Interim Report

Level 4

Context-Aware Damage Detection and Text Restoration in Sinhala Handwritten Documents

Group Name: Reformers

Faculty of Information Technology

University of Moratuwa

2025

Interim Report

Level 4

Context-Aware Damage Detection and Text Restoration in Sinhala Handwritten Documents

Group Name: Reformers

Group Members

Index Number	Name
204009V	**Athukorala D.A.Y.S**
204044X	Disara M.P.A
204065L	Gunathilaka M.D.K.L
204190N	Sankalpana B.L.P

Supervisor: Dr. L. Ranathunga

Faculty of Information Technology

University of Moratuwa

2025

Declaration

We declare that this thesis is our own work and has not been submitted in any form for another degree or diploma at any university or other institution of tertiary education. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

Acknowledgements

First of all, we would like to express our sincere gratitude to our research supervisor, Dr.(Mr.)L.Ranathunga, for his invaluable guidance, continuous support, and insightful feedback throughout this research. His expertise and encouragement has always been instrumental in shaping the direction of this study.

We would like to thank our friends, family and all those who assisted us in collecting damaged Sinhala handwritten resources to create a sound dataset for our research. We are grateful for the guidance and mentorship Dr. L.Ranathunga provided us through his Digital Image Processing(DIP) course module, which laid the basic foundation for our research work and assisted us to kick start with a great impression about image processing, and made us confident working with them.

We are also thankful for the academic and non-academic staff at the Faculty of Information Technology,University of Moratuwa, for their support provided throughout. We are especially grateful to Dr.Ranathunga and the lab staff for letting us use the Multimedia Lab of the Faculty of IT for scanning the Sinhala handwritten documents.Furthermore, we would like to extend us our sincere gratitude to all our relatives and well wishers for extending us their support in many ways.

Abstract

Sinhala is the primary language of Sinhalese who are the largest ethnic group of Sri Lanka. From the ancient times itself the documents, books, written by philosophers, teachers, leaders, students, lyricists and artists are primarily written by hand. Over years, when files are been piled up in boxes, offices,etc, they get subjected to a lot of damage, wear and tear and insect damages which makes it very difficult to preserve them for the future.Hence, it mirrors forth the importance of taking measures to reconstruct damaged Sinhala handwritten documents and preserve them.Although similar research have been done in other countries for English language, there are a very less number of studies carried out for the restoration of Sinhala texts. The unique Sinhala script, with rounded characters and complex structure, makes restoration difficult by making the restoration techniques used for English unsuitable for Sinhala.This report includes a descriptive strategy to reconstruct damaged Sinhala handwritten documents, with the aid of image processing, Natural Language Processing (NLP) and Machine Learning(ML) techniques.A group of four members try to achieve this goal by dividing this project into four sub modules,where each member implements a one out of the four modules.The first module is to detect damaged area in handwritten Sinhala documents, and label them on what damage type it belongs to (ink blotches, wear and tear , blurred text , insect attacks, peeled off areas). The second module involves classifying detected damage into minor or contextual and estimating the extent of missing content within each damaged area whereas the third module restores minor damages based on the damage type identified by the first module (eg: ink blotches, wear and tear , insect attacks, peeled off areas) and restores blurred parts of words, to improve the document's readability.Finally, the fourth module reconstructs larger, contextually significant missing sections, including entire letter, words and phrases, ensuring coherence with the surrounding text.

Table of Content

Chapter 1 - Introduction

1.1 Introduction

Sri Lanka has a rich history that dates back thousands of years and continues to this day in the 21st century. The ancient ancestors dealt with several languages in advancing the culture, including Pali and Sanskrit, which is believed to be the language that gave rise to Sinhala. With most of its letters being rounded, Sinhala writing differs greatly from most other languages used today. Sinhala is not for the faint-hearted; only a very motivated individual can learn to read and write it.

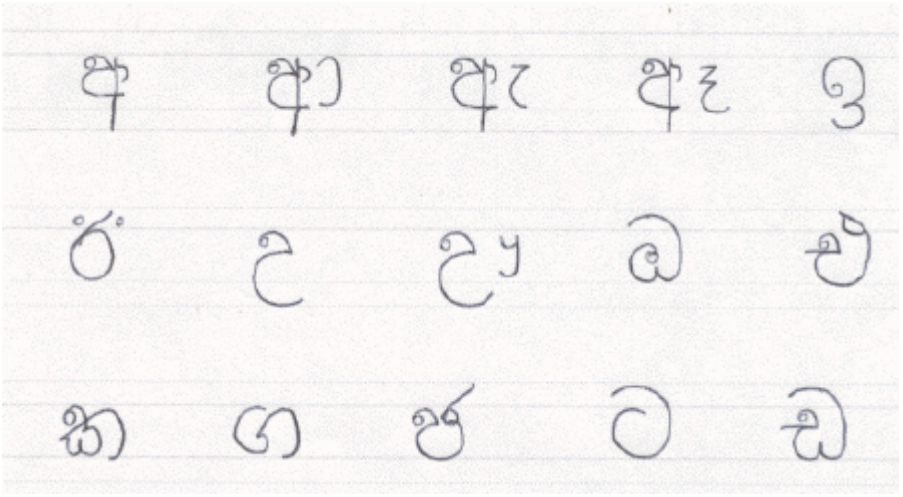


Fig: 1: A few Sinhala handwritten letters

A significant portion of our historical knowledge is derived from numerous old books, pamphlets, manuscripts, and chronicles authored by our ancestors. A large number of them are accessible in Sinhala. Unfortunately, insect attacks, fading handwritten ink, wear and tear, and ink blotches cause valuable documents to be destroyed and content to be lost. It might be difficult to reconstruct handwritten Sinhala documents after they have been destroyed while maintaining their value. The project that has been suggested could be presented as a solution in this specific situation.

The project's goal is to assist in resolving some of the problems pertaining to the damage restoration of handwritten Sinhala documents, which are crucial to the preservation of scholarly materials and can also aid in preserving historical records and legal documents as well. This includes estimating the missing text, narrowing the damaged areas, and returning the document to its initial state through the application

of Digital Image Processing(DIP) and NLP techniques. Using a customized algorithm to detect the damaged letters and words, this project extends towards the restoration of missing content with the help of image processing and reconstruction of missing words and phrases using Natural Language Processing(NLP).

It's crucial to restore a handwritten document that has been damaged for several reasons. It aids in the preservation of scholarly materials,historical and cultural records, which are essential for helping present and future generations comprehend and learn from the past. These documents can then be made available online for global access, study, and learning after they have been restored. This encourages the exchange of knowledge, making these priceless resources available to a far wider audience.

1.2 Background and Motivation

The motivation stems from the need to restore Sinhala handwritten documents, which contain culturally significant information relevant to many aspects of ancient Sri Lanka. As Sri Lanka is the only country that uses Sinhala as a native language, it is obvious that it still has very little research done. The lack of technology and implementation to work with Sinhala letters stressed the importance of this initiative even more.

Regarding the research that was previously done for restoration of damaged Sinhala written text using image processing, the necessity to address the limitations they have faced was felt important. In that particular research, a green colored background was placed to detect the damaged part of a particular letter, which cannot be considered a perfect approach for damage detection. Moreover, the studies have only addressed the restoration of a part of the letter by linking broken edges using techniques. While this method achieved a reasonable level of accuracy it was primarily focused on reconstructing parts of single letters and could not restore entire words or phrases. Additionally, the accuracy of these techniques decreased when handling complex characters or 90-degree angled letters. Moreover, this research is entirely image processing-based and this approach, while useful for reconstructing small segments, cannot infer missing textual content in complex or large-scale damages where significant portions of text are missing. It also fails to incorporate contextual

understanding, leading to errors when reconstructing complex letters or phrases. [1],[2]

Apart from that, handwritten documents often tend to deteriorate over time due to environmental factors, aging, and mishandling. This project seeks to develop a custom damage detection and restoration pipeline tailored for handwritten documents, improving the accuracy and quality of restoration. By developing an automated damage detection and restoration approach, this project aims to improve the process of document preservation and make valuable information more accessible.

1.3 Problem In Brief

The project concerns the development of a system able to identify the location and restore damaged portions within Sinhala handwritten documents. Handwritten documents can be destroyed by ink spills, tears, bug damage, and fading due to aging, environmental factors, and improper handling. Due to this fact, much valuable information has been lost, which is very hard to maintain about the cultural, historical, and legal values of the documents.

The problem mainly decomposes into two parts: finding the location of the damaged areas in this document correctly is one part. In other words, it is the detection of edges of the damaged spots-for example, small torn-out pieces, lack of ink, or faded areas-without necessarily needing human intervention. The task requires a robust identification system for damages that should handle multiple types of damages while recognizing damaged portions from undamaged portions of the document.

Once the location of the damages has been found, the next task is to determine what is missing in these places. That is, by observing the document's layout, such as the distance between the letters, the length of the words, and the space between paragraphs, the system can guess what text might be missing. The system needs to find out whether the damage hurts just one letter, part of a word, or even several phrases. The important thing is to make the missing content cohere appropriately with its surrounding text, both semantically and syntactically.

The models used in this research, such as transformers, use the surrounding context of the text to fill in the missing words or phrases by merely guessing. The solution

ensures that the restored text fits within the document making sure that it delivers the proper meaning and that it is clear and accurate.

This method is different from regular image processing techniques. It aims to provide a smarter system that understands the context for fixing damaged handwritten documents, using advanced image processing techniques and NLP techniques to improve the accuracy of outputs making sure they are saved for future use.

1.4 Aim and Objectives

1.4.1 Aim

Develop an automated system for detecting and restoring damaged areas in Sinhala handwritten documents using Computer Vision based algorithms.

1.4.2 Objectives

- Create a customized model to spot the boundaries of damage in texts.
- Use techniques to detect the damaged areas of the document and mark them with bounding boxes and to train a model to identify the type of damage in each bounding box
- Employ image processing to restore blurred and bleed through parts of documents.
- Reconstruct damaged parts of letters according to damage type and maintain original stroke pattern and styles.(Minor damages)
- Completion of missing words or phrases by transformer models, like Sinhala GPT-2
- Identify how the solution will work for different kinds of handwritten notes.
- Evaluate the readability and neatness of Sinhala handwritten text by analyzing the consistency and pattern of handwriting
- To complete a missing text(a complete letter,word or phrase) based on its surrounding context. (Contextual damagesTo develop a customized model to detect damages and label the damage (whether it is a blurred text, insect attack, peeled off area, wear and tear ,etc)
- Estimates the extent of damage by counting the number of missing letters or words for contextual damages.

1.5 Proposed Solutions

As a solution to the above problem, this project develops an intelligent restoration system to detect damages in Sinhala handwritten documents and restore them, with the aid of Computer Vision and NLP techniques to achieve its goal. This system indeed provides a massive support in preserving old Sinhala documents of significant cultural and social importance.

Module 1: Detect damaged areas and recognize the type of damage.

This module initializes the process of restoring the damaged text areas in Sinhala handwritten texts. It uses different types of Digital Image Processing (DIP) techniques to identify the damaged areas in the text and mark them with bounding boxes, then trains a Convolutional Neural Network (CNN) model to label those bounding boxes with the damage type, based on their characteristics. The damage types can include ink blotches, wear and tear, blurred text, insect attacks, peeled-off areas, etc.

Module 2: Evaluating readability, classifying detected damages, and estimating the extent of damage

The second module performs two major functions basically.

1. Evaluating Handwriting Neatness/Readability:

    This function evaluates whether the handwriting is consistently neat or not, essentially determining if it follows a recognizable pattern. It involves analyzing the characteristics of neat handwriting. The decision on whether the handwriting is neat or not is based on a rating and confidence level, which are assigned by comparing the handwriting to a standard of neat handwriting.

2. Classifying and Estimating Missing Content After Damage Detection:

    After damage detection is completed by Module 1, the second module categorizes the missing content into four main types of damage:

    - Part of a letter is missing.
    - A complete letter is missing.
    - A complete word is missing.
    - A few letters of a complete word are missing.

    After categorizing the damage, the module counts the number of missing

letters or words, which applies specifically to the third and fourth categories (complete word and few letters of a complete word missing).

Module 3 : Blurred Text Restoration and Letter-Level Damage Reconstruction

This module is responsible for two key functions in the document restoration process:

1. **Restoration of Blurred or Bleed-Through Text**
   After Module 1 detects areas affected by blurred or bleed-through text, Module 3 applies image enhancement techniques to remove these distortions. By improving text clarity and separating any interfering ink from the background, the module ensures that the text remains readable. Once the necessary corrections are made, the processed text is sent back to Module 1 for further classification, allowing for additional refinement if needed.
2. **Reconstruction of Minor Letter-Level Damage**
   When Module 2 identifies certain damages as minor damages(limited to missing parts of letters without requiring contextual reconstruction) , these affected characters are processed in Module 3. Each damaged area is already classified by Module 1 based on its type, such as ink blotches, wear and tear, insect attacks, or peeled-off sections. Using this classification, Module 3 reconstructs the affected letters by preserving their original stroke patterns and handwriting style, ensuring consistency with the document's natural appearance.

Module 4: Restoring missing words based on surrounding context

The damage, like missing an entire letter or word in a paragraph, is categorized as a contextual damage at the second module. These type of contextual damages arrive at the fourth module for restoration purposes.This includes reconstruction of larger, contextually significant missing sections, including entire letter, letters, words and phrases, ensuring coherence with the surrounding text.

Using this intelligent Sinhala text restoration system, the ability to reconstruct missing letters and words in Sinhala texts becomes easier than ever. It is crucial for preserving linguistic integrity, improving digital accessibility, and enhancing natural language processing (NLP) applications to have the ability to reconstruct missing letters and words in any language. This research contributes to the development of AI-driven text

restoration methods, which can benefit historical document preservation, automated text correction, and better OCR performance for Sinhala. By leveraging GPT models, this study helps bridge gaps in Sinhala NLP, supporting broader applications in education, communication, and AI-powered language tools, ultimately advancing technology for low resourced languages.

High Level Architecture of the Proposed Solution

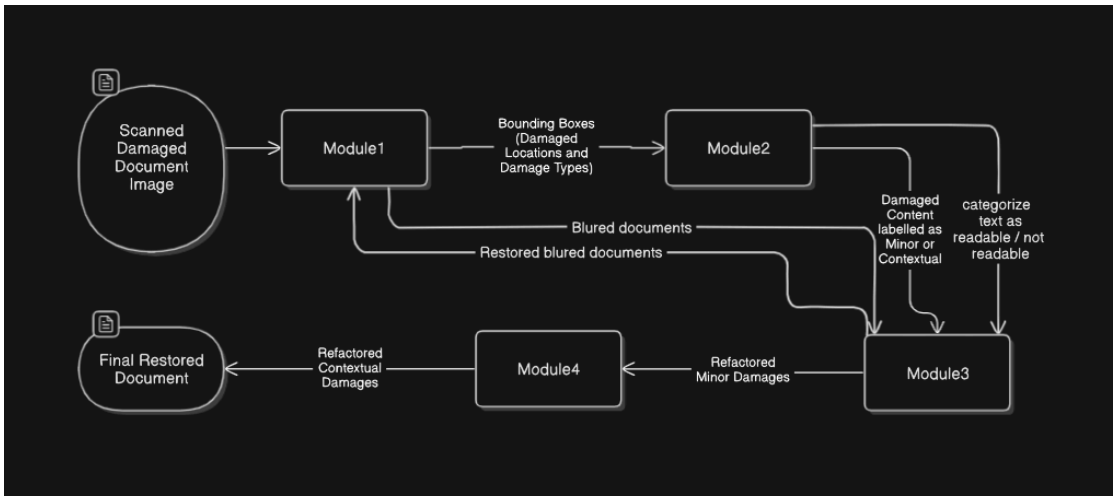


Fig: High Level Architecture of the Proposed Solution

1.6 Chapter Summary

The paper begins with Chapter 1, which provides an introduction to the research and provides an overview of the importance of preserving and restoring the Sinhala handwritten resources, its unique style of writing. The background and motivation of emphasizes what pushed us to pursue this attempt to solve this particular problem, why and how critical the matter we are concerned should be addressed.As Sri Lanka is the only country that uses the language Sinhala, there is very less number of research done in this particular area.Although a few research has been already done in Sri Lanka for this, the issue has not been perfectly addressed and still, the previous approaches have certain fallbacks themselves.

The problem in brief highlights the major problem that the research addresses , in a concise and precise manner.Moreover,the aim and objectives demonstrates what the research aims and its major objectives are. The paper explains how it intends to develop an automated system for detecting and restoring damaged areas in Sinhala

handwritten documents using Computer Vision based algorithms, NLP libraries and ML technologies. The authors have elaborated their solution in the proposed solution section of chapter 1, where the functions of the four modules have been included precisely. The high level diagram of the proposed diagram has also been included for better clarification.The chapter ends by giving a sound idea picture of what the research is about to be going forth and what sort of benefit the research is supposed to deliver to the world of AI and NLP.

Chapter 2 describes the literature review for each module, Chapter 3 describes the technology adapted, chapter 4 describes proposed solution, chapter 5 describes analysis and design, chapter 6 describes implementation and finally chapter 7 gives the conclusion.

Chapter 2 - Literature Review

2.1 Chapter Overview

This chapter will provide a more thorough description of the current solutions. There will be a description of the problem that has been discovered by past research, how it has been approached using various approaches, and the shortcomings of those earlier efforts. This section also includes a summary of the current works for each module.

2.2 Literature Review on Module 1 - Detection and Classification of Damaged Areas

2.2.1 Introduction

Handwritten documents must be preserved in order to preserve academic, legal, and historical records. Nevertheless, these records frequently experience deterioration in the form of physical harm, blurring, insect attacks, and ink smudges.This affects their integrity and readability. The current research on automated document damage detection and classification methods is examined in this overview of the literature. The methods that use Deep Learning, and Digital Image Processing (DIP) to recognize and classify various types of damage are also highlighted. This review looks at earlier research in order to obtain knowledge that can direct the creation of a reliable system for identifying and categorizing damaged areas in handwritten Sinhala documents, guaranteeing their repair and long-term preservation.

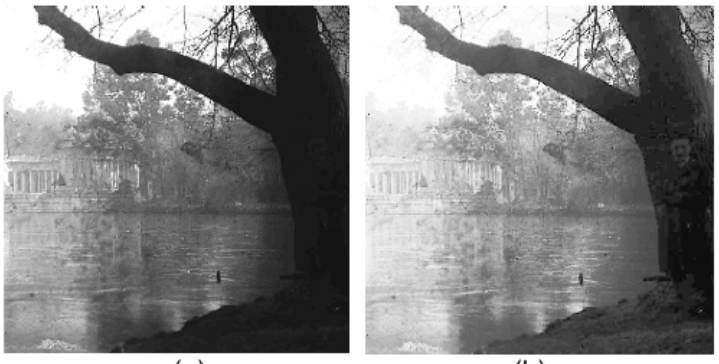
2.2.2 Review of Other's Work

2.2.2.1 Image Processing Techniques for Noise Removal and Damage Enhancement

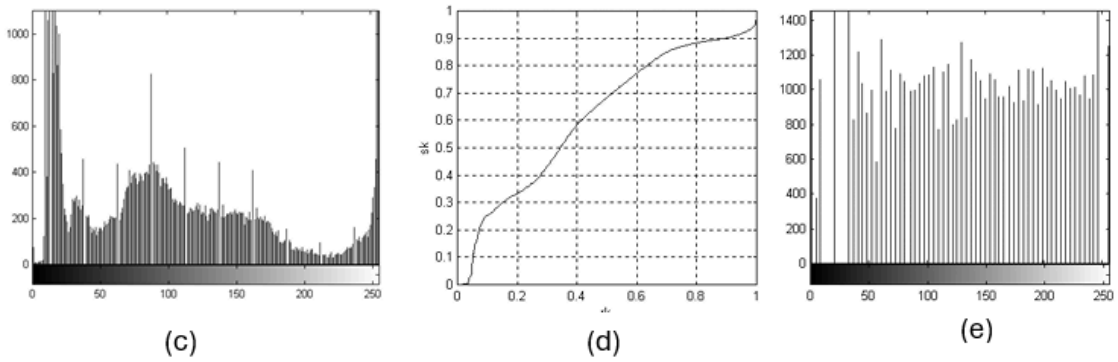
Because of the texture of the paper, stains, or inconsistent ink, handwritten documents frequently contain noise. Before damage is detected and categorised, effective noise removal techniques are crucial. A number of noise reduction and enhancement strategies for restoring damaged visual material were examined in the paper "Digital image processing techniques applied to the recovering and conservation of the graphical heritage.[1]" One important technique was a modified median filter, which

increases the accuracy of non-text noise detection by substituting noisy pixels with the median value while excluding the central pixel.

In order to create a more balanced representation, the study also used illumination extraction which uses Discrete Wavelet Transform (DWT) to separate an image's illumination and reflection components [1]



Example of illumination degradation (a) and histogram equalization example (b)

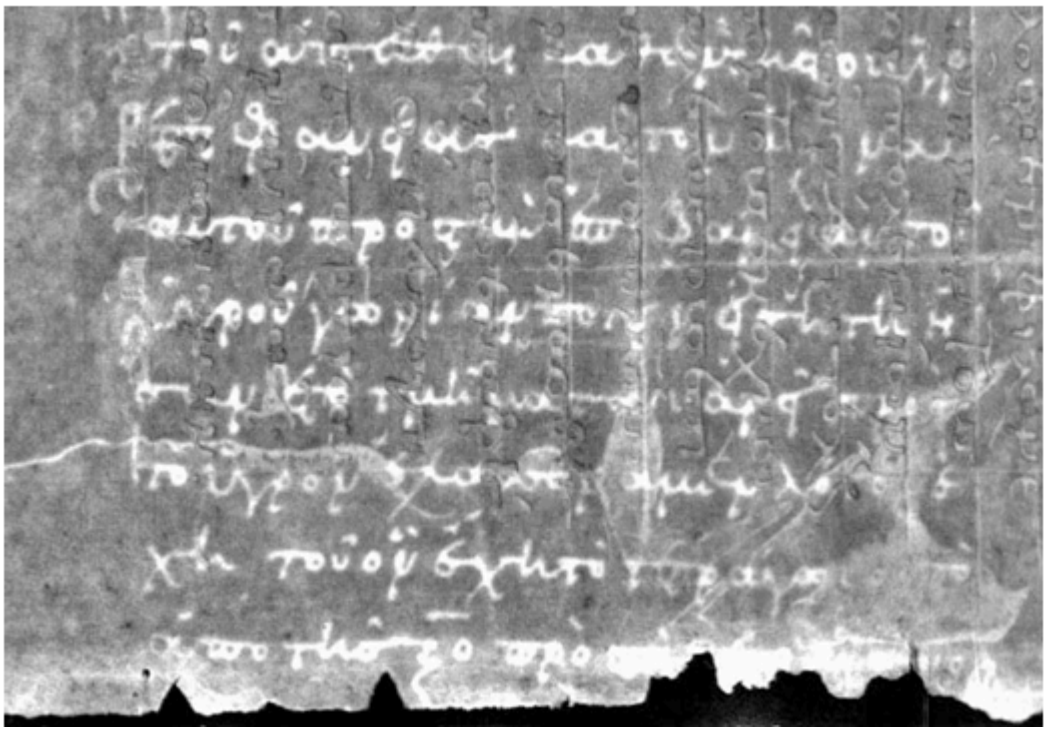


Histogram equalization example. (c) Histogram of image 'a'. (d) Transformation function. (e) Histogram of image 'b'.

Furthermore, contrast enhancement is essential for improving the distinguishability of damaged text. Histogram equalisation was used in the study to improve low-contrast photographs by distributing pixel values over a wider range[1] . This technique greatly enhanced the visibility of fading or smudged text, a typical problem in ancient texts.

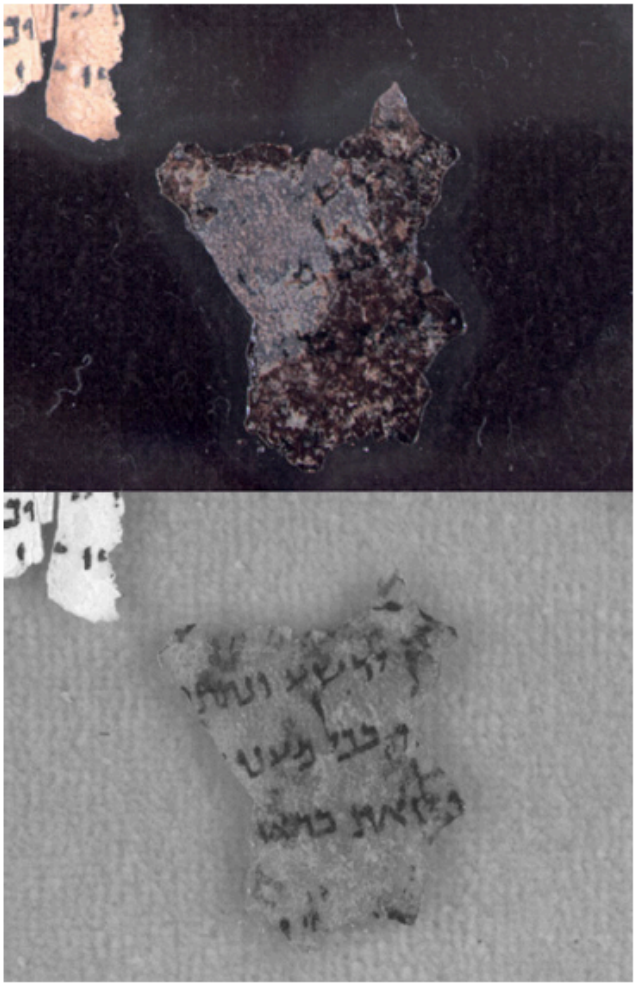
2.2.2.2 Identification and Segmentation of Damaged Areas

One of the most important restoration steps is identifying damaged sections in historical and handwritten manuscripts. To improve the contrast between ink and background and make damaged areas easier to identify, a variety of image processing techniques have been used. Advanced image processing techniques, such as reflection, transmission, and fluorescence imaging, were used in the study "Digital Restoration of Erased and Damaged Manuscripts" to recover text that has degraded to the point where it is nearly invisible to the human eye. The study successfully identified damaged areas while separating the original text from later alterations by classifying photos into distinct object classes. [2]



Example of an individual image representing a specific object class.

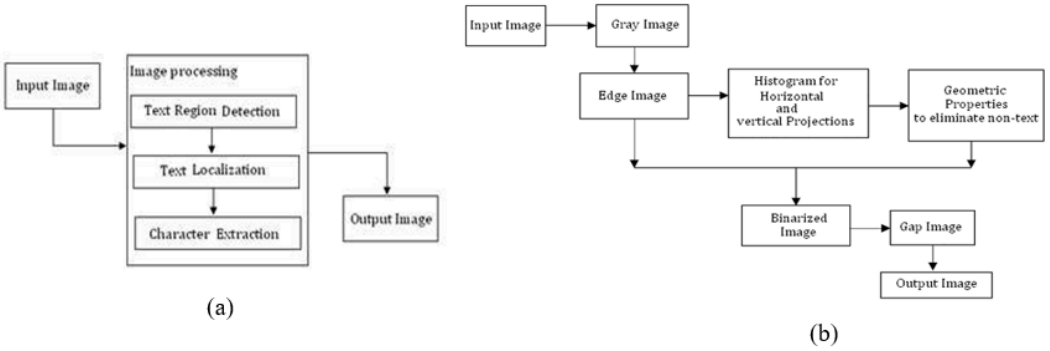
The use of infrared photography to uncover concealed text written with carbon-based ink was investigated in a different publication titled "Image restoration of damaged or erased manuscripts."[3]



Near infrared light reveals text characters that are hidden on the damaged surface

In order to improve the visibility of hidden writing, UV reflectance was also utilized to enhance erased characters. Additionally, the study used X-ray fluorescence (XRF) to find iron gall ink under opaque materials and contrast stretching techniques to improve low-contrast characters in scanned photos[3]. These methods show that multispectral imaging is a useful tool for recovering and segmenting damaged text.

Additionally, "Text Information Extraction and Analysis from Images Using Digital Image Processing Techniques" concentrated on text detection in images, which is important for differentiating between undamaged and damaged sections. In order to find text regions in noisy and damaged images, the study employed an edge-based detection methodology and a connected-component analysis method[4] .

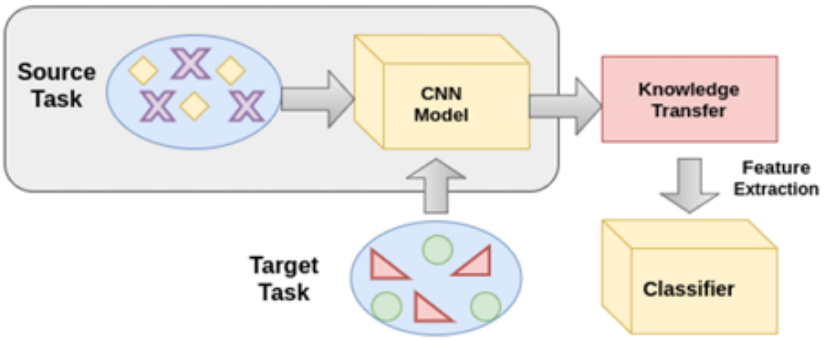


(a) Bloock diagram for edge based text extraction (b)Basic Block diagram for Connected Component based text extraction

Morphological operations were also employed for visibility enhancement and the text was analyzed at various scales using a Gaussian pyramid. Also, in order to identify the damaged regions in handwritten documents, the process had contrast-based segmentation[4] applied.

2.2.2.3 Classification of Damage Types Using Machine Learning

Deep learning and machine learning techniques have been thoroughly investigated for picture classification, including damage detection. Convolutional Neural Networks (CNNs) have demonstrated effectiveness in recognizing various forms of document damage through the acquisition of complex patterns from labelled datasets. Through the use of methods like transfer learning and data augmentation, a related study on car damage classification [5] shows how effective CNNs are in identifying different forms of damage with high accuracy. In order to train a model to classify document damage into categories like ink blotches, smudges, or torn sections, such as deep learning-based car damage classification, the study demonstrates how pre-trained models on large datasets, like ImageNet, can be refined for domain-specific applications.



A CNN model trained on a source task extracts features, which are then used for classification in a target task

2.3 Literature Review on Module 2 - Evaluating readability, classifying detected damages, and estimating the extent of damage

2.3.1 Introduction

Handwritten text recognition and analysis have been extensively studied in various languages, including Sinhala, which presents unique challenges due to its complex script structure. The segmentation and recognition of Sinhala handwritten text require specialized techniques to address overlapping characters, curvature, and non-uniform spacing. Unlike Latin-based scripts, Sinhala comprises intricate circular and semicircular characters that often connect fluidly, making character segmentation a significant challenge. Recent research has focused on improving segmentation accuracy, feature extraction, and classification methodologies to enhance recognition rates. Moreover, the field has witnessed the integration of machine learning and deep learning techniques to improve accuracy and efficiency. This literature review presents an overview of existing studies on segmentation, feature extraction, and classification of handwritten text, with a particular emphasis on Sinhala script.

2.3.2 Review of Other's Works

2.3.2.1 Handwriting Segmentation Techniques

Handwriting segmentation plays a crucial role in recognition, with various methods employed to separate lines, words, and characters effectively. Horizontal and vertical projection techniques have been widely used to identify text boundaries, particularly in handwritten scripts [6]. Studies have explored the effectiveness of these methods in segmenting connected and overlapping characters, demonstrating their applicability in Sinhala handwriting [7]. Additionally, contour-based and morphological operations have been examined to enhance segmentation accuracy, particularly in cases where characters merge or touch.

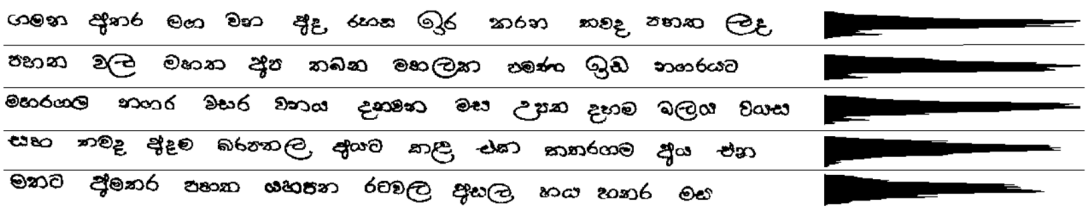


Figure - Horizontal Projection

Figure - Vertical Projection for Word Segmentation

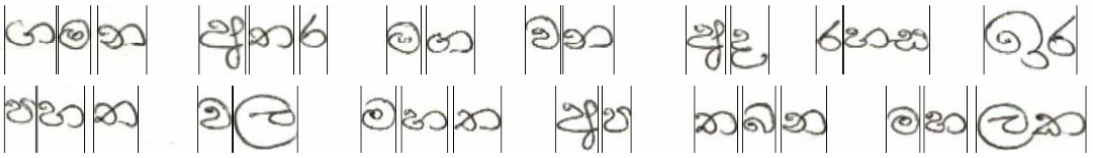


Figure - Vertical Projection for Character Segmentation

A significant challenge in Sinhala handwriting segmentation is the presence of ligatures and diacritics, which influence character structure. Researchers have explored improved linear density techniques and Connected Component Analysis (CCA) to handle touching and overlapping characters [8]. These methods analyze pixel distributions and stroke connectivity to distinguish individual characters from handwritten text. Comparative studies between projection-based techniques and neural network-driven segmentation approaches indicate that hybrid models, combining traditional methods with deep learning, yield better results for complex scripts such as Sinhala [9]. Moreover, the introduction of region-based segmentation has proven beneficial in isolating specific character components, ensuring better segmentation accuracy.

2.3.2.2 Feature Extraction in Handwritten Text Recognition

The feature extraction process is essential in quantifying distinct characteristics of handwritten text, aiding in its classification. Commonly extracted features include stroke width variation, letter spacing, baseline alignment, and curvature consistency. Research has indicated that combining multiple feature extraction techniques leads to improved recognition accuracy, particularly when dealing with handwritten text that exhibits variability in writing styles [10]. Some studies have utilized zoning and

directional gradient-based feature extraction techniques to capture fine-grained details in Sinhala handwriting. Additionally, Histogram of Oriented Gradients (HOG) and Scale-Invariant Feature Transform (SIFT) have been employed to enhance feature representation. [11]

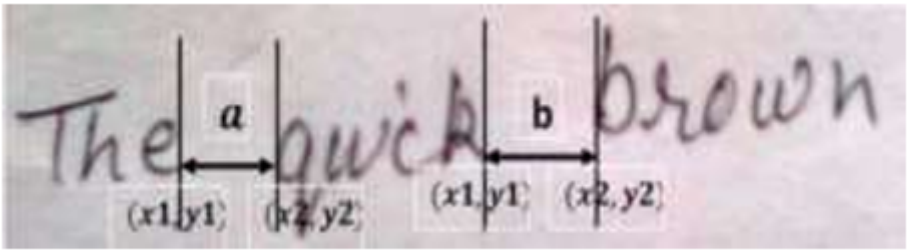


Figure - Word Spacing

2.3.2.3 Readability and Handwriting Quality Evaluation

Several approaches have been proposed for evaluating the readability of handwritten documents. Bounding box symmetry analysis, stroke consistency measurement, and spatial alignment checks have been explored as methods to distinguish between legible and difficult-to-read handwriting [10]. Research indicates that Sinhala handwriting with consistent stroke widths, well-aligned baselines, and uniform spacing is more readable, whereas irregularities in these aspects lead to decreased legibility. The role of machine learning in automatically predicting readability has also been investigated, with researchers experimenting with Support Vector Machines (SVM) and Convolutional Neural Networks (CNNs) to classify handwriting into readable and hard-to-read categories. Furthermore, deep reinforcement learning has been explored as a technique to refine handwriting evaluation models.

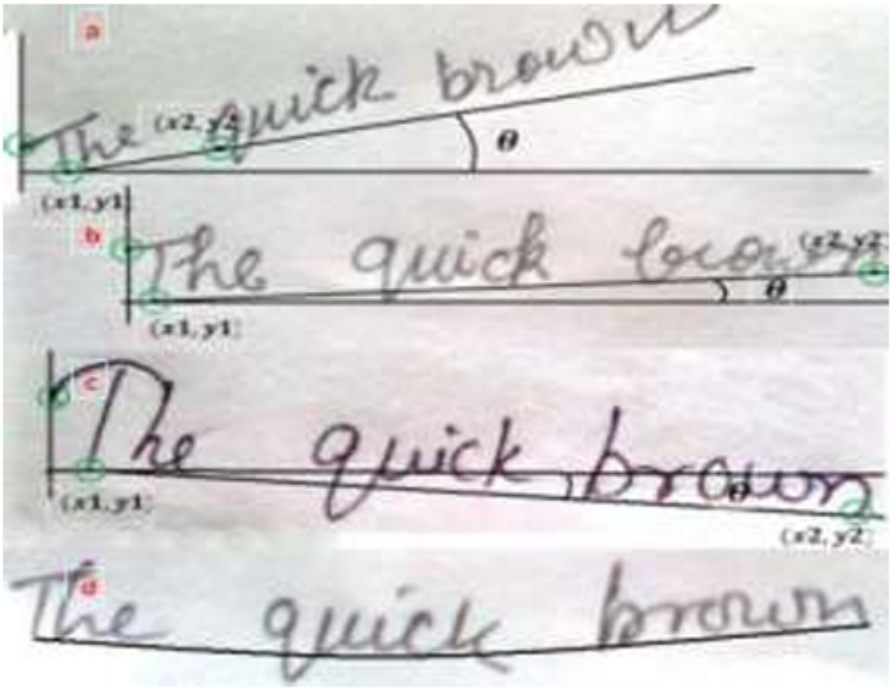


Figure - Type of baselines (a) rising, (b) straight, (c) falling, (d) erratic.

2.3.2.4 Classification Methods for Handwritten Text Recognition

Classification methods for handwritten text recognition have evolved significantly, incorporating machine learning and deep learning approaches. Traditional methods such as Support Vector Machines (SVM) and k-Nearest Neighbors (k-NN) have been used for classification, whereas recent advancements leverage Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) to achieve higher accuracy in recognizing complex scripts [12]. These methods have shown promise in identifying and categorizing handwritten characters, particularly in scripts with intricate structures such as Sinhala. Hybrid approaches combining CNNs with attention mechanisms have demonstrated potential in improving recognition accuracy, particularly in the presence of noise and damaged characters. Recent studies have also explored the effectiveness of Vision Transformers (ViTs) in capturing contextual relationships within handwritten text.

2.3.2.5 Handling Damaged and Degraded Handwritten Text

Several studies have focused on addressing the challenges associated with damaged handwritten documents. Identifying missing or incomplete characters remains a

critical task, with researchers employing connected component analysis (CCA) and contour-based techniques to detect character damage [13]. The classification of damage types, including partial letter loss and complete word omissions, has been explored through statistical and deep learning models, enhancing the restoration of degraded text. Researchers have proposed the use of generative adversarial networks (GANs) to reconstruct missing characters and words in handwritten documents, with promising results in enhancing readability and recognition performance. Moreover, adversarial training techniques have been introduced to improve model robustness against distorted or incomplete character inputs.

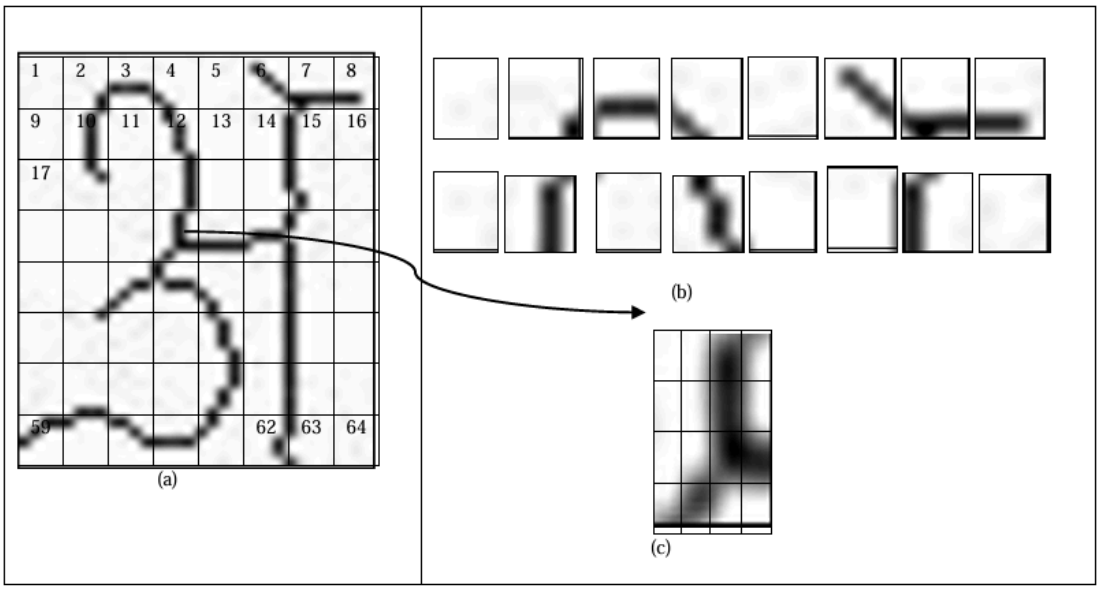


Figure - (a) Handwritten character 'අ' divded into 8x8 blocks. with feature indices (b) top 2 rows after zoning and (c) 4x4 grid placed on one block to compute the Pixel density feature

2.3.2.6 Insights from Related Scripts

Research on character segmentation in related scripts, such as Brahmi, Arabic, and Devanagari, has provided insights applicable to Sinhala handwriting recognition [14]. Comparative analyses have revealed similarities in segmentation challenges, with proposed solutions including ligature detection and density-based segmentation methods [15]. The application of these techniques in Sinhala script has demonstrated improvements in text separation and recognition accuracy. For example, improved segmentation techniques from Arabic handwriting recognition, such as ligature-based

segmentation, have been adapted to handle the curved nature of Sinhala script effectively. Additionally, multilingual handwriting datasets have been leveraged to develop cross-lingual handwriting recognition models, improving generalization performance across different scripts.

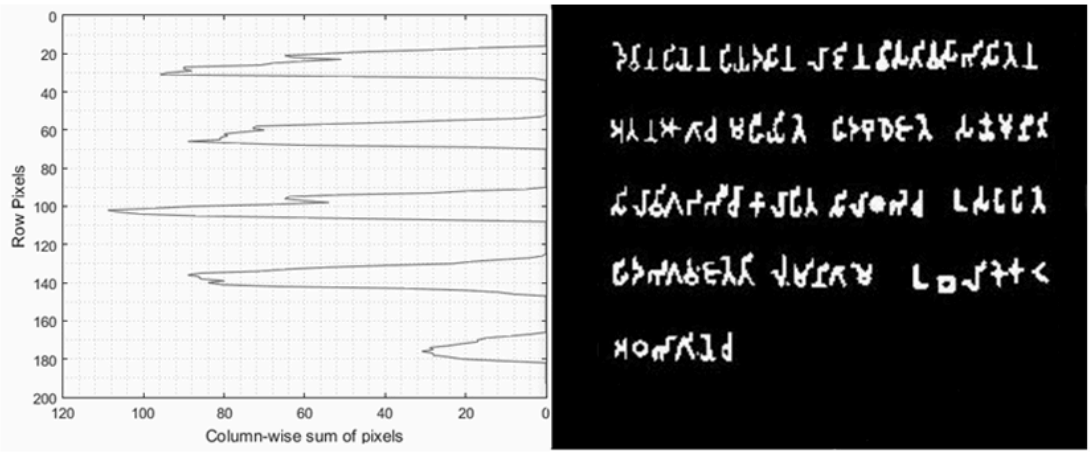


Figure - Horizontal projection profile for image segmentation.

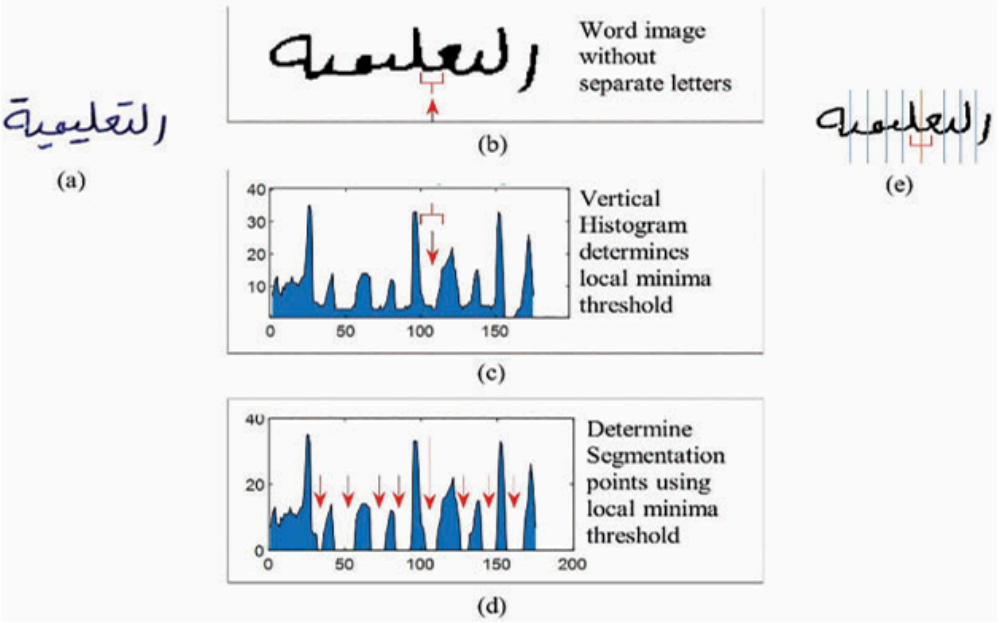


Figure - Determine segmentation points using local minima threshold (a) word image before preprocessing (b) word image after preprocessing, (c) vertical histogram of the image, (d) vertical histogram after determining ligature segmentation points, and (e) the word image after segmentation.

2.3.2.7 Advancements in Optical Character Recognition (OCR) for Sinhala Handwriting

Advancements in Optical Character Recognition (OCR) technology have further contributed to the field, with studies emphasizing the adaptation of OCR models for complex scripts [9]. The effectiveness of OCR in recognizing compound characters and diacritics has been evaluated, highlighting the need for script-specific enhancements to improve accuracy. Pre-training deep learning OCR models on synthetic handwritten Sinhala datasets has been explored as a method to improve character recognition rates. Moreover, Transformer-based OCR architectures have shown promise in capturing long-range dependencies in handwritten text, leading to significant improvements in recognition accuracy.

2.4 Literature Review on Module 3 - Blurred Text Restoration and Letter-Level Damage Reconstruction

2.4.1 Introduction

Restoration of handwritten documents is an essential task in preserving historical, legal, and cultural records. The primary challenges in this domain involve reconstructing damage strokes in letters and enhancing blurred text while ensuring that the handwriting style remains unchanged. Traditional image processing techniques, machine learning approaches, and deep learning-based restoration have been extensively researched, each offering different levels of accuracy and efficiency.

Earlier works on Sinhala handwritten text restoration primarily focused on character-level repair, often failing when extensive portions of text were missing. Morphological operations, inpainting, and stroke reconstruction have been used for partial letter restoration, This section provides a structured review of past research efforts in blurred text restoration and letter-level damage reconstruction and highlighting their applications and limitations.

2.4.2 Review of Other's Work

2.4.2.1 Traditional Image Processing Techniques

Traditional methods for stroke restoration rely on image binarization, morphological operations, and edge linking algorithms. These techniques are fast and computationally inexpensive but are less effective in reconstructing highly degraded text.

The research mainly doing restoration of historical documents handwritten documents using the combination of special local and global binarization techniques, by eliminating of non-uniformly illuminated background restoration. Study has applied binarization techniques such as Otsu's thresholding and Adaptive Gaussian Thresholding to segment handwritten text from background noise. Morphological dilation and erosion have been used to reconstruct letter strokes that have minor gaps or missing parts. However, these techniques fail when text is severely degraded or when entire segments of a stroke are missing. [16]

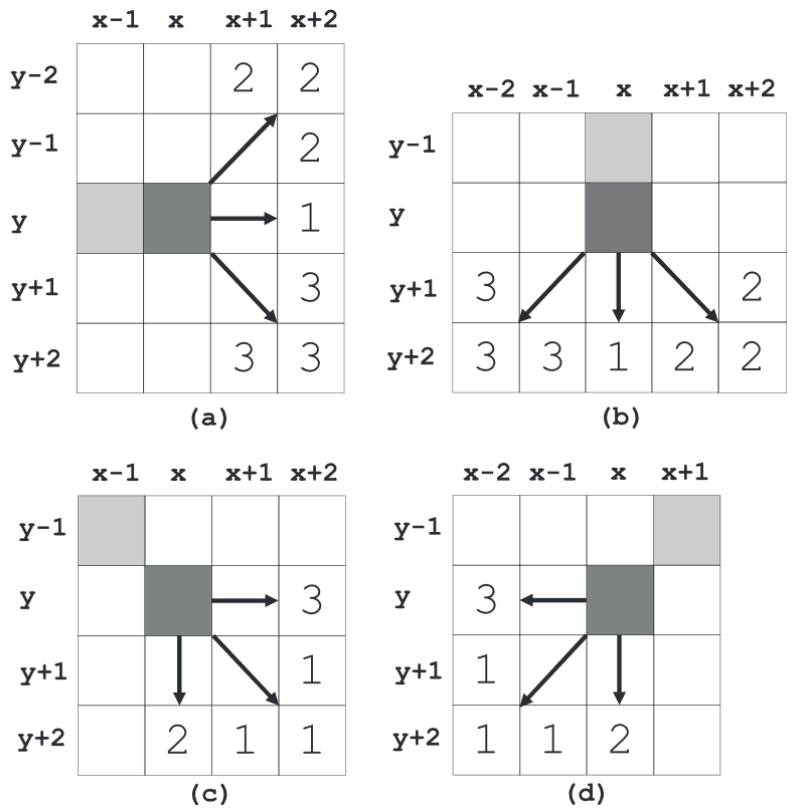
Edge-linking methods are widely used for reconstructing broken handwritten characters.Following are the mainly used edge linking methodologies used for reconstruct damage character reconstruction. Jiang et al. introduced morphological techniques to detect and enhance thin edge details, particularly in low-contrast image regions. Meanwhile, Wei et al. proposed a Sequential Edge Linking Algorithm to improve edge connectivity, though it is primarily applicable to simplified two-region edge detection problems[17]. Shih et al. developed an approach using adaptive structuring elements, which dilates damaged edges by aligning them with the direction of their slope[18].

Researchers proposed Predictive Edge Linking (PEL) algorithm to enhance edge linking in binary edge maps. Unlike traditional edge detectors that leave gaps, noise, and multi-pixel-wide edges, PEL refines edge detection results by predicting the next movement based on previous steps. This approach makes PEL suitable for real-time applications such as Optical Character Recognition (OCR), object detection, and image segmentation.The PEL algorithm takes a Binary Edge Map (BEM) as input and converts it into a set of refined edge segments.

The process consists of four key steps.

1. Filling One-Pixel Gaps in the Edge Map

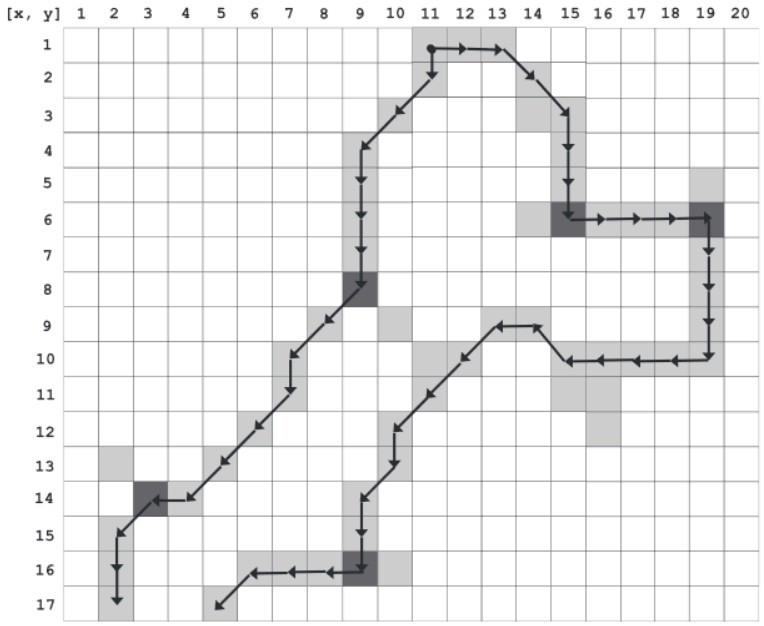
   Traditional edge detectors, such as Canny, often create small gaps between detected edge pixels (edgels). To resolve this, PEL identifies edge endpoints (tips) by detecting pixels with only one neighboring pixel. It then applies heuristic rules to connect the nearest edgel in the same direction of movement, ensuring continuity in edge structures and reducing fragmentation[19]



The process involves filling one-pixel gaps between edge segments by identifying tip pixels (dark gray) that have only one neighboring pixel (light gray).

2. Creating Edge Segments

After filling the gaps, PEL proceeds with forming continuous edge segments. It employs an eight-directional walking mechanism, starting from an arbitrary edge pixel and moving in both forward and reverse directions. The system maintains a history of the last 8 movements, which helps it predict the next most probable pixel to be part of the same segment. If an edge extends diagonally, PEL prioritizes maintaining the diagonal instead of making abrupt shifts, ensuring smooth and accurate edge linking.[19]



The process starts at pixel (11,2), creating two chains that are later merged into a single edge segment. Dark gray pixels indicate points where prediction guides the movement, ensuring smooth edge continuity.

3. Joining Nearby Edge Segments

   PEL further identifies unconnected edge segments that should be merged into the same structure. Two edge segments are classified as neighbors if their endpoints are within five pixels of each other and they appear to be structurally aligned. The system then removes overlapping pixels and merges segments into a single, unified edge, reducing disjointed edges and enhancing edge-based object detection accuracy.[19]

4. Thinning and Cleaning Up Edge Segments

   To refine the results, PEL applies thinning techniques to eliminate multi-pixel-wide staircase edges, which often occur in diagonal structures. The thinning process detects redundant pixels, removes excess ones, and maintains the original direction and shape of the edge. Additionally, noise reduction is applied by eliminating short edge segments to prevent random speckles from being mistakenly identified as significant edges.

The Predictive Edge Linking (PEL) algorithm offers significant improvements over conventional edge linking techniques by addressing key limitations in existing

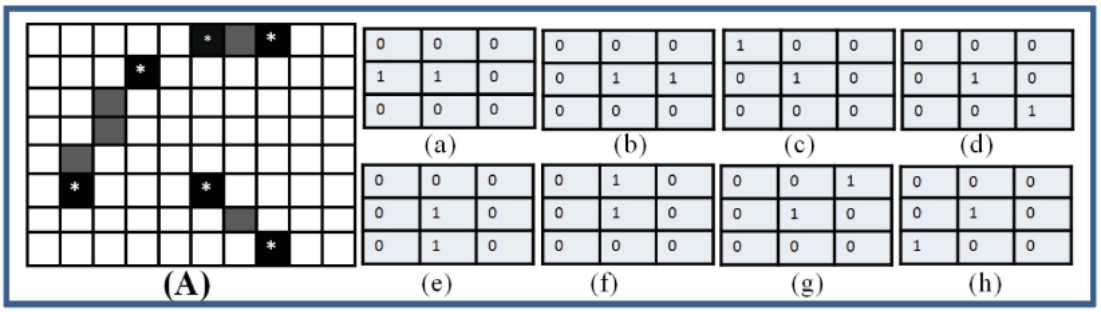
methods. Compared to Canny with Traditional Edge Linking, where broken edges require additional post-processing, PEL automatically fills gaps, reduces noise, and refines edges, enhancing edge continuity. Unlike CannySR (Smart Routing for Edge Drawing), which is limited to Canny's binary edge maps, PEL is more flexible and does not require prior knowledge of the smoothing function, making it adaptable to various preprocessing conditions. Similarly, Sequential Edge Linking (SEL) by Wei et al., which is designed for two-region edge problems, lacks the robustness needed for complex, multi-region images, whereas PEL effectively handles such cases. PEL achieves faster processing with lower computational overhead, making it a more efficient choice for real-time applications in image processing and character recognition.

Studies also discussed Distance Based Edge Linking Algorithm to enhance edge linking by focusing on minimum distance between edge segments, unlike traditional methods that rely on morphological operations.[19]

The key steps of the DEL algorithm are,

1. Finding End Tips of Edge Segments

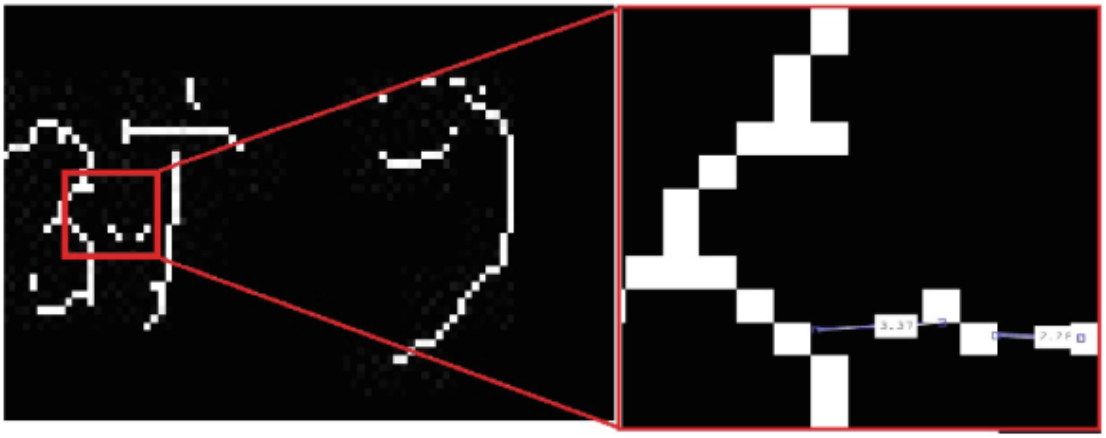
   The Binary Edge Image Map (BEIM) is generated using a Canny edge detector, which helps in detecting prominent edges in an image. Each edge segment is labeled, and its end tips are identified to recognize the discontinuities in the edge structure. A mask-based approach is then used to locate and highlight disconnected edge tips, ensuring effective feature extraction for further processing.[20]



shows the three binary edge segment the dark pixel is the end tip and gray pixel is neighbors of tip pixel. Figure (a) to (h) mask are used for finding the end tip of binary edge segments.

2. Calculating Minimum Distance Between Edge Segments

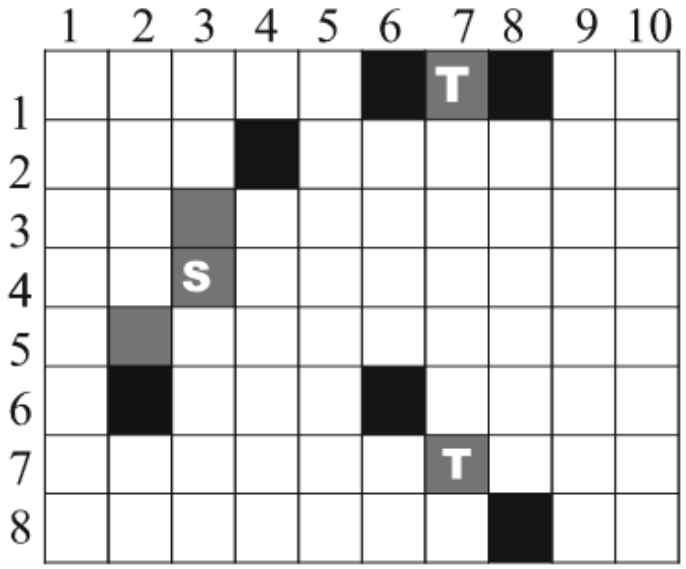
   The Euclidean Distance Calculation method is applied to measure the shortest distance between two edge segments. To refine this process, a 5×5 search window is used to scan for the closest edges. A voting mechanism is then employed to select the best matching edge segment pairs, ensuring accurate connections between broken edges.[20]



   Shows the calculated minimum distance between corresponding edge segments

3. Filling the Gaps Between Edge Segments

   This is achieved using Bresenham's Algorithm, which effectively links the identified source and target edge segments by drawing a straight line between them. The process reconstructs the broken edges, significantly improving the readability and recognition accuracy of the characters, making them suitable for further processing in Optical Character Recognition (OCR) and related applications.[20]



Source Point	Target Point	Distance (Euclidean)
S1(4,2)	T1(6,1)	**2.2361**
S1(4,2)	T2(6.6)	4.4721
S1(4,2)	T1(8,1)	4.1231
S1(4,2)	**T2(6,8)**	**6.3246**
S2(2,6)	T1(6,1)	6.4031
S2(2,6)	T2(6.6)	4
S2(2,6)	T1(8,1)	7.8102
S2(2,6)	T2(6,8)	4.4721

Sample edge segment map ‘S’ denote source edge segment and ‘T’ denotes Target segment. Table shows Minimum distance of source ‘S1’, ‘S2’ and target ‘T1’, ‘T2’ edge segment.

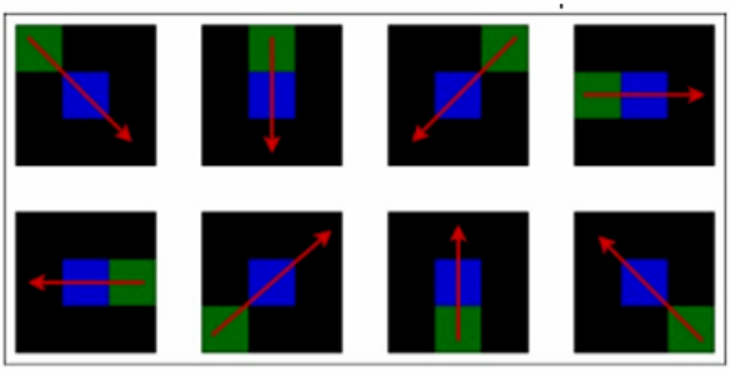
The proposed Distance-Based Edge Linking (DEL) algorithm outperforms both Predictive Edge Linking (PEL) and traditional Canny edge detection techniques by effectively preserving edge details, thereby ensuring more accurate character reconstruction. Traditional approaches, which often result in loss of fine edge structures and introduce false connections, DEL significantly reduces noise and enhances edge connectivity. Furthermore, by minimizing the gaps between broken edge segments, the DEL algorithm improves the overall recognition accuracy, particularly for degraded character images, making it a more robust solution for character restoration and optical character recognition (OCR) tasks.

The recent study that is focused on Damage Sinhala Handwritten Document reconstruction proposed Guided Edge Linking Approach that utilized distance-based end-tip coupling and Bezier curve linking techniques. [21]

The key steps of the Guided Edge Linking Approach algorithm are,

1. Identification of Damaged End-Tip Pixels

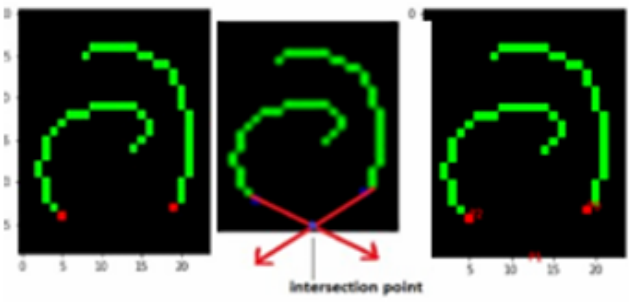
Thinning operations are applied, reducing the stroke width to a single-pixel representation. This ensures accurate identification of damaged end-tip pixels, which are marked as potential connection points for the reconstruction process. The system then analyzes these end tips to determine suitable pairs for linking.[21]



Possible damaged end-tip pixel patterns

2. End-Tip Coupling Based on Distance and Direction

The Euclidean distance formula is employed to identify the nearest end-tip pairs, ensuring that closely positioned segments are linked together. Additionally, local area thresholding is applied to determine whether the end tips are within an acceptable range. If they fall outside this range, direction mapping is used to analyze their slope and orientation, ensuring that only structurally appropriate tips are linked. A recursive function refines the pairing process to improve accuracy.[21]



Obtaining control points for Bezier curve

3. End-Tip Linking Using Bezier Curve

The system first determines whether the missing stroke should be reconstructed using a straight line or a curved connection by assessing the character's structural complexity. If the missing segment requires curvature, control points are identified based on intersection or boundary points, and a Bezier curve is applied to restore the natural shape of the character.[21]

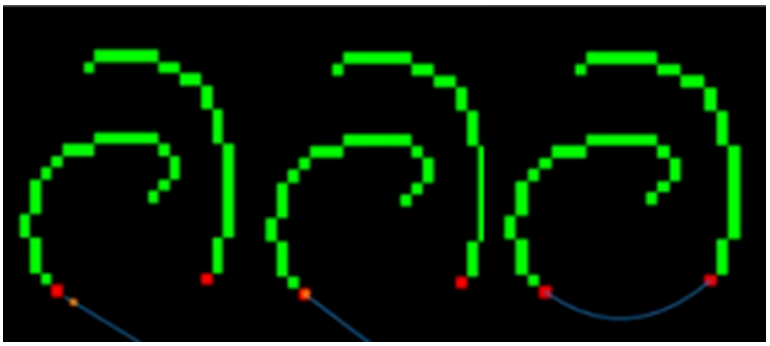


Fig: Bezier curve approach in damaged ”ಅ” character

One major issue in the proposed solution is the lack of damage-specific restoration techniques, as most methods apply a single reconstruction approach regardless of the type of damage, leading to suboptimal accuracy.Additionally, existing methods struggle with reconstruct high-damage cases and complex letters with stroke damages.

2.4.2.2 Restoration of Blurred or Bleed-Through Text

Super-resolution techniques enhance the resolution of low-quality text images, making blurred characters more readable. Deep learning-based approaches such as SRCNN (Super-Resolution Convolutional Neural Network) and ESRGAN (Enhanced Super-Resolution GAN) have been widely used for text sharpening in historical documents. These methods utilize convolutional networks to reconstruct fine details and improve OCR recognition rates. However, while they significantly enhance image sharpness, they do not restore missing strokes, which limits their effectiveness for severely degraded text[22]

Adaptive sharpening techniques focus on enhancing text clarity by dynamically adjusting edge sharpness while minimizing noise artifacts. Wiener deconvolution and Laplacian variance detection have been widely applied to detect and restore blurred

text edges. These methods analyze the intensity gradients in text images and apply edge-preserving filters to improve legibility. While these approaches are effective for moderate blur removal, excessive sharpening can introduce artifacts or distortions, making text less readable in cases of extreme degradatio[23]

2.5 Literature Review on Module 4 - Restoring missing letters and words based on surrounding context

2.5.1. Review of Other's Work

2.5.1.2. LLMS and low resourced languages

GPT, Llama, and Claude are examples of recent developments in Large Language Models (LLMs) that have demonstrated potential in Natural Language Processing (NLP) tasks. But they still don't perform well enough on low-resource languages like Sinhala. With an emphasis on their suitability for Sinhala, Jayakody et.al's study on Performance of Recent Large Language[24] Models for a Low-Resourced Language study assesses four modern LLMs—Claude, GPT 4o, Llama 3, and Mistral 7B—on tasks including translation, summarization, and text production. The results show that Claude and GPT 4o outperform earlier models and function well when used outside of the box. Llama and Mistral, on the other hand, perform worse but have room for improvement with tweaking. Notably, despite Llama's overall restrictions, Google and GPT performed better when translating from Sinhala to English, while Google and Llama performed better when translating from English to Sinhala.

The study mirrorsforth the requirement for developing fine-tuning techniques for LLMs to enhance their effectiveness in low-resourced languages[12]. Results of this study show how important it is to increase NLP resources in order to enhance digital communication and cultural representation in languages such as Sinhala 1.

Tan et al. (2021)[25] suggest a new approach called Progressive Generation of Text (ProGen) that uses pretrained language models to produce coherent lengthy text passages. By first creating domain-specific content keywords and then gradually honing these into whole passages over several steps, their approach tackles the difficulties of writing lengthy and contextually cohesive prose. This strategy outperforms conventional fine-tuning and planning-then-generation techniques by efficiently utilizing pretrained language models such as BART to improve coherence

and quality. Extensive analyses show notable gains in sampling efficiency and text coherence in a variety of contexts, such as CNN News and Writing-Prompts 1.

The proposed method of progressive text generation focuses on generating text in stages, beginning with high-level content and eventually falls down into a complete passage. This can be a very useful method for handling handwritten text, as it may allow for starting with rough interpretations of the handwritten characters and progressively refining them into accurate text.

In order to address the data scarcity issues that large language models (LLMs) encounter in low-resource languages like Sinhala , Hettiarachchi et al.'s research presents NSina[26], a comprehensive news corpus for the Sinhala language that includes over 500,000 items from prominent Sri Lankan news websites. It describes three NLP tasks with different training and testing sets: news media identification, news category prediction, and news headline generation. The need for Sinhala-specific models and better assessment measures was highlighted by the evaluation of many transformer models, the best of which only received a BLEU score of 0.17 for headline generation. Because NSina is open-access, it fosters collaboration in Sinhala NLP research, which is especially important for restoring damaged Sinhala handwritten texts because it can supply baseline training data, guide model development, and inspire text analysis techniques.This can be useful for training a sound base model for Sinhala before subjecting it to any fine tuning.

Chapter 3 - Technology Adapted

3.1 Introduction

This chapter describes the technologies used in implementing the solution for this research study in brief.

3.2 Technologies Adopted for Implementation

3.2.1 Programming Languages

Python is a programming language that aims to keep it simple for both fresh and experienced programmers to transform their ideas into code. Python for Computer Vision is the most prominent, matured, and supported programming language in the field of machine learning, which is why many developers use it.

C++ is a powerful and efficient programming language widely used in computer vision and machine learning due to its high-performance capabilities and robust libraries. C++ for Computer Vision is one of the most prominent and well-supported languages, especially in real-time applications, thanks to libraries like OpenCV. Many developers prefer C++ in Visual Studio for developing high-speed image processing, AI, and machine learning models, as it offers better optimization and lower-level hardware control compared to other languages.

3.2.2 Libraries

3.2.2.1 OpenCV

The OpenCV library was utilized since the research highly depends on image processing. It's a free, open-source computer vision library that can run on a wide range of systems. OpenCV assists researchers with such a range of tasks.

3.2.2.2 TensorFlow

TensorFlow is an end-to-end open-source platform for machine learning. It incorporates a comprehensive, adaptable biological system of instruments, libraries

and community assets that lets analysts thrust the state-of-the-art in ML and designers effortlessly construct and send ML fueled applications.

3.2.2.3 Matplotlib

Matplotlib may be a comprehensive library for making inactive, energized, and intuitively visualizations in Python. Matplotlib makes simple things simple and difficult things possible. Create distribution quality plots. Make intelligently figures that can zoom, skillet, update. Customize visual fashion and layout. Export to numerous record designs. Embed in Jupiter and Graphical Client Interfaces. Use a wealthy cluster of third-party bundles built on Matplotlib.

3.2.2.4 Pandas

Pandas could be a quick, capable, adaptable, and simple to utilize open-source information investigation and control tool, built on best of the Python programming dialect

3.2.3 Development Tool

3.2.3.1 PyCharm

PyCharm is the most effective IDE to adopt since Python is the key programming language for development. Because the PyCharm IDE has advanced features like intelligent coding aid, built-in database tools, built-in terminal, and python profiler.

3.2.3.2 Colaboratory

Colaboratory, or "Colab" for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. More technically, Collab is a hosted Jupiter notebook service that requires no setup to use, while providing free access to computing resources including GPUs.

3.2.3.3 Visual Studio Code

Visual Studio Code was used as an additional development tool, particularly for writing and debugging C++ code. It offers extensive extensions, an intuitive user

interface, and built-in Git support, making it an effective environment for multi-language development.

3.2.4 Version Controlling System

Since this research is done in a group, a mechanism was needed to effectively synchronize everyone's contribution. In the software development process, Git is used as a source code management system. In the software development lifecycle, it keeps track of changes within the development process as versions and enables a collaborative work environment.

3.3 Summary

This chapter took a brief overview of the technologies used in this research study, which were classified into three parts: Machine Learning, Image Processing, and Python.

Chapter 4 - Proposed Solution and Methodology

4.1 Chapter Overview

This chapter provides a detailed description of the suggested solution to the noted issue. Determine the research gap for each module by examining the literature reviews; this chapter will present a fresh strategy based on that analysis. This chapter will cover the model's inputs, working method, suggested solution, output for each module, and evaluation techniques for each model.

4.2 Module 1 - Detection and Classification of Damaged Areas

This module aims detects the damage areas in the text document with advanced image processing techniques and machine leaning techniques.Noise removal techniques including Non-Local Means (NLM) filtering, the system enhances the quality of scanned document images while minimising distortions and maintaining text integrity.simultaneously, mode filtering combined with the Intensity-Based Classification approach efficiently separates text, background, and damaged areas. This preprocessing and segmentation method's integration with contour-based bounding box generation ensures accurate localisation and makes it possible to precisely identify damages in sinhala handwritten documents. Then classify the damaged areas into different types of damages (insect attacks, fading handwritten ink, wear and tear, and ink blotches) using a CNN model.

4.2.1. Proposed Solution

4.2.1.1 Detection of Damaged Areas

The module utilizes various Digital Image Processing (DIP) techniques to identify the damaged regions in Sinhala handwritten texts. This submodule aims to segment the damaged areas without considering the type of damage.

4.2.2 Methodology for Damage Detection

4.2.2.1 Data Collection

A collection of handwritten Sinhala documents that have been damaged gathered, guaranteeing a range of writing styles, paper textures, and forms of damages. The collection contains typical damage types like physical holes, blurring, insect attacks, and ink smudges.

4.2.2.2 Preprocessing

Old text documents commonly have noises that can affect the damage detection like texture of the papers.This module follows below preprocessing methods to improve image quality and the accuracy of damage detection:

- Grayscale conversion is use to remove the color complexities so can focus on intensity variations.
- Use NLM filtering to remove unwanted texture noises.
- HSI conversion is used to discrete intensity data helps in the identification of faded text and stains.
- Morphological techniques to enhance the segmentation of damaged areas and remove minor unwanted noise.

4.2.2.3 Damage Detection and Bounding Box Generation

The output of this submodule to output text images that has marked the damaged regions with bounding boxes.So to do that this module follow below steps.

- Adaptive thresholding and Otsu thresholding will apply to the damaged text images.
- Canny edge detection use to define the contours of the damaged areas.
- Connected component analysis (CCA) is used to group pixels belonging to the same damaged area.
- Finally draw the bounding boxes around the areas that has identified as damaged.

4.2.1.2 Classification of Damage Types

After completing the damage area detection this sub module aims to categorize those damages in to multiple types, such as blurred text, ink smudges, physical tears, and insect-caused holes.For that module hopes to utilize a deep learning-based classification model.

4.2.3 Methodology for Damage Classification

4.2.3.1 Feature Extraction

To ensure accurate classification, key visual features are extracted from the detected regions:

- Texture analysis to distinguish ink smudges from handwritten text.
- Edge patterns to identify holes and physical tears.
- Intensity variations to detect faded or blurred areas.

4.2.3.2 Model Training

To train the Convolutional Neural Network (CNN) model, we use predefined damage types with datasets for each. Then, the training process can proceed with the following steps.

- Data augmentation (rotation, flipping, contrast adjustments) to improve generalization.
- Categorical labeling where each bounding box is assigned a damage type.
- Optimization techniques like batch normalization and dropout to improve accuracy and prevent overfitting.

4.2.3.3 Validation and Performance Evaluation

After training the model, the validation is done with another dataset. It assists in evaluating the performance.

- Precision, recall, and F1-score to assess classification accuracy.
- Confusion matrix for analyzing misclassifications.
- Hyperparameter optimization to improve the efficiency of the model before deployment.

The classification procedure guarantees that every identified damage is precisely categorized

4.2 Module 2 - Evaluating readability, classifying detected damages, and estimating the extent of damage

4.2.1 Overview

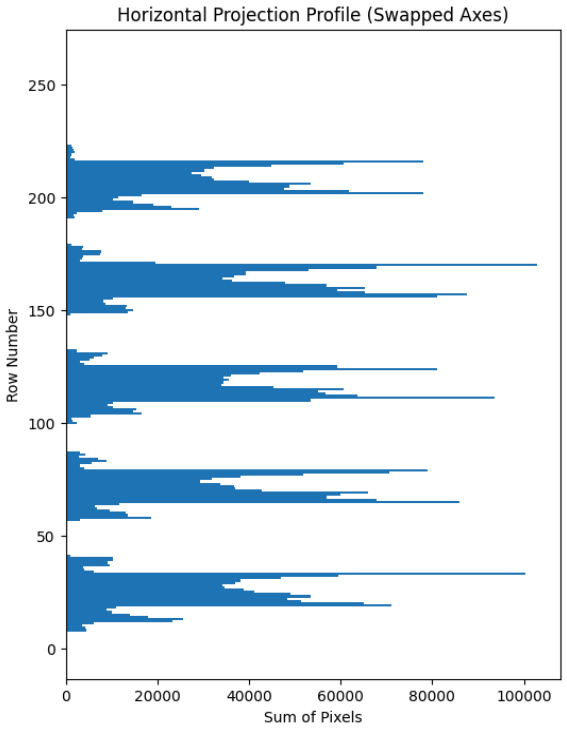
The proposed methodology aims to automate the evaluation of handwriting readability, classify detected damage, and estimate the extent of missing content. The approach integrates image processing techniques, feature extraction, and machine learning models to analyze Sinhala handwritten documents. The system consists of three main stages: handwriting pre-processing, readability assessment, and damage classification.

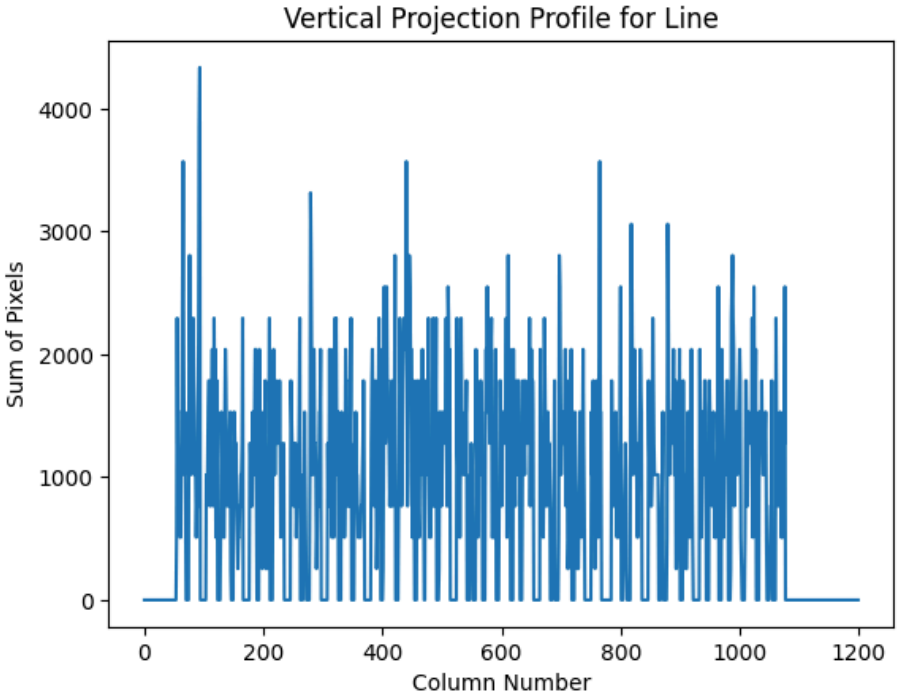
4.2.2 Handwritten document Pre-processing

Pre-processing is an essential step to prepare handwritten documents for analysis. This includes noise removal, binarization, and segmentation into lines, words, and characters. The pre-processing phase ensures that the text is properly aligned and cleaned for feature extraction.

1. Image Acquisition and Normalization:
    - Handwritten documents are digitized using scanning or direct image input.
    - The images undergo noise removal techniques such as Gaussian filtering and thresholding.
    - Skew correction is applied to align the text properly.
2. Segmentation:
    - Line Segmentation: Horizontal projection analysis is used to identify gaps between lines and segment the text accordingly.
    - Word Segmentation: Vertical projection analysis detects spaces between words to segment them accurately.
    - Character Segmentation: A combination of horizontal and vertical projection methods is used to isolate individual characters based on pixel intensity variations.
    - Advantages of Projection-Based Segmentation: These methods are particularly effective for Sinhala handwriting, as they account for variations in stroke density, letter connections, and script-specific

spacing characteristics. By analyzing pixel intensity distributions along horizontal and vertical axes, projection-based segmentation effectively handles complex handwritten structures.





4.2.3 Readability Evaluation

Handwriting readability is assessed based on various handwriting characteristics, including size, slant, spacing, and pen pressure. A numerical readability score is assigned by evaluating these features.

1. Feature Extraction for Readability Analysis
    - Size of Letters: Measures the height and width of characters to assess uniformity. Large handwriting may indicate emphasis, while small handwriting may suggest compactness.
    - Slant of Words and Letters: Determines the angle of handwriting. Right slant, left slant, and vertical orientations are analyzed to identify consistency in writing.

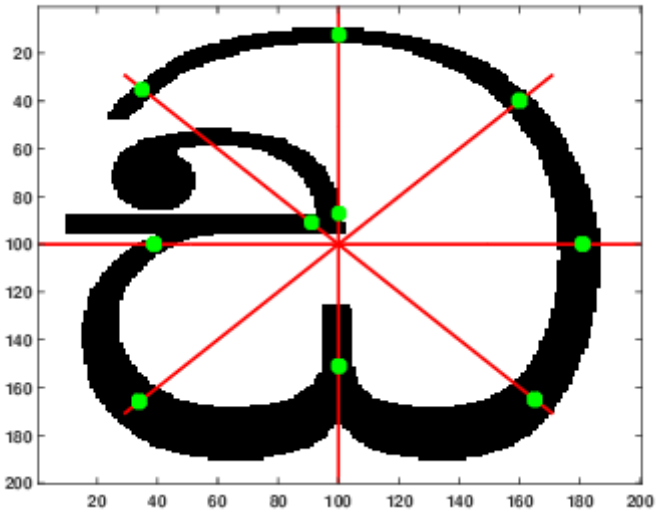


Figure - Letter Angle and size.

- Baseline Consistency: Examines the stability of the writing baseline to determine if the handwriting follows a structured flow.
- Pen Pressure: Measures pixel intensity to determine whether the writer applied light or heavy pressure, impacting readability.
- Spacing Between Letters and Words: Evaluates the uniformity of spacing between characters and words, affecting overall readability.
- Pixel Density of Characters: Analyzes stroke density to detect inconsistencies in letter formation.

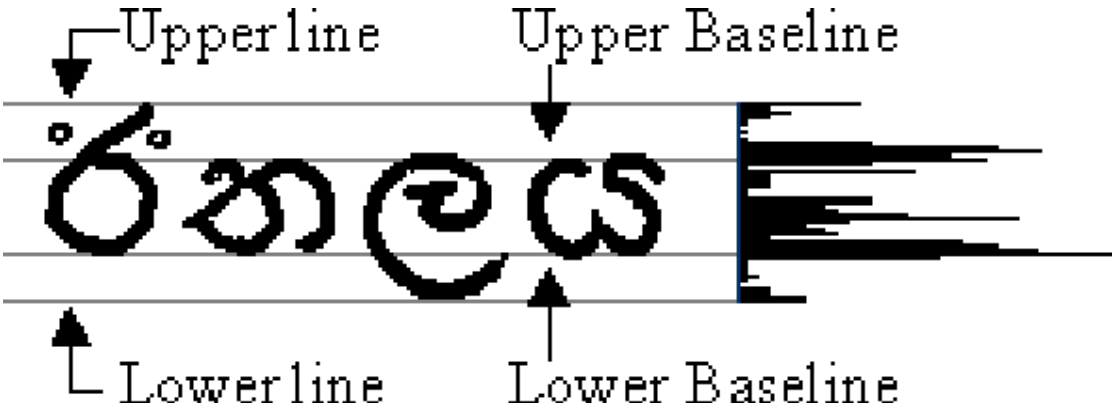


Figure - Baseline Distribution.

2. Readability Scoring System:
    - Each extracted feature is assigned a score based on predefined thresholds.
    - A weighted sum of these scores generates an overall readability score.

4.2.4 Character Recognition and Confidence Evaluation

To further validate the readability score, the system integrates Optical Character Recognition (OCR) using Tesseract OCR. The confidence level of recognized characters is extracted, and a comparison is made between the calculated readability score and the OCR confidence level.

1. Tesseract OCR Implementation:
    - Extracts textual content from the handwritten document.
    - Generates a confidence score indicating the reliability of recognition.
2. Comparison and Decision Making:
    - If the readability score aligns with a high OCR confidence level, the handwriting is classified as readable.
    - If there is a discrepancy between the readability score and OCR confidence, further evaluation is conducted to refine results.

4.2.5 Damage Classification and Estimation of Missing Content

Once damage detection is completed in module 1, the missing content is classified into four categories:

- Part of a letter is missing.
- A complete letter is missing.
- A complete word is missing.
- A few letters of a word are missing.

The extent of missing content is estimated by counting the number of missing letters or words in the third and fourth categories.

1. Damage Detection Techniques:
    - Letter Missing: Connected Component Analysis (CCA) is used to detect missing bounding boxes of letters.
    - Partial Letter Missing: Contour & Stroke Gap Analysis is applied to identify incomplete letter structures.
    - Word Missing: Bounding Box Analysis is used to detect empty word spaces.
    - Partial Word Missing: Word Width Comparison is employed to estimate missing letters based on size differences.
2. Quantifying Damage:
    - Word Length Estimation Using Bounding Box Width: Compares the width of damaged words to undamaged words to estimate the number of missing letters based on character width statistics.
    - Statistical Analysis of Character Widths: A dataset of undamaged handwriting samples is used to create a statistical distribution of character widths, which helps in estimating missing letters more accurately.

The proposed methodology integrates advanced segmentation, feature extraction, and OCR-based validation techniques to evaluate Sinhala handwritten text. By incorporating various handwriting characteristics, the system effectively determines readability, classifies missing content, and estimates the extent of damage, providing a comprehensive solution for handwritten document analysis. Future improvements may include deep learning-based segmentation and classification approaches to further enhance accuracy and robustness.

All these methodologies are currently in the research phase to determine the most effective approach for achieving the highest accuracy. While specific techniques may evolve, the fundamental process remains consistent. Future iterations may refine segmentation, feature extraction, and classification methods to improve performance and adaptability.

4.4 Module 3 - Blurred Text Restoration and Letter-Level Damage Reconstruction

4.4.1 Proposed Solution

4.4.1.1 Restoration of Blurred or Bleed-Through Text

After Module 1 detects areas affected by blurred or bleed-through text, Module 3 applies image enhancement techniques to remove these distortions. By improving text clarity and separating any interfering ink from the background, the module ensures that the text remains readable. Once the necessary corrections are made, the processed text is sent back to Module 1 for further classification, allowing for additional refinement if needed.

4.4.1.2 Reconstruction of Minor Letter-Level Damage

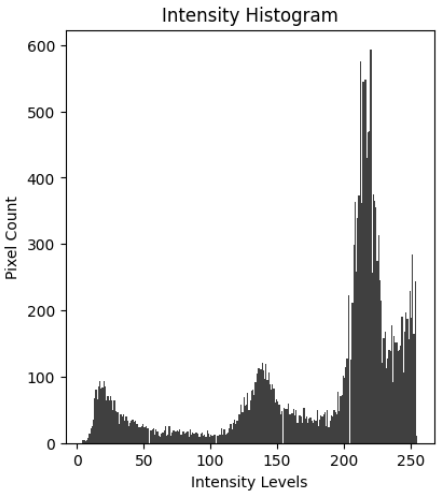
When Module 2 identifies certain damages as minor damages(limited to missing parts of letters without requiring contextual reconstruction) , these affected characters are processed in Module 3. Each damaged area is already classified by Module 1 based on its type, such as ink blotches, wear and tear, insect attacks, or peeled-off sections. Using this classification, Module 3 reconstructs the affected letters by preserving their original stroke patterns and handwriting style, ensuring consistency with the document's natural appearance.

4.4.2 Methodology

4.4.2.1 Restoration of Blurred or Bleed-Through Text

4.4.2.1.1 Text-Blur Separation Using K-Means Clustering

To differentiate between text, background, and blurred/bleed-through areas, the image is first converted into a one-dimensional feature space by vectorizing pixel intensity values. Using K-Means Clustering (K=3), pixels are classified into three categories: background (highest intensity), text (medium intensity), and blurred/bleed-through areas (lowest intensity). Once the clusters are identified, the blurred regions are replaced with the background intensity, reducing unwanted noise and improving text contrast.



4.4.2.1.2 Enhancing Blurred Text Using Super-Resolution and Sharpening

To restore degraded handwriting or faded printed text, Super-Resolution CNN (SRCNN) or ESRGAN is applied to the detected blurred text regions. These models reconstruct sharp edges in letter strokes and enhance the clarity of characters. Additionally, an adaptive sharpening kernel (high-pass filter) is used to emphasize text boundaries, ensuring fine details are preserved while preventing over-sharpening, which could distort the characters.

4.4.2.1.3 Stroke Refinement Using Morphological Thinning

To maintain text consistency and improve OCR readability, morphological thinning is performed. This involves iterative erosion operations to extract the skeletal representation of characters while ensuring that the stroke width remains uniform. A selective enhancement process is applied where only text areas undergo thinning, preventing over-processing of already well-defined characters and ensuring the structural integrity of the text is maintained.

4.4.2.1.4 Final Reconstruction and Smoothing

After text restoration and enhancement, Gaussian blur is selectively applied to remove any harsh edges left from the processing steps. Stroke Width Normalization (SWT - Stroke Width Transform) is then implemented to ensure uniform stroke thickness across the entire text.

4.4.2.2 Letter-Level Damage Reconstruction

This methodology provides a approach to reconstructing damage letter strokes. By dynamically identifying and classifying damage, grouping endpoints intelligently, and using Bezier curve interpolation for smooth stroke reconstruction, this method ensures that handwritten text is restored with high accuracy. The final output maintains the natural flow, structure, and texture of the original handwriting, making the restoration process both effective and visually seamless.

4.4.2.2.1 Extracting Text, Background, and Damaged Regions

The first step involves preprocessing the input image to separate text, background, and damaged regions. The image is first converted to the HSI color space, where the intensity (I) channel is extracted since it holds the most critical information for grayscale-based segmentation. To reduce noise, a Gaussian blur is applied before K-Means clustering, which classifies pixels into three main categories: text (darkest), damage (mid-gray), and background (brightest). The damaged regions are identified based on their intensity values, forming a damage mask that highlights missing parts of the letters. Since the damage divider may be angled, this step dynamically detects the orientation of the damage to ensure an accurate reconstruction.

4.4.2.2.2 Skeletonization and Endpoint Detection

Once the damaged text and background are separated, the next step involves skeletonization, which reduces the text to one-pixel-wide strokes, preserving the original handwriting structure while eliminating unnecessary noise. This step is crucial for identifying the core shape of each letter and its missing parts. After skeletonization, endpoint detection is performed by scanning pixel neighborhoods: points with only one connected neighbor are classified as endpoints. These endpoints act as key reference markers for the upcoming stroke reconstruction phase.

4.4.2.2.3 Identifying the Angled Damage Divider and Grouping Endpoints

Unlike conventional methods that assume a horizontal or vertical break in the text, this approach identifies an angled damage divider by analyzing the shape and contour of the damaged region. A best-fit line is drawn along the damage, dividing the detected

endpoints into two groups. Group 1 contains endpoints on one side of the divider, while Group 2 contains endpoints on the opposite side. This grouping ensures that only logical stroke connections are attempted, preventing errors that could distort the natural handwriting flow.

4.4.2.2.4 Matching Endpoints Across the Angled Damage Divider

With the endpoints grouped, the next step is to find matching pairs for reconstruction. A multi-step matching strategy is used, starting with closest endpoint detection using Euclidean distance. Additional validation checks ensure that the stroke continuity is aligned with the original text, and an angle-based similarity measure prevents incorrect pairings. If an immediate match is not found, the search radius is gradually expanded, allowing for a more flexible reconstruction while maintaining accuracy. This step ensures that broken strokes are correctly reconnected even when text curvature varies.

4.4.2.2.5 Reconstructing Missing Strokes Using Bezier Curves

Once endpoints are successfully paired, the missing strokes are reconstructed using Bezier curves. A smooth, natural-looking stroke is generated by placing control points along the detected stroke gradient, ensuring fluidity in the reconstruction. B-Spline smoothing is applied to refine the curves and eliminate unnatural bends. Additionally, stroke thickness is dynamically adjusted to match the original handwriting, preserving consistency across the restored text. Since the damage divider is angled, reconstructed strokes are carefully aligned with the original letter orientation to avoid distortions.

4.4.2.2.6 Integrating Reconstructed Strokes into the Original Image

The final step involves blending the reconstructed strokes into the original text seamlessly. This is achieved by ensuring that the stroke width, intensity, and texture match the surrounding text, preventing visible inconsistencies. Adaptive blending techniques merge the new strokes smoothly while preserving the handwriting's natural appearance. To prevent visual artifacts, region-growing segmentation is used to restore the background, ensuring that the repaired area does not appear artificially modified. This step guarantees that the restored text remains visually coherent and indistinguishable from the original document.

4.5. Module 4 - Restoring missing letters and words based on surrounding context

It is a known fact that historical documents, manuscripts, or printed materials often suffer from wear and tear, leading to missing or illegible text. The content in such resources is more vulnerable to destruction when it they are in a low-resourced language. Since Sinhala is one such low-resourced language used only in Sri Lanka, the reconstruction of the damaged Sinhala handwritten texts was considered an action that was of immense importance.

4.5.1. Proposed Solution

The approach here is to fine-tune a GPT-based model to reconstruct missing letters and words using its contextual understanding capabilities. The model will predict missing elements based on surrounding text, ensuring grammatical and syntactical correctness.

4.5.1.1.Data Collection

The dataset sourced for the entire research is based on G.C.E. A/L Physics notes. Thereby, the model will be trained using the G.C.E. A/L Sinhala Medium Physics Resource Book.

4.5.1.2. Data Preprocessing

Text will be extracted from the obtained Resource book which exists in pdf format, using tools like PyPDF2. Moreover,The data will be cleaned removing unnecessary parts like headers, footers, unwanted spaces, etc. And made ready for the fine-tuning process.

4.5.1.3. Model Selection

Research was done with different pre-trained models (Sinhala-GPT-2, XLM-R, mT5) for compatibility with Sinhala texts. As per the current findings, Sinhala-GPT 2 seems to be the best to go with restoring missing Sinhala texts.

Justification for my choice is as follows,

1. Sinhala-GPT-2 follows a causal language modeling (CLM) approach, which means it generates text from left to right and is well-suited for reconstructing missing sequences (letters or words).
2. XLM-R is a masked language model (MLM) trained to fill in missing words but is not optimized for letter-level reconstruction.
3. As in this case the dataset has both missing words and missing letters, GPT-based models similar to Sinhala-GPT-2 are better because they generate sequences, rather than just predicting masked tokens.

4.5.1.4. Fine-Tuning the Model

For fine-tuning, transfer learning is used with pre-trained transformer models. Moreover, several hyperparameters (learning rate, batch size, number of epochs) . The model is trained by using masked text and complete text as input-output pairs.

4.5.1.5. Evaluation Metrics

1. Accuracy can be taken by comparing the generated text with ground truth.
2. BLEU Score can be used to measure the similarity between reconstructed and actual text.
3. Human Evaluation can also be done with the aid of native speakers to assess fluency and correctness.

4.5 Chapter Summary

This chapter explains our approach to the proposed solution. We describe the inputs, outputs and processes of the system and what we are going to implement. In the next chapter we describe the analysis and design part of our proposed solution.

Chapter 5 - Analysis and Design

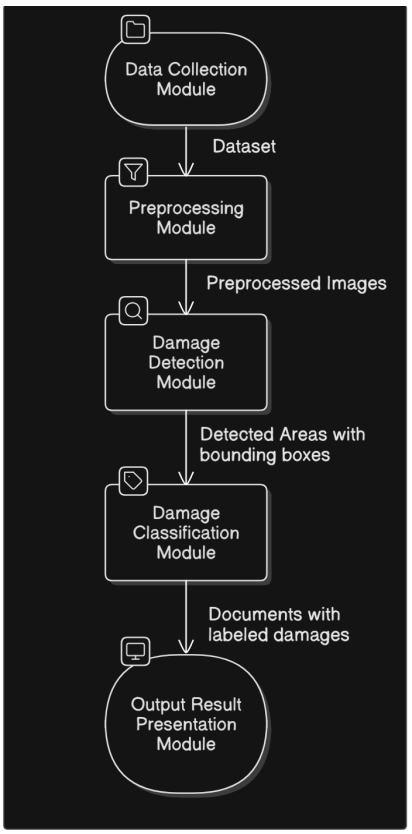
5.1 Chapter Overview

In this chapter, the suggested solution for each module will be explained together with analysis and design. This is where the development process for each module will be shown.

5.2 Analysis and Design of Module 1 - Detection and Classification of Damaged Areas

5.2.1 High-Level System Design

A method for identifying and categorizing damaged areas in handwritten Sinhala documents is examined and designed in this section. The system is composed of several modules, each of which is in responsible for a certain function and works in along with the others to deliver a seamless solution. The data flow and interactions between the modules are depicted in the high-level design diagram that follows.



High-Level Design of the Damaged Area Detection and Classification System

1. **Data Collection Module:** This is a common module for each member. This module is responsible for collecting various types of damaged Sinhala handwritten documents, including different writing patterns and damage types such as ink smudges, blurring, and physical tears.
2. **Preprocessing Module:** this module is responsible to enhance suitable image features by removing unnecessary noises , so the damage detection task can be done accurately.It includes grayscale conversion, NLM filtering, and HSI conversion to extract intensity information.
3. **Damage Detection Module:** This module is used to detect damaged areas using thresholding techniques, edge detection, and morphological operations. It captures the damaged areas, applies mode filtering, and draws bounding boxes around the detected damaged regions.
4. **Damage Classification Module:** This module classifies the damaged areas based on predefined damage types such as blurred text, ink smudges, holes, or

wear and tear. A Convolutional Neural Network (CNN) is used to train the model and classify each damaged region.

5. **Output/Result Presentation Module:** This module presents the results, displaying the detected damaged areas and their corresponding damage types on the document images.

5.2.2 Workflow and Interaction Between Modules

The workflow begins with the Data Collection Module, where a diverse set of damaged handwritten documents is gathered. The collected data is then passed to the Preprocessing Module for noise reduction and enhancement. Preprocessed images are fed into the Damage Detection Module, which identifies the damaged areas by drawing bounding boxes around them.

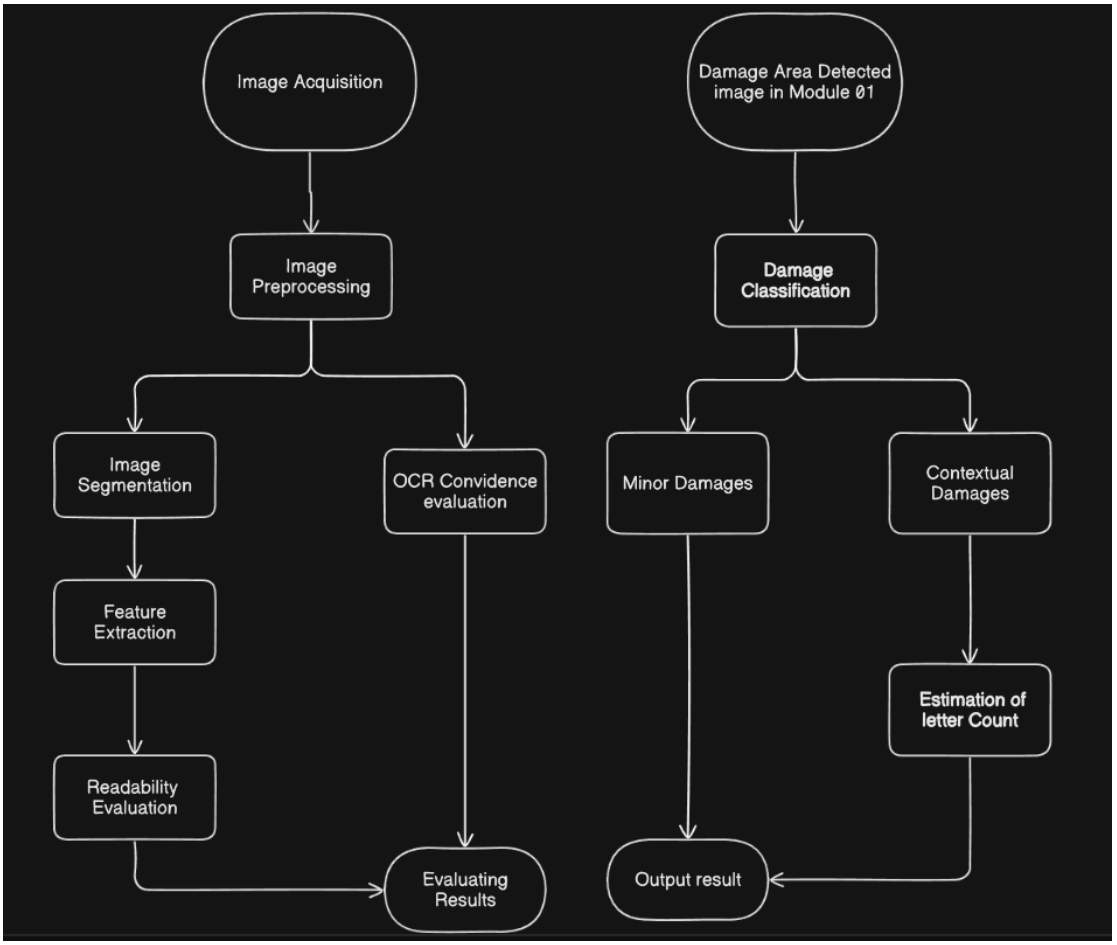
Once the bounding boxes are generated, the Damage Classification Module classifies each damaged region into specific damage types using a CNN-based model. The output, including the identified damage types and bounding box coordinates, is then passed to the Output/Result Presentation Module for display.

The system's modular design ensures that each part functions independently but is integrated effectively to achieve the overall goal of detecting and classifying damaged areas in handwritten Sinhala documents.

5.3 Analysis and Design of Module 2 - Evaluating Readability, Classifying Detected Damages, and Estimating the Extent of Damage

This section outlines the design and analysis of the system for evaluating readability, classifying damages, and estimating the extent of missing content in Sinhala handwritten documents. The system addresses challenges inherent to the Sinhala script, such as its complex structure, variability in handwriting styles, and the difficulty of accurately detecting and estimating missing or damaged content. Below is the detailed technical framework that provides a solution.

5.3.1 Architecture of the Model



High Level Design for Evaluating Readability, Classifying Detected Damages, and Estimating the Extent of Damage

Input Layer:

The system receives scanned images of handwritten Sinhala documents, which are pre-processed for further analysis.

Pre-processing Layer:

This layer is responsible for cleaning up the images before segmentation. This includes noise removal, skew correction, and initial image enhancement.

Segmentation Layer:

Segmentation plays a critical role in the recognition process. This layer handles the task of segmenting the pre-processed image into distinct lines, words, and characters. The system uses:

- **Line Segmentation**: Horizontal projection techniques identify gaps between lines, enabling the segmentation of the document into individual lines of text.

- **Word Segmentation**: Vertical projection techniques detect spaces between words, accurately segmenting the text at the word level.
- **Character Segmentation**: A combination of horizontal and vertical projection methods is used to isolate characters within words.

Feature Extraction Layer:
In this layer, various handwriting features are extracted to assess readability. Features like character size, slant, spacing, and pen pressure are measured. Pixel intensity data is also analyzed to detect inconsistencies in stroke formation and spacing.

Damage Classification Layer:
Once damage is detected, this layer classifies the missing content into four categories:

- Part of a letter is missing.
- A complete letter is missing.
- A complete word is missing.
- A few letters of a word are missing.

Readability Scoring Layer:
In this layer, a numerical readability score is generated based on the extracted features. The score represents how legible the handwritten text is, based on characteristics such as letter size, slant, and spacing.

Output Layer:
The system outputs the reconstructed text with the estimated missing content, classified damage types, and the overall readability score.

5.3.2 Basic System Workflow

Step 1: Input
Handwritten Sinhala documents are scanned or digitized and fed into the system as input images.

Step 2: Data Preprocessing
The input image undergoes several pre-processing steps:

- **Noise Removal**: Gaussian filtering and thresholding techniques are applied to clean up the image.
- **Skew Correction**: Alignment of the text is performed to ensure that the handwriting follows a consistent baseline.
- **Segmentation**: The pre-processed image is segmented into lines, words, and characters using projection-based segmentation methods.

Step 3: Segmentation

Segmentation is a crucial step in ensuring that the individual components of the handwritten text (lines, words, and characters) are accurately isolated for further analysis. This step uses:

- **Line Segmentation**: Horizontal projections to detect line gaps.
- **Word Segmentation**: Vertical projections to detect spaces between words.
- **Character Segmentation**: A hybrid projection method that identifies individual characters within words.

Step 4: Readability Evaluation

Once segmentation is completed, features like size, slant, baseline consistency, and spacing are extracted from the segmented text. These features are used to calculate a readability score for the document, reflecting how easily it can be read based on these factors.

Step 5: OCR Confidence Evaluation

Optical Character Recognition (OCR) is applied to extract textual content from the segmented text. The OCR output is compared with the calculated readability score to validate the handwriting quality. If the OCR confidence is low, further analysis is performed to refine the evaluation.

Step 6: Damage Classification and Estimation

After detecting damaged areas, the system classifies the missing content into the following categories:

- **Missing Part of a Letter**: Identified using Connected Component Analysis (CCA).
- **Missing Complete Letter**: Detected through bounding box analysis.

- **Missing Complete Word**: Identified by recognizing empty word spaces.
- **Partial Word Missing**: Estimated using word width comparison.

Quantitative techniques are applied to estimate the extent of the missing content (e.g., comparing word lengths and character widths).

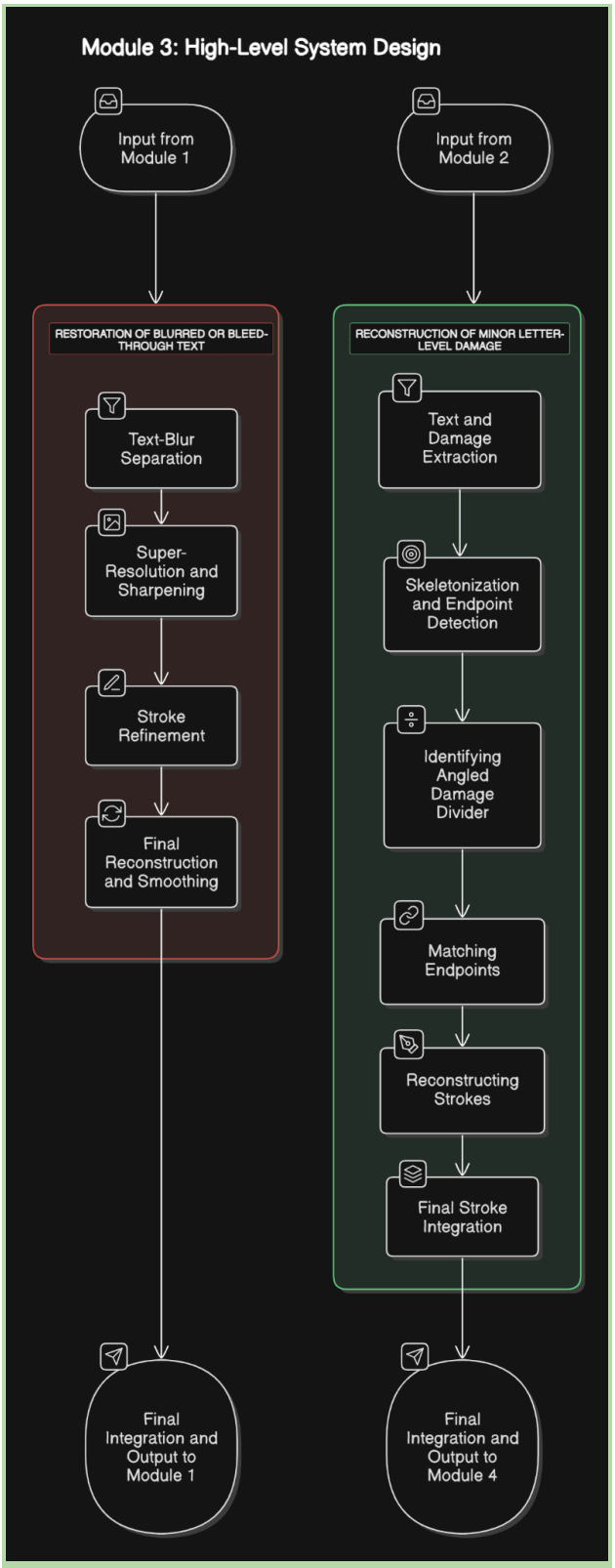
5.3.3 Summary of Module Design

The system's architecture is composed of interconnected layers that perform distinct functions: input processing, pre-processing, segmentation, feature extraction, damage classification, readability scoring, and post-processing. Each layer contributes to the overall accuracy of the final output, which includes a reconstructed, readable document with classified damage and estimates of missing content.

By separating segmentation as its own layer, we ensure that the system is better equipped to handle complex handwriting structures, such as overlapping characters and irregular spacing. The integration of both traditional image processing and modern machine learning techniques allows for robust handling of Sinhala handwritten text, despite its unique challenges.

Future iterations of the system may incorporate more advanced deep learning techniques to further improve segmentation accuracy, damage classification, and overall system performance.

5.4 Analysis and Design of Module 3



High-Level System Design diagram outlines the processing steps for restoring blurred or bleed-through text and reconstructing minor letter-level damage. The module receives input from Module 1 and Module 2, classifies the type of damage, and applies appropriate restoration techniques before passing the processed text forward.

5.4.1 Restoration of Blurred or Bleed-Through Text

When Module 1 detects blurred or bleed-through text, the system applies image enhancement techniques to improve text clarity. The restoration process follows these key steps:

1. Text-Blur Separation: K-Means clustering is used to classify pixels into background, text, and blur regions, allowing unwanted noise to be removed.
2. Super-Resolution and Sharpening: Deep learning models such as SRCNN or ESRGAN reconstruct fine details, while adaptive sharpening enhances text boundaries.
3. Stroke Refinement: Morphological thinning ensures consistent stroke width, improving OCR readability.
4. Final Reconstruction and Smoothing: Gaussian blur removes harsh artifacts, and Stroke Width Transform (SWT) normalizes stroke thickness.
5. Final Integration: The restored text is returned to Module 1 for further classification and verification if needed.

5.4.2 Reconstruction of Minor Letter-Level Damage

When Module 2 identifies minor damages (such as broken letter strokes), the system reconstructs missing portions while preserving handwriting consistency. The process follows these steps:

1. Text and Damage Extraction: K-Means clustering is used to segment text, background, and damaged regions.
2. Skeletonization and Endpoint Detection: Text is converted to one-pixel-wide strokes, and endpoints of broken strokes are identified.
3. Identifying the Angled Damage Divider: The orientation of missing strokes is analyzed to ensure logical reconstruction.
4. Matching Endpoints: Endpoints are paired using Euclidean distance and angle-based similarity to restore missing connections.
5. Reconstructing Strokes: Bezier curve interpolation and B-Spline smoothing generate natural-looking strokes that match the handwriting style.
6. Final Stroke Integration: The reconstructed strokes are blended into the document, ensuring seamless restoration.
7. Final Integration: The processed text is sent to Module 4 for further verification and document-level corrections.

This proposed approach allows Module 3 to handle both blurred text restoration and minor letter damage reconstruction, ensuring that handwritten text retains its natural flow, structure, and readability.

5.5 Analysis and Design of Module 4 - Restoring missing letters and words based on surrounding context

This section gives a general description of the technical framework and theoretical analysis behind the GPT model.

Dealing with Sinhala language in NLP might not be an easy task as it seems because of the following challenges.

- Sinhala letters possess a significant amount of complex character structures and diacritic variations.
- Another challenge that the Sinhala language has is that it has limited availability of large-scale, high-quality Sinhala datasets as well.
- Apart from that, its morphological complexity affects word segmentation and prediction in most cases of reconstruction.

By taking the above challenges into consideration, the below procedure was adhered to.

5.5.1. Architecture of the Model

Input Layer: Consists of preprocessed incomplete text of the dataset.

Transformer-based Layers: Utilizes self-attention mechanisms to understand context and predict missing elements, at this layer.

Output Layer: This layer outputs the reconstructed text with all the missing words/letters filled.

5.5.2. Basic System Workflow

Step 1. Input

This is the initiative step of inputting the paragraph with missing letters/words (e.g., damaged text) obtained from the dataset.

Step 2. Data Preprocessing

At this step, the dataset undergoes tokenization and normalization of text to handle diacritics, special characters, etc. Apart from that, controlled missing letters/words (simulate real-world conditions) are also introduced.

Additionally, the dataset will be split into training (90%), validation (10%).

Step 3. Fine-Tuning the Model

During this step, the pre-trained Sinhala-GPT 2 model is loaded. The data should be prepared by masking text as input, and complete text as output. Thereafter, the model should be fine-tuned using transfer learning. The hyperparameters like learning rate, batch size, and epoch are to be optimized here.

Step 4. Model Inference

At this step, incomplete text is input into the fine-tuned model. Model processes context and predicts missing letters/words and output reconstructed text.

Step 5. Post-Processing

During this step, unwanted spaces, diacritic mismatches, or incorrect predictions are removed and language-specific rules are applied, if any, to improve readability.

Step 6. Evaluation

To evaluate, the reconstructed text is compared with ground truth using metrics like accuracy, BLEU Score, and human accuracy**.**

7. Output

This denotes the final reconstructed context with missing letters/words accurately filled.

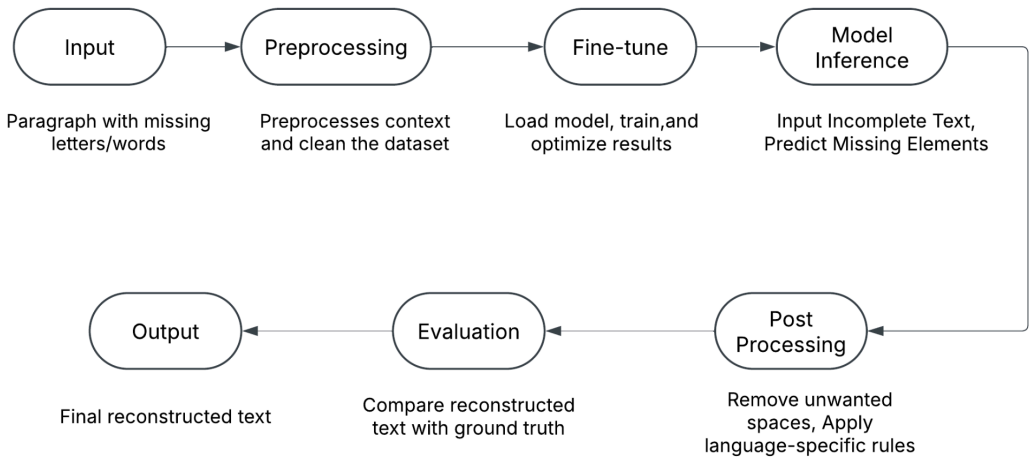


Fig: Basic system workflow of Module 4

5.6 Chapter Summary

In this chapter we explained the analysis and design for each module in the system. These four modules collectively contribute essential components to the Intelligent Sinhala text reconstruction System. Module 1 detects the damages seamlessly, module 2 categorizes the damages and ranks the handwriting , the third module addresses the blurred and bleed out parts,and replaces the missing parts of a single letter, whereas finally the GPT model fills out the rest of the context whether it is a letter or a word.Together, they form a robust and adaptable system aimed at reconstructing damaged handwritten documents which will eventually benefit the Sri Lankan society and the world inevitably. In the next chapter the implementation process of each module will be discussed.

Chapter 6 - Implementation

6.1 Chapter Overview

The use of the technologies and models covered in the previous chapter is covered in this chapter. A range of information on the models and preprocessed datasets is also provided.

6.2 Module 1 - Detection and Classification of Damaged Areas

In this section, we provide the implementation details for the modules in the system, focusing on damage detection and the drawing of bounding boxes for the damaged areas in Sinhala handwritten documents. The approach involves several image processing techniques, including grayscale conversion, noise filtering, intensity extraction using HSI conversion, and classification of damage areas. We then use contour detection to identify the damaged areas and draw bounding boxes around them.

6.2.1.Modules in the Design

6.2.1.1. Image Preprocessing

The first step in damage detection is preprocessing the images. The images are first converted to grayscale, followed by noise reduction. A Non-Local Means (NLM) filter is applied to the grayscale image to remove noise while preserving edges. This process ensures that the subsequent steps can focus on relevant features without being disturbed by noise artifacts.

Key Code Segment for Preprocessing

```
# Convert the image to grayscale
grayscale_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply NLM filter to reduce noise
```

```
filtered_image = cv2.fastNlMeansDenoising(grayscale_image,
None, 30, 7, 21)
```

6.2.1.2. HSI Conversion and Intensity Extraction

The filtered grayscale image is then converted to the HSI color space (Hue, Saturation, Intensity), focusing on the intensity channel. This step aids in identifying areas of the document that are either damaged or exhibit significant visual anomalies. Intensity variations can often correspond to damage, such as blurred or smudged areas.

Key Code Segment for HSI Conversion and Intensity Extraction:

```
# Convert grayscale to HSI and extract the Intensity channel
def convert_to_hsi_and_extract_intensity(image):
    gray_3_channel = cv2.merge([image, image, image])

# Convert to 3-channel grayscale
    hsi_image = cv2.cvtColor(gray_3_channel,
cv2.COLOR_BGR2HSV)

# Convert to HSI
    intensity_channel = hsi_image[:, :, 2]  # Extract the
intensity channel
    return intensity_channel
```

6.2.1.3. Classification of Damaged Areas

The next step involves classifying the areas in the image as either background, text, or damaged regions. Thresholding is applied to the intensity channel to create masks for each region. Damaged areas typically have high intensity values, while background and text areas have lower intensity values.

Key Code Segment for Classification:

```
def classify_areas(intensity_image):
    # Thresholds for background, text, and damaged areas
    background_thresh = 80
    text_thresh = 150
    damaged_thresh = 255

    # Create masks based on intensity thresholds
    background_mask = intensity_image < background_thresh
```

```
    text_mask = (intensity_image >= background_thresh) &
(intensity_image < text_thresh)
    damaged_mask = intensity_image >= text_thresh

    return background_mask, text_mask, damaged_mask
```

6.2.1.4. Highlighting the Damaged Areas

The classified damaged areas are then highlighted by drawing small squares on the detected regions. These highlighted areas are further processed using a mode filter to smooth out noise in the detected damage areas.

6.2.1.5. Mode Filtering for Noise Reduction

To enhance the detection of the damaged regions, a mode filter is applied to the highlighted image. The mode filter helps in removing small noise artifacts that may have been wrongly identified as damage.

Key Code Segment for Mode Filtering:

```
# Apply mode filter to the image
def apply_mode_filter(image, kernel_size=5):
    height, width = image.shape[:2]
    filtered_image = image.copy()
    kernel_half = kernel_size // 2

    for y in range(kernel_half, height - kernel_half):
        for x in range(kernel_half, width - kernel_half):
            neighborhood = image[y - kernel_half:y +
kernel_half + 1, x - kernel_half:x + kernel_half + 1]
            neighborhood_flat = neighborhood.flatten()
            mode_value =
Counter(neighborhood_flat).most_common(1)[0][0]
            filtered_image[y, x] = mode_value

    return filtered_image
```

6.2.1.6. Contour Detection and Bounding Box Drawing

After filtering the image to enhance the visibility of the damaged regions, contours are detected in the non-black areas (damaged regions). Bounding boxes are drawn around these contours, highlighting the damaged areas in the document.

Key Code Segment for Bounding Box Detection:

```
# Find contours and draw bounding boxes
contours, _ =
cv2.findContours(non_black_mask.astype(np.uint8),
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Draw bounding boxes around detected damage areas
image_with_bboxes = image.copy()
for contour in contours:
    if cv2.contourArea(contour) > 50:
 # Ignore small contours
        x, y, w, h = cv2.boundingRect(contour)
        cv2.rectangle(image_with_bboxes, (x, y), (x + w, y +
h), (0, 0, 255), 2)  # Red bounding box
```

6.2.1.7 Displaying the Result

Finally, the image with the drawn bounding boxes is displayed, providing a clear visualization of the detected damaged areas.

Key Code Segment for Displaying the Image:

```
# Display the original image with bounding boxes
plt.figure(figsize=(8, 8))
plt.imshow(cv2.cvtColor(image_with_bboxes,
cv2.COLOR_BGR2RGB))
plt.title(f"Bounding Boxes on Original Image: {image_path}")
plt.axis('off')
plt.show()
```

This module's implementation combines greyscale conversion, noise reduction, intensity analysis, and contour recognition to effectively identify damaged areas in handwritten Sinhala documents. Bounding boxes are used to sketch and highlight the regions that have been identified as damaged, giving a clear picture of the damage. After that, this output can be utilized for labeling the damage types , additional analysis, restoration, or classification.

6.3 Module 2 - Evaluating Readability, Classifying Detected Damages, and Estimating the Extent of Damage

Implementation for Image preprocessing and line segmentation, word segmentation and character segmentation

```
# Step 1: Install Required Libraries
!pip install opencv-python-headless numpy matplotlib

# Step 2: Import Libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Step 3: Load and Preprocess the Image
def preprocess_image(image_path):
    # Load the original image
    original_image = cv2.imread(image_path)

    # Display the original image
                    plt.imshow(cv2.cvtColor(original_image,
cv2.COLOR_BGR2RGB))
    plt.title('Original Image')
    plt.show()

    # Convert the image to grayscale
              gray_image    =    cv2.cvtColor(original_image,
cv2.COLOR_BGR2GRAY)

    # Display the grayscale image
    plt.imshow(gray_image, cmap='gray')
    plt.title('Grayscale Image')
    plt.show()

    # Apply Gaussian Blur to reduce noise
    blurred = cv2.GaussianBlur(gray_image, (5, 5), 0)
```

```
    # Display the blurred image
    plt.imshow(blurred, cmap='gray')
    plt.title('Blurred Image')
    plt.show()

      # Apply Binary Thresholding (invert the image for
projection)
        _, binary = cv2.threshold(blurred, 128, 255,
cv2.THRESH_BINARY_INV)

    # Display the binary image
    plt.imshow(binary, cmap='gray')
    plt.title('Binary Image (Inverted)')
    plt.show()

    return binary

# Step 4: Line Segmentation using Horizontal Projection
def segment_lines(binary):
     # Calculate horizontal projection (sum of pixels along
rows)
    horizontal_projection = np.sum(binary, axis=1)

     # Plot the horizontal projection histogram with swapped
axes
    plt.figure(figsize=(6, 8))
                plt.barh(range(len(horizontal_projection)),
horizontal_projection, height=1.0)
       plt.title('Horizontal Projection Profile (Swapped
Axes)')
    plt.xlabel('Sum of Pixels')
    plt.ylabel('Row Number')
    plt.show()

    # Find lines based on peaks and valleys
```

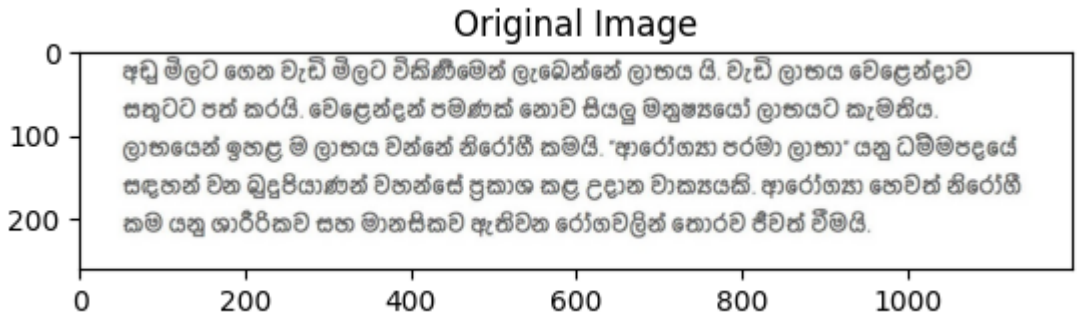
```
    peaks = np.where(horizontal_projection > 0)[0]
    lines = []
    start = peaks[0]

    for i in range(1, len(peaks)):
        if peaks[i] - peaks[i-1] > 1:  # Threshold for line
separation
            lines.append((start, peaks[i-1]))
            start = peaks[i]
    lines.append((start, peaks[-1]))

    # Display segmented lines
    for i, (start, end) in enumerate(lines):
        line = binary[start:end, :]
        plt.imshow(line, cmap='gray')
        plt.title(f'Line {i+1}')
        plt.show()

    return lines, binary

# Step 5: Word Segmentation using Vertical Projection
def segment_words(line_image):
    # Calculate vertical projection (sum of pixels along
columns)
    vertical_projection = np.sum(line_image, axis=0)

    # Plot the vertical projection histogram
    plt.plot(vertical_projection)
    plt.title('Vertical Projection Profile for Line')
    plt.xlabel('Column Number')
    plt.ylabel('Sum of Pixels')
    plt.show()

    # Find words based on peaks and valleys
    peaks = np.where(vertical_projection > 0)[0]
    words = []
```

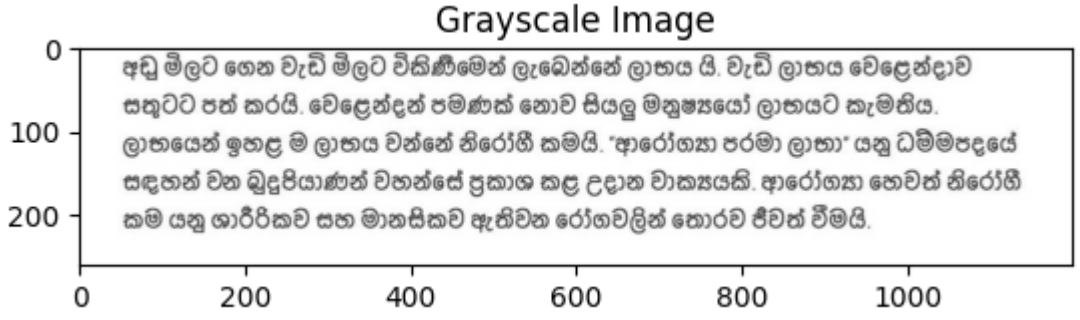
```
    start = peaks[0]

    for i in range(1, len(peaks)):
        if peaks[i] - peaks[i-1] > 10:  # Adjust threshold
for word separation
            words.append((start, peaks[i-1]))
            start = peaks[i]
    words.append((start, peaks[-1]))

    # Create a copy of the line image to draw rectangles
        line_with_rectangles  =  cv2.cvtColor(line_image,
cv2.COLOR_GRAY2BGR)

    # Draw rectangles around each word
    for (start, end) in words:
            cv2.rectangle(line_with_rectangles, (start, 0),
(end, line_image.shape[0]), (0, 255, 0), 2)

    # Display the entire line with rectangles around words
            plt.imshow(cv2.cvtColor(line_with_rectangles,
cv2.COLOR_BGR2RGB))
    plt.title('Line with Segmented Words')
    plt.show()

    # Display each word individually
    for i, (start, end) in enumerate(words):
        word = line_image[:, start:end]
        plt.imshow(word, cmap='gray')
        plt.title(f'Word {i+1}')
        plt.show()

    return words, line_image

# Step 6: Character Segmentation using Vertical Projection
def segment_characters(word_image):
```

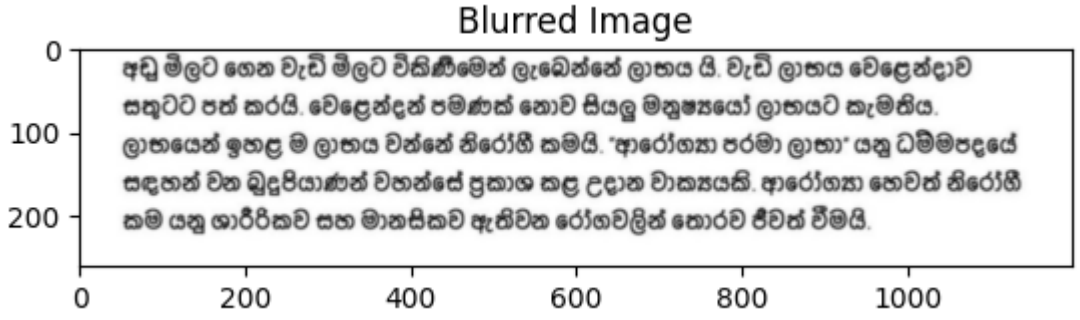
```
    # Calculate vertical projection (sum of pixels along
columns)
    vertical_projection = np.sum(word_image, axis=0)

    # Plot the vertical projection histogram
    plt.plot(vertical_projection)
    plt.title('Vertical Projection Profile for Word')
    plt.xlabel('Column Number')
    plt.ylabel('Sum of Pixels')
    plt.show()

    # Find characters based on peaks and valleys
    peaks = np.where(vertical_projection > 0)[0]
    characters = []
    start = peaks[0]

    for i in range(1, len(peaks)):
        if peaks[i] - peaks[i-1] > 2:  # Adjust threshold
for character separation
            characters.append((start, peaks[i-1]))
            start = peaks[i]
    characters.append((start, peaks[-1]))

    # Create a copy of the word image to draw rectangles
        word_with_rectangles = cv2.cvtColor(word_image,
cv2.COLOR_GRAY2BGR)

    # Draw rectangles around each character
    for (start, end) in characters:
            cv2.rectangle(word_with_rectangles, (start, 0),
(end, word_image.shape[0]), (0, 255, 0), 2)

    # Display the entire word with rectangles around
characters
            plt.imshow(cv2.cvtColor(word_with_rectangles,
cv2.COLOR_BGR2RGB))
```

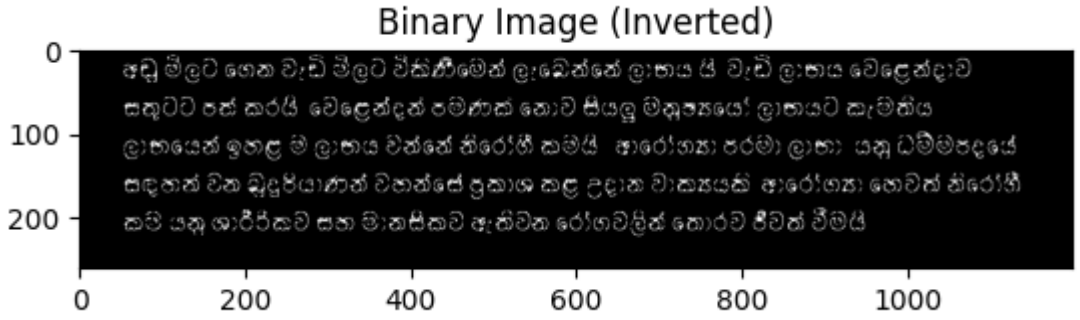
```
    plt.title('Word with Segmented Characters')
    plt.show()

    # Display each character individually
    for i, (start, end) in enumerate(characters):
        character = word_image[:, start:end]
        plt.imshow(character, cmap='gray')
        plt.title(f'Character {i+1}')
        plt.show()

# Main Function
def main(image_path):
    # Step 1: Preprocess the image
    binary = preprocess_image(image_path)

    # Step 2: Segment lines
    lines, binary = segment_lines(binary)

    # Step 3: Segment words in the first line
    if len(lines) > 0:
         line_image = binary[lines[0][0]:lines[0][1], :]  # Take the first line
        words, line_image = segment_words(line_image)

        # Step 4: Segment characters in the first word
        if len(words) > 0:
                                word_image  =  line_image[:, words[0][0]:words[0][1]]  # Take the first word
            segment_characters(word_image)

# Run the program
if __name__ == "__main__":
    image_path = '/content/sinhala4.jpg'
    main(image_path)
```

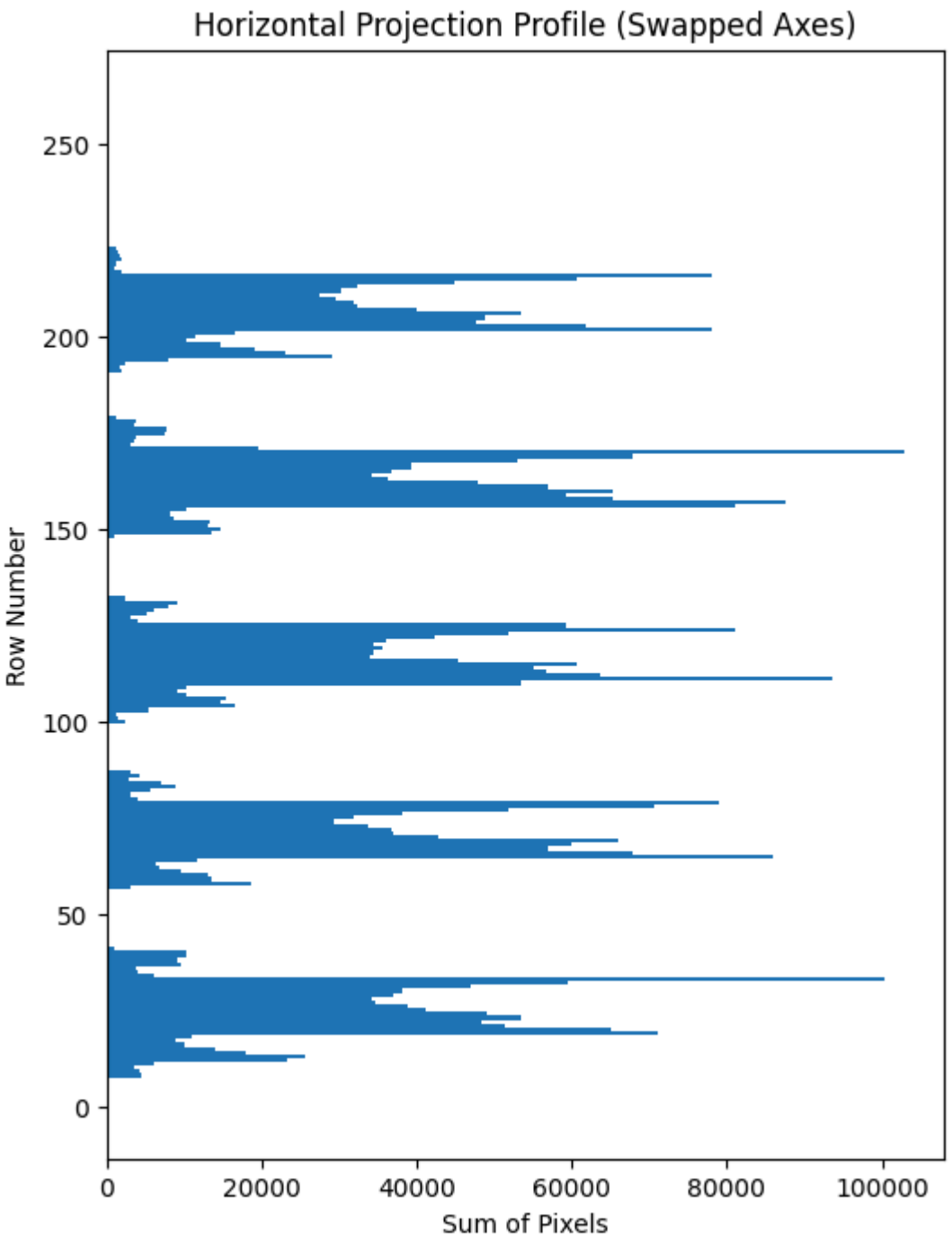
Output Results

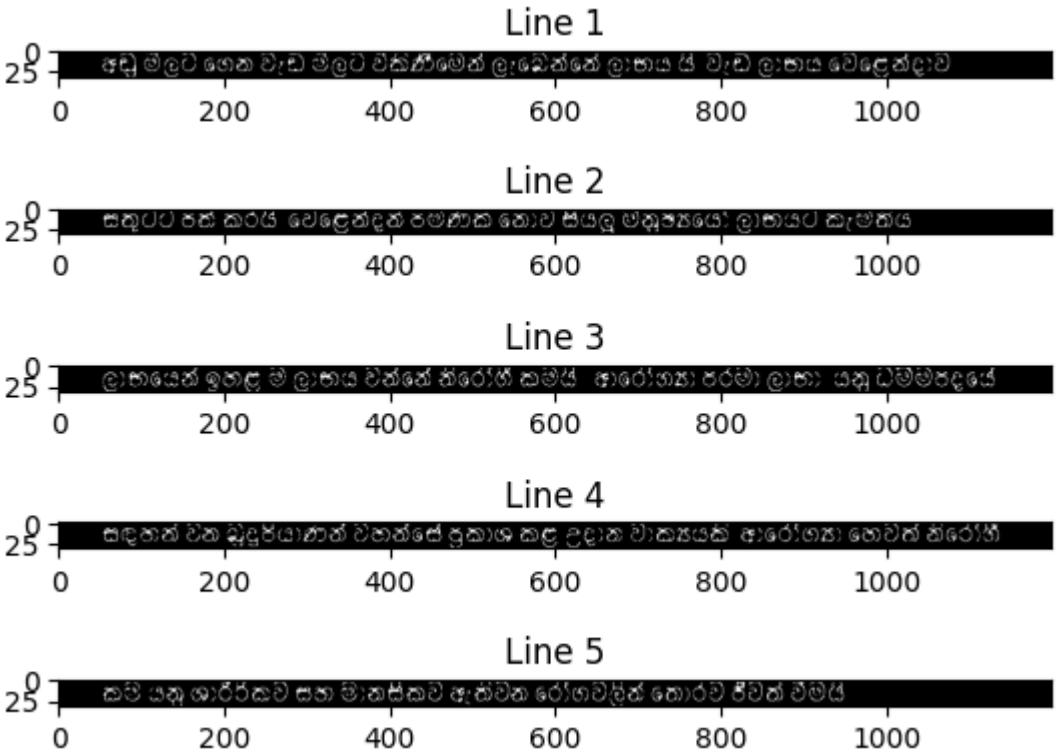


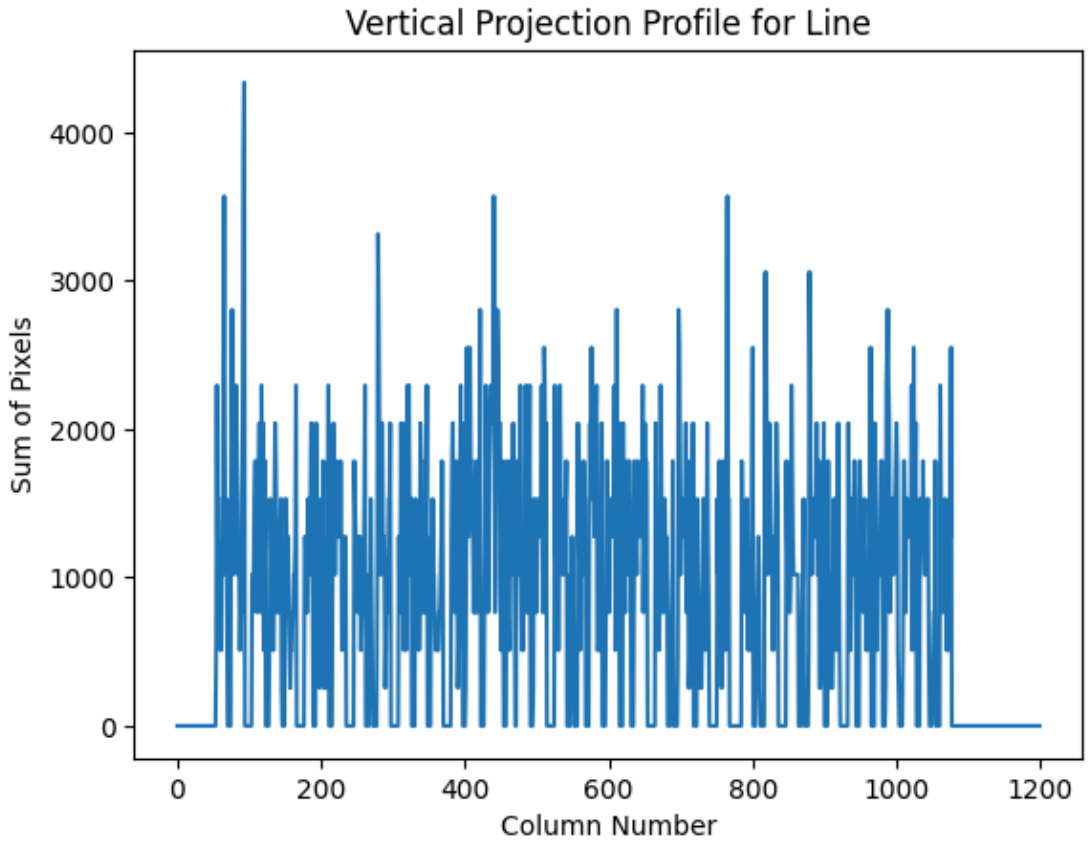


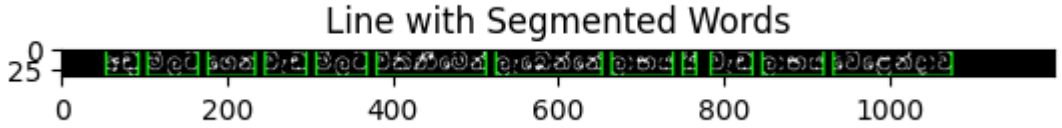


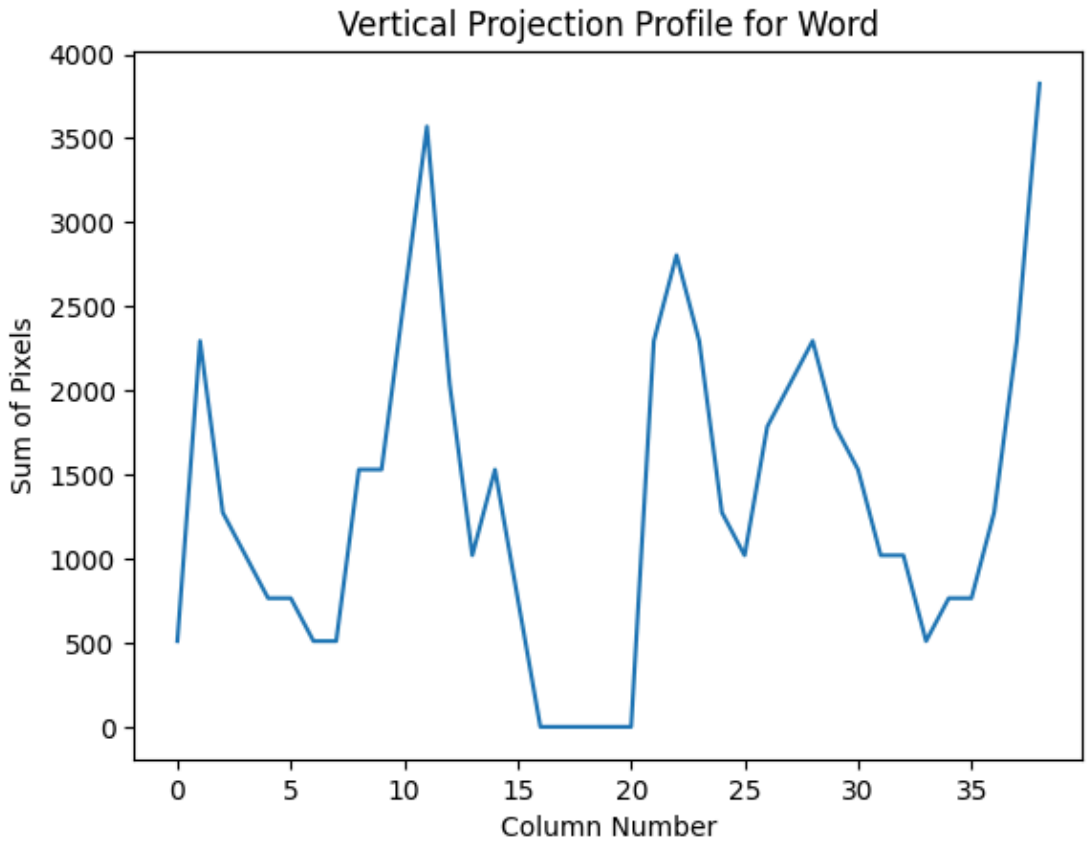


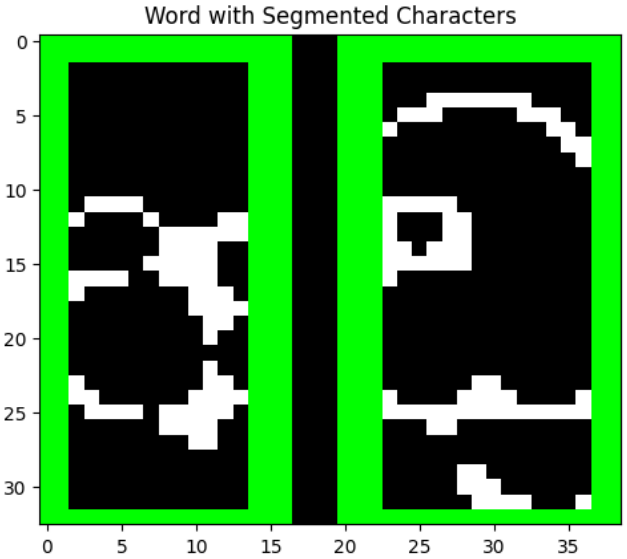

Horizontal Projection Profile (Swapped Axes)
Row Number
250
200
150
100
50
0
0
20000
40000
60000
80000
100000
Sum of Pixels


Line 1
Line 2
Line 3
Line 4
Line 5
0
25
0
200
400
600
800
1000


Vertical Projection Profile for Line
Sum of Pixels
Column Number
0
1000
2000
3000
4000
0
200
400
600
800
1000
1200


Line with Segmented Words
0
25
0
200
400
600
800
1000


Vertical Projection Profile for Word
Sum of Pixels
4000
3500
3000
2500
2000
1500
1000
500
0
0
5
10
15
20
25
30
35
Column Number


Word with Segmented Characters
0
5
10
15
20
25
30
0
5
10
15
20
25
30
35

6.4 Module 3 - Blurred Text Restoration and Letter-Level Damage Reconstruction

6.4.1 Restoration of Blurred or Bleed-Through Text

The restoration process begins with Text-Blur Separation using K-Means clustering (K=3) to classify pixels into text, blurred areas, and background, replacing blurred regions with background intensity to remove noise. Next, Super-Resolution (SRCNN/EDSR) enhances fine text details, while adaptive sharpening with a high-pass filter sharpens text boundaries without distortion. Morphological thinning is then applied to refine stroke width, ensuring text consistency and improving OCR readability. The Final Reconstruction & Smoothing step applies Gaussian blur to remove harsh edges and Stroke Width Transform (SWT) to normalize stroke thickness. Finally, the restored text is reintegrated into Module 1 for further classification and verification.

```
#include "cv.h"
#include "highgui.h"
#include <highgui.hpp>
#include <iostream>
#include <imgproc.hpp>
#include <opencv2/opencv.hpp>
#include <iostream>
#include <vector>

using namespace cv;
using namespace std;

// Function to perform thinning
Mat thinning(const Mat& inputImage, int iterations) {
    Mat thinned = inputImage.clone();
    Mat kernel = getStructuringElement(MORPH_CROSS, Size(3,
3)); // Using MORPH_CROSS for better thinning
    Mat eroded;
    Mat dilated;

    for (int i = 0; i < iterations; i++) {
        erode(thinned, eroded, kernel);
```

```
        /*dilate(eroded, dilated, kernel);*/
        Mat diff;
        subtract(thinned, eroded, diff);
        thinned = diff.clone(); // Update thinned for the
next iteration
    }

    return thinned;
}

int main(int argc, char* argv[]) {
    if (argc != 2) {
        cout << "Usage: Project.exe <ImageFileName>" << endl;
        return -1;
    }

    Mat image = imread(argv[1], IMREAD_UNCHANGED);
    if (image.empty()) {
        cout << "Error: Couldn't open the image file." <<
endl;
        return -1;
    }

    Mat grayImage, binaryImage, horizontalLines,
verticalLines, noLinesImage, clusteredImage, restoredImage,
enhancedImage, thinnedImage;



    // Convert the image to grayscale
    cvtColor(image, grayImage, COLOR_BGR2GRAY);

    // Step 2: Binarization with Adaptive Thresholding
    adaptiveThreshold(grayImage, binaryImage, 255,
ADAPTIVE_THRESH_GAUSSIAN_C, THRESH_BINARY_INV, 15, 10);

    // Step 3: Detect Horizontal and Vertical Lines
    // Detect Horizontal Lines
    Mat horizontalKernel = getStructuringElement(MORPH_RECT,
Size(40, 1)); // Wider kernel for horizontal lines
    morphologyEx(binaryImage, horizontalLines, MORPH_OPEN,
horizontalKernel);
```

```
    // Detect Vertical Lines
    Mat verticalKernel = getStructuringElement(MORPH_RECT,
Size(1, 40)); // Taller kernel for vertical lines
    morphologyEx(binaryImage, verticalLines, MORPH_OPEN,
verticalKernel);

    // Combine Horizontal and Vertical Lines
    Mat allLines = horizontalLines | verticalLines;

    // Step 4: Subtract Detected Lines from Original Binary
Image
    Mat noLinesBinary;
    subtract(binaryImage, allLines, noLinesBinary);

    // Step 5: Apply Gaussian Blur to Smooth Transitions
    GaussianBlur(noLinesBinary, noLinesBinary, Size(3, 3),
0);

    // Step 6: Use Hough Transform for Remaining Gridline
Artifacts
    // Detect edges using Canny
    Mat edges;
    Canny(noLinesBinary, edges, 50, 150, 3);

    // Perform Hough Line Transform
    vector<Vec2f> lines;
    HoughLines(edges, lines, 1, CV_PI / 180, 200); // Adjust
the threshold based on the image

    // Mask the detected lines
    Mat mask = Mat::zeros(grayImage.size(), CV_8U);
    for (size_t i = 0; i < lines.size(); i++) {
        float rho = lines[i][0], theta = lines[i][1];
        Point pt1, pt2;
        double a = cos(theta), b = sin(theta);
        double x0 = a * rho, y0 = b * rho;
        pt1.x = cvRound(x0 + 1000 * (-b));
        pt1.y = cvRound(y0 + 1000 * (a));
        pt2.x = cvRound(x0 - 1000 * (-b));
        pt2.y = cvRound(y0 - 1000 * (a));
        line(mask, pt1, pt2, Scalar(255), 2, LINE_AA);
```

```
    }

    // Invert the mask and apply it to the binary image
    bitwise_not(mask, mask);
    noLinesImage = grayImage.clone();
    noLinesImage.copyTo(noLinesImage, mask);


    // Step 3: K-Means Clustering
    Mat samples = noLinesImage.reshape(1, noLinesImage.rows *
noLinesImage.cols); // Flatten the image
    samples.convertTo(samples, CV_32F);

    int k = 3; // Clustering into background, text, and blur
    Mat labels, centers;
    kmeans(samples, k, labels, TermCriteria(TermCriteria::EPS
+ TermCriteria::MAX_ITER, 100, 0.2), 20,
KMEANS_RANDOM_CENTERS, centers);

    // Reshape labels back to the original image size
    labels = labels.reshape(1, noLinesImage.rows);
    clusteredImage = Mat(noLinesImage.size(), CV_8U);
    for (int i = 0; i < clusteredImage.rows; i++) {
        for (int j = 0; j < clusteredImage.cols; j++) {
            clusteredImage.at<uchar>(i, j) =
static_cast<uchar>(centers.at<float>(labels.at<int>(i, j),
0));
        }
    }

    // Step 4: Identify clusters (background, text, blur)
    vector<pair<int, float>> sortedCenters;
    for (int i = 0; i < centers.rows; i++) {
        sortedCenters.push_back({ i, centers.at<float>(i, 0)
});
    }
    sort(sortedCenters.begin(), sortedCenters.end(), [](const
pair<int, float>& a, const pair<int, float>& b) {
        return a.second < b.second;
        });

    int blurCluster = sortedCenters[1].first;       // Lowest
```

```
intensity
    int textCluster = sortedCenters[0].first;      // Medium
intensity
    int backgroundCluster = sortedCenters[2].first; //
Highest intensity

    // Step 5: Replace blur cluster with background intensity
    restoredImage = noLinesImage.clone();
    for (int i = 0; i < restoredImage.rows; i++) {
        for (int j = 0; j < restoredImage.cols; j++) {
            if (labels.at<int>(i, j) == blurCluster) {
                restoredImage.at<uchar>(i, j) =
static_cast<uchar>(centers.at<float>(backgroundCluster, 0));
            }
        }
    }

    // Step 6: Smooth transitions
    GaussianBlur(restoredImage, restoredImage, Size(3, 3),
0);

    // Step 7: Sharpen the final image
    Mat sharpeningKernel = (Mat_<float>(3, 3) <<
        0, -1, 0,
        -1, 5, -1,
        0, -1, 0);
    Mat sharpenedImage;
    filter2D(restoredImage, sharpenedImage, -1,
sharpeningKernel);
    enhancedImage = sharpenedImage.clone();

    // Step 8: Apply Thinning to the enhanced Image
    thinnedImage = thinning(enhancedImage, 1);
    for (int i = 0; i < enhancedImage.rows; i++) {
        for (int j = 0; j < enhancedImage.cols; j++) {
            if (noLinesImage.at<uchar>(i, j) == 0) {
                enhancedImage.at<uchar>(i, j) = 0; // Keep
only the text regions
            }
        }
    }
```

```
    // Step 9: Display and save the results
    imshow("Original Image", image);
    imshow("Gray Image", grayImage);
    imshow("No Lines Image", noLinesImage);
    imshow("Clustered Image", clusteredImage);
    imshow("Sharpened Image", sharpenedImage);
    imshow("Restored Image", enhancedImage);
    imshow("thinnedImage", thinnedImage);
    imshow("Detected Lines", allLines);
    imwrite("restored_image.jpg", enhancedImage);

    waitKey(0);
    return 0;
}
```

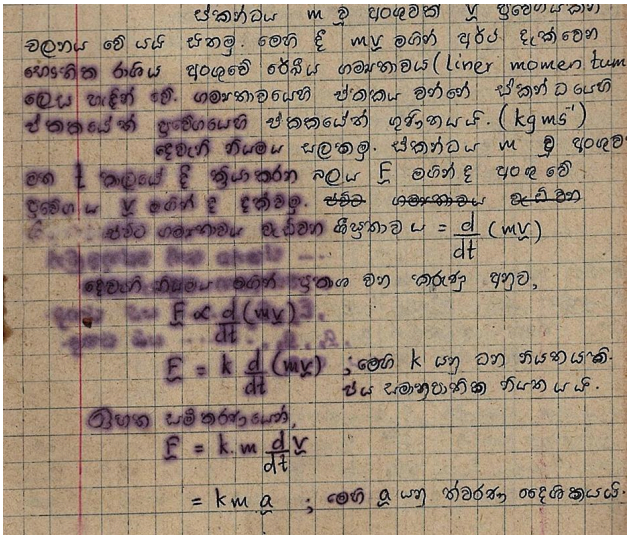


Fig: Input Image

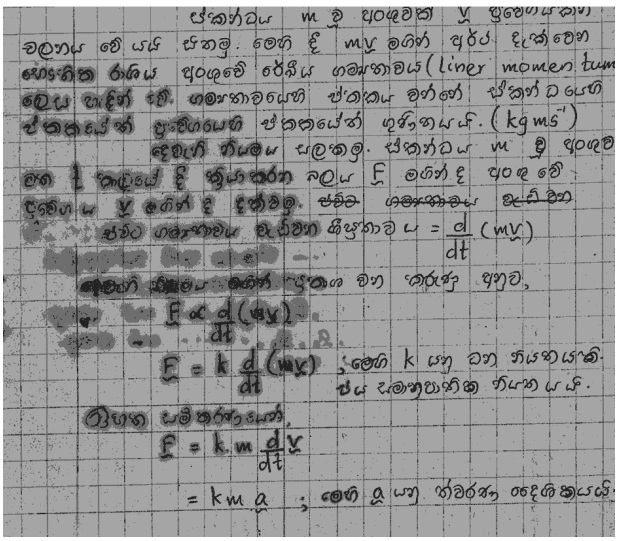


Fig: Clustered Image

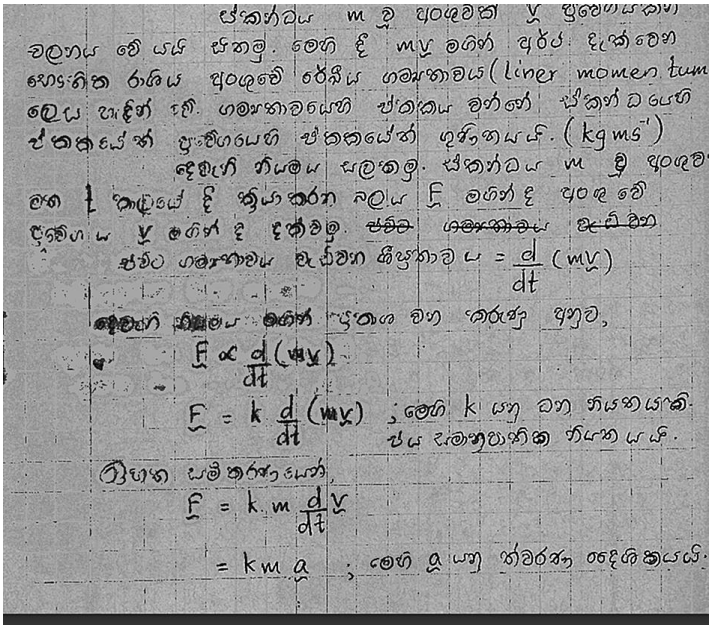


Fig: Restored Image

6.4.1 Reconstruction of Minor Letter-Level Damage

6.4.1.1 Extracting Text, Background, and Damaged Regions

```
def extract_text_and_damage(image_path, k=3):
    """Segment text, background, and damage regions using
```

```
K-Means clustering."""
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    pixels = img.reshape(-1, 1)

    # Apply K-Means clustering
    kmeans = KMeans(n_clusters=k, random_state=0,
n_init=10).fit(pixels)
    clustered = kmeans.labels_.reshape(img.shape)

    # Identify clusters (sorted by intensity)
    cluster_means = [np.mean(pixels[kmeans.labels_ == i]) for
i in range(k)]
    sorted_clusters = np.argsort(cluster_means)

    text_cluster = sorted_clusters[0]  # Darkest (Text)
    damage_cluster = sorted_clusters[1]  # Mid-gray (Damage)
    background_cluster = sorted_clusters[2]  # Brightest
(Background)

    # Create a damage mask
    damage_mask = (clustered ==
damage_cluster).astype(np.uint8) * 255

    return damage_mask
```

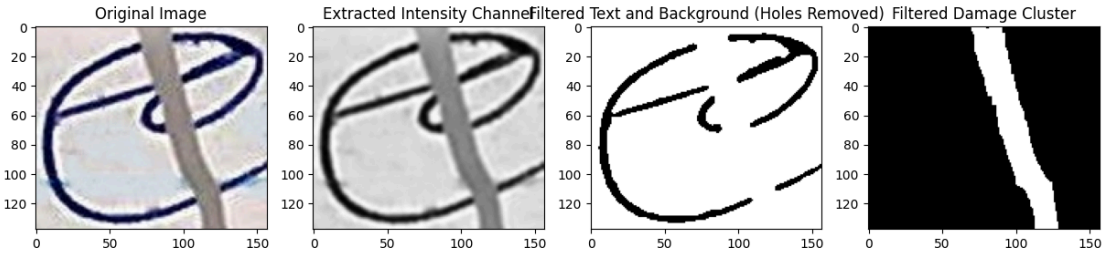


Fig : Output 1

6.4.1.2 Skeletonization and Endpoint Detection

```
def detect_endpoints(skeleton):
    """Detect stroke endpoints in the skeletonized image."""
    endpoints = []
    for y in range(1, skeleton.shape[0] - 1):
```

```
        for x in range(1, skeleton.shape[1] - 1):
            if skeleton[y, x] == 255:
                neighbors = [
                    skeleton[y-1, x-1], skeleton[y-1, x],
skeleton[y-1, x+1],
                    skeleton[y, x-1],
skeleton[y, x+1],
                    skeleton[y+1, x-1], skeleton[y+1, x],
skeleton[y+1, x+1]
                ]
                if np.sum(neighbors) == 255:  # Only one
white neighbor (endpoint)
                    endpoints.append((x, y))
    return endpoints
```

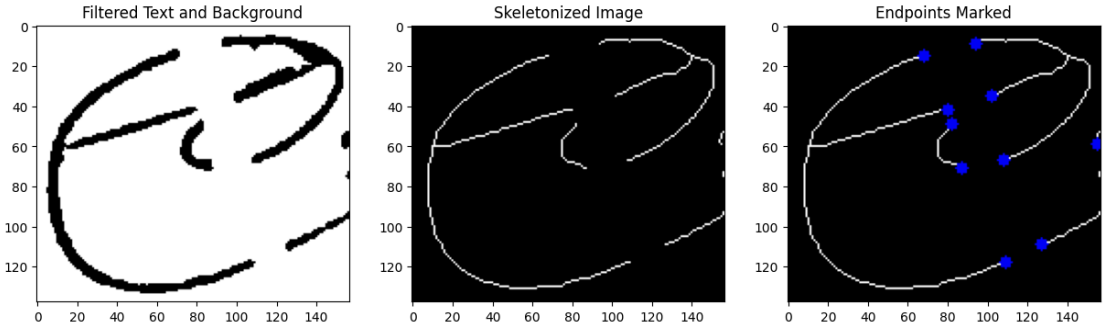


Fig : Output 2

6.4.1.3 Identifying the Angled Damage Divider and Grouping Endpoints

```
def identify_damage_divider(damage_mask):
    """Identify the damage divider by analyzing contours."""
    contours, _ = cv2.findContours(damage_mask,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    if contours:
        largest_contour = max(contours, key=cv2.contourArea)
        x, y, w, h = cv2.boundingRect(largest_contour)
        centroid_x = x + w // 2
        centroid_y = y + h // 2
        damage_orientation = "vertical" if h > w else
"horizontal"
        return centroid_x, centroid_y, damage_orientation
    return None, None, None
```

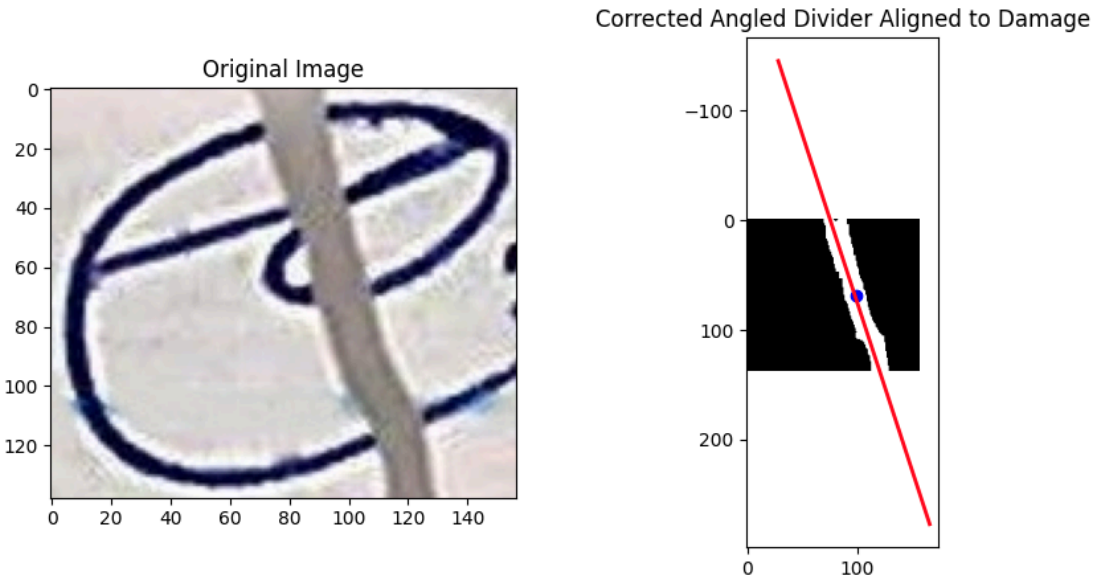


Fig : Output 3

6.4.1.4 Matching Endpoints Across the Angled Damage Divider

```
def match_endpoints(endpoints, centroid_x, centroid_y,
damage_orientation):
    """Pair matching endpoints based on Euclidean
distance."""
    if damage_orientation == "vertical":
        left_side = [pt for pt in endpoints if pt[0] <
centroid_x]
        right_side = [pt for pt in endpoints if pt[0] >=
centroid_x]
        group1, group2 = left_side, right_side
    else:
        top_side = [pt for pt in endpoints if pt[1] <
centroid_y]
        bottom_side = [pt for pt in endpoints if pt[1] >=
centroid_y]
        group1, group2 = top_side, bottom_side

    matched_pairs = []
    for p1 in group1:
        distances = cdist([p1], group2)
        if distances.size > 0:
            min_idx = np.argmin(distances)
            matched_pairs.append((p1, group2[min_idx]))
```

```
    return matched_pairs
```

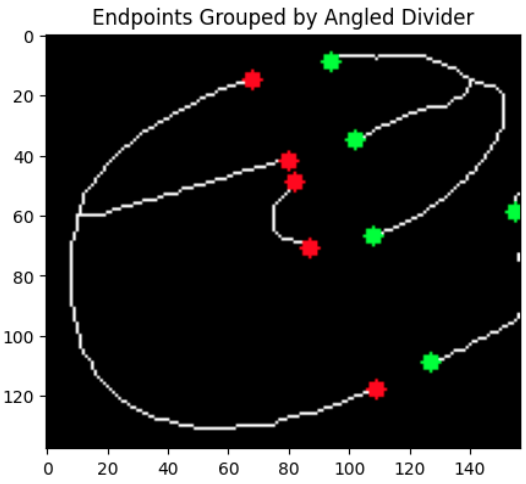


Fig : Output 4

6.4.1.5 Reconstructing Missing Strokes Using Bezier Curves

```
def bezier_curve(p1, p2, control1, control2, num_points=300):
    """Generates a high-resolution cubic Bezier curve."""
    t = np.linspace(0, 1, num_points)
    bezier_x = (1 - t) ** 3 * p1[0] + 3 * (1 - t) ** 2 * t *
control1[0] + 3 * (1 - t) * t ** 2 * control2[0] + t ** 3 *
p2[0]
    bezier_y = (1 - t) ** 3 * p1[1] + 3 * (1 - t) ** 2 * t *
control1[1] + 3 * (1 - t) * t ** 2 * control2[1] + t ** 3 *
p2[1]
    return bezier_x, bezier_y

def fit_smooth_bezier_curve(p1, p2):
    """Generates a smooth Bezier curve with adaptive control
points."""
    control1 = (p1[0] + 10, p1[1])
    control2 = (p2[0] - 10
```

```
def integrate_strokes(image, matched_pairs):
    """Blend reconstructed strokes into the document
image."""
    result_img = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)
    for (p1, p2) in matched_pairs:
        bezier_x, bezier_y = fit_smooth_bezier_curve(p1, p2)
        for i in range(len(bezier_x) - 1):
```

```
            cv2.line(result_img, (int(bezier_x[i]),
int(bezier_y[i])),
                     (int(bezier_x[i+1]),
int(bezier_y[i+1])),
                     (0, 255, 0), 1)  # Draw in green
    return result_img
```

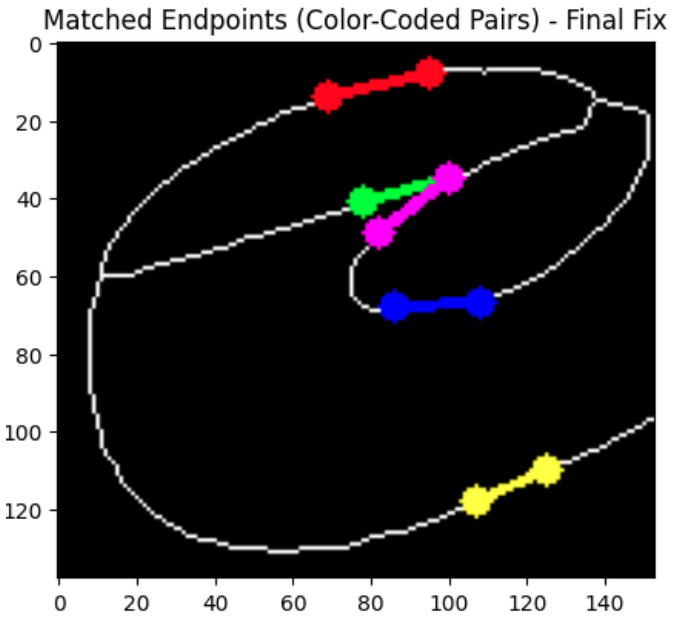


Fig : Output 5

6.4.1.6 Integrating Reconstructed Strokes into the Original Image

```
def integrate_strokes(image, matched_pairs):
    """Blend reconstructed strokes into the document
image."""
    result_img = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)
    for (p1, p2) in matched_pairs:
        bezier_x, bezier_y = fit_smooth_bezier_curve(p1, p2)
        for i in range(len(bezier_x) - 1):
            cv2.line(result_img, (int(bezier_x[i]),
int(bezier_y[i])),
                     (int(bezier_x[i+1]),
int(bezier_y[i+1])),
                     (0, 255, 0), 1)  # Draw in green
    return result_img
```

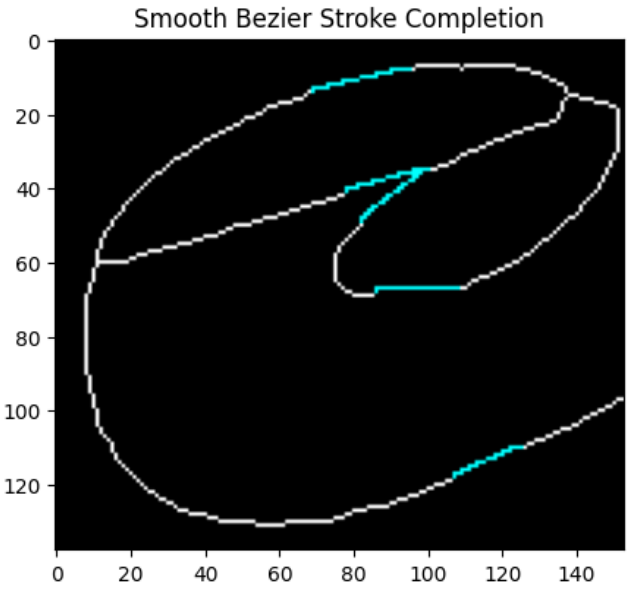

Smooth Bezier Stroke Completion
0
20
40
60
80
100
120
0
20
40
60
80
100
120
140

Fig : Output 6

6.5 Module 4 - Restoring missing letters and words based on surrounding context

As Sinhala is such a low resourced language, it is rather difficult to find a corpus itself to train a model to involve in any NLP related task. As per the review of other's work , some corpus was found but as we decided to go with Sinhala GPt 2 which is a already trained base model for Sinhala, the corpus was not much necessary.This section details how the solution is implemented, including tools and techniques used in training the model in the process of reconstruction of missing Sinhala text.

6.5.1.Input PDF

6.5.1.1. Extract Text from a PDF

```
import PyPDF2
pdf_file = open("AL-Physics-Resource-Book.pdf","rb") # Open
the PDF file
pdf_reader = PyPDF2.PdfReader(pdf_file)
text = ""   # Extract text from each page
for page in pdf_reader.pages:
    text += page.extract_text()
pdf_file.close()
with open("physics_textbook.txt", "w", encoding="utf-8") as
f:   # Save the extracted text to a file
    f.write(text)
```

6.5.1.2.Clean and Preprocess the Text

```
import re
# Clean the text
cleaned_text = re.sub(r"\s+", " ", text)
cleaned_text = re.sub(r"[^\w\s.,!?]", "", cleaned_text)
paragraphs = cleaned_text.split("\n\n")  # Split into
paragraphs or sentences
```

6.5.2. Format the Dataset

Create the dataset in the particular dataset format suitable for training.

```
from datasets import Dataset
data = {"text": paragraphs} # Create a list of examples
dataset = Dataset.from_dict(data)# Convert to Hugging Face Dataset
dataset.save_to_disk("physics_dataset")
```

6.5.3. Fine-Tune the Model

6.5.3.1. Load the Pre-trained Model and Tokenizer

```
from transformers import GPT2LMHeadModel, GPT2Tokenizer

# Load the Sinhala GPT-2 model and tokenizer
model =
GPT2LMHeadModel.from_pretrained("keshan/sinhala-gpt2")
tokenizer =
GPT2Tokenizer.from_pretrained("keshan/sinhala-gpt2")
if tokenizer.pad_token is None:
       tokenizer.pad_token = tokenizer.eos_token
```

6.5.3.2.Tokenize the Dataset

```
def tokenize_function(examples):
    return tokenizer(
        examples["text"],
        padding="max_length",
        truncation=True,
        max_length=512,
        return_tensors="pt",
    )


tokenized_dataset = dataset.map(tokenize_function, batched=True)
```

6.5.3.3.Fine-Tune the Model

```
from transformers import GPT2LMHeadModel, GPT2Tokenizer,
```

```
Trainer, TrainingArguments
from datasets import load_dataset, DatasetDict

# Load the dataset
dataset = load_dataset("wikitext", "wikitext-2-raw-v1")  #
Replace with your dataset

tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
tokenizer.pad_token = tokenizer.eos_token  # Use EOS token as
padding token

def tokenize_function(examples):
    tokenized_inputs = tokenizer(
        examples["text"],
        padding="max_length",
        truncation=True,
        max_length=512,
        return_tensors="pt",
    )

    tokenized_inputs["labels"] =
tokenized_inputs["input_ids"].clone()
    return tokenized_inputs

tokenized_dataset = dataset.map(tokenize_function,
batched=True)


if isinstance(tokenized_dataset, DatasetDict):
    # Assuming the dataset has a single split (e.g., "train")
    dataset = tokenized_dataset["train"]
else:
    dataset = tokenized_dataset

# Split the dataset into train and validation sets
split_dataset = dataset.train_test_split(test_size=0.1)  #
90% train, 10% validation

# Load the GPT-2 model
model = GPT2LMHeadModel.from_pretrained("gpt2")

# Define training arguments
```

```
training_args = TrainingArguments(
    output_dir="./results",
    per_device_train_batch_size=4,
    num_train_epochs=3,
    logging_dir="./logs",
    logging_steps=10,
    save_steps=500,
    save_total_limit=2,
    report_to="none",
)

# Define Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=split_dataset["train"],  # Use the training
split
    eval_dataset=split_dataset["test"],   # Use the
validation split
)

# Start fine-tuning
trainer.train()


trainer.save_model("./fine-tuned-model")
tokenizer.save_pretrained("./fine-tuned-model")
```

6.5 Chapter Summary

This chapter discussed the core of our system of four modules:the implementation of damage detection, categorization of damages, counting of letters and words missing,

ranking handwriting, reconstructing blurred and parts of letters and predicting missing letters and words.

The next chapter focuses on the conclusion of each module and future implementation of them.

Chapter 7 - Conclusion and Further Work

7.1 Chapter Overview

A conclusion as per the current status of work is given and the plans for future work is discussed here for all four modules .

7.2 Module 1 -Detection and Classification of Damaged Areas

The implemented system effectively detects damaged areas in handwritten Sinhala documents using image processing techniques. Initial testing indicates successful noise reduction, segmentation, and bounding box generation. Compared to existing methods, our approach enhances accuracy through HSI-based intensity analysis and mode filtering.

Further work involves developing a CNN model to classify different types of document damage, improving classification accuracy. Future evaluations will focus on model performance metrics such as precision, recall, and computational efficiency to ensure reliability and scalability.

7.3 Module 2 - Evaluating Readability, Classifying Detected Damages, and Estimating the Extent of Damage

This solution stands out from similar works by focusing not only on improving segmentation and OCR accuracy but also on addressing the evaluation of readability and the classification of document damage. Unlike previous studies that primarily focus on clean text or character recognition, our system incorporates both damage detection and content restoration. By combining multiple feature extraction techniques and creating a comprehensive readability score, we handle the complex challenges posed by degraded Sinhala handwriting.

Our approach also differs by utilizing projection-based segmentation to handle overlapping or distorted text, while future improvements could explore deep learning methods like CNNs for more robust segmentation. Additionally, integrating generative models, such as GANs, could enhance the restoration of missing content.

In conclusion, our solution offers a comprehensive approach to Sinhala handwriting analysis, combining segmentation, readability assessment, and damage classification, with potential for further enhancements in deep learning.

7.4 Module 3

Module 3 plays a critical role in the restoration of degraded textual content by addressing both blurred or bleed-through text and minor letter-level damages. The module applies a combination of image enhancement techniques, deep learning-based super-resolution, and stroke refinement to improve text clarity while maintaining the original document structure. The restoration of blurred text is achieved through K-Means clustering for segmentation, SRCNN/ESRGAN for super-resolution, and morphological thinning for stroke consistency, ensuring enhanced readability and improved OCR accuracy.

For letter-level damage reconstruction, the module employs skeletonization, endpoint detection, and Bezier curve interpolation to seamlessly restore missing strokes while preserving the handwriting style. By integrating adaptive smoothing techniques and stroke width normalization, the reconstructed strokes blend naturally into the document. Additionally, the iterative feedback mechanism with Modules 1 and 4 allows for further refinement, ensuring high accuracy in text restoration.

Overall, Module 3 effectively enhances the readability and structural integrity of degraded documents by reconstructing missing or distorted text while preserving their original appearance. Its implementation ensures a robust and efficient restoration process, contributing to the overall accuracy and reliability of the document reconstruction framework.

7.5 Module 4 -Restoring missing letters and words based on surrounding context

The goal of this project is to apply refined language models, such as GPT-2, to recover damaged Sinhala handwritten text documents. The system successfully predicts and reconstructs missing or damaged text by modifying pre-trained models to the Sinhala language and training them on carefully selected datasets. Successful fine-tuning, dataset preparation, and enhanced model performance—shown by a reduction in training loss and the production of coherent text—are among the major accomplishments. Limited dataset availability, handwriting diversity, and processing resource requirements are among the difficulties. Future research will focus on building user-friendly interfaces, enhancing model structures, and growing datasets. Future generations could benefit greatly from the digitization of old records and the preservation of Sinhala cultural heritage made possible by this research.

7.6 Chapter Summary

The research on the reconstruction of damaged Sinhala handwritten text documents using digital image processing and computer vision techniques has demonstrated significant potential in preserving and restoring valuable cultural and historical manuscripts. By combining advanced image processing methods with state-of-the-art language models, the system effectively detects damaged regions in handwritten documents, extracts text, and reconstructs missing. Key achievements include the successful application of techniques such as image segmentation, optical character recognition (OCR), and masked language modeling to restore text with high accuracy. Challenges such as variability in handwriting styles, document degradation, and computational resource requirements were addressed through innovative solutions and iterative improvements.

Chapter 9 - References

[1] H. G. Moreno, “DIGITAL IMAGE PROCESSING TECHNIQUES APPLIED TO THE RECOVERING AND CONSERVATION OF THE GRAPHICAL HERITAGE”.

[2] R. Easton, “Digital Restoration of Erased and Damaged Manuscripts,” 2004.

[3] K. Knox, R. Easton, and W. Christens-Barry, “Image restoration of damaged or erased manuscripts,” 2008.

[4] P. S. Giri, “Text Information Extraction And Analysis From Images Using Digital Image Processing Techniques,” vol. 2, no. 1.

[5] K. Patil, M. Kulkarni, A. Sriraman, and S. Karande, “Deep Learning Based Car Damage Classification,” in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Cancun, Mexico: IEEE, Dec. 2017, pp. 50–54. doi: 10.1109/ICMLA.2017.0-179.

[6] C. Silva and C. Kariyawasam, “Segmenting Sinhala Handwritten Characters,” vol. 2, no. 4.

[7] K. S. A. Walawage and L. Ranathunga, “Segmentation of Overlapping and Touching Sinhala Handwritten Characters,” in *2018 3rd International Conference on Information Technology Research (ICITR)*, Moratuwa, Sri Lanka: IEEE, Dec. 2018, pp. 1–6. doi: 10.1109/ICITR.2018.8736129.

[8] H. A. Al Hamad, L. Abualigah, M. Shehab, K. H. A. Al-Shqeerat, and M. Otair, “Improved linear density technique for segmentation in Arabic handwritten text recognition,” *Multimed. Tools Appl.*, vol. 81, no. 20, pp. 28531–28558, Aug. 2022, doi: 10.1007/s11042-022-12717-2.

[9] H. A. Alhamad *et al.*, “Handwritten Recognition Techniques: A Comprehensive Review,” *Symmetry*, vol. 16, no. 6, p. 681, Jun. 2024, doi: 10.3390/sym16060681.

[10] S. Prasad, V. K. Singh, and A. Sapre, “Handwriting Analysis based on Segmentation Method for Prediction of Human Personality using Support Vector Machine,” *Int. J. Comput. Appl.*, vol. 8, no. 12, pp. 25–29, Oct. 2010, doi: 10.5120/1256-1758.

[11] Asith Ishantha, “Sinhala Handwriting Character Recognition based on Feature Extraction,” 2021, *Unpublished*. doi: 10.13140/RG.2.2.16292.40327.

[12] C. M. Silva, N. D. Jayasundere, and C. Kariyawasam, “State of Handwriting Recognition of modern Sinhala Script,” vol. 1, no. 9, 2014.

[13] O. V. R. Murthy, “A Study on the Effect of Outliers in Devanagari Character Recognition,” *Int. J. Comput. Appl.*, vol. 31.

[14] A. P. Singh and A. K. Kushwaha, “Analysis of Segmentation Methods for Brahmi Script,” *DESIDOC J. Libr. Inf. Technol.*, vol. 39, no. 2, pp. 109–116, Mar. 2019, doi: 10.14429/djlit.39.2.13615.

[15] H. A. A. Hamad and M. Shehab, “Improving the Segmentation of Arabic Handwriting Using Ligature Detection Technique,” *Comput. Mater. Contin.*, vol. 79, no. 2, pp. 2015–2034, 2024, doi: 10.32604/cmc.2024.048527.

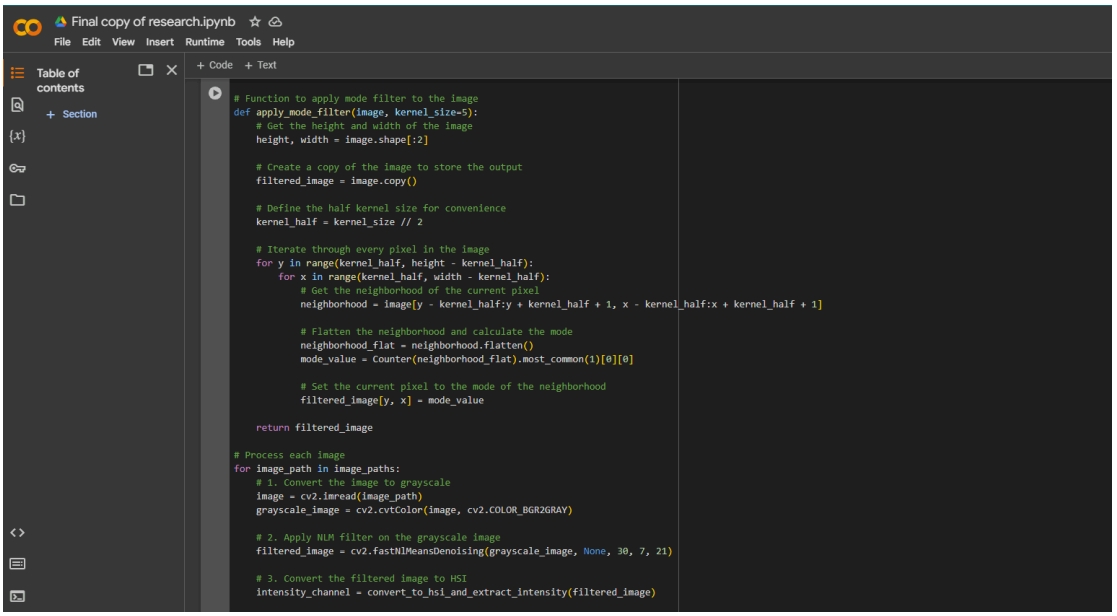
[16] P. Bannigidad and C. Gudada, “Restoration of Degraded Historical Kannada Handwritten Document Images Using Image Enhancement Techniques,” in *Proceedings of the Eighth International Conference on Soft Computing and Pattern Recognition (SoCPaR 2016)*, vol. 614, A. Abraham, A. K. Cherukuri, A.

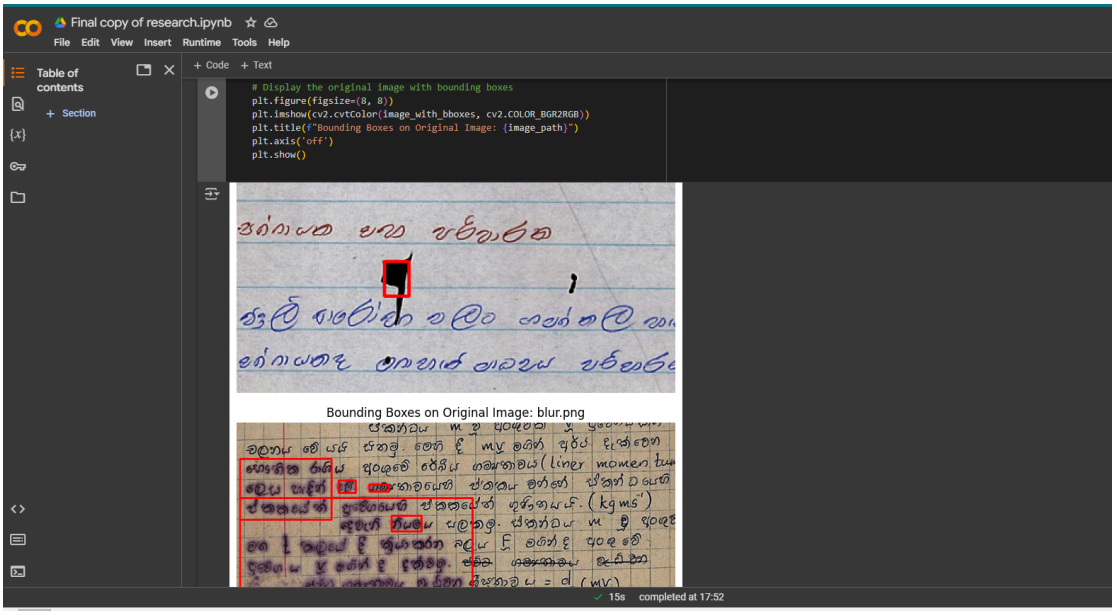
M. Madureira, and A. K. Muda, Eds., in Advances in Intelligent Systems and Computing, vol. 614. , Cham: Springer International Publishing, 2018, pp. 498–508. doi: 10.1007/978-3-319-60618-7_49.
[17] “An improved sequential edge linking model for contour detection in medical images | Request PDF,” in *ResearchGate*, doi: 10.1109/ICIEA.2009.5138906.
[18] F. Y. Shih and S. Cheng, “Adaptive mathematical morphology for edge linking,” *Inf. Sci.*, vol. 167, no. 1, pp. 9–21, Dec. 2004, doi: 10.1016/j.ins.2003.07.020.
[19] C. Akinlar and E. Chome, “PEL: A Predictive Edge Linking algorithm,” *J. Vis. Commun. Image Represent.*, vol. 36, pp. 159–171, Apr. 2016, doi: 10.1016/j.jvcir.2016.01.017.
[20] P. M. Kamble, R. S. Hegadi, and R. S. Hegadi, “Distance Based Edge Linking (DEL) for Character Recognition,” in *Recent Trends in Image Processing and Pattern Recognition*, vol. 1037, K. C. Santosh and R. S. Hegadi, Eds., in Communications in Computer and Information Science, vol. 1037. , Singapore: Springer Singapore, 2019, pp. 261–268. doi: 10.1007/978-981-13-9187-3_23.
[21] H. K. I. L. Madhuwanthi and L. Ranathunga, “Reconstruct Damaged Sinhala Handwritten Characters by Guided Edge Linking Approach,” *2023 3rd Int. Conf. Adv. Res. Comput. ICARC*, pp. 96–101, Feb. 2023, doi: 10.1109/ICARC57651.2023.10145633.
[22] C. Dong, C. C. Loy, and X. Tang, “Accelerating the Super-Resolution Convolutional Neural Network,” Aug. 01, 2016, *arXiv*: arXiv:1608.00367. doi: 10.48550/arXiv.1608.00367.
[23] Kaushick Parui and A. Minj, “Image Restoration and de-blurring using Inverse and Wiener filtering,” 2018, doi: 10.13140/RG.2.2.25372.10880.
[24] Ravindu Jayakody, “Performance of Recent Large Language Models for a Low-Resourced Language,”
[25] Bowen Tan1, “Progressive Generation of Long Text with Pretrained Language Models,”
[26] Hansi Hettiarachchi1, “NSina: A News Corpus for Sinhala,”

Appendix

Module 1 - Detection and Classification of Damaged Areas

Implementation of Damage Detection in Google Colab





```python
from transformers import pipeline


generator = pipeline('text-generation', model='keshan/sinhala-gpt2')


output = generator(
    "අපි අද ආවේ",  # Input prompt
    max_length=50,  # Maximum length of generated text
    num_return_sequences=5,  # Number of sequences to generate
    truncation=True,
)

for i, result in enumerate(output):
    print(f"Generated Text {i+1}: {result['generated_text']}")
```

```
Device set to use cpu
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Generated Text 1: අපි අද ආවේ දිනුකගේ කතාව බලාගෙන.. වෙනදා වගේම කතාව රහයි පින්තූර කතාවටත් විචාරක මහත්
Generated Text 2: අපි අද ආවේ අපේ දරුවන් වෙනුවෙන් කෙරෙන අනුස්මරණ දිනයක් තියන්න.
අපි ආවේ ලේ දන්දෙ
Generated Text 3: අපි අද ආවේ අපේ අනිනය සොයා යන්න...හැමදාම තුනී පුදකරලා යනවා අපේ රට ගැන පැහැදිලි දැක්
Generated Text 4: අපි අද ආවේ පාප් වහන්සේගෙන් ඉල්ලීමක් කරන්න. ඒ මාවෙන මගේ හදවතේ දොරගුළු විවර වන්නට අවශ
Generated Text 5: අපි අද ආවේ චාර්ලි වගේ දක්ෂ නලුවෙක් ගැන බලන්න.චාර්ලි මැරුණා කියල නමයි කිව්
```

Fig - Output of the base model of Sinhala GPT-2 model

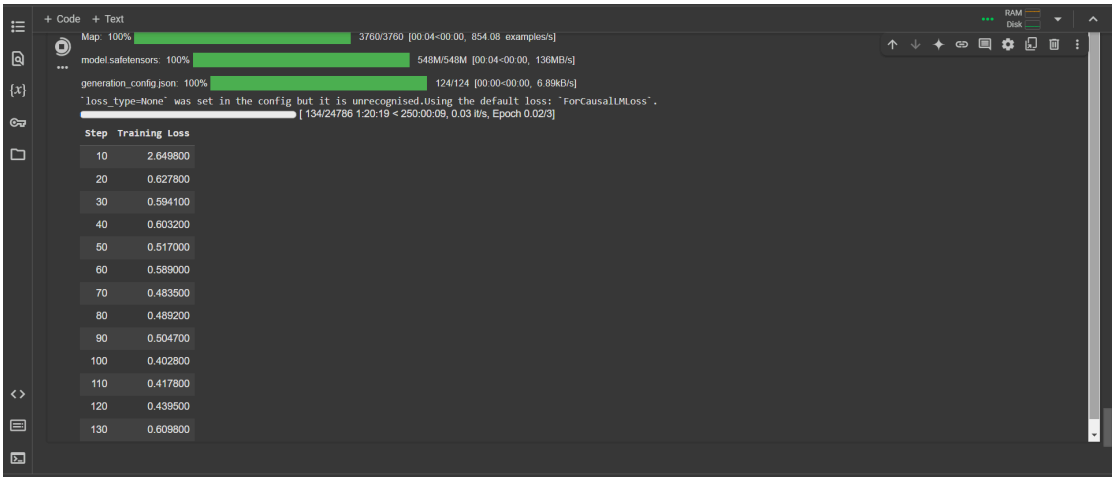


Fig - Showing the process of fine tuning the model