IMAGE BINARIZATION USING DEEP LEARNING TECHNIQUE CONVOLUTION NEURAL NETWORK

A MINI-PROJECT REPORT SUBMITTED IN PARTIAL FULFILLMENT OF REQUIREMENTS TO RGUKT-SRIKAKULAM FOR THE AWARD OF THE DEGREE OF

BACHELOR OF TECHNOLOGY (AY: 2020-2024)

In

Computer Science and Engineering

SUBMITTED BY

Anusha Renangi	S190245
Bhavani Pisini	S191132
Venkata Kalyani Kanaka	S190238
Uma Reddy K	S190712

SUBMITTED TO

Department of Computer Science and Engineering

RGUKT – SRIKAKULAM, ETCHERLA

April 2024

Department of Computer Science Engineering

Rajiv Gandhi University of Knowledge Technologies

Srikakulam, Etcherla

CERTIFICATE

This is to certify that the mini project report titled "**IMAGE BINARIZATION USING DEEP LEARNING TECHNIQUE CONVOLUTION NEURAL NETWORK**" was successfully completed by **ANUSHA RENANGI (S190245) , BHAVANI PISINI (S191132), VENKATA KALYANI KANAKA (S190238) , UMA REDDY K (S190712)** under the guidance of **MR.si** In partial fulfilment of the requirements for the Mini Project in Computer Science and Engineering of Rajiv Gandhi University of Knowledge Technologies under my guidance and output of the work carried out is satisfactory.

Project Guide	**Head of the department**
Siva Rama Sastry	**Lakshmi Bala**
Assistant professor	**Head of the department**
Department of CSE	**Department of CSE**
RGUKT, SRIKAKULAM	**RGUKT, SRIKAKULAM**

BONAFIDE CERTIFICATE

Certified that this project work titled "**IMAGE BINARIZATION USING DEEP LEARNING TECHNIQUE CONVOLUTION NEURAL NETWORK"** is the bonafide work of **ANUSHA RENANGI (S190245) , BHAVANI PISINI (S191132), VENKATA KALYANI KANAKA (S190238) , UMA REDDY K (S190712)** who carried out the work under my supervision, and submitted in partial fulfilment of the requirements for the award of the degree, Bachelor Of Technology, during the year 2023 - 2024.

Project Guide	**Head of the department**
Siva Rama Sastry	**Lakshmi Bala**
Assistant professor	**Head of the department**
Department of CSE	**Department of CSE**
RGUKT, SRIKAKULAM	**RGUKT, SRIKAKULAM**

ACKNOWLEDGMNT

We would like to articulate my profound gratitude and indebted ness to our project guide **Mr. Siva Rama Sastry,** who has always been a constant motivation and guiding factor throughout the project time. It has been a great pleasure for us to get an opportunity to work under his guidance and complete the thesis work successfully.

We wish to extend our sincere thanks **Ass.Prof.Lakshmi Bala** Head of the Computer Science and Engineering Department, for his constant encouragement throughout the project.

We are also grateful to other members of the department without their support our work would have not been carried out so successfully.

I thank one and all who have rendered help to me directly or indirectly in the completion of my thesis work.

Project Associate

ANUSHA RENANGI - S190245

BHAVANI PISINI - S191132

UMA REDDY K - S190712

VENKATA KALYANI KANAKA - S190238

ABSTRACT

Retrieving insightful cultural and scientific information often relies on historical documents as a primary source. Telugu Language is one of the prominent languages in south India, where more than 80 million people speak worldwide. During the olden days, people stored information and wrote on palm leaves. Due to their delicate nature and varying conditions, these palm leaves are not appropriately preserved. In this work, we introduce the first-ever free and open-source Telugu palm leaf dataset to advance digitization research and researchers to work on the ancient Telugu language. This work would be a starting point in Optical Character Recognition in ancient Telugu scripts.

Our dataset comprises 36 images of Telugu palm leaf manuscripts, which are binarized using the Althetia software. We investigated various cutting-edge binarization algorithms, such as Otsu, Niblack, and Sauvola, while creating ground truth images to check the performance of these algorithms on the dataset. Moreover, we explored the effectiveness of deep learning methodologies such as U-Net, LinkNet, and PSPNet in improving binarization quality. Our efforts not only aid in the preservation of cultural heritage but also propel forward the advance of image processing and machine learning techniques utilized in the analysis of historical documents.

TABLE OF CONTENTS

Chapter-1

INTRODUCTION

1.1. Introduction

Historical documents are more valuable as they connect the past with the present. Earlier, information was recorded on palm leaves, which is particularly prevalent in South Asian countries. These documents played a crucial role in preserving knowledge. Sundanese and Balinese historical documents have been subject to extensive research. Indic scripts contain valuable cultural heritage information, including pictures, tables, and decorative elements, often arranged in non-standard layouts. Notably, Indic manuscripts feature holes in the middle for attaching other sheets.

Telugu is the primary language in Andhra Pradesh and Telangana, and it is spoken by approximately 80 million people worldwide. It belongs to the Dravidian language family and has a unique history and roots. Telugu boasts an unbroken literary tradition of over a thousand years. The language consists of 16 vowels and 36 consonants, along with 16 matras and 36 gunithams, with each gunitham comprising 16 akharas.

Before the advent of paper, Palm leaf manuscripts were one of the oldest writing mediums, especially in Southern India. Palm-leaf manuscripts play a significant role in sustaining and transmitting traditional knowledge, cultural heritage, and literature in the Telugu language. As time passes, palm leaf manuscripts often experience degradation in the form of noise, blemishes, faint ink strokes lacking contrast and bleed through. So, it is a challenging task to understand semantic meanings. In the document analysis, primary preprocessing stages include document enhancement and the subsequent binarization process.

The objective of document binarization is to transform the provided grayscale or colored document into a binary representation, differentiate between foreground and background elements. In binarization, enhancing the contrast between the foreground and its background facilitates the readability and comprehension of Palm leaf content by eliminating background noise and degradation.

Removing the noise from palm leaves helps researchers perceive the content in a more readable and accessible format. We were greatly inspired by the work done on Sundanese scripts and applied the same strategy to scan the original palm leaf images to create ground truth images. In this paper, we will see the process of creation of ground truth images, various traditional ML algorithms existing in literature with their results, deep learning strategies that were useful in binarization.

1.2 Motivation

The motivation for developing an image binarization solution for ancient Telugu palm leaf manuscripts is to preserve and make accessible these invaluable cultural artifacts. These manuscripts, stored on delicate palm leaves, face significant preservation challenges due to their age and environmental conditions. Manual efforts are often labor-intensive and insufficient for long-term preservation.

To address this, the project introduces the first-ever free and open-source dataset of 36 Telugu palm leaf manuscript images, facilitating research in digitization and preservation. Advanced binarization techniques like Otsu, Niblack, and Sauvola are used to convert grayscale images to binary, enhancing legibility and preparing them for Optical Character Recognition (OCR) systems. Additionally, modern deep learning models such as U-Net, LinkNet, and PSPNet are employed to further refine the binarization process, improving the clarity and quality of the images.

This initiative not only aids in cultural preservation but also advances image processing and machine learning techniques in historical document analysis, promoting broader accessibility and understanding of Telugu heritage for researchers, historians, and linguists worldwide.

.

1.3 Problem Statement

The aim of this project is to address the challenges in the preservation and digital processing of ancient Telugu palm leaf manuscripts, which include significant variations in writing styles and the presence of unique characters no longer used in modern Telugu. These challenges complicate the OCR process, particularly in recognition and post-processing phases. The project explores traditional and deep learning-based binarization techniques to enhance the quality and readability of these manuscripts, aiding in their digital preservation and analysis.

1.4 Objectives

1. **Enhance Readability of Historical Manuscripts:** Image binarization aims to improve the readability of ancient Telugu manuscripts by converting grayscale or colored images into a binary format. This process clearly distinguishes the text from the background, significantly enhancing clarity and legibility. Improved readability is essential for accurate transcription and preservation, making these historical texts accessible and understandable for scholars, historians, and researchers**.**

2. **Remove Background Noise and Degradation:** The objective is to eliminate background noise and degradation in images of palm leaf manuscripts. These documents often suffer from stains, smudges, and fading, which obscure the text. Image binarization techniques are used to remove these imperfections, restoring

visual integrity and preserving the authenticity and readability of the manuscripts for accurate analysis and interpretation.

3. **Facilitate Text Line and Character Segmentation**: The binarization process aims to aid in accurately identifying and separating text lines and characters from the background. Clear binary images are crucial for efficient text line and character segmentation, enhancing OCR system performance and enabling precise extraction of textual information from manuscripts.

4. **Preserve Historical Knowledge**: Binarization techniques help preserve traditional knowledge in ancient manuscripts by converting them into a digital binary format. This process protects the documents from physical deterioration and loss, ensuring they are permanently accessible for future generations. Digital preservation is essential for maintaining cultural heritage and allowing future researchers to study and learn from these historical documents.

5. **Create Ground Truth Datasets for Machine Learning**: Image binarization aims to create accurate ground truth datasets for training machine learning models. These manually annotated binary images serve as reference standards, improving algorithm performance. High-quality datasets enable better model training, automating the binarization process and enhancing document analysis efficiency and accuracy.

6. **Support Digital Humanities Research**: Image binarization supports digital humanities research by enabling the digitization and analysis of historical manuscripts. Digital humanities researchers rely on high-quality digital representations of historical texts for their studies. Binarized images provide a clear and accurate representation of the manuscripts, allowing researchers to perform various computational analyses, such as text mining, pattern recognition, and linguistic analysis. By facilitating these analyses, binarization contributes to the advancement of knowledge in the humanities and the discovery of new insights from historical documents.

1.5 Goal

The primary goal of the image binarization document is to develop and implement effective techniques to convert historical Telugu manuscript images into a clear binary format, enhancing readability, removing background noise, and preserving the content for future research and analysis. This process aims to facilitate accurate text segmentation, support digital preservation efforts, and create high-quality datasets for training machine learning models in document analysis.

1.6 Scope

The scope for the paper on image binarization techniques for preserving Telugu manuscripts is extensive and impactful. This project focuses on using advanced binarization techniques to preserve and enhance the readability of ancient Telugu manuscripts, crucial for their cultural preservation.

It explores algorithms like Otsu, Niblack, Sauvola, and deep learning models such as U-Net, LinkNet, and PSPNet. The project aims to create accessible datasets for OCR research, advance image processing methods, and foster collaboration in document analysis and computational linguistics.

Overall, the scope of this paper encompasses the preservation of cultural heritage, the development of innovative binarization techniques, the creation of valuable datasets, and the advancement of technology and research, Ultimately, it aims to make these manuscripts more accessible to researchers, historians, and the public, preserving and promoting Telugu cultural heritage.

1.7 Applications

The enhanced image binarization system for ancient Telugu palm leaf manuscripts has a wide range of applications, contributing significantly to the fields of cultural preservation, research, education, and technology.

1. **Cultural Preservation**: Binarized images of ancient manuscripts can be stored in digital archives, ensuring their preservation for future generations. This helps protect cultural heritage from physical degradation over time.High-quality binarized images can assist in the restoration and reconstruction of damaged manuscripts, providing a clearer basis for conservation efforts.

2. **Historical Research:** Researchers and historians can access binarized images to study ancient texts, facilitating the analysis of historical documents and the understanding of past cultures and languages.Linguistic Studies: Linguists can use the binarized texts to analyze the evolution of the Telugu language, studying changes in script and usage over centuries.
3. **Technology Development:** The improved quality of binarized images significantly enhances OCR performance, aiding in the digitization and searchable text extraction from ancient manuscripts.

4. **Digital Exhibits:** Libraries and museums can create digital exhibits featuring binarized images of ancient manuscripts, allowing a wider audience to explore these cultural artifacts online.
5. **Cataloging and Indexing:** Improved binarization aids in the cataloging and indexing of manuscript collections, making it easier for institutions to manage and provide access to their holdings.

6. **Public Access:** Binarized manuscripts can be made available in online databases, providing public access to historical documents that were previously difficult to read or physically inaccessible.Clearer binarized images can be used in crowdsourced transcription projects, where volunteers help transcribe and digitize ancient texts.

By enhancing the readability and preservation of ancient Telugu palm leaf manuscripts, the proposed image binarization system opens up numerous applications that benefit cultural heritage, academic research, education, technology development, and public access.

1.8 Limitations

1. **Variability in Image Quality**: Image binarization performance can be significantly affected by the quality of the input images. Factors such as low resolution, high noise levels, uneven lighting, or poor contrast can degrade the effectiveness of the binarization process, leading to suboptimal results.

2. **Sensitivity to Parameters:** Many image binarization techniques require careful tuning of parameters to achieve optimal performance. These parameters can vary widely depending on the specific characteristics of the images being processed, which can make the binarization process less robust and more difficult to generalize.

3. **Computational Complexity:** Some advanced binarization algorithms, especially those involving deep learning or complex image processing techniques, can be computationally intensive. This can result in longer processing times and higher resource consumption, which may not be feasible for real-time applications or devices with limited computational power.

4. **Environmental Conditions:** The performance of the binarization algorithm can be influenced by varying environmental conditions, such as changes in lighting or background variations. Consistency in image capture conditions is crucial for achieving reliable binarization results.

Chapter – 2

LITERATURE SURVEY

2.1 Paper 1:

Title: Document Enhancement and Binarization using Iterative Deep Learning

Authors: Sheng He, Lambert Schomaker

Focus: Image binarization for Convolutional Neural Networks (CNNs), Iterative Deep Learning, Recurrent Refinement

Methodology:

- The paper proposes an iterative deep learning approach named DeepOtsu.
- The technique involves a recurrent refinement process that enhances and binarizes documents through multiple iterations.

- The approach is particularly designed to improve the performance of CNNs by providing cleaner, binarized images for better feature extraction and classification.

Results: The iterative approach demonstrated significant improvements in document binarization tasks, leading to enhanced image quality and better performance of subsequent image processing and analysis tasks.

Institution: Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, PO Box 407, 9700 AK, Groningen, The Netherlands.

2.2 Paper 2:

Title: U-Net: Convolutional Networks for Biomedical Image Segmentation

Authors: O. Ronneberger, P. Fischer, T. Brox

Focus: Biomedical image segmentation using Convolutional Neural Networks (CNNs)

Methodology:

- The paper introduces the U-Net architecture, a type of convolutional neural network specifically designed for biomedical image segmentation.
- U-Net consists of a contracting path to capture context and a symmetric expanding path for precise localization.
- The network employs data augmentation to effectively utilize the limited available annotated images.

Results:

- U-Net achieved state-of-the-art performance on various biomedical segmentation tasks.
- The architecture demonstrated superior accuracy and efficiency, making it highly suitable for medical image analysis.

2.3 Paper 3:

Title: A Threshold Selection Method from Gray-Level Histograms

Author: N. Otsu

Focus: Image binarization using a threshold selection method based on gray-level histograms

Methodology:

- The paper proposes Otsu's method, an automatic thresholding technique.
- This method calculates the optimal threshold value by maximizing the between-class variance in the gray-level histogram of the image.
- The approach is non-parametric and unsupervised, making it widely applicable across different image types and conditions.

Results:

- Otsu's method is highly effective in separating foreground from background in an image.
- The technique has become a standard benchmark for image binarization and is extensively used in various image processing applications.

2.4 Summary

These surveys illustrate the advancements in image binarization techniques, from classical methods like Otsu's thresholding to modern deep learning approaches such as DeepOtsu and U-Net. Otsu's method provides a reliable, unsupervised technique for image thresholding. DeepOtsu enhances document binarization through iterative deep learning, demonstrating neural networks' capabilities in complex tasks.

U-Net excels in biomedical image segmentation with high accuracy and efficiency. These techniques collectively enhance image binarization, facilitating improved performance in various image analysis applications.

Chapter 3

SYSTEM ANALYSIS

3.1 Existing System

The existing systems mentioned in the literature survey primarily focused on traditional and deep learning approaches for the binarization of ancient manuscripts, including techniques like global and local thresholding methods, as well as advanced neural network models. These methods involve preprocessing images, applying various algorithms and models, and achieving varying levels of success in enhancing the readability and quality of historical documents.

3.1.1 Disadvantages

1. **Limited Adaptability:** Traditional binarization methods like Otsu, Niblack, and Sauvola often struggle to adapt to the diverse and degraded conditions of ancient manuscripts. These methods may not perform consistently across different types of degradation and writing styles present in the manuscripts.

2. **Performance Variability** : Different binarization techniques, including deep learning models like U-Net, LinkNet, and PSPNet, show varying levels of effectiveness.Some models may excel in certain conditions while underperforming in others, leading to inconsistent results.

3. **High Computational Requirements**: Deep learning-based binarization methods require substantial computational power and resources.Training and deploying these models can be time-consuming and resource-intensive, making them less accessible for widespread use.

4. **Manual Pre-processing**: Pre-processing of manuscript images often requires manual effort to remove noise, correct distortions, and enhance contrast. This manual intervention can be labor-intensive, prone to human error, and may not scale well for large collections of manuscripts.
5. **Limited Dataset Availability**: The availability of comprehensive and high-quality datasets for training and evaluating binarization models is limited. This constraint can hinder the development and benchmarking of new techniques, impacting their effectiveness and reliability.

6. **Lower Accuracy Rates**: Despite advancements, existing binarization systems may still produce suboptimal results in terms of accuracy and readability enhancement. There is a need for further improvement to achieve higher precision in character recognition and image quality.

3.2 Proposed System

The proposed system aims to develop an enhanced image binarization solution for ancient Telugu palm leaf manuscripts using a combination of traditional methods and advanced deep learning techniques. The system utilizes a carefully curated dataset of high-resolution images of palm leaf manuscripts, incorporating both global and local thresholding methods alongside state-of-the-art ne ural network models like U-Net, LinkNet, and PSPNet.

3.2.1 Advantages

1. **High Accuracy**: By leveraging advanced deep learning models, the proposed system aims to achieve superior accuracy in binarization compared to traditional methods. This enhances the readability and quality of digitized manuscripts, facilitating better OCR performance and preservation.

2. **Comprehensive Dataset:** The project utilizes a dataset consisting of high-resolution images of Telugu palm leaf manuscripts, encompassing various levels of degradation and writing styles. This comprehensive dataset ensures the robustness and generalizability of the proposed system across different types of manuscripts.

3. **Automatic Pre-processing:** The proposed system includes automated pre-processing techniques to remove noise, correct distortions, and enhance contrast.This reduces the

manual effort required and minimizes human error, making the process more efficient and scalable.

4. **Versatile Binarization Techniques:** The integration of both traditional methods (Otsu, Niblack, Sauvola) and deep learning models (U-Net, LinkNet, PSPNet) provides a versatile and adaptive approach to binarization. This allows the system to perform well under varying conditions and types of manuscript degradation.

5. **Improved Readability and Preservation:** By enhancing the quality of binarized images, the proposed system aids in the preservation of ancient manuscripts, making them more accessible for research and cultural heritage purposes. Improved readability ensures that these historical documents can be better understood and studied by scholars and the general public.

6. **Enhanced OCR Performance:** The high-quality binarized images produced by the proposed system significantly improve the performance of OCR technologies.This leads to more accurate text extraction, aiding in the digitization and analysis of ancient manuscripts.

7. **Scalability and Efficiency:**The automated and robust nature of the proposed system makes it scalable for large collections of manuscripts. Efficient processing ensures that a large number of documents can be binarized and preserved in a timely manner.

3.3 System Requirements

3.3.1 Software Requirements

- Python
- TensorFlow
- OpenCV
- Matplotlib
- Aletheia

3.3.2 Hardware Requirements

- Processor with GPU
- RAM with 16GB minimum
- Storage
- High-Resolution Scanner
- Internet Connectivity

Chapter 4

ALGORITHMS

4.1 Traditional ML Algorithms for Binarization

In image binarization, we convert an image into a binary format by assigning each pixel to either foreground (text or object of interest) or background. There are two main types of binarization techniques: global thresholding and local (adaptive) thresholding.

Global Thresholding:

- **Description:** Global thresholding uses a single threshold value for the entire image. This method is straightforward and effective when the background lighting is uniform. One commonly used global thresholding method is Otsu's method.

Local (Adaptive) Thresholding:

- **Description:** Local thresholding calculates thresholds for small groups of neighboring pixels, adapting to local variations in the image. This approach is beneficial for images with varying lighting conditions and complex backgrounds. Examples include Niblack, Sauvola, and Su methods**.**

4.2 Algorithms and Their Descriptions

1. **Otsu Thresholding:**

   Type: Global Thresholding

   Description: Divides the image into foreground and background by evaluating all possible thresholds. Selects the threshold that maximizes the between-class variance. Widely used for its simplicity and effectiveness.

2. **Niblack Method:**

   Type: Local (Adaptive) Thresholding

   Description: Calculates the threshold for each pixel based on the mean and standard deviation of its neighborhood. Adapts to local variations in the image. Suitable for documents with varying lighting conditions.

3. **Sauvola Method:**

   Type: Local (Adaptive) Thresholding

   Description: An enhancement of Niblack's method, designed for complex textures and intricate foregrounds. Considers local mean and standard deviation for threshold calculation. Effective for detailed document analysis.

4. **Su Method:**

   Type: Local (Adaptive) Thresholding

   Description: Uses a local extrema analysis approach to separate foreground text from the background in document images. Finds the minimum and maximum intensity values within a window surrounding each pixel. Calculates the threshold by averaging the means of local minima and maxima.

4.3 Deep Learning Architectures for Image Binarization

Besides traditional ML models, deep learning architectures are highly flexible for image segmentation. Image binarization, similar to segmenting an image, involves categorizing each pixel as either background or text, resulting in a binary image that highlights the textual content against the background. In this section, we discuss successful architectures from the literature.

Architectures

UNet:

Description: A common convolutional neural network (CNN) architecture for image segmentation. It comprises a contracting path (encoder) with convolutional layers, activation functions, and pooling layers to extract features from the input image. The expanding path (decoder) involves upsampling layers, skip connections, and concatenation to increase feature map resolution and precise object location. An activation function like a sigmoid follows the final convolutional layer. Optimized using the Adam optimizer.

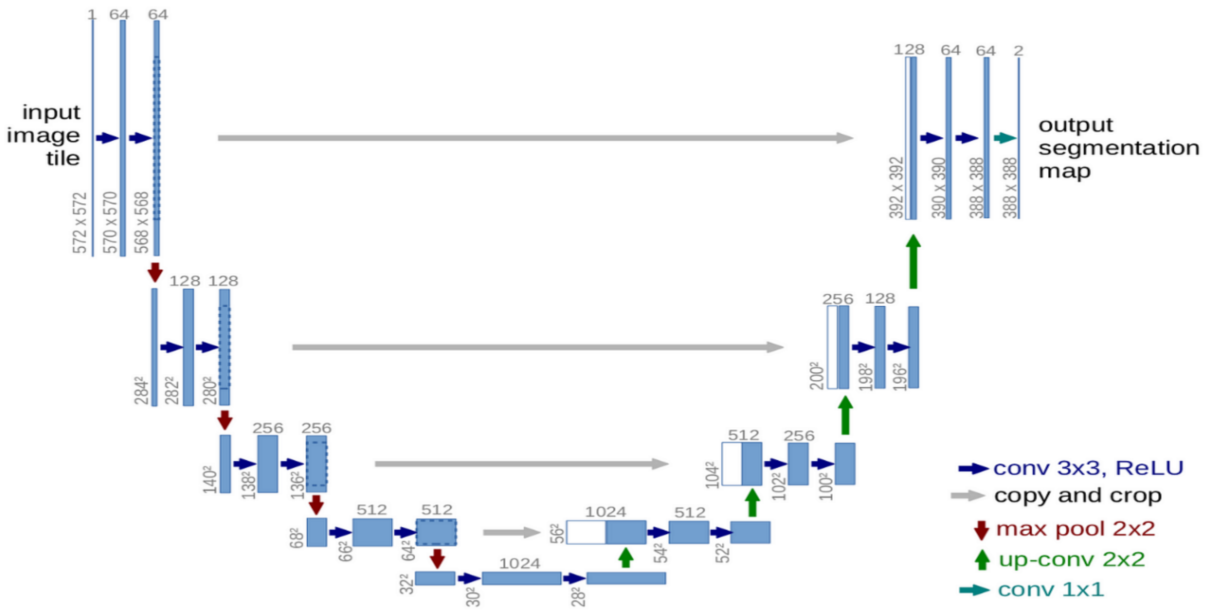


Fig : UNet

UNet++:

Description: Builds nested skip paths instead of single skip connections like UNet. It creates multiple skip paths at different resolutions and integrates refinement blocks within the decoder path to improve segmentation accuracy and reduce errors. Data augmentation and batch normalization enhance generalization and prevent overfitting.

PSPnet:

Description: Pyramid Scene Parsing Network (PSPnet) comprises a pyramid pooling module that extracts features at different scales. Its architecture combines the FCN module with a pyramid structure, capturing information at various scales for better global context understanding and accurate segmentation, particularly useful for complex images with overlapping or partially shaded elements.

LinkNet:

Description: Contains a down-sampling network (contracting path) followed by an upsampling network (expansive path). Uses pre-trained encoders like ResNet or VGG to extract features, which the decoder enlarges to create the segmentation mask. Link connections directly connect corresponding layers in the encoder and decoder to improve communication between network parts.

Feature Pyramid Network (FPN):

- **Description**: Improves the robustness and accuracy of foreground text by employing a dynamically selected background color through a linear transformation process. It serves as a framework for learning pixel classifications, refining a continuous variant of the Pseudo F-measure metric. FPN uses a base network (typically a CNN) to generate a hierarchy of feature maps with top-down and lateral connections.

These deep learning architectures provide various approaches to enhance image binarization, contributing to more accurate and effective segmentation in diverse applications.

Chapter 5

SYSTEM IMPLEMENTATION

5.1 Programming Languages and Implementation Tools

- **Python:** Python is a powerful, high-level programming language that is renowned for being easy to learn and understand. It has extensive libraries and frameworks for numerous applications, including web development, scientific computing, and artificial intelligence, and benefits from a sizable and vibrant community.

- **Google Colab:** Colab is a free Jupyter notebook environment that runs entirely in the cloud. It requires no setup, and the notebooks you create can be simultaneously edited by team members, similar to Google Docs. Colab supports many popular machine learning libraries, which can be easily loaded into your notebook.

Frameworks:

TensorFlow: TensorFlow is an open-source machine learning framework that provides a comprehensive ecosystem for developing and deploying machine learning models. It offers tools and functionalities for building deep learning models, including Convolutional Neural Networks (CNNs). TensorFlow is widely used in computer vision and provides efficient GPU support for accelerated model training.

PyTorch: PyTorch is an open-source machine learning library based on the Torch library, known for its flexibility and ease of use. It offers a dynamic computational graph, allowing for easier debugging and model visualization during development. PyTorch is widely used for applications in natural language processing, image processing, and more recently, computer vision tasks.

OpenCV: OpenCV (Open Source Computer Vision Library) is a popular computer vision library offering a variety of image and video processing functions. It provides tools for image preprocessing, manipulation, and feature extraction, making it suitable for tasks related to image-based machine learning projects. In this project, OpenCV is used for image resizing, normalization, and visualization.

NumPy: NumPy is a fundamental library for numerical computing in Python. It supports large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. NumPy is often used in conjunction with TensorFlow and PyTorch for data manipulation, preprocessing, and numerical computations.

Matplotlib: Matplotlib is a widely used plotting library in Python, providing various functions for creating static, animated, and interactive visualizations. Matplotlib is used for visualizing images, displaying training/validation curves, and generating plots to present project findings.

Chapter 6

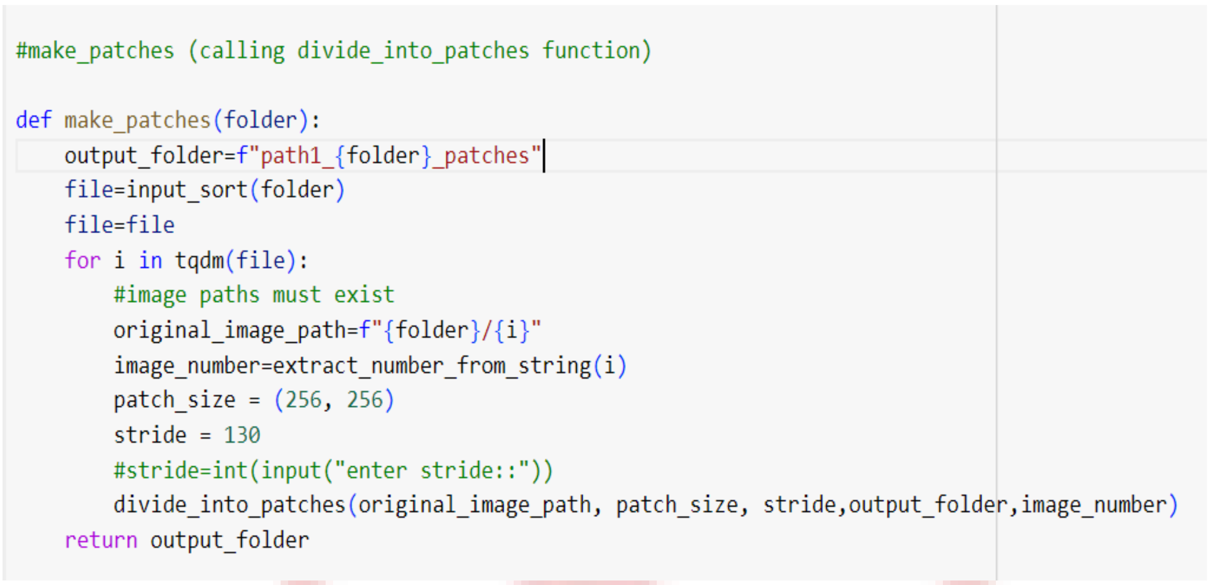
SOURCE CODE

6.1 Code:

Importing Libraries and Path Specifying

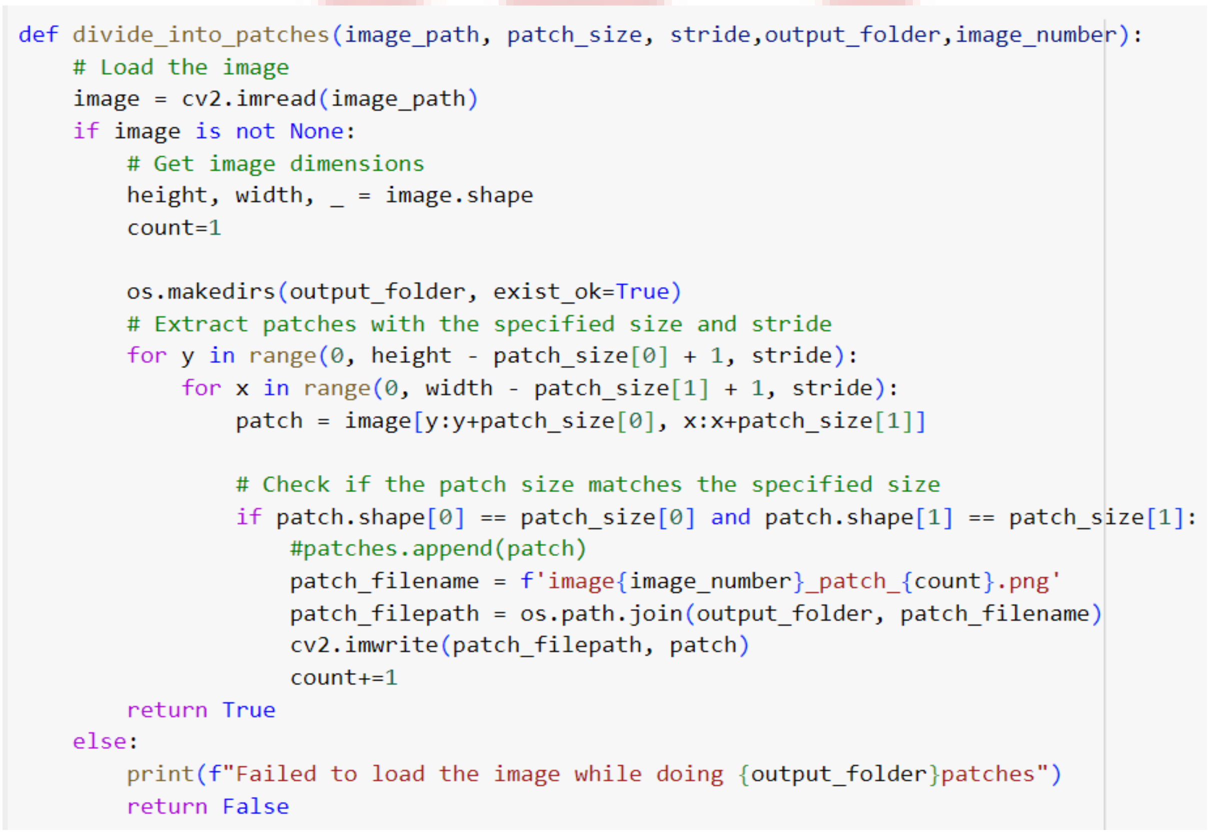
```
import cv2
import os
import numpy as np
from tqdm.auto import tqdm
import re
import torch
from torch.utils.data import DataLoader, TensorDataset
from torch import nn
import segmentation_models_pytorch as smp
from albumentations import Compose, RandomResizedCrop, HorizontalFlip, Rotate
from tabulate import tabulate
```

Making Data into Patches



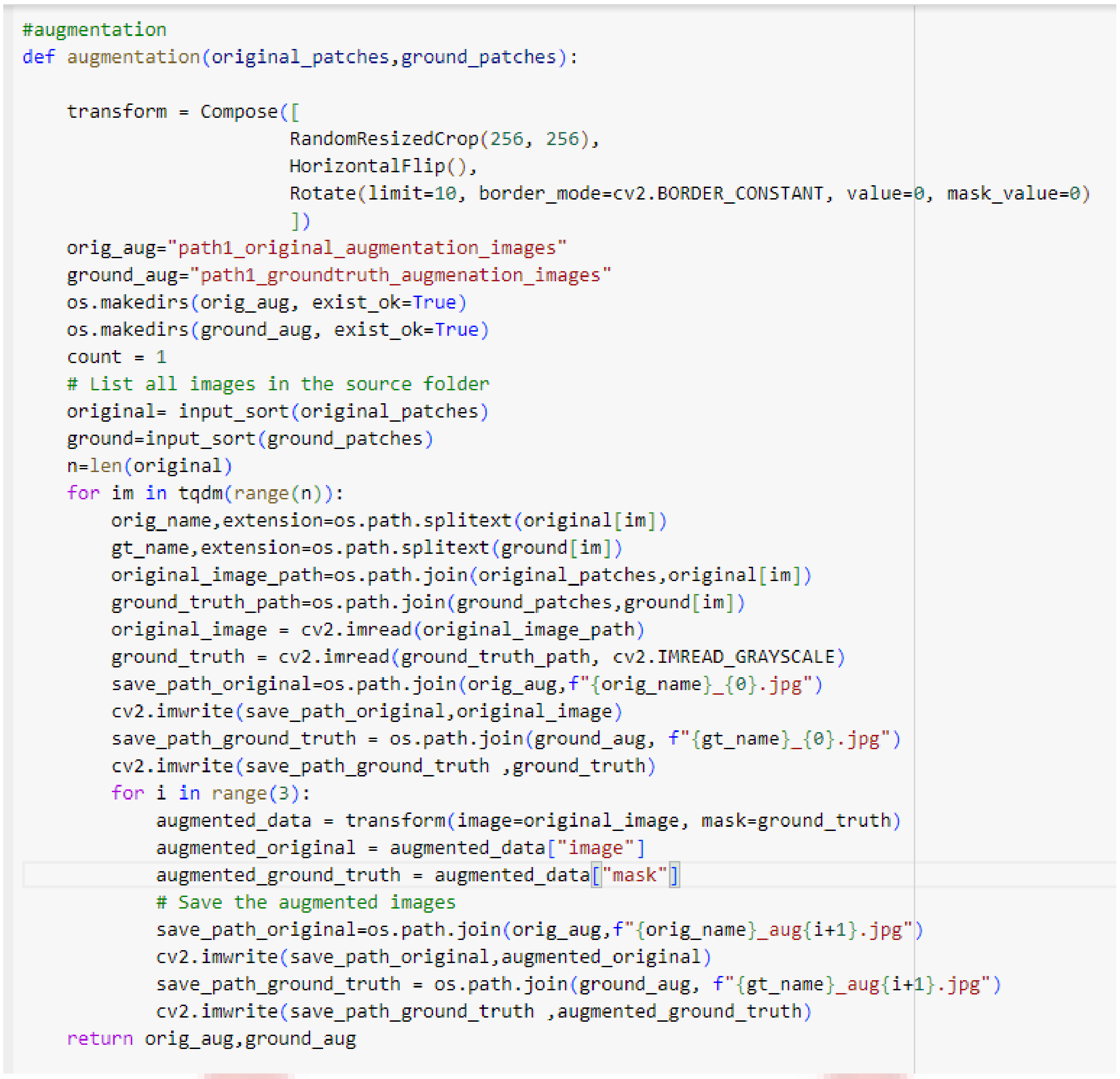
```python
#make_patches (calling divide_into_patches function)

def make_patches(folder):
    output_folder=f"path1_{folder}_patches"
    file=input_sort(folder)
    file=file
    for i in tqdm(file):
        #image paths must exist
        original_image_path=f"{folder}/{i}"
        image_number=extract_number_from_string(i)
        patch_size = (256, 256)
        stride = 130
        #stride=int(input("enter stride::"))
        divide_into_patches(original_image_path, patch_size, stride,output_folder,image_number)
    return output_folder
```

Divide into Patches



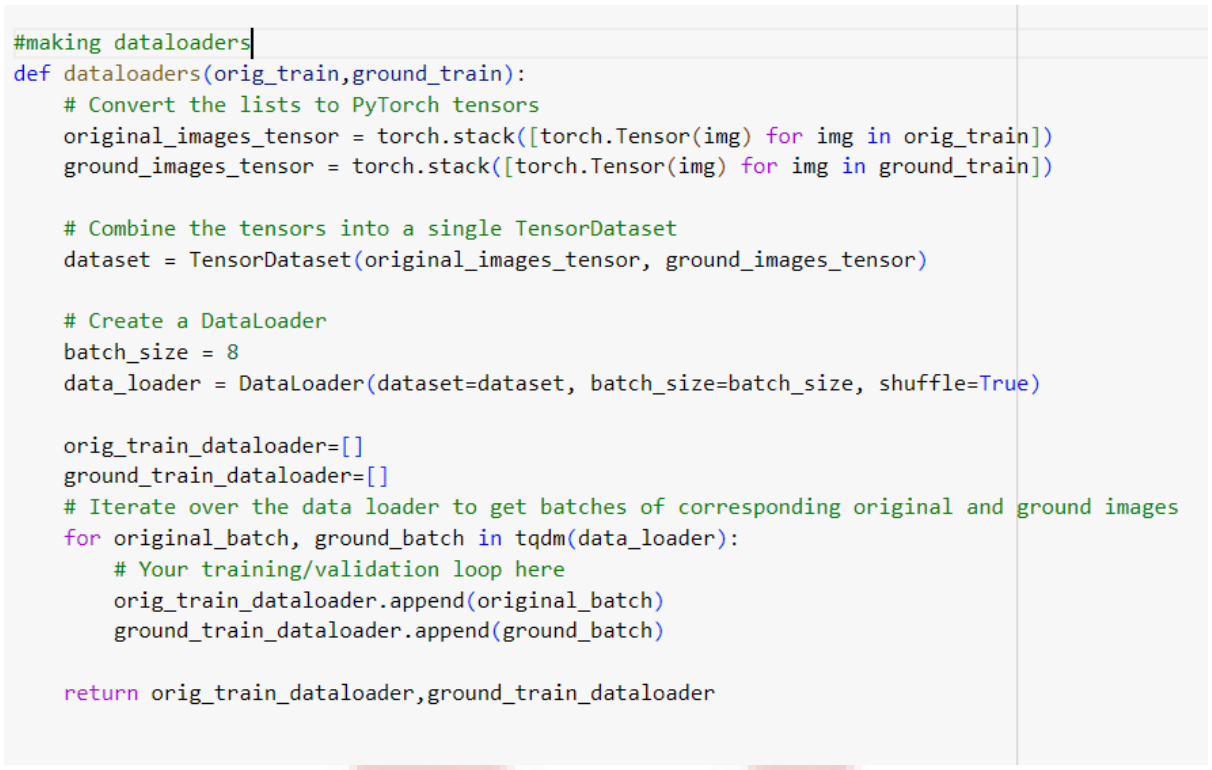
```python
def divide_into_patches(image_path, patch_size, stride,output_folder,image_number):
    # Load the image
    image = cv2.imread(image_path)
    if image is not None:
        # Get image dimensions
        height, width, _ = image.shape
        count=1

        os.makedirs(output_folder, exist_ok=True)
        # Extract patches with the specified size and stride
        for y in range(0, height - patch_size[0] + 1, stride):
            for x in range(0, width - patch_size[1] + 1, stride):
                patch = image[y:y+patch_size[0], x:x+patch_size[1]]

                # Check if the patch size matches the specified size
                if patch.shape[0] == patch_size[0] and patch.shape[1] == patch_size[1]:
                    #patches.append(patch)
                    patch_filename = f'image{image_number}_patch_{count}.png'
                    patch_filepath = os.path.join(output_folder, patch_filename)
                    cv2.imwrite(patch_filepath, patch)
                    count+=1
        return True
    else:
        print(f"Failed to load the image while doing {output_folder}patches")
        return False
```

Data Augmentation



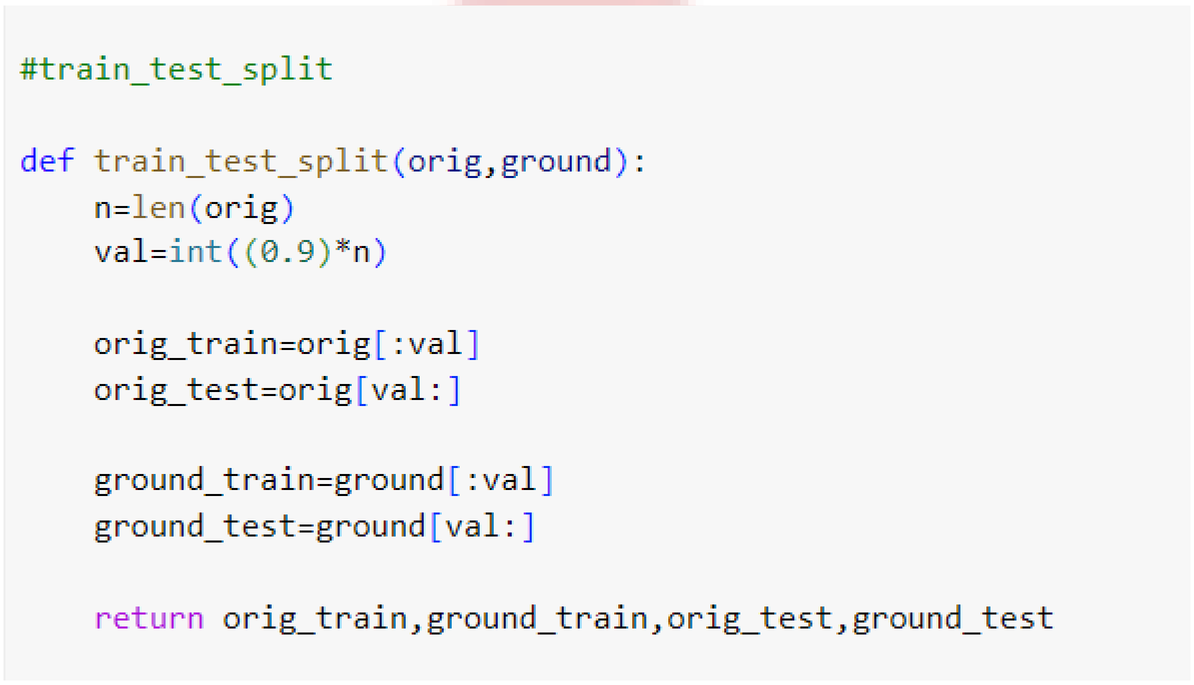
```python
#augmentation
def augmentation(original_patches,ground_patches):

    transform = Compose([
                        RandomResizedCrop(256, 256),
                        HorizontalFlip(),
                        Rotate(limit=10, border_mode=cv2.BORDER_CONSTANT, value=0, mask_value=0)
                        ])
    orig_aug="path1_original_augmentation_images"
    ground_aug="path1_groundtruth_augmenation_images"
    os.makedirs(orig_aug, exist_ok=True)
    os.makedirs(ground_aug, exist_ok=True)
    count = 1
    # List all images in the source folder
    original= input_sort(original_patches)
    ground=input_sort(ground_patches)
    n=len(original)
    for im in tqdm(range(n)):
        orig_name,extension=os.path.splitext(original[im])
        gt_name,extension=os.path.splitext(ground[im])
        original_image_path=os.path.join(original_patches,original[im])
        ground_truth_path=os.path.join(ground_patches,ground[im])
        original_image = cv2.imread(original_image_path)
        ground_truth = cv2.imread(ground_truth_path, cv2.IMREAD_GRAYSCALE)
        save_path_original=os.path.join(orig_aug,f"{orig_name}_{0}.jpg")
        cv2.imwrite(save_path_original,original_image)
        save_path_ground_truth = os.path.join(ground_aug, f"{gt_name}_{0}.jpg")
        cv2.imwrite(save_path_ground_truth ,ground_truth)
        for i in range(3):
            augmented_data = transform(image=original_image, mask=ground_truth)
            augmented_original = augmented_data["image"]
            augmented_ground_truth = augmented_data["mask"]
            # Save the augmented images
            save_path_original=os.path.join(orig_aug,f"{orig_name}_aug{i+1}.jpg")
            cv2.imwrite(save_path_original,augmented_original)
            save_path_ground_truth = os.path.join(ground_aug, f"{gt_name}_aug{i+1}.jpg")
            cv2.imwrite(save_path_ground_truth ,augmented_ground_truth)
    return orig_aug,ground_aug
```

Data Loaders



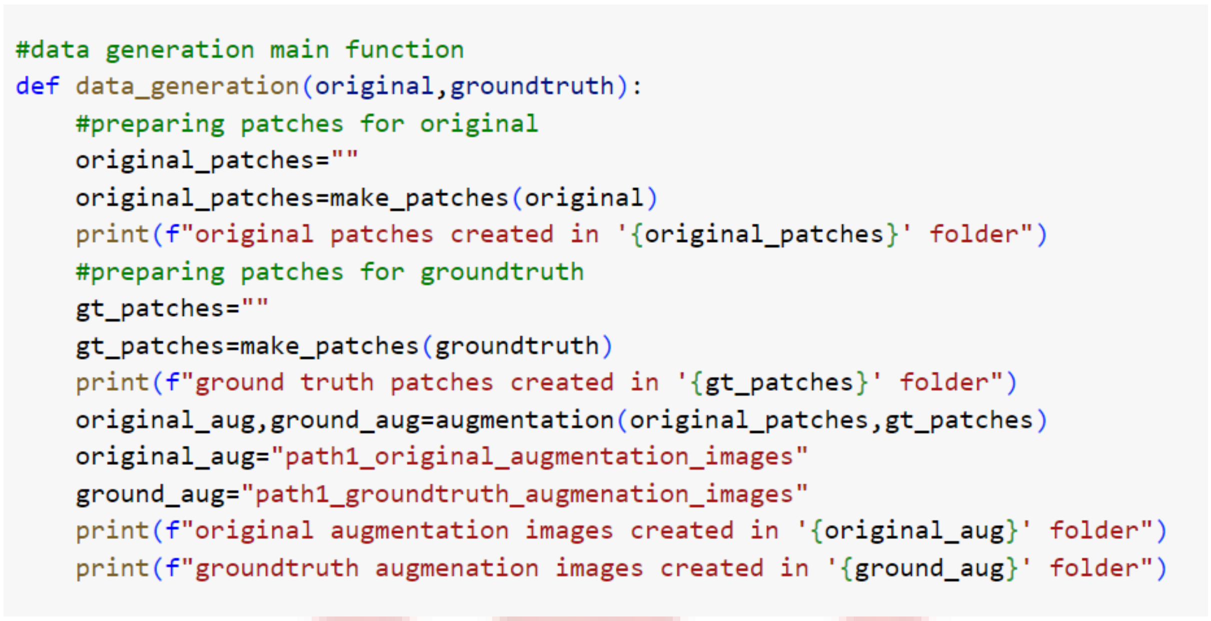
```python
#making dataloaders
def dataloaders(orig_train,ground_train):
    # Convert the lists to PyTorch tensors
    original_images_tensor = torch.stack([torch.Tensor(img) for img in orig_train])
    ground_images_tensor = torch.stack([torch.Tensor(img) for img in ground_train])

    # Combine the tensors into a single TensorDataset
    dataset = TensorDataset(original_images_tensor, ground_images_tensor)

    # Create a DataLoader
    batch_size = 8
    data_loader = DataLoader(dataset=dataset, batch_size=batch_size, shuffle=True)

    orig_train_dataloader=[]
    ground_train_dataloader=[]
    # Iterate over the data loader to get batches of corresponding original and ground images
    for original_batch, ground_batch in tqdm(data_loader):
        # Your training/validation loop here
        orig_train_dataloader.append(original_batch)
        ground_train_dataloader.append(ground_batch)

    return orig_train_dataloader,ground_train_dataloader
```

Train Test Split

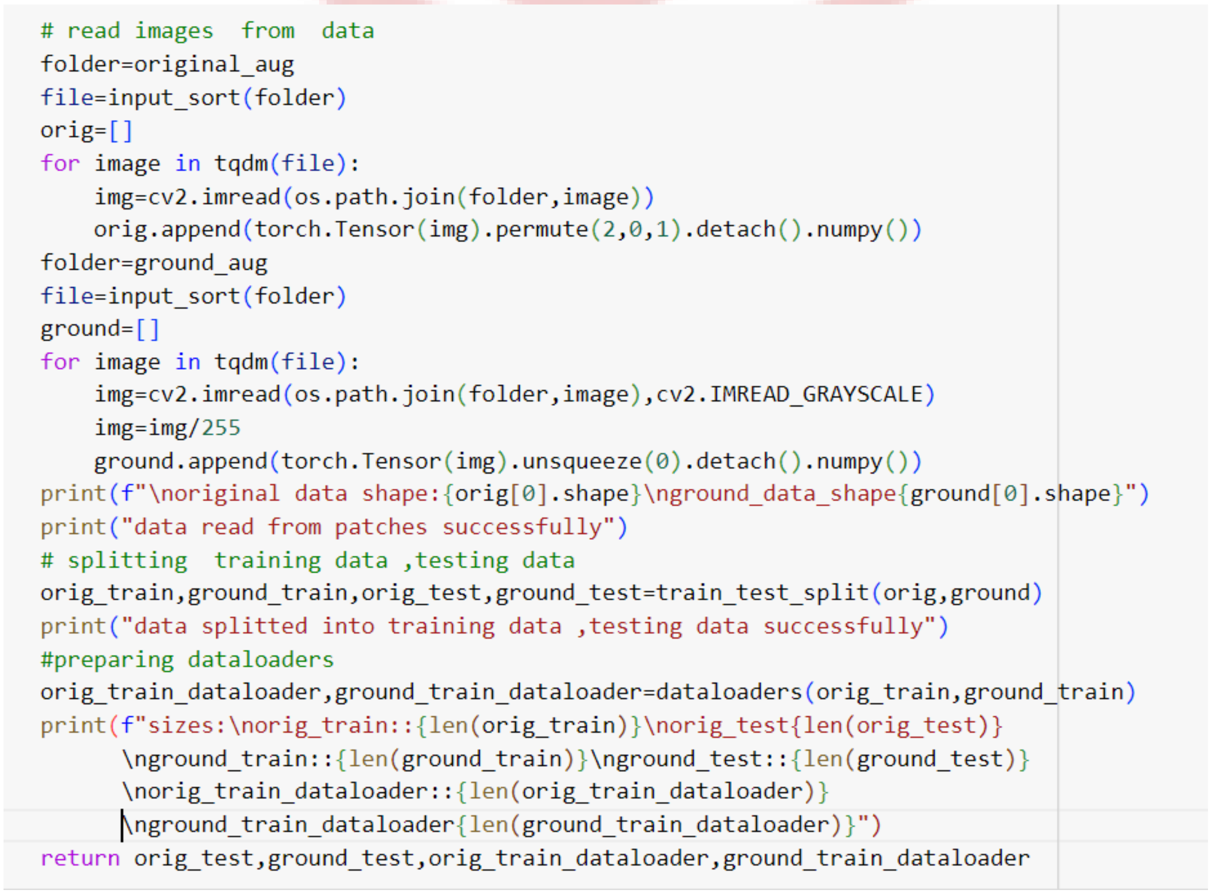


```python
#train_test_split

def train_test_split(orig,ground):
    n=len(orig)
    val=int((0.9)*n)

    orig_train=orig[:val]
    orig_test=orig[val:]

    ground_train=ground[:val]
    ground_test=ground[val:]

    return orig_train,ground_train,orig_test,ground_test
```

Data Generation

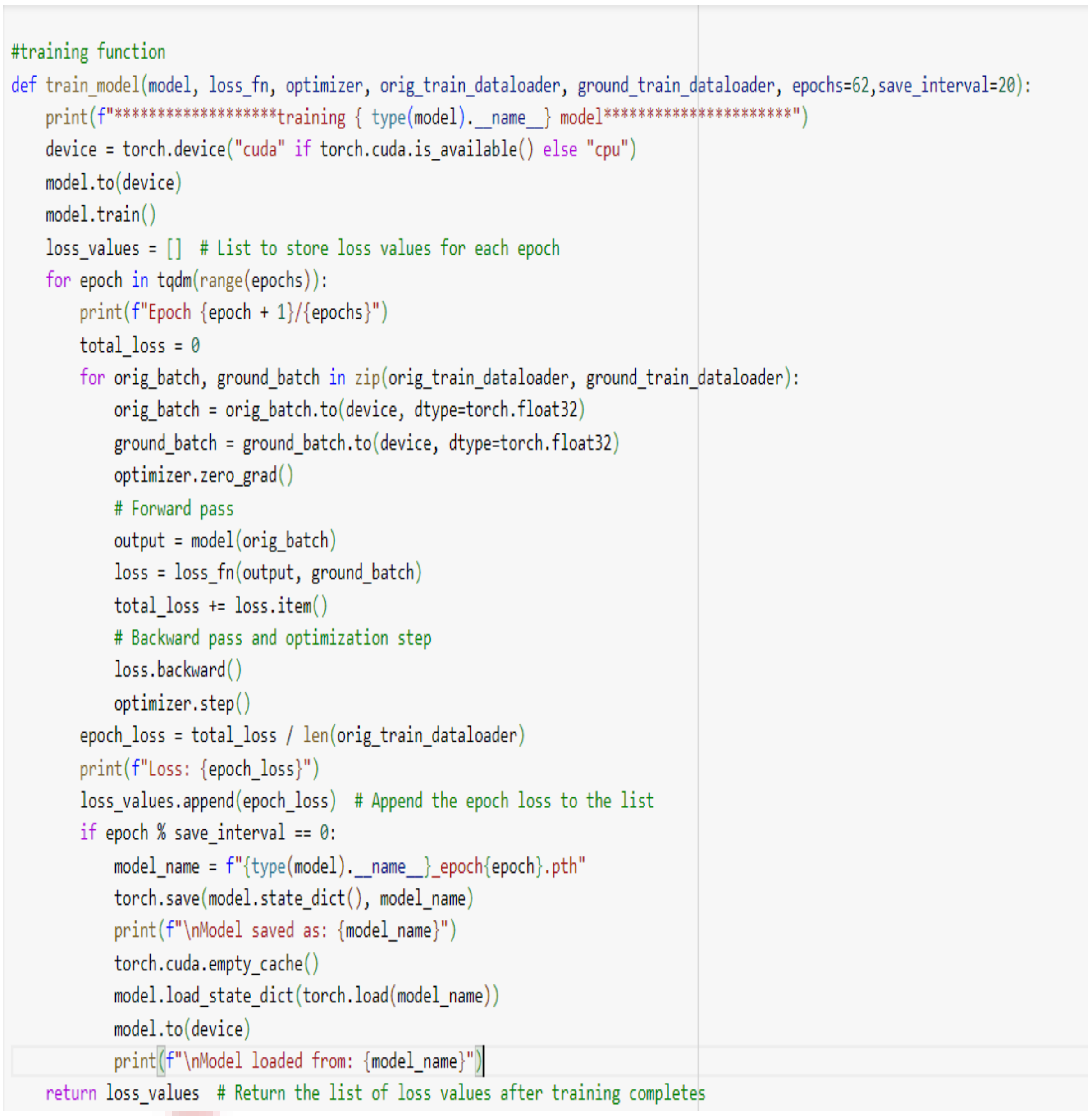


```python
#data generation main function
def data_generation(original,groundtruth):
    #preparing patches for original
    original_patches=""
    original_patches=make_patches(original)
    print(f"original patches created in '{original_patches}' folder")
    #preparing patches for groundtruth
    gt_patches=""
    gt_patches=make_patches(groundtruth)
    print(f"ground truth patches created in '{gt_patches}' folder")
    original_aug,ground_aug=augmentation(original_patches,gt_patches)
    original_aug="path1_original_augmentation_images"
    ground_aug="path1_groundtruth_augmenation_images"
    print(f"original augmentation images created in '{original_aug}' folder")
    print(f"groundtruth augmenation images created in '{ground_aug}' folder")
```



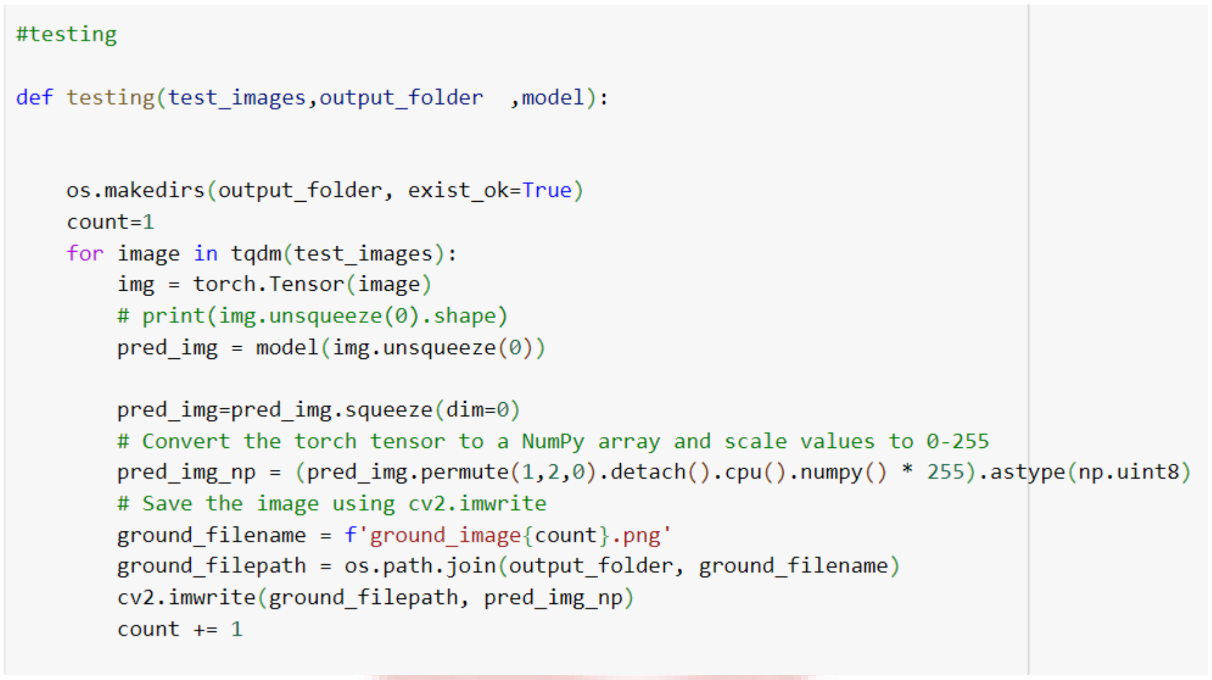
```python
# read images  from  data
folder=original_aug
file=input_sort(folder)
orig=[]
for image in tqdm(file):
    img=cv2.imread(os.path.join(folder,image))
    orig.append(torch.Tensor(img).permute(2,0,1).detach().numpy())
folder=ground_aug
file=input_sort(folder)
ground=[]
for image in tqdm(file):
    img=cv2.imread(os.path.join(folder,image),cv2.IMREAD_GRAYSCALE)
    img=img/255
    ground.append(torch.Tensor(img).unsqueeze(0).detach().numpy())
print(f"\noriginal data shape:{orig[0].shape}\nground_data_shape{ground[0].shape}")
print("data read from patches successfully")
# splitting  training data ,testing data
orig_train,ground_train,orig_test,ground_test=train_test_split(orig,ground)
print("data splitted into training data ,testing data successfully")
#preparing dataloaders
orig_train_dataloader,ground_train_dataloader=dataloaders(orig_train,ground_train)
print(f"sizes:\norig_train::{len(orig_train)}\norig_test{len(orig_test)}
      \nground_train::{len(ground_train)}\nground_test::{len(ground_test)}
      \norig_train_dataloader::{len(orig_train_dataloader)}
      \nground_train_dataloader{len(ground_train_dataloader)}")
return orig_test,ground_test,orig_train_dataloader,ground_train_dataloader
```

Training Model



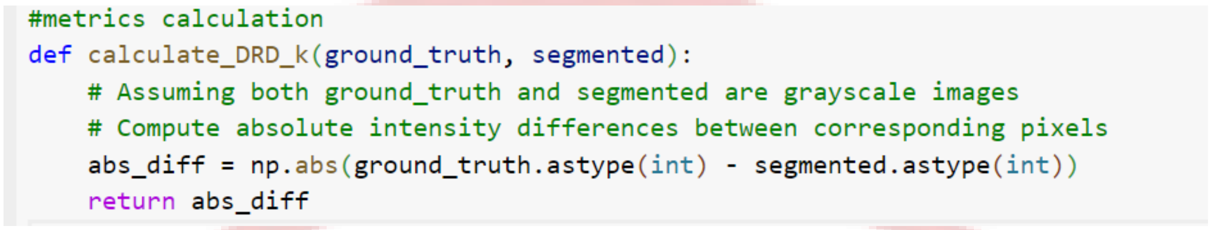
```
#training function
def train_model(model, loss_fn, optimizer, orig_train_dataloader, ground_train_dataloader, epochs=62,save_interval=20):
    print(f"******************training { type(model).__name__} model*********************")
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.to(device)
    model.train()
    loss_values = []  # List to store loss values for each epoch
    for epoch in tqdm(range(epochs)):
        print(f"Epoch {epoch + 1}/{epochs}")
        total_loss = 0
        for orig_batch, ground_batch in zip(orig_train_dataloader, ground_train_dataloader):
            orig_batch = orig_batch.to(device, dtype=torch.float32)
            ground_batch = ground_batch.to(device, dtype=torch.float32)
            optimizer.zero_grad()
            # Forward pass
            output = model(orig_batch)
            loss = loss_fn(output, ground_batch)
            total_loss += loss.item()
            # Backward pass and optimization step
            loss.backward()
            optimizer.step()
        epoch_loss = total_loss / len(orig_train_dataloader)
        print(f"Loss: {epoch_loss}")
        loss_values.append(epoch_loss)  # Append the epoch loss to the list
        if epoch % save_interval == 0:
            model_name = f"{type(model).__name__}_epoch{epoch}.pth"
            torch.save(model.state_dict(), model_name)
            print(f"\nModel saved as: {model_name}")
            torch.cuda.empty_cache()
            model.load_state_dict(torch.load(model_name))
            model.to(device)
            print(f"\nModel loaded from: {model_name}")
    return loss_values  # Return the list of loss values after training completes
```

Testing Data

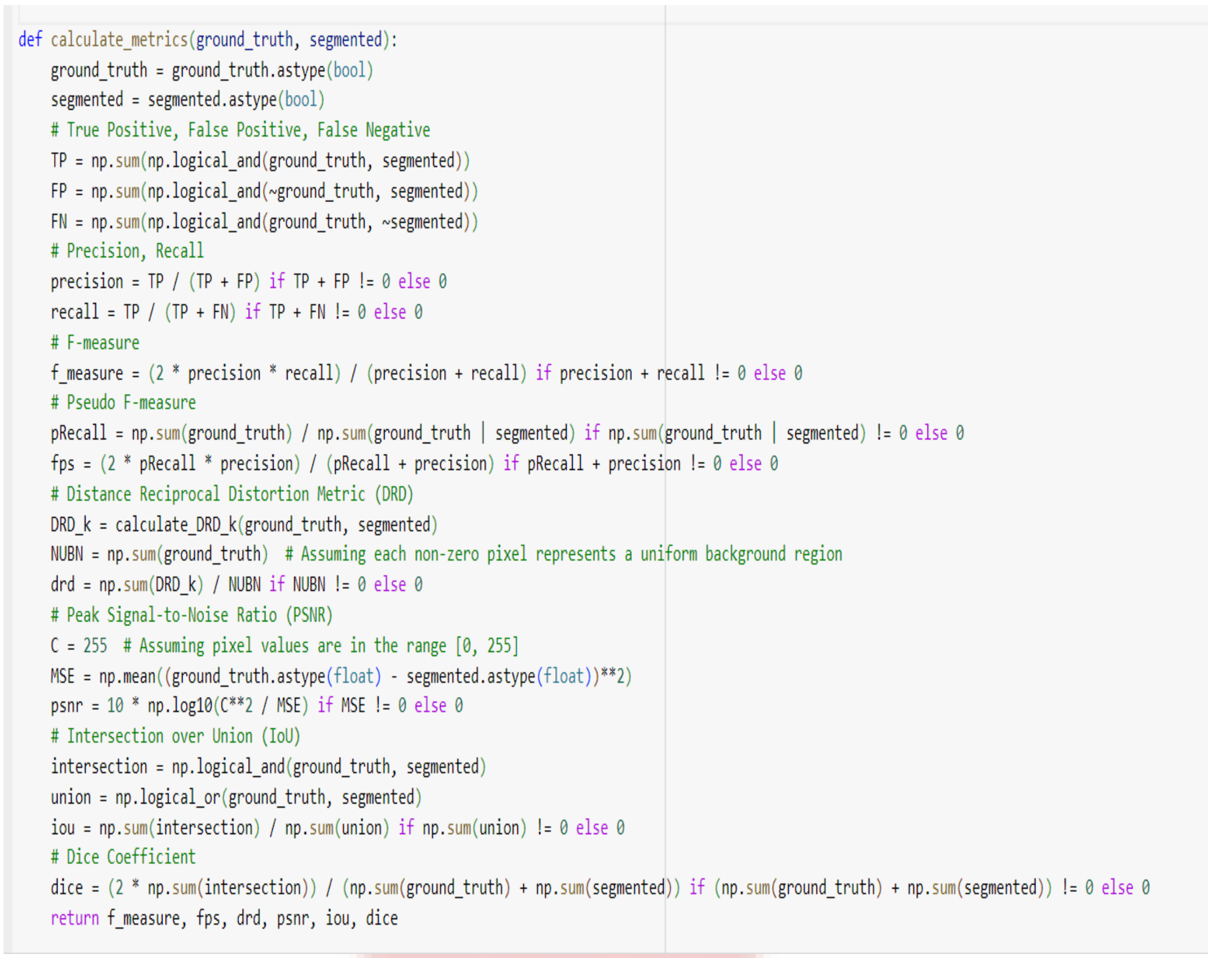


```
#testing

def testing(test_images,output_folder  ,model):


    os.makedirs(output_folder, exist_ok=True)
    count=1
    for image in tqdm(test_images):
        img = torch.Tensor(image)
        # print(img.unsqueeze(0).shape)
        pred_img = model(img.unsqueeze(0))

        pred_img=pred_img.squeeze(dim=0)
        # Convert the torch tensor to a NumPy array and scale values to 0-255
        pred_img_np = (pred_img.permute(1,2,0).detach().cpu().numpy() * 255).astype(np.uint8)
        # Save the image using cv2.imwrite
        ground_filename = f'ground_image{count}.png'
        ground_filepath = os.path.join(output_folder, ground_filename)
        cv2.imwrite(ground_filepath, pred_img_np)
        count += 1
```

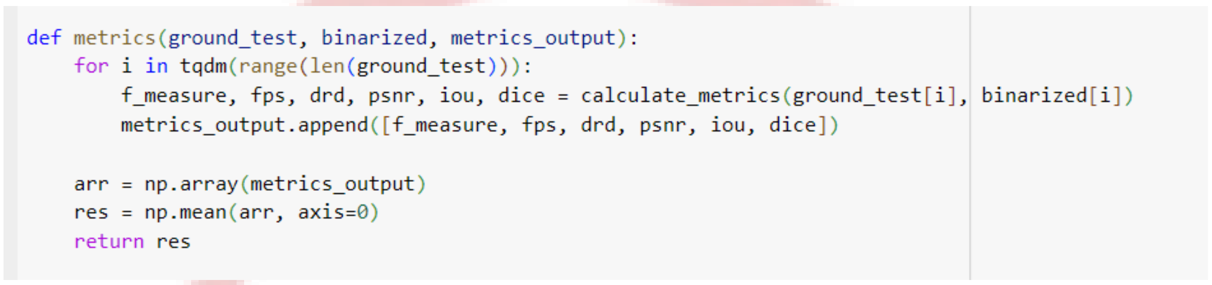
Metrics Calculation



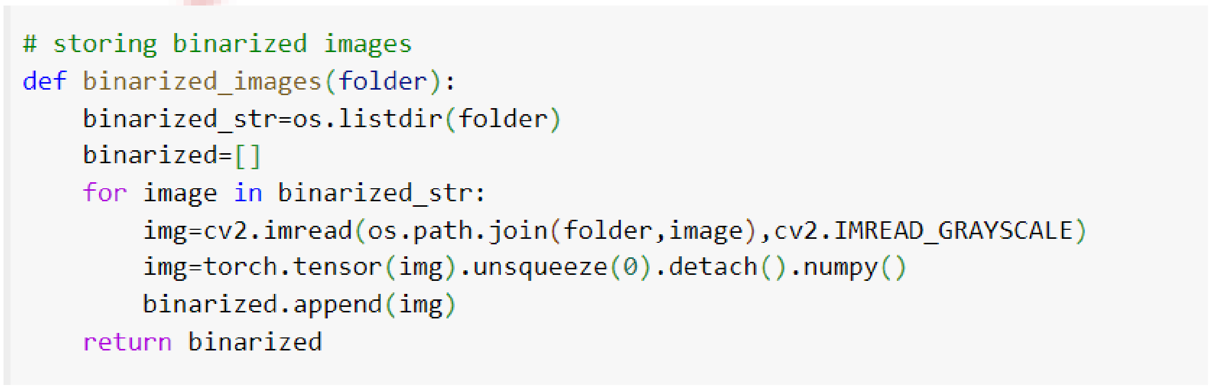
```
#metrics calculation
def calculate_DRD_k(ground_truth, segmented):
    # Assuming both ground_truth and segmented are grayscale images
    # Compute absolute intensity differences between corresponding pixels
    abs_diff = np.abs(ground_truth.astype(int) - segmented.astype(int))
    return abs_diff
```



```python
def calculate_metrics(ground_truth, segmented):
    ground_truth = ground_truth.astype(bool)
    segmented = segmented.astype(bool)
    # True Positive, False Positive, False Negative
    TP = np.sum(np.logical_and(ground_truth, segmented))
    FP = np.sum(np.logical_and(~ground_truth, segmented))
    FN = np.sum(np.logical_and(ground_truth, ~segmented))
    # Precision, Recall
    precision = TP / (TP + FP) if TP + FP != 0 else 0
    recall = TP / (TP + FN) if TP + FN != 0 else 0
    # F-measure
    f_measure = (2 * precision * recall) / (precision + recall) if precision + recall != 0 else 0
    # Pseudo F-measure
    pRecall = np.sum(ground_truth) / np.sum(ground_truth | segmented) if np.sum(ground_truth | segmented) != 0 else 0
    fps = (2 * pRecall * precision) / (pRecall + precision) if pRecall + precision != 0 else 0
    # Distance Reciprocal Distortion Metric (DRD)
    DRD_k = calculate_DRD_k(ground_truth, segmented)
    NUBN = np.sum(ground_truth)  # Assuming each non-zero pixel represents a uniform background region
    drd = np.sum(DRD_k) / NUBN if NUBN != 0 else 0
    # Peak Signal-to-Noise Ratio (PSNR)
    C = 255  # Assuming pixel values are in the range [0, 255]
    MSE = np.mean((ground_truth.astype(float) - segmented.astype(float))**2)
    psnr = 10 * np.log10(C**2 / MSE) if MSE != 0 else 0
    # Intersection over Union (IoU)
    intersection = np.logical_and(ground_truth, segmented)
    union = np.logical_or(ground_truth, segmented)
    iou = np.sum(intersection) / np.sum(union) if np.sum(union) != 0 else 0
    # Dice Coefficient
    dice = (2 * np.sum(intersection)) / (np.sum(ground_truth) + np.sum(segmented)) if (np.sum(ground_truth) + np.sum(segmented)) != 0 else 0
    return f_measure, fps, drd, psnr, iou, dice
```



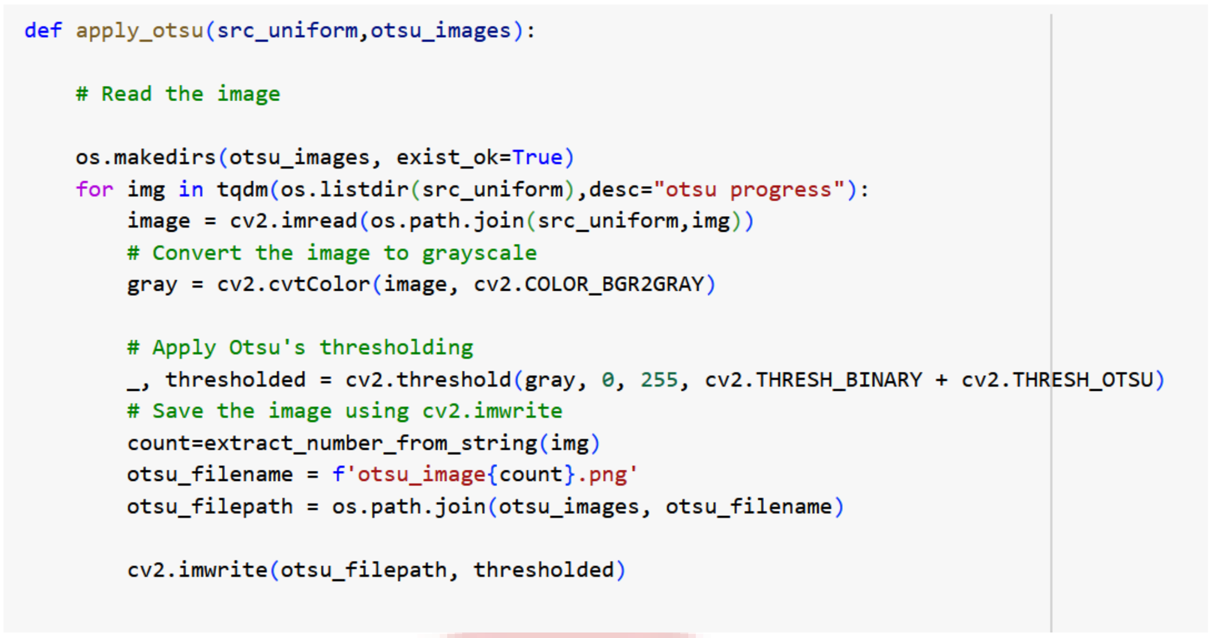
```python
def metrics(ground_test, binarized, metrics_output):
    for i in tqdm(range(len(ground_test))):
        f_measure, fps, drd, psnr, iou, dice = calculate_metrics(ground_test[i], binarized[i])
        metrics_output.append([f_measure, fps, drd, psnr, iou, dice])

    arr = np.array(metrics_output)
    res = np.mean(arr, axis=0)
    return res
```



```python
# storing binarized images
def binarized_images(folder):
    binarized_str=os.listdir(folder)
    binarized=[]
    for image in binarized_str:
        img=cv2.imread(os.path.join(folder,image),cv2.IMREAD_GRAYSCALE)
        img=torch.tensor(img).unsqueeze(0).detach().numpy()
        binarized.append(img)
    return binarized
```

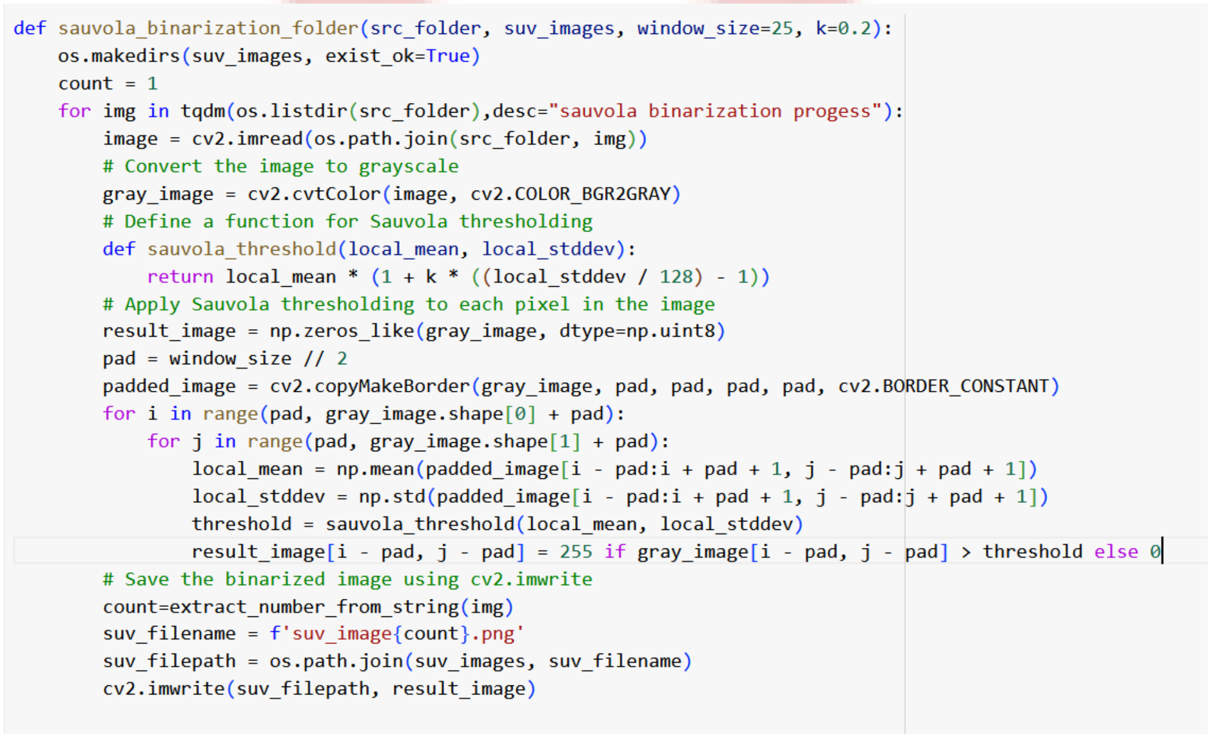
Traditional ML Algorithms

OTSU



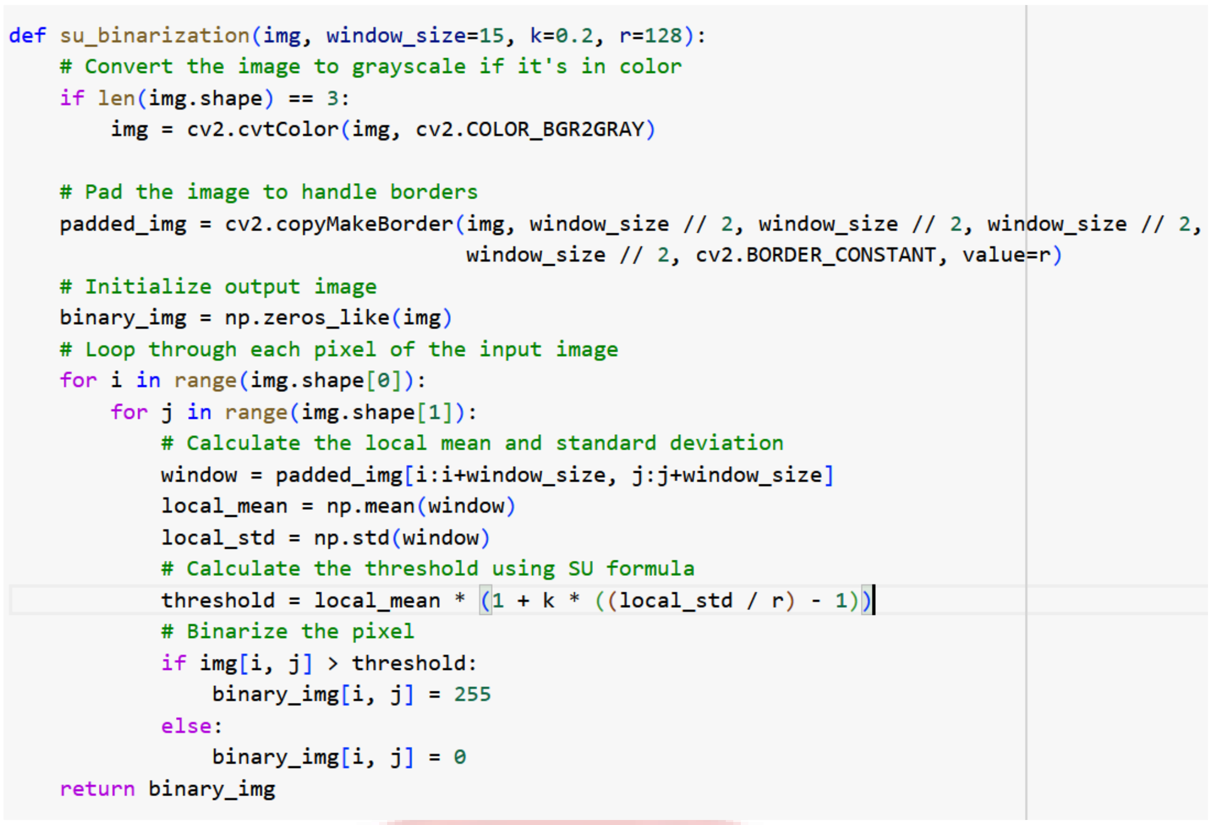
```python
def apply_otsu(src_uniform,otsu_images):

    # Read the image

    os.makedirs(otsu_images, exist_ok=True)
    for img in tqdm(os.listdir(src_uniform),desc="otsu progress"):
        image = cv2.imread(os.path.join(src_uniform,img))
        # Convert the image to grayscale
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        # Apply Otsu's thresholding
        _, thresholded = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
        # Save the image using cv2.imwrite
        count=extract_number_from_string(img)
        otsu_filename = f'otsu_image{count}.png'
        otsu_filepath = os.path.join(otsu_images, otsu_filename)

        cv2.imwrite(otsu_filepath, thresholded)
```

Sauvola

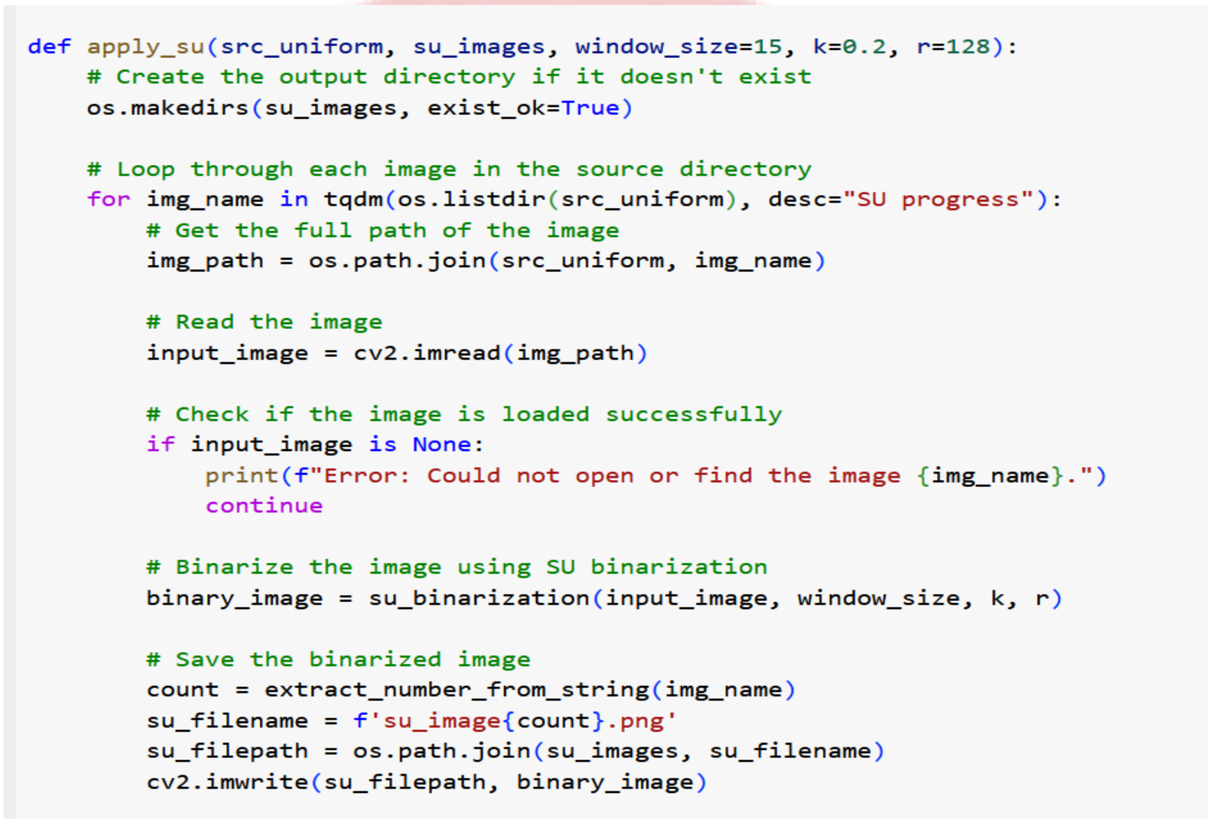


```python
def sauvola_binarization_folder(src_folder, suv_images, window_size=25, k=0.2):
    os.makedirs(suv_images, exist_ok=True)
    count = 1
    for img in tqdm(os.listdir(src_folder),desc="sauvola binarization progess"):
        image = cv2.imread(os.path.join(src_folder, img))
        # Convert the image to grayscale
        gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        # Define a function for Sauvola thresholding
        def sauvola_threshold(local_mean, local_stddev):
            return local_mean * (1 + k * ((local_stddev / 128) - 1))
        # Apply Sauvola thresholding to each pixel in the image
        result_image = np.zeros_like(gray_image, dtype=np.uint8)
        pad = window_size // 2
        padded_image = cv2.copyMakeBorder(gray_image, pad, pad, pad, pad, cv2.BORDER_CONSTANT)
        for i in range(pad, gray_image.shape[0] + pad):
            for j in range(pad, gray_image.shape[1] + pad):
                local_mean = np.mean(padded_image[i - pad:i + pad + 1, j - pad:j + pad + 1])
                local_stddev = np.std(padded_image[i - pad:i + pad + 1, j - pad:j + pad + 1])
                threshold = sauvola_threshold(local_mean, local_stddev)
                result_image[i - pad, j - pad] = 255 if gray_image[i - pad, j - pad] > threshold else 0
        # Save the binarized image using cv2.imwrite
        count=extract_number_from_string(img)
        suv_filename = f'suv_image{count}.png'
        suv_filepath = os.path.join(suv_images, suv_filename)
        cv2.imwrite(suv_filepath, result_image)
```

SU-Binarization

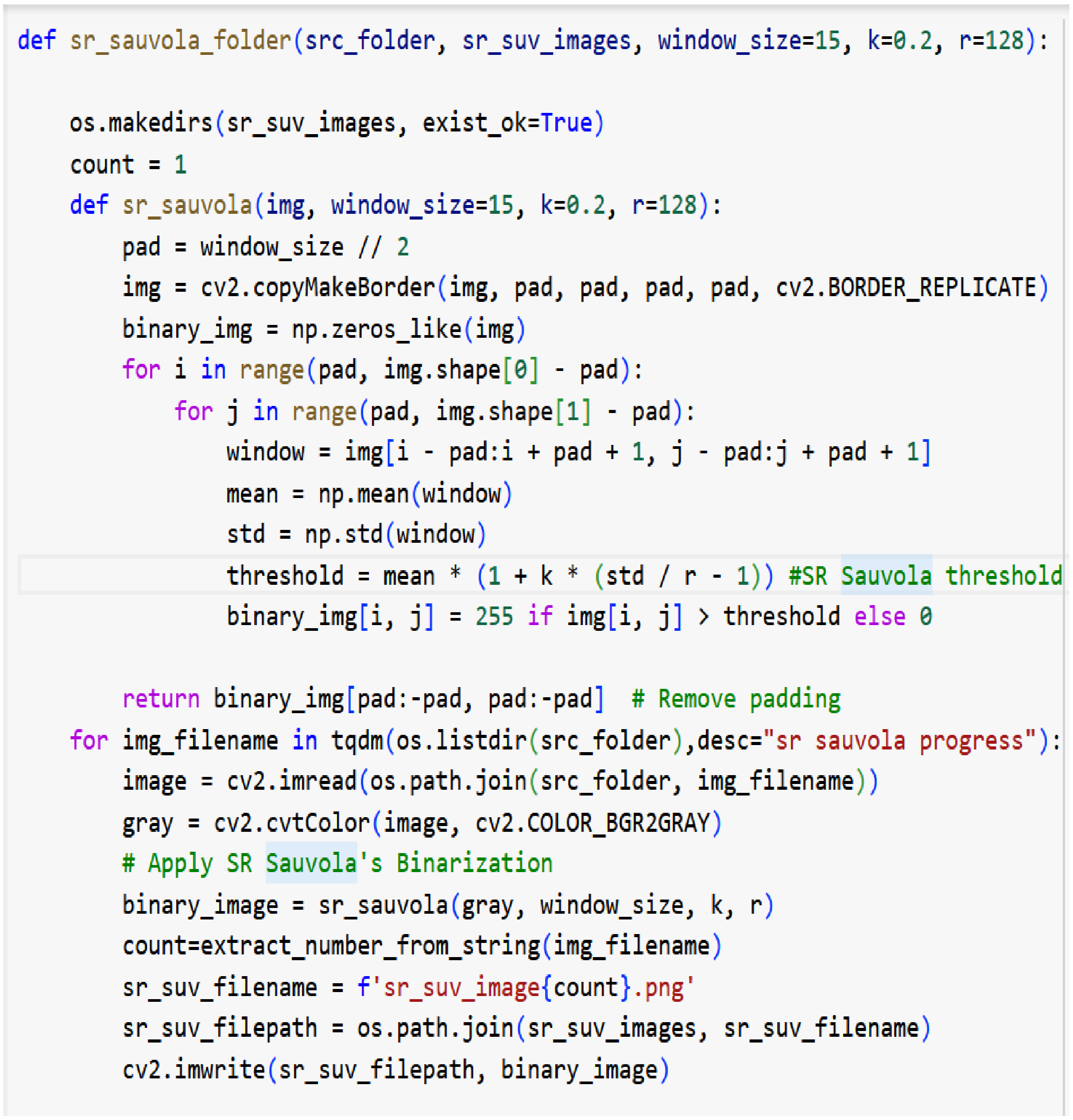


```
def su_binarization(img, window_size=15, k=0.2, r=128):
    # Convert the image to grayscale if it's in color
    if len(img.shape) == 3:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Pad the image to handle borders
    padded_img = cv2.copyMakeBorder(img, window_size // 2, window_size // 2, window_size // 2,
                                    window_size // 2, cv2.BORDER_CONSTANT, value=r)
    # Initialize output image
    binary_img = np.zeros_like(img)
    # Loop through each pixel of the input image
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            # Calculate the local mean and standard deviation
            window = padded_img[i:i+window_size, j:j+window_size]
            local_mean = np.mean(window)
            local_std = np.std(window)
            # Calculate the threshold using SU formula
            threshold = local_mean * (1 + k * ((local_std / r) - 1))
            # Binarize the pixel
            if img[i, j] > threshold:
                binary_img[i, j] = 255
            else:
                binary_img[i, j] = 0
    return binary_img
```



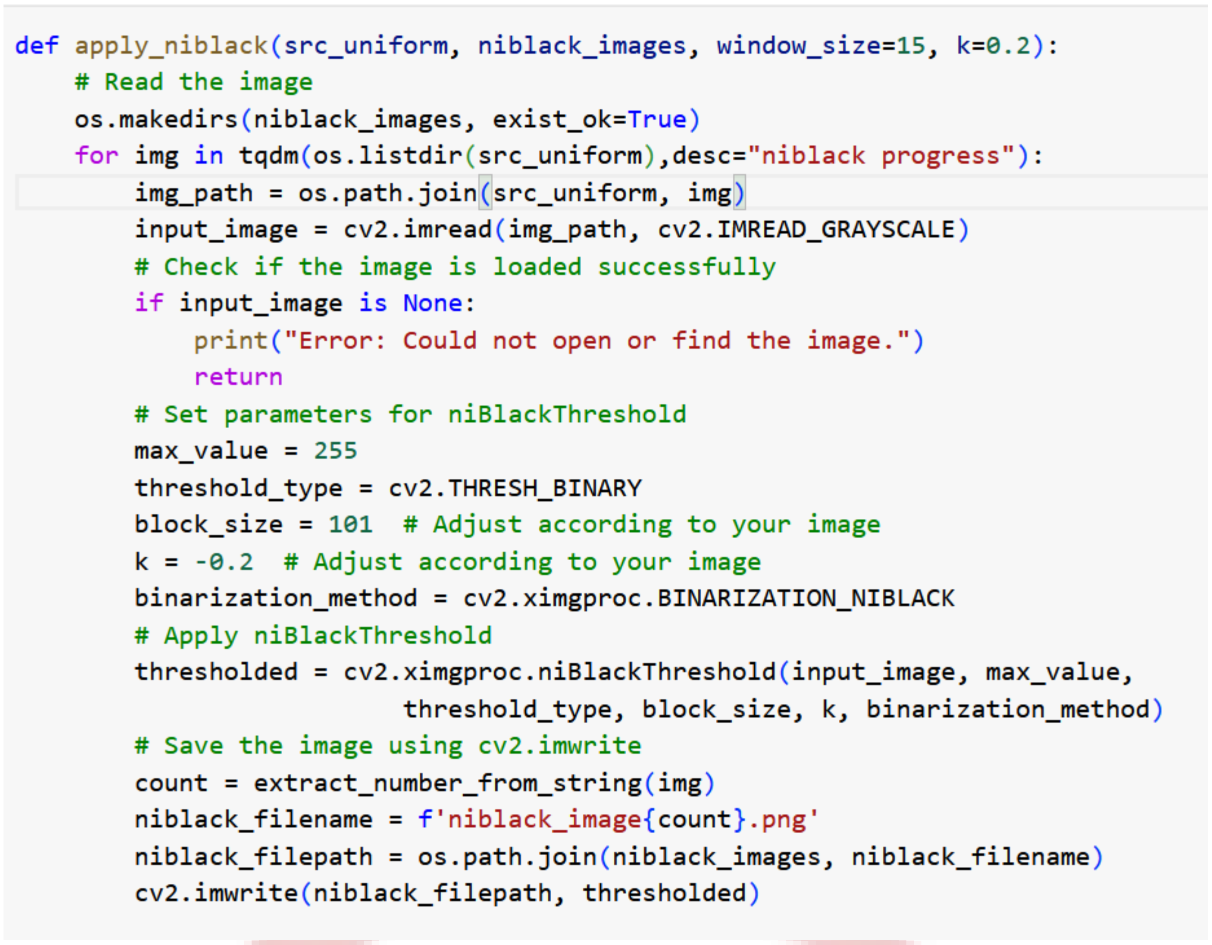
```
def apply_su(src_uniform, su_images, window_size=15, k=0.2, r=128):
    # Create the output directory if it doesn't exist
    os.makedirs(su_images, exist_ok=True)

    # Loop through each image in the source directory
    for img_name in tqdm(os.listdir(src_uniform), desc="SU progress"):
        # Get the full path of the image
        img_path = os.path.join(src_uniform, img_name)

        # Read the image
        input_image = cv2.imread(img_path)

        # Check if the image is loaded successfully
        if input_image is None:
            print(f"Error: Could not open or find the image {img_name}.")
            continue

        # Binarize the image using SU binarization
        binary_image = su_binarization(input_image, window_size, k, r)

        # Save the binarized image
        count = extract_number_from_string(img_name)
        su_filename = f'su_image{count}.png'
        su_filepath = os.path.join(su_images, su_filename)
        cv2.imwrite(su_filepath, binary_image)
```

sr_sauvola

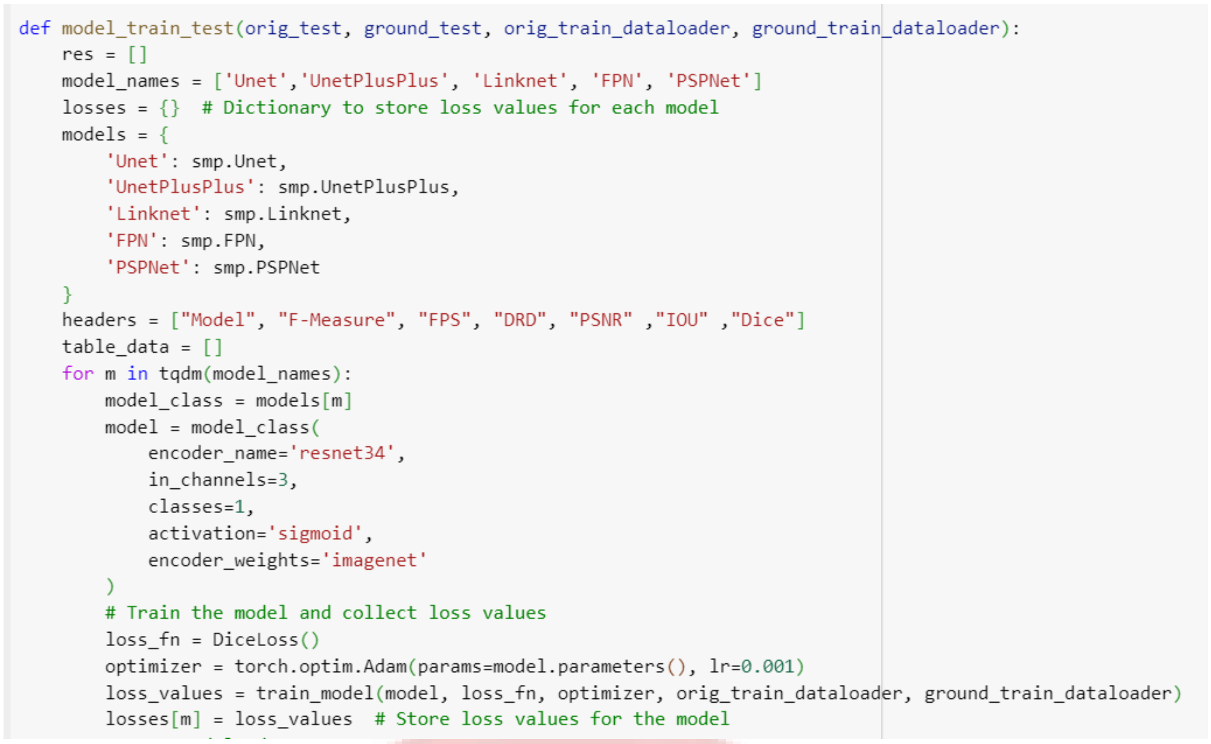


```python
def sr_sauvola_folder(src_folder, sr_suv_images, window_size=15, k=0.2, r=128):

    os.makedirs(sr_suv_images, exist_ok=True)
    count = 1
    def sr_sauvola(img, window_size=15, k=0.2, r=128):
        pad = window_size // 2
        img = cv2.copyMakeBorder(img, pad, pad, pad, pad, cv2.BORDER_REPLICATE)
        binary_img = np.zeros_like(img)
        for i in range(pad, img.shape[0] - pad):
            for j in range(pad, img.shape[1] - pad):
                window = img[i - pad:i + pad + 1, j - pad:j + pad + 1]
                mean = np.mean(window)
                std = np.std(window)
                threshold = mean * (1 + k * (std / r - 1)) #SR Sauvola threshold
                binary_img[i, j] = 255 if img[i, j] > threshold else 0

        return binary_img[pad:-pad, pad:-pad]  # Remove padding
    for img_filename in tqdm(os.listdir(src_folder),desc="sr sauvola progress"):
        image = cv2.imread(os.path.join(src_folder, img_filename))
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        # Apply SR Sauvola's Binarization
        binary_image = sr_sauvola(gray, window_size, k, r)
        count=extract_number_from_string(img_filename)
        sr_suv_filename = f'sr_suv_image{count}.png'
        sr_suv_filepath = os.path.join(sr_suv_images, sr_suv_filename)
        cv2.imwrite(sr_suv_filepath, binary_image)
```

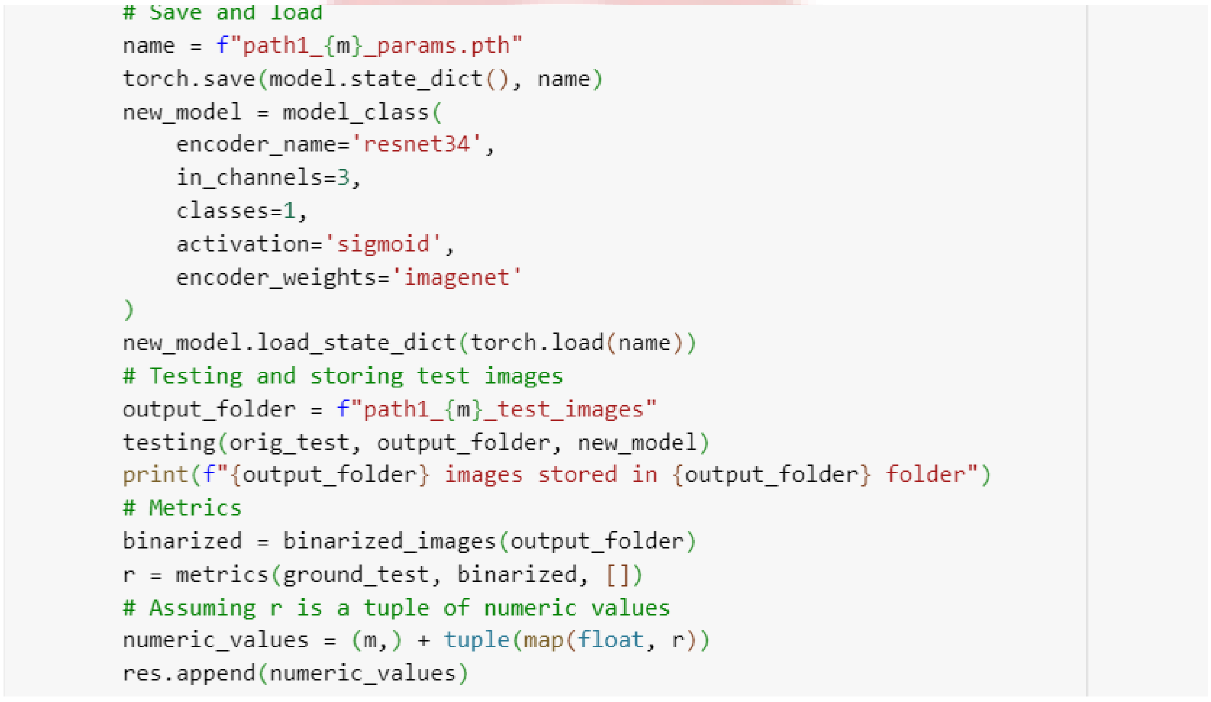
NIBLACK



```python
def apply_niblack(src_uniform, niblack_images, window_size=15, k=0.2):
    # Read the image
    os.makedirs(niblack_images, exist_ok=True)
    for img in tqdm(os.listdir(src_uniform),desc="niblack progress"):
        img_path = os.path.join(src_uniform, img)
        input_image = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
        # Check if the image is loaded successfully
        if input_image is None:
            print("Error: Could not open or find the image.")
            return
        # Set parameters for niBlackThreshold
        max_value = 255
        threshold_type = cv2.THRESH_BINARY
        block_size = 101  # Adjust according to your image
        k = -0.2  # Adjust according to your image
        binarization_method = cv2.ximgproc.BINARIZATION_NIBLACK
        # Apply niBlackThreshold
        thresholded = cv2.ximgproc.niBlackThreshold(input_image, max_value,
                          threshold_type, block_size, k, binarization_method)
        # Save the image using cv2.imwrite
        count = extract_number_from_string(img)
        niblack_filename = f'niblack_image{count}.png'
        niblack_filepath = os.path.join(niblack_images, niblack_filename)
        cv2.imwrite(niblack_filepath, thresholded)
```

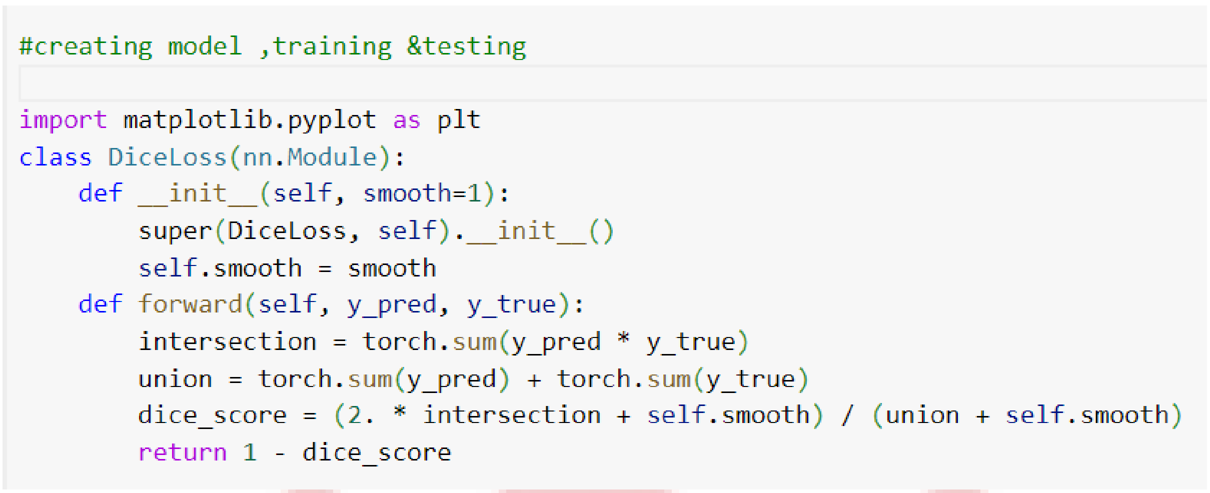


```python
def model_train_test(orig_test, ground_test, orig_train_dataloader, ground_train_dataloader):
    res = []
    model_names = ['Unet','UnetPlusPlus', 'Linknet', 'FPN', 'PSPNet']
    losses = {}  # Dictionary to store loss values for each model
    models = {
        'Unet': smp.Unet,
        'UnetPlusPlus': smp.UnetPlusPlus,
        'Linknet': smp.Linknet,
        'FPN': smp.FPN,
        'PSPNet': smp.PSPNet
    }
    headers = ["Model", "F-Measure", "FPS", "DRD", "PSNR" ,"IOU" ,"Dice"]
    table_data = []
    for m in tqdm(model_names):
        model_class = models[m]
        model = model_class(
            encoder_name='resnet34',
            in_channels=3,
            classes=1,
            activation='sigmoid',
            encoder_weights='imagenet'
        )
        # Train the model and collect loss values
        loss_fn = DiceLoss()
        optimizer = torch.optim.Adam(params=model.parameters(), lr=0.001)
        loss_values = train_model(model, loss_fn, optimizer, orig_train_dataloader, ground_train_dataloader)
        losses[m] = loss_values  # Store loss values for the model
```

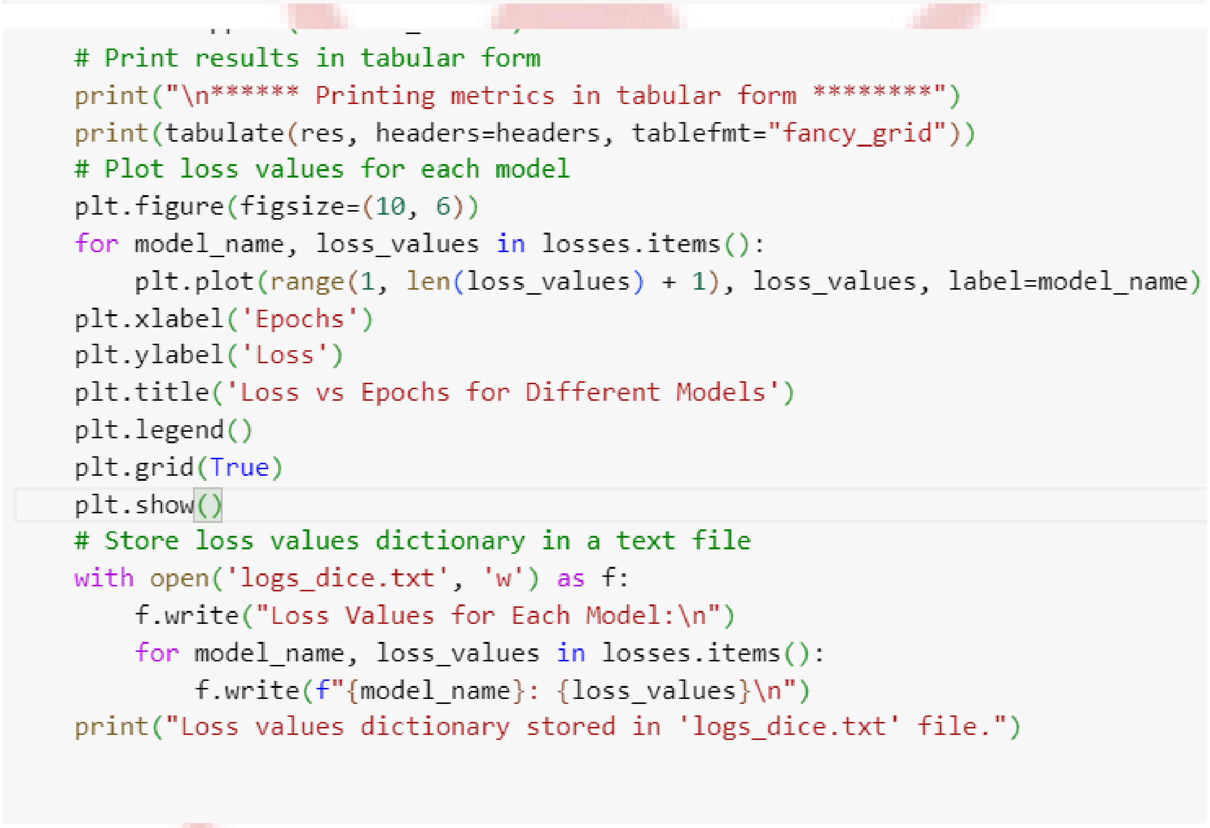


```python
        # Save and load
        name = f"path1_{m}_params.pth"
        torch.save(model.state_dict(), name)
        new_model = model_class(
            encoder_name='resnet34',
            in_channels=3,
            classes=1,
            activation='sigmoid',
            encoder_weights='imagenet'
        )
        new_model.load_state_dict(torch.load(name))
        # Testing and storing test images
        output_folder = f"path1_{m}_test_images"
        testing(orig_test, output_folder, new_model)
        print(f"{output_folder} images stored in {output_folder} folder")
        # Metrics
        binarized = binarized_images(output_folder)
        r = metrics(ground_test, binarized, [])
        # Assuming r is a tuple of numeric values
        numeric_values = (m,) + tuple(map(float, r))
        res.append(numeric_values)
```

Data-Visualization

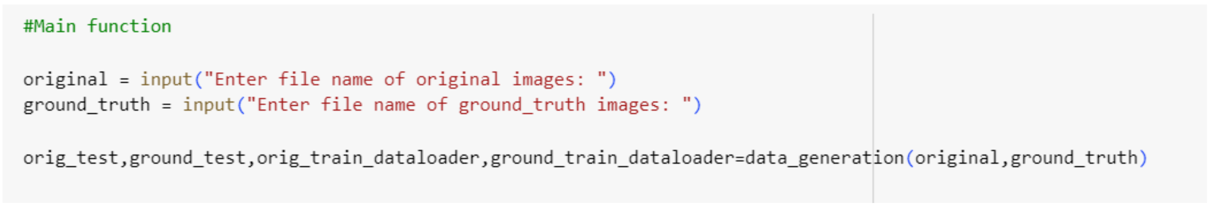


```
#creating model ,training &testing

import matplotlib.pyplot as plt
class DiceLoss(nn.Module):
    def __init__(self, smooth=1):
        super(DiceLoss, self).__init__()
        self.smooth = smooth
    def forward(self, y_pred, y_true):
        intersection = torch.sum(y_pred * y_true)
        union = torch.sum(y_pred) + torch.sum(y_true)
        dice_score = (2. * intersection + self.smooth) / (union + self.smooth)
        return 1 - dice_score
```



```
    # Print results in tabular form
    print("\n****** Printing metrics in tabular form ********")
    print(tabulate(res, headers=headers, tablefmt="fancy_grid"))
    # Plot loss values for each model
    plt.figure(figsize=(10, 6))
    for model_name, loss_values in losses.items():
        plt.plot(range(1, len(loss_values) + 1), loss_values, label=model_name)
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title('Loss vs Epochs for Different Models')
    plt.legend()
    plt.grid(True)
    plt.show()
    # Store loss values dictionary in a text file
    with open('logs_dice.txt', 'w') as f:
        f.write("Loss Values for Each Model:\n")
        for model_name, loss_values in losses.items():
            f.write(f"{model_name}: {loss_values}\n")
    print("Loss values dictionary stored in 'logs_dice.txt' file.")
```

Main Function



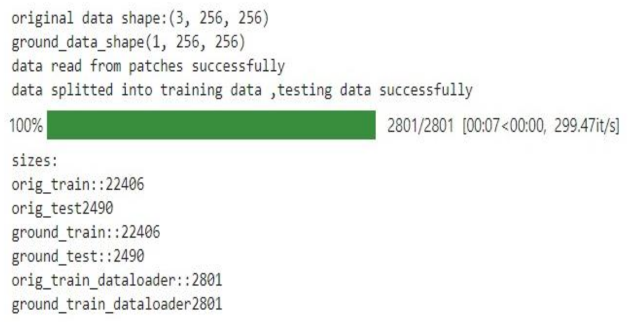
```
#Main function

original = input("Enter file name of original images: ")
ground_truth = input("Enter file name of ground_truth images: ")

orig_test,ground_test,orig_train_dataloader,ground_train_dataloader=data_generation(original,ground_truth)
```

Chapter-7

MODEL EVALUATION AND RESULTS

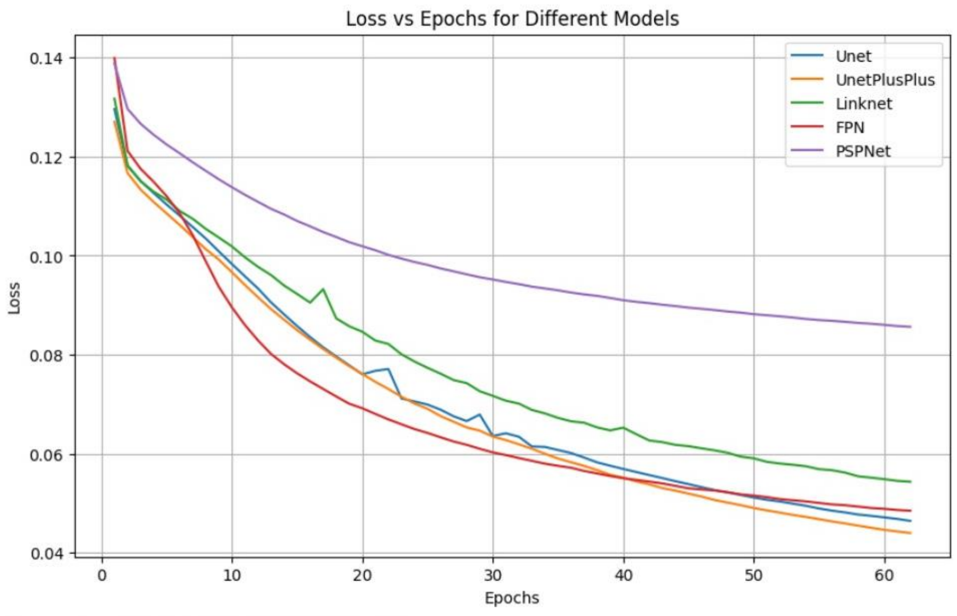
7.1 Model Evaluation

Train Accuracy and Test Accuracy

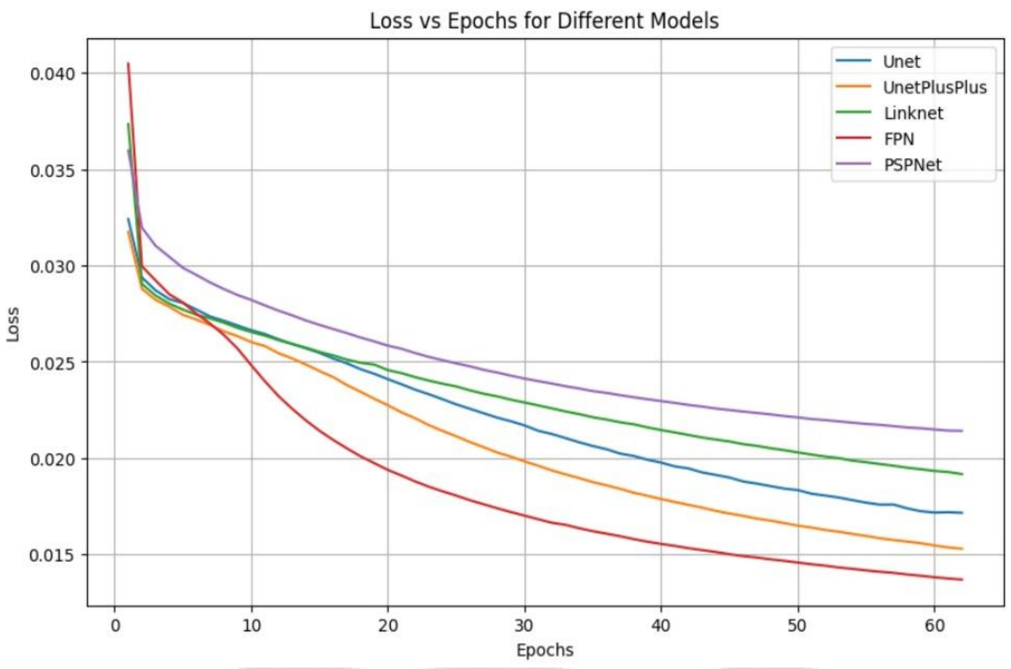


```
original data shape:(3, 256, 256)
ground_data_shape(1, 256, 256)
data read from patches successfully
data splitted into training data ,testing data successfully
100%                                      2801/2801 [00:07<00:00, 299.47it/s]
sizes:
orig_train::22406
orig_test2490
ground_train::22406
ground_test::2490
orig_train_dataloader::2801
ground_train_dataloader2801
```

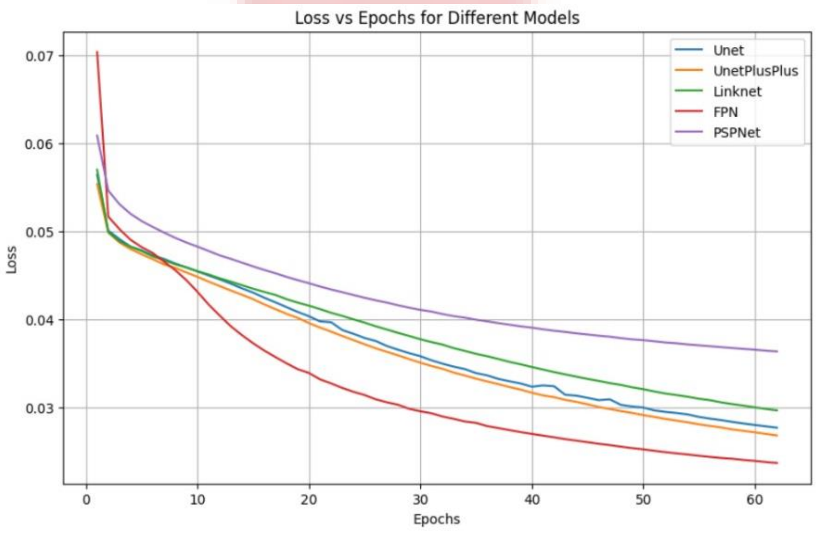
Training Losses Graph for Various Loss functions



a) BCE Loss Function

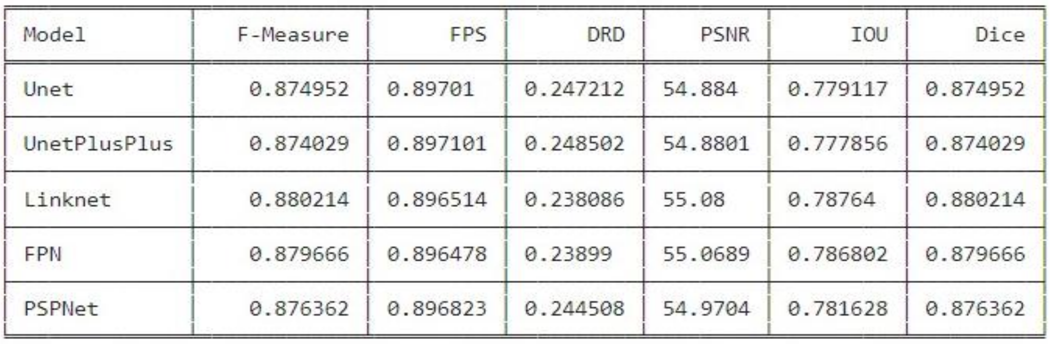


b) Dice Loss Function



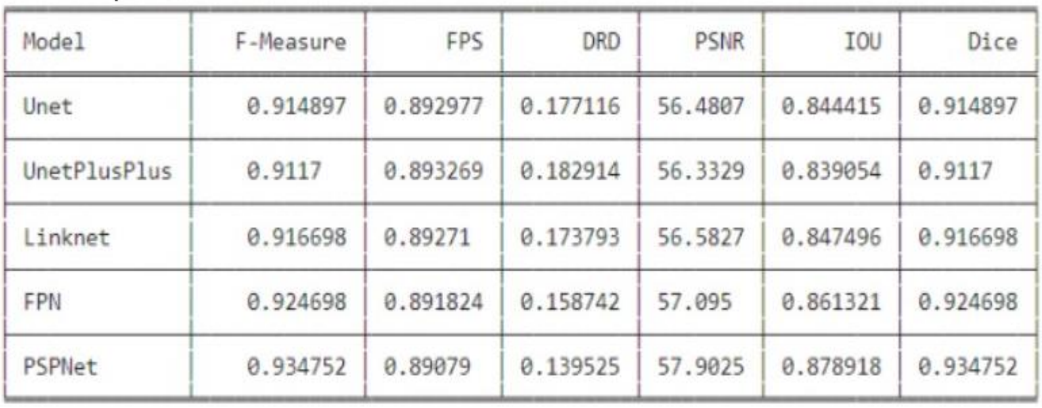
c) L1 Loss Function graph

7.2 Evaluation Metrics



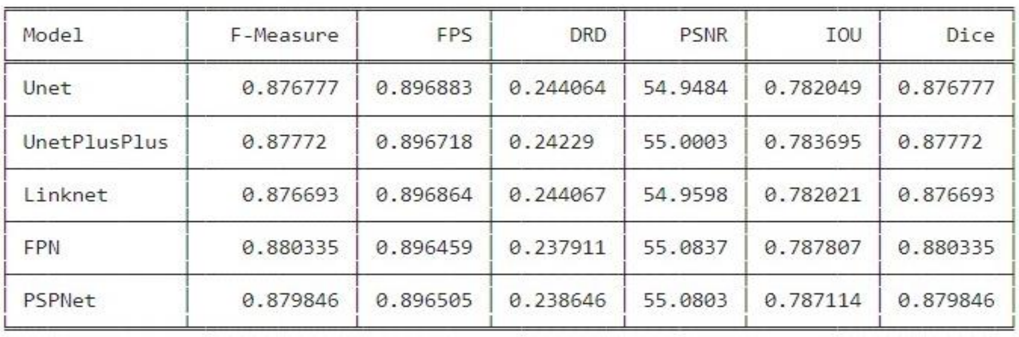
Model	F-Measure	FPS	DRD	PSNR	IOU	Dice
Unet	0.874952	0.89701	0.247212	54.884	0.779117	0.874952
UnetPlusPlus	0.874029	0.897101	0.248502	54.8801	0.777856	0.874029
Linknet	0.880214	0.896514	0.238086	55.08	0.78764	0.880214
FPN	0.879666	0.896478	0.23899	55.0689	0.786802	0.879666
PSPNet	0.876362	0.896823	0.244508	54.9704	0.781628	0.876362

a) L1 Metrics



Model	F-Measure	FPS	DRD	PSNR	IOU	Dice
Unet	0.914897	0.892977	0.177116	56.4807	0.844415	0.914897
UnetPlusPlus	0.9117	0.893269	0.182914	56.3329	0.839054	0.9117
Linknet	0.916698	0.89271	0.173793	56.5827	0.847496	0.916698
FPN	0.924698	0.891824	0.158742	57.095	0.861321	0.924698
PSPNet	0.934752	0.89079	0.139525	57.9025	0.878918	0.934752

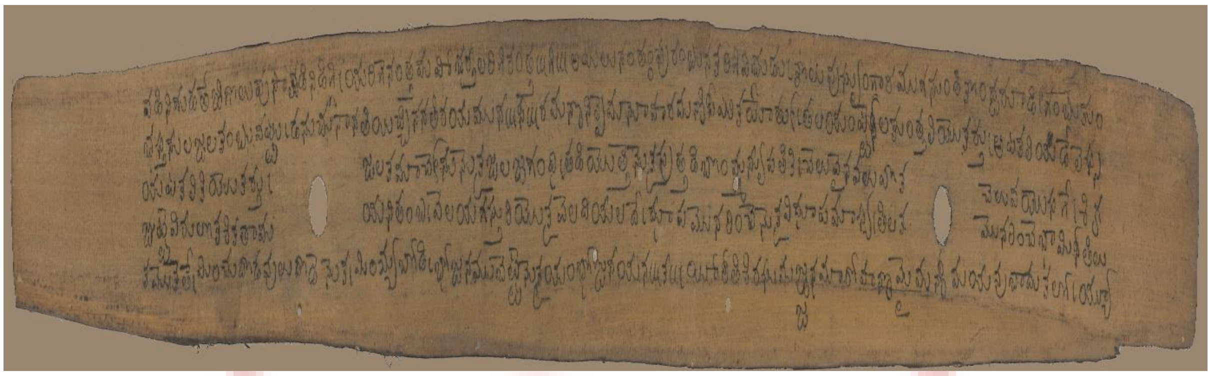
b) BCE Matrics



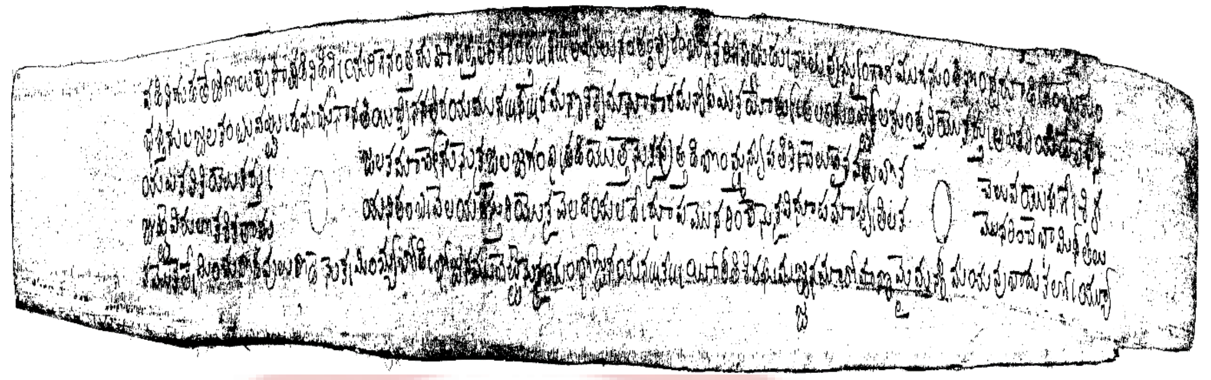
Model	F-Measure	FPS	DRD	PSNR	IOU	Dice
Unet	0.876777	0.896883	0.244064	54.9484	0.782049	0.876777
UnetPlusPlus	0.87772	0.896718	0.24229	55.0003	0.783695	0.87772
Linknet	0.876693	0.896864	0.244067	54.9598	0.782021	0.876693
FPN	0.880335	0.896459	0.237911	55.0837	0.787807	0.880335
PSPNet	0.879846	0.896505	0.238646	55.0803	0.787114	0.879846

C) DICE Metrics

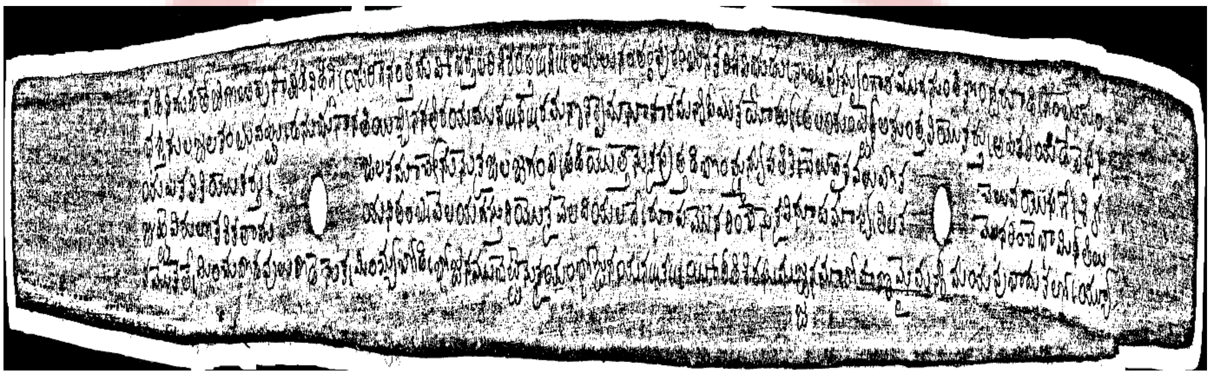
7.3 Traditional Algorithms Results



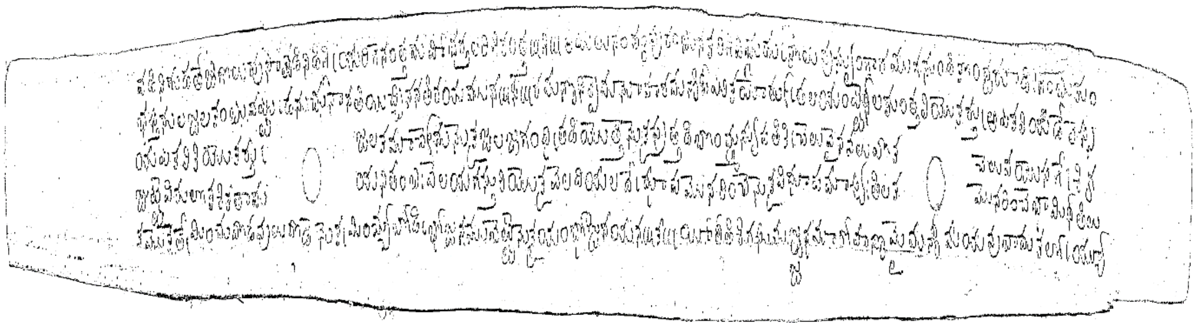
a) Original Image



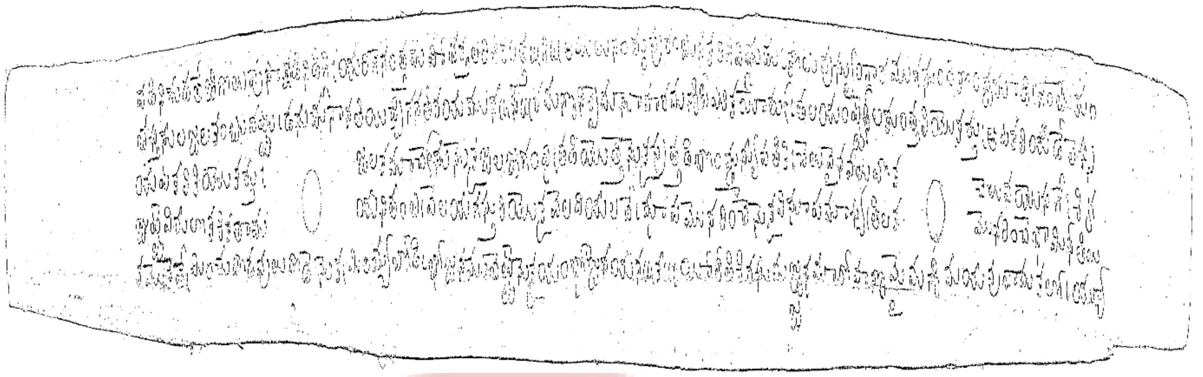
b) OTSU Algorithm Output



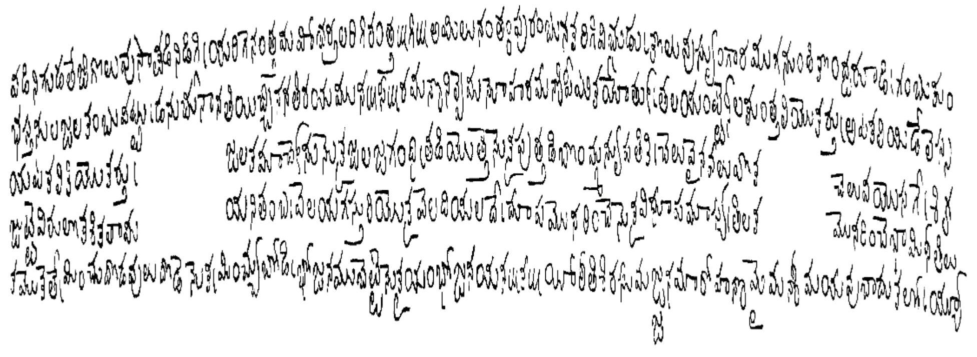
c) Niblack Output



d) Savoula Algorithm Output

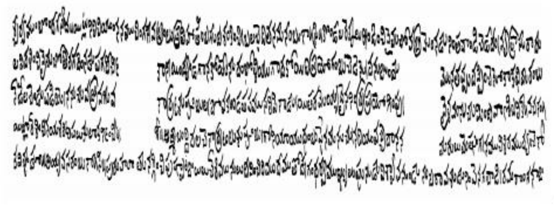


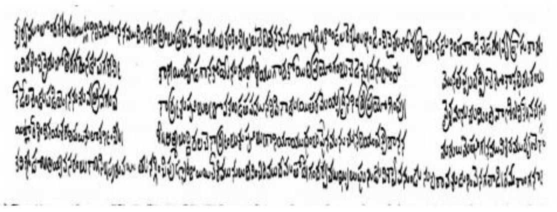
e) SU Algorithm Output

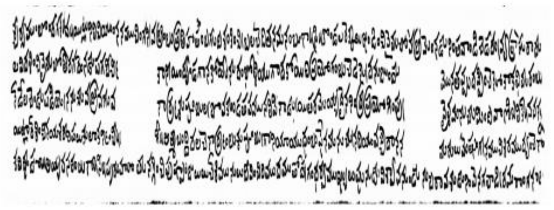


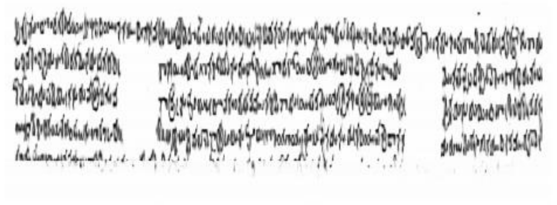
f) Ground Truth Image

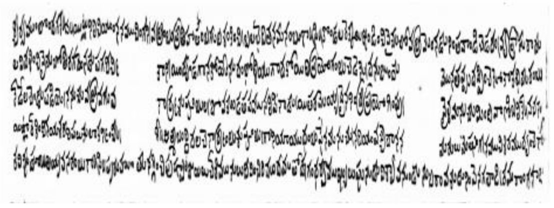
7.4 Deep Learning Algorithm Results

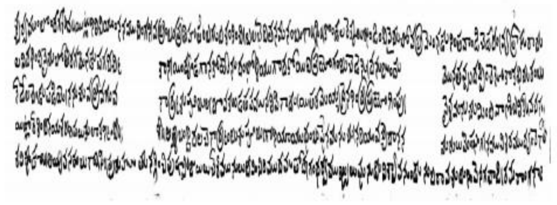

(a) Ground Truth Image

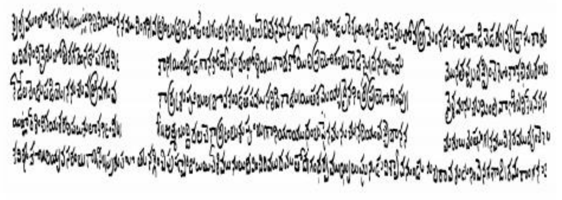

(b) Unet-BCE Loss

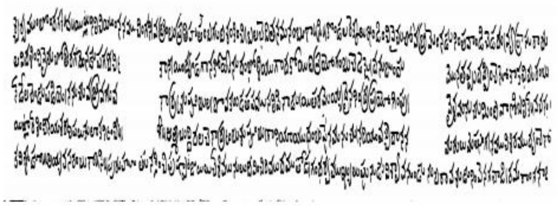

(c) U-Net++ - BCE Loss

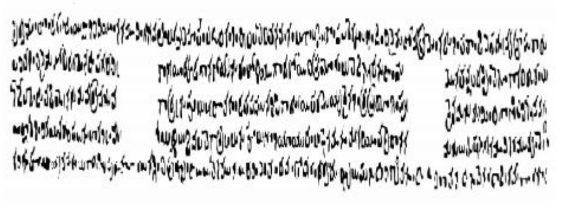

(d) PSP Net - BCE Loss

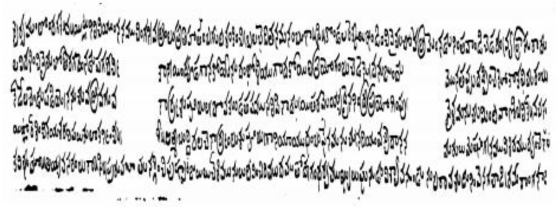

(e) LinkNet - BCE Loss

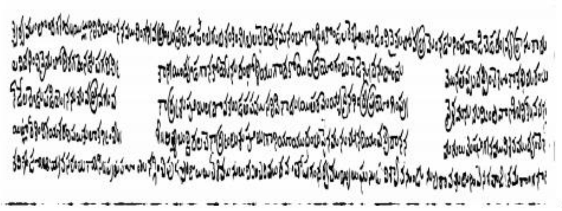

(f) FPN Network - BCE Loss

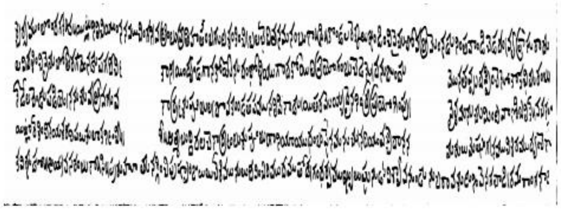

(g) UNet - DICE Loss

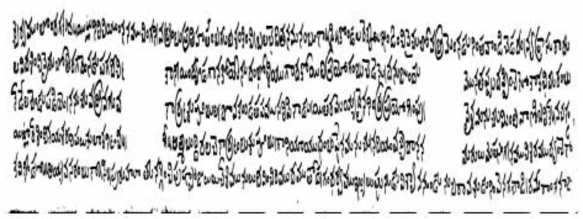

(h) UNet++ - DICE Loss


(i) PSP Net - DICE Loss

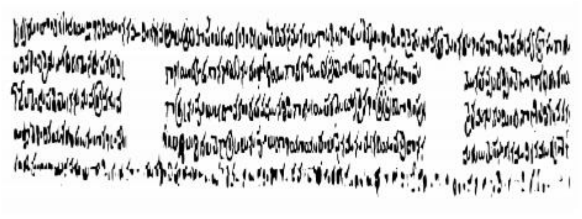

(j) LinkNet - DICE Loss


(k) FPN Network - DICE Loss

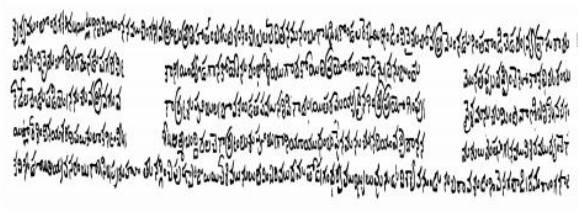

(l) UNet - L1 Loss



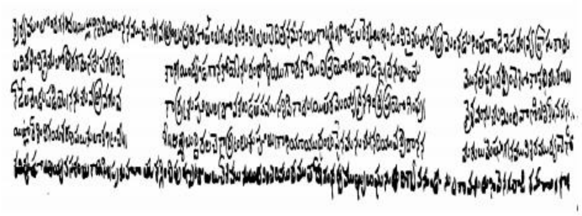
(m) UNet++ - L1 Loss)



(n) PSP Net - L1 Loss



(o) Linknet - L1 Loss)



(p) FPN Net - L1 Loss

Conclusion

This project successfully developed an automated system for accurately binarizing historical manuscript images, with a specific focus on Telugu manuscripts. By leveraging advanced image processing techniques and machine learning algorithms, the project achieved high accuracy in distinguishing between text and background noise, even in degraded and complex document conditions. The implementation of methodologies such as the Otsu method, Niblack's method, and deep learning-based approaches demonstrated the effectiveness of these techniques in enhancing the readability and preservation of historical documents. The trained models and developed algorithms, built using frameworks such as TensorFlow showcased the potential of automated binarization in digitizing and preserving cultural heritage. The project's outcomes hold significant promise for advancing document analysis and preservation practices in libraries, museums, and cultural institutions worldwide.

FUTURE ENHANCEMENTS

1. **Integration with Advanced Preprocessing Techniques**: Develop and integrate more advanced preprocessing techniques to handle complex noise patterns and document degradations, such as stains, bleed-through, and uneven lighting. Techniques like background subtraction, inpainting, and advanced noise filtering can be explored.
2. **Real-Time Binarization**: Implement real-time binarization capabilities for immediate feedback during the digitization process. This can be achieved by integrating the binarization algorithms into mobile applications or scanner software, allowing users to see binarized results instantaneously.
3. **Multimodal Data Fusion**: Combine data from various sources, such as hyperspectral imaging, infrared imaging, and other non-visible spectrum data, to enhance the accuracy of binarization. By leveraging multiple data types, the system can better distinguish between foreground text and background noise.
4. **Transfer Learning and Model Optimization:** Apply transfer learning techniques to leverage pre-trained models from related tasks or large datasets. Fine-tuning these models specifically for historical manuscript binarization can improve performance, especially with limited data. Explore model optimization methods, including neural architecture search, hyperparameter tuning, and regularization techniques.
5. **User-Friendly Interface and Tools**: Develop a user-friendly interface with interactive tools for manual corrections and enhancements. Features such as adjustable thresholding sliders, real-time previews, and annotation tools can help users refine binarized images easily.
6. **Automated Ground Truth Generation**: Create automated or semi-automated methods for generating ground truth data. This can involve using advanced segmentation algorithms to produce initial ground truth images, which can then be refined by human annotators, significantly reducing the time and effort required.
7. **Evaluation and Benchmarking Framework**: Establish a comprehensive evaluation and benchmarking framework for binarization algorithms. This framework should include various metrics to assess the performance of

different algorithms across diverse manuscript types and degradation conditions.

8. **Expansion of Dataset**: Expand the dataset to include more diverse and representative samples of historical manuscripts. This can involve collaborating with libraries, museums, and cultural institutions to digitize and binarize a broader range of documents, enhancing the generalizability of the binarization algorithms.

9. **Adaptive and Context-Aware Binarization**: Develop adaptive binarization techniques that can adjust to different document types and degradation levels dynamically. Context-aware algorithms that can understand and adapt to the specific characteristics of a document, such as text density and ink type, can improve binarization quality.

10.**Crowdsourced and Community-Driven Improvements:** Engage the community in crowdsourced initiatives to improve binarization results. Platforms where users can contribute by refining binarized images, providing feedback, and sharing their binarization techniques can accelerate advancements in the field.

REFERENCES

Conference Papers

Paper1:

https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://arxiv.org/abs/1901.06081&ved=2ahUKEwjAyYzwZaHAxWh2TgGHeGFD_YQFnoECBQQAQ&usg=AOvVaw2NpJjkCYlKl7qg-AD_ktvj

Paper2:
https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://ieeexplore.ieee.org/document/4310076&ved=2ahUKEwjj8db9wZaHAxXB3DgGHcH1A7wQFnoECBEQAQ&usg=AOvVaw32uEmoY-YLN2jBH8fEBTzc

Paper3:

https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://arxiv.org/pdf/1709.01782&ved=2ahUKEwim0MDQwpaHAxVa3TgGHcLYNRsQFnoECCQQAQ&usg=AOvVaw0oRhUrIc1961mWwQlyZPoc