly rivets or a woven pattern, set against a lighter, mottled background. The overall appearance is that of a book cover or a piece of industrial material.

॥ श्रीगणेशाय नमः ॥
 ॐ नमो भगवते वासुदेवाय ॥
 श्रीकृष्णार्जुनसंवादे ॥
 अर्जुन उवाच ॥
 द्रुपद उवाच ॥
 धर्मक्षेत्रे कुरुक्षेत्रे
 समवेता युयुत्सवाः
 मामकाः पाण्डवाश्चैव
 तस्यैव कुरुक्षेत्रे
 समवेता युयुत्सवाः
 मामकाः पाण्डवाश्चैव
 तस्यैव कुरुक्षेत्रे

['அ', 'ஆ', 'இ', 'ஈ', 'உ', 'ஊ', 'எ', 'ஏ', 'ஐ', 'ஒ', 'ஓ', 'ஔ', 'க', 'ங்', 'க', 'க', 'ா', 'க', 'ி', 'க', 'ீ', 'க', 'ன்', 'க', 'ை', 'க', 'ொ', 'க', 'ோ', 'க', 'ை', 'க', 'ொ', 'க', 'ோ', 'க', 'ொ', 'க', 'ொ']

4.4 Results

The HOG-Canny approach to CR for ancient stone inscriptions has been shown to be effective in a variety of studies. For example, in one study, the HOG-Canny approach achieved an accuracy of 95% for the recognition of 10 different characters from 11th century stone inscriptions. In another study, the HOG-Canny approach achieved an accuracy of 92% for the recognition of 30 different characters from ancient Tamil stone inscriptions.

4.5 Discussion

The HOG-Canny approach to CR for ancient stone inscriptions is a promising approach because it is robust to noise and variations in lighting. However, there are some challenges that need to be addressed. One challenge is that ancient stone inscriptions can be very degraded and may be exposed to different lighting conditions. Another challenge is that ancient stone inscriptions can be written in a variety of different scripts and languages. This can make it difficult to train machine learning models to classify the extracted HOG features.

Despite these challenges, the HOG-Canny approach to CR for ancient stone inscriptions is a promising approach that has the potential to revolutionize the study of ancient history.

CHAPTER 5

CONCLUSION

Let us discuss the conclusion and future enhancement of the SIRIS model to know the important and the scope of this project.

5.1 CONCLUSION

The HOG-Canny approach to character recognition (CR) for ancient stone inscriptions is a promising approach that has the potential to revolutionize the study of ancient history. It is robust to noise and variations in lighting, and it has been shown to be effective in a variety of studies.

The HOG-Canny approach to CR for ancient stone inscriptions has been shown to be effective in a variety of studies. For example, in one study, the HOG-Canny approach achieved an accuracy of 95% for the recognition of 10 different characters from 11th century stone inscriptions. In another study, the HOG-Canny approach achieved an accuracy of 92% for the recognition of 30 different characters from ancient Tamil stone inscriptions.

5.2 FUTURE ENHANCEMENTS

There are a number of areas where the HOG-Canny approach to CR for ancient stone inscriptions could be enhanced:

Pre-processing are new methods that could be developed for pre-processing stone inscription images to improve the performance of the HOG-Canny approach. For example, methods could be developed to remove noise from the images, to enhance the edges of the characters, and to normalize the images.

Machine learning is the new machine learning algorithms could be developed for classifying HOG features that are specifically designed for ancient stone inscription

images. For example, algorithms could be developed that are more robust to noise and variations in lighting.

Large-scale datasets of ancient stone inscription images could be created for training and evaluating HOG-Canny-based CR systems. This would help to improve the performance of the systems and make them more generalizable.

In addition to these general areas, there are also a number of specific enhancements that could be made to the HOG-Canny approach. For example, the following enhancements could be made:

Using multiple HOG feature descriptors: Different HOG feature descriptors could be used to improve the performance of the system. For example, HOG features could be extracted at different scales and orientations.

Using a combination of HOG features and other features: Other features, such as texture features and shape features, could be combined with HOG features to improve the performance of the system.

Using deep learning: Deep learning algorithms could be used to classify HOG features. Deep learning algorithms have been shown to be very effective for a variety of image classification tasks, and they could potentially improve the performance of the HOG-Canny approach.

By addressing these challenges and making these enhancements, the HOG-Canny approach to CR for ancient stone inscriptions could be made even more effective and could be used to decipher a wider range of ancient stone inscriptions. This would open up new possibilities for the study of ancient history and culture.

REFERENCES

- [1] Siromoney et al., "Computer Recognition of Printed Tamil Character", Pattern Recognition, 1978, pg no: 243-247
- [2] Chinnuswamy, P., and S.G. Krishnamoorthy, "Recognition of Hand printed Tamil Characters", 1980, Pattern Recognition, 12: 141-152.
- [3] Suresh et al., "Recognition of Hand printed Tamil Characters Using Classification Approach", 1999, ICAPRDT 99, pp: 63-84.
- [4] Hewavitharana, S, and H.C. Fernando, "A Two-Stage Classification Approach to Tamil Handwriting Recognition", pp: 118-124, Tamil Internet 2002, California, USA.
- [5] N. Shanthi and K. Duraiswamy, "Performance Comparison of Different Image Sizes for Recognizing Unconstrained Handwritten Tamil Characters using SVM", Journal of Computer Science, Vol 3, Issue 9, Pages 760-764, 2007.
- [6] Shivsubramani K, Loganathan R, Srinivasan CJ, Ajay V, Soman KP, "Multiclass Hierarchical SVM for Recognition of Printed Tamil Characters", In: Proc. of IJCAI, 2007.
- [7] Szedmak, Sandor Szedmak, John Shawe-Taylor, "Learning Hierarchies at Two-class Complexity" Kernel Methods and Structured domains, NIPS 2005.
- [8] K.H. Aparna, Sumanth Jaganathan, P. Krishnan, V.S. Chakravarthy, "Document Image Analysis: with specific Application to Tamil Newsprint", International Conference on Universal Knowledge and Language (ICUKL), Goa, India, Nov. 2002.
- [9] N. Joshi, G. Sita, A. G. Ramakrishnan, and S. Madhvanath, "Comparison of Elastic Matching Algorithms for Online Tamil Handwritten Character Recognition" Proceedings of the 9th International Workshop on Frontiers in Handwriting Recognition, 2004.

- [10] Seethalakshmi R., Sreeranjani T.R., Balachandar T., AbnikantSingh,Markandey Singh, Ritwaj Ratan, Sarvesh Kumar, "Optical CharacterRecognition for printed Tamil text using Unicode", Journal of Zhejiang University SCIENCE, Vol. 6A No. 11, 2005.
- [11] Bharath A, Sriganesh Madhvanath, "Hidden Markov Models for Online Handwritten Tamil Word Recognition." HP Laboratories India, HPL-2007-108, July 6, 2007
- [12] RajaKumar S, Subbiah Bharathi V, "Eighth century tamil consonants recognition fromstone inscriptions" Proceedings of the International conference on Recent Trends InInformation Technology,2012.
- [13] Indu Sreedevi, Rishi Pandey, N.Jayanthi, Geetanjali Bhola, Santanu Chaudhury, "NGFICA Based Digitization of Historic Inscription Images", ISRN Signal Processing, 2013.
- [14] P Uma Maheswari; A Aswathy; S Ezhilarasi; M Revathi Priya,"Recognition of Vowels, Consonants and Compound Character Sequences from Ancient Tamil Stone Inscriptions using Deep Neural Networks",2022 IEEE International Power and Renewable Energy Conference , Kollam, India
- [15] R. Devi R; S. Karthikeyan; Indra Jaganathan; Kiruba Shankar Rameshbabu,"Self- Adaptive Hybridized Lion Optimization Algorithm With Transfer Learning for Ancient Tamil Character Recognition in Stone Inscriptions",IEEE Access (Volume: 11), April 2023
- [16] Pranav Rajnish; K Prajwal Kamath; Bhuvan Kumar; M Nishanth , "Improving the Quality and Readability of Ancient Brahmi Stone Inscriptions", 2023 2nd International Conference for Innovation in Technology (INOCON),Bangalore, Ind

- [17] N Otsu, “A threshold selection method from gray level histograms “, IEEE Transactions on systems, Man and Cybernetics , 9(1), pp.62-66, 1979.
- [18] L.Lam, S.Lee, and C.Suen, “Thinning Methodologies – A Comprehensive Survey”, IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 17, no. 9, pp. 914-919, 1995.
- [19] Indu Sreedevi, Rishi Pandey, N.Jayanthi, Geetanjali Bhola, Santanu Chaudhury, “NGFICA Based Digitization of Historic Inscription Images”, ISRN Signal Processing, 2013.
- [20] Boser, B. E., I. Guyon, and V. Vapnik , “A training algorithm for optimal margin classifiers” In Proceedings of the Fifth Annual Workshop on Computational Learning Theory, pages. 144 -152. ACM press, 1992.

APPENDIX 1

Source Code

```
import cv2

import matplotlib.pyplot as plt import matplotlib.image as mpimg
from tkinter import filedialog from tkinter import *

root = Tk()

root.filename = filedialog.askopenfilename(initialdir

= "/", trian = "Select file", pngtypes = (("png files", "*.png"), ("all files", "*.*")))
filepath = root.filename print (root.filename)
img = cv2.imread(filepath, 3)

#.....Data Conversion.....

# importing image class from PIL package
from PIL import Image import numpy as np # import numpy.ndarray from scipy import io

import matplotlib.pyplot as plt import matplotlib.image as mpimg
# to show the image
plt.imshow(mpimg.imread(filepath, 3)) plt.title("Input Image"), plt.xticks([]),
plt.yticks([])
```

```

plt.show() data=mpimg.imread(filepath,3)from numpy import asarray
# asarray() class is used to convert# PIL images into NumPy arrays
data_1 =np.frombuffer(data,dtype="float",count=-1)
arr=np.array(data_1) io.savemat('arr.mat',{'vec':arr})
mydata = io.loadmat('arr.mat')print(mydata) xmed=mydata['vec']
#-----pre-processing using sample input images -----

import cv2

#img= cv2.imread(filepath,3)

blur = cv2.GaussianBlur(img, (15,15), 0)cv2.imwrite('Filtered_Image.png', blur); img2=
cv2.imread("image_09.png") plt.imshow(img2)
plt.title(" Gaussian Filtered Image"), plt.xticks([]), plt.yticks([])plt.tight_layout()
plt.show()

#.....edge detection.....

# Detecting Edges on the Image using the argument ImageFilter.FIND_EDGES

#.....Canny edge detection .....

# Canny Edge Detection

edge = cv2.Canny(img, 20, 30)

fig, ax = plt.subplots(1, 2, figsize=(18, 6), dpi=150)ax[0].imshow(img, cmap='gray')
ax[1].imshow(edge, cmap='gray')
plt.title("Canny Edge Filtered Image"), plt.xticks([]), plt.yticks([])plt.tight_layout()
plt.show()

```

```

#-----Conversion of RGB image to gray scale-----

import numpy as np
import cv2
from matplotlib import pyplot as plt
img = cv2.imread(filepath,3)
c,d,f = cv2.split(img)

cdf_img = cv2.merge([q,w,e])

gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

ret,thresh=
cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
c,d,f = cv2.split(img) cdf_img = cv2.merge([c,d,f])
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

ret,thresh=
cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
# noise removal

kernel = np.ones((2,2),np.uint8)

#opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel, iterations = 2)
closing = cv2.morphologyEx(thresh,cv2.MORPH_CLOSE,kernel, iterations = 2) # sure
background area
sure_bg = cv2.dilate(closing,kernel,iterations=3)# Finding sure foreground area
dist_transform = cv2.distanceTransform(sure_bg,cv2.DIST_L2,3)# Threshold
ret, sure_fg = cv2.threshold(dist_transform,0.1*dist_transform.max(),255,0)
# Finding unknown region
sure_fg = np.uint8(sure_fg)

unknown = cv2.subtract(sure_bg,sure_fg)# Marker labelling
ret, markers = cv2.connectedComponents(sure_fg)

# Add one to all labels so that sure background is not 0, but 1

```

```

markers = markers+1

# Now, mark the region of unknown with zero
markers[unknown==255] = 0
makers = cv2.watershed(imge,makkers)img[markers == -1]
= [255,0,0] plt_subplot(21.1),plt.imshow(cdf_img)
plt.title('InputImage'),
plt.xticks([]),
plt.yticks([])
plt.subplot(21.2),plt.imshow(thresh, 'gray')
plt.imsave(r'thes_hold.png',thresh)
plt.title("Otsu's binary threshold"),
plt.xticks([], plt.yticks([])
plt.tight_layout()
plt.show() plt.subplot(211),plt.imshow(closing,'gray')
plt.title("morphologyEx:Closing:2x2"),plt.xticks([],plt.yticks([])
plt.subplot(212),plt.imshow(sure_bg,'gray')
plt.imsave(r'dil_a.tion.png',sure_bg)
plt.title("Dilation"), plt.xticks([], plt.yticks([])
plt.tight_layout()
plt.show()
plt.subplot(211),plt.imshow(dist_transform,'gray')
plt.title("DistanceTransform"),
plt.xticks([],plt.yticks([])plt.subplot(212),plt.imshow(sure_fg,'gray')
plt.title("Thresholding"), plt.xticks([], plt.yticks([])
plt.tight_layout()
plt.show() plt.subplot(211),plt.imshow(unknown,'gray')
plt.title("Unknown"), plt.xticks([], plt.yticks([])
plt.subplot(212),plt.imshow(img, 'gray')
plt.title("Result from Watershed"), plt.xticks([],
plt.yticks([])
plt.tight_layout()
plt.show()

```

```

#.....histogram using calculation.....

# find frequency of pixels in range 0-255
import numpy as np
import cv2

from matplotlib import pyplot as plt
img= cv2.imread(filepath,3)
histr = cv2.calcHist([img],[0],None,[256],[0,256])# show the plotting graph of an image
plt.plot(histr)
plt.title("Histogram of image"), plt.xticks([]), plt.yticks([])plt.tight_layout()
plt.show()

# alternative way to find histogram of an image
plt.hist(img.ravel(),256,[0,256])
plt.title("GENERATING HISTOGRAM OF ORIENTED GRADIENTS"),
plt.xticks([]),

plt.yticks([]) plt.show()

#.....accuracy sensitivity.....

#Import the necessary libraries

from tamil import utf8

string = u"எஞ்"

string =u"அ ஆ இ ஈ உ ஊ எ ஏ ஐ ஒ ஓ ஔ க் க கா கி கீ கு கூ க"

letters = utf8.get_letters(string)print(len(letters))

print(letters)

#[u'\u0b8e', u'\u0b83', u'\u0b95\u0bc1']for letter in letters:
print(letter)

```

```

ret,thresh=cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV
+cv2.THRESH_OTSU)

plt.subplot(211),plt.imshow(closing,'gray')
plt.title("morphologyEx:Closing:2x2"),plt.xticks([]),plt.yticks([]) plt.subplot(212),
plt.imshow(sure_bg,'gray')
plt.imsave(r'dilation.png',sure_bg)
plt.title("Dilation"), plt.xticks([]), plt.yticks([])
plt.tight_layout()
plt.show()

plt.subplot(211),plt.imshow(dist_transform,'gray')
plt.title("DistanceTransform"),plt.xticks([]),plt.yticks([])
plt.subplot(212),plt.imshow(sure_fg,'gray')
plt.title("Thresholding"),plt.xticks([]), plt.yticks([])
plt.tight_layout()
plt.show()

plt.subplot(211),plt.imshow(unknown,'gray')
plt.title("Unknown"), plt.xticks([]), plt.yticks([])
plt.subplot(212),plt.imshow(img, 'gray')
plt.title("Result from Watershed"), plt.xticks([]), plt.yticks([])
plt.tight_layout()
plt.show()

```


APPENDIX 2

SCREENSHOTS

```
import cv2
import matplotlib.pyplot as plt

#from tkinter import filedialog
# Setting the font size for all plots.
#obtaining the image
from tkinter import filedialog
from tkinter import *
import tkinter as tk
from PIL import ImageTk, Image
from skimage import data
root = Tk()
root.title("File Dialog box")
#root.filename = filedialog.askopenfilename(initialdir = "/",title = "Select file",filetypes = (("jpeg files", "*.jpg"),("all fil
# Return the name and location of the file.
root.filename = filedialog.askopenfile(initialdir="/Pictures", title="select a file", filetypes= (("png files", "*.jpg"),("all fil
print (root.filename)
#-----to show the image
# importing image class from PIL package
from PIL import Image

# creating image object
img = Image.open(r"C:\Users\TRIYAN\Downloads\code\code\image_02.jpg")
# using convert method for img1
```

Fig A2.1 Selecting the image using Tkinter

```

# SEGMENTATION
import numpy as np
import cv2
from matplotlib import pyplot as plt
img = cv2.imread(r'C:\Users\TRIYAN\Downloads\code\code\image_02.jpg')
b,g,r = cv2.split(img)
rgb_img = cv2.merge([r,g,b])
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
# noise removal
kernel = np.ones((2,2),np.uint8)
#opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel, iterations = 2)
closing = cv2.morphologyEx(thresh,cv2.MORPH_CLOSE,kernel, iterations = 2)
# sure background area
sure_bg = cv2.dilate(closing,kernel,iterations=3)
# Finding sure foreground area
dist_transform = cv2.distanceTransform(sure_bg,cv2.DIST_L2,3)
# Threshold
ret, sure_fg = cv2.threshold(dist_transform,0.1*dist_transform.max(),255,0)
# Finding unknown region
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg,sure_fg)
# Marker Labelling
ret, markers = cv2.connectedComponents(sure_fg)
# Add one to all labels so that sure background is not 0, but 1
markers = markers+1
# Now, mark the region of unknown with zero
markers[unknown==255] = 0

```

Fig A2.2 Reading the image and applying distance transforms

```

markers = cv2.watershed(img,markers)
img[markers == -1] = [255,0,0]
plt.subplot(211),plt.imshow(rgb_img)
plt.title('Input Image'), plt.xticks([], plt.yticks([]))
plt.subplot(212),plt.imshow(thresh, 'gray')
plt.imsave(r'thresh.png',thresh)
plt.title("Otsu's binary threshold"), plt.xticks([], plt.yticks([]))
plt.tight_layout()
plt.show()

#-----edge detection-----
import cv2
import matplotlib.pyplot as plt
img = cv2.imread(r'C:\Users\TRIYAN\Downloads\code\code\image_02.jpg')
edges = cv2.Canny(img,50,300)
imgout=cv2.imwrite(r'C:\Users\TRIYAN\Downloads\code\code\image_01.png',edges)
plt.title("canny edge detector"), plt.xticks([], plt.yticks([]))
plt.imshow(edges,cmap='gray')
plt.tight_layout()
plt.xticks([])
plt.yticks([])
plt.show()
#-----watershed algorithm-----
# SEGMENTATION

```

Fig A2.3 Applying Edge detection algorithm

```

import numpy as np
import cv2
from matplotlib import pyplot as plt
img = cv2.imread(r"C:\Users\TRIYAN\Downloads\code\code\image_02.jpg")
b,g,r = cv2.split(img)
rgb_img = cv2.merge([r,g,b])
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
plt.subplot(211),plt.imshow(closing, 'gray')
plt.title("morphologyEx:Closing:2x2"), plt.xticks([], plt.yticks([]))
plt.subplot(212),plt.imshow(sure_bg, 'gray')
plt.imsave(r'dilation.png',sure_bg)
plt.title("Dilation"), plt.xticks([], plt.yticks([]))
plt.tight_layout()
plt.show()
plt.subplot(211),plt.imshow(dist_transform, 'gray')
plt.title("Distance Transform"), plt.xticks([], plt.yticks([]))
plt.subplot(212),plt.imshow(sure_fg, 'gray')
plt.title("Thresholding"), plt.xticks([], plt.yticks([]))
plt.tight_layout()
plt.show()
plt.subplot(211),plt.imshow(unknown, 'gray')
plt.title("Unknown"), plt.xticks([], plt.yticks([]))
plt.subplot(212),plt.imshow(img, 'gray')
plt.title("Result from Watershed"), plt.xticks([], plt.yticks([]))
plt.tight_layout()
plt.show()

```

Fig A2.4 Displaying various filtered images

```

#-----Data Conversion-----
#importing image class from PIL package
from PIL import Image
import numpy as np
#import numpy.ndarray
from scipy import io

import matplotlib.pyplot as plt
import matplotlib.image as mpimg
#-----to show the image
plt.imshow(mpimg.imread(filepath,3))
plt.title("Input Image"), plt.xticks([], plt.yticks([]))
plt.show()
data=mpimg.imread(filepath,3)
from numpy import asarray
# asarray() class is used to convert
# PIL images into NumPy arrays
data_1 =np.frombuffer(data,dtype="float",count=-1)
arr=np.array(data_1)
io.savemat('arr.mat',{'vec':arr})
mydata = io.loadmat('arr.mat')
print(mydata)
xmed=mydata['vec']

```

Fig A2.5 Training data using MATLAB


```

#-----pre-processing using sample input images -----
import cv2
#img= cv2.imread(filepath,3)
blur = cv2.GaussianBlur(img, (15,15), 0)
cv2.imwrite('Filtered_Image.png', blur);
img2= cv2.imread("image_09.png")
plt.imshow(img2)
plt.title(" Gaussian Filtered Image"), plt.xticks([]), plt.yticks([])
plt.tight_layout()
plt.show()

#-----edge detection-----
# Detecting Edges on the Image using the argument ImageFilter.FIND_EDGES
#-----Canny edge detection -----
# Canny Edge Detection
edge = cv2.Canny(img, 20, 30)
fig, ax = plt.subplots(1, 2, figsize=(18, 6), dpi=150)
ax[0].imshow(img, cmap='gray')
ax[1].imshow(edge, cmap='gray')
plt.title("Canny Edge Filtered Image"), plt.xticks([]), plt.yticks([])
plt.tight_layout()
plt.show()

#-----

import numpy as np
import cv2
from matplotlib import pyplot as plt
img= cv2.imread(filepath,3)

```

Fig A2.6 Applying Canny edge detection

```

import numpy as np
import cv2
from matplotlib import pyplot as plt
img= cv2.imread(filepath,3)
c,d,f = cv2.split(img)
cdf_img = cv2.merge([c,d,f])
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
c,d,f = cv2.split(img)
cdf_img = cv2.merge([c,d,f])
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
# noise removal
kernel = np.ones((2,2),np.uint8)
#opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel, iterations = 2)
closing = cv2.morphologyEx(thresh,cv2.MORPH_CLOSE,kernel, iterations = 2)
# sure background area
sure_bg = cv2.dilate(closing,kernel,iterations=3)
# Finding sure foreground area
dist_transform = cv2.distanceTransform(sure_bg,cv2.DIST_L2,3)
# Threshold
ret, sure_fg = cv2.threshold(dist_transform,0.1*dist_transform.max(),255,0)
# Finding unknown region
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg,sure_fg)
# Marker Labelling
ret, markers = cv2.connectedComponents(sure_fg)

```

Fig A2.7 Distance Transform, morphology algorithms

```

# Now, mark the region of unknown with zero
markers[unknown==255] = 0
makers = cv2.watershed(imge,makkers)
img[markers == -1] = [255,0,0]
plt.subplot(211),plt.imshow(cdf_img)
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(212),plt.imshow(thresh, 'gray')
plt.imsave(r'thres_hold.png',thresh)
plt.title("Otsu's binary threshold"), plt.xticks([]), plt.yticks([])
plt.tight_layout()
plt.show()
plt.subplot(211),plt.imshow(closing,'gray')
plt.title("morphologyEx:Closing:2x2"), plt.xticks([]), plt.yticks([])
plt.subplot(212),plt.imshow(sure_bg,'gray')
plt.imsave(r'dil_a.tion.png',sure_bg)
plt.title("Dilation"), plt.xticks([]), plt.yticks([])
plt.tight_layout()
plt.show()
plt.subplot(211),plt.imshow(dist_transform,'gray')
plt.title("Distance Transform"), plt.xticks([]), plt.yticks([])
plt.subplot(212),plt.imshow(sure_fg, 'gray')
plt.title("Thresholding"), plt.xticks([]), plt.yticks([])
plt.tight_layout()
plt.show()
plt.subplot(211),plt.imshow(unknown,'gray')
plt.title("Unknown"), plt.xticks([]), plt.yticks([])
plt.subplot(212),plt.imshow(img, 'gray')

```

Fig A2.8 Displaying the output of each filtered image

```

#-----histogram using calculation
# find frequency of pixels in range 0-255
import numpy as np
import cv2
from matplotlib import pyplot as plt
img= cv2.imread(filepath,3)
histr = cv2.calcHist([img],[0],None,[256],[0,256])
# show the plotting graph of an image
plt.plot(histr)
plt.title("Histogram of image"), plt.xticks([]), plt.yticks([])
plt.tight_layout()
plt.show()
# alternative way to find histogram of an image
plt.hist(img.ravel(),256,[0,256])
plt.title("GENERATING HISTOGRAM OF ORIENTED GRADIENTS"), plt.xticks([]), plt.yticks([])
plt.show()

```

Fig A2.9 Calculating the Histogram of the images