Satellite Image Segmentation for Agriculture Mapping using Mask R-CNN based Auto-encoder

Natnael Dawit

A Thesis Submitted to the Department of Computer Science and Engineering

School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Master's in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

June, 2023

Adama, Ethiopia

Satellite Image Segmentation for Agriculture Mapping using Mask R-CNN based Auto-encoder

Natnael Dawit

Advisor: Dr. Worku Jifara Sori(Ph.D)

A Thesis Submitted to the department of Computer Science and Engineering
School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Master's in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

June, 2023
Adama, Ethiopia

DECLARATION

I hereby declare that this Master Thesis entitled "**Satellite Image Segmentation for Agriculture Mapping using Mask R-CNN based Auto-encoder"** is my original work and has not been submitted any university for similar purpose. The reference used in this thesis and duly recognized by proper citations.

Natnael Dawit	_____	_____
Name of student	Signature	Date

RECOMMENDATION

I, the major advisor of this research thesis, hereby certify that I have closely advised the student while developing this thesis and read the draft thesis entitled "**Satellite Image Segmentation for Agriculture Mapping using Mask R-CNN based Auto-encoder**" prepared under my guidance by Natnael Dawit. Therefore, I recommend the submission of the thesis to the department for further review and evaluation.

Dr Worku jifara (Ph.D)	_____	_____
Major advisor	Signature	Date

APPROVAL PAGE

We, the undersigned, members of the Board of Reviewers of the thesis open defence by **Natnael Dawit** have read and evaluated his thesis entitled "**Satellite Image Segmentation for Agriculture Mapping using Mask R-CNN based Auto-encoder**" and assessed the understanding of thesis is accepted and we recommend the implementation of the thesis.

_____	_____	_____
Chairperson	Signature	Date
_____	_____	_____
Internal Examiner	Signature	Date
_____	_____	_____
External Examiner	Signature	Date

Final approval and acceptance of the thesis is contingent upon submission of its final copy to the Office of Postgraduate Studies (OPGS) through the Department Graduate Council (DGC) and School Graduate Committee (SGC).

_____	_____	_____
Department Head	Signature	Date
_____	_____	_____
School Dean	Signature	Date
_____	_____	_____
Office of Postgraduate Studies, Dean	Signature	Date

ACKNOWLEDGMENT

All praise and gratitude are due to Almighty GOD, who is the most kind and benevolent being, who endowed me with skills and enabled me to finish this thesis. In addition, I want to express my gratitude to Adama Science and Technology, School of Electrical Engineering and Computing for providing me with the opportunity to advance my professional expertise. Likewise, the Department of Computer Science and Engineering gave me the opportunity to explore topics, which allowed me to complete my thesis.

I want to express my sincere gratitude to **Dr. Worku Jifara**, who served as my advisor and supported me with my work. His helpful suggestions, wise leadership, and insightful criticism of my thesis work are immeasurable. My technical and non-technical skills were improved by suggestions, which allowed me to write my research with assurance.

Moreover, several friends and coworkers have offered me valuable advice and constructive criticism while I completed this thesis. All names are registered in my memory but are not included here. I appreciate all your support and insightful comments. Furthermore, I want to thank my instructors for their support and for helping me understand the course material.

Last but not least, I would want to extend my gratitude to my dear family for their prayers and to my friends for their encouragement and unending love in helping me accomplish my goals.

Natnael Dawit

June 2023

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SAMPLE CODE

LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
AP	Average Precision
BN	Batch Normalization
CAE	Convolutional Auto-Encoder
CNN	Convolutional Neural Network
DAE	Denoising Auto-Encoder
FCN	Fully Convolutional Network
GAN	Generative Adversarial Network
GIS	Geographic Information Systems
IOU	Intersection Over Union
mIOU	Mean Average Precision
N2V	Noise2Void
R-CNN	Region- Convolutional Neural Network
ROI	Region of Interest
RPN	Region Proposal Network
VAE	Variational Auto-Encoder
VIA	VGG Image Annotator

ABSTRACT

This research study focuses on agricultural mapping, which involves the use of satellite techniques to map and monitor agricultural lands, crops, and vegetation without physical contact with the Earth's surface. Satellite imaging, aerial photography, and other satellite methods are employed to gather information about crop growth patterns, soil moisture, and other environmental factors influencing crop health and yield. However, obtaining high-quality satellite images for agricultural mapping is challenging due to the introduction of noise during image acquisition. In this thesis, we propose a novel approach called Satellite Image Segmentation for Agriculture Mapping using Mask R-CNN based Auto-encoder. The proposed method consists of two components: an auto-encoder for image denoising and a Mask R-CNN for region-based segmentation. The auto-encoder comprises an encoder that encodes the image into a latent space representation and a decoder that reconstructs the image from the latent space. RGB images are utilized in the model, and the Mask R-CNN is integrated with the decoder of the auto-encoder. Furthermore, the "blind-spot" training technique known as Noise2Void is adopted for data augmentation in image segmentation. This technique encourages the neural network to learn how to denoise the image based on the available information instead of memorizing the training data. The simulation results demonstrate that the proposed Mask R-CNN approach achieves higher accuracy and prediction in agricultural mapping.

Keywords*: Mask R-CNN, auto-encoder, satellite image, agricultural mapping, RGB,* N2V

CHAPTER ONE

INTRODUCTION

1.1 Background of the study

Satellite is the acquisition of information about earth surface via measuring its reflectance and emittance radiation from longer distances (Campbell & Wynne, 2011). Image employed in satellite can be either RGB, Multispectral or Hyperspectral images which can be analyzed and used in many application area (van der Meer et al., 2012). Agriculture mapping is the process of making geographical maps that reflects land distribution of Agricultural production by analyzing satellite images (Huang et al., 2018). Agriculture map is used to show distribution of crops and livestock, crop status for precision agriculture, livestock productivity, and specific agronomic maps (Huang et al., 2018). Detection, classification, segmentation and recognition are some of the ways to analyze satellite images for agricultural imaging (García-Pedrero et al., 2017).

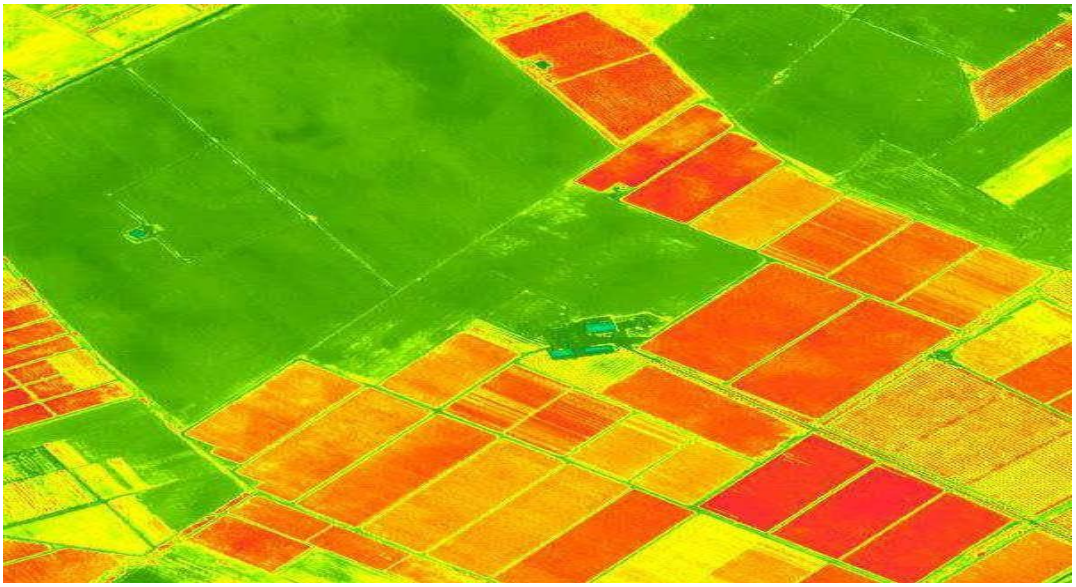


Figure 1. 1 Agricultural map of farming land distribution

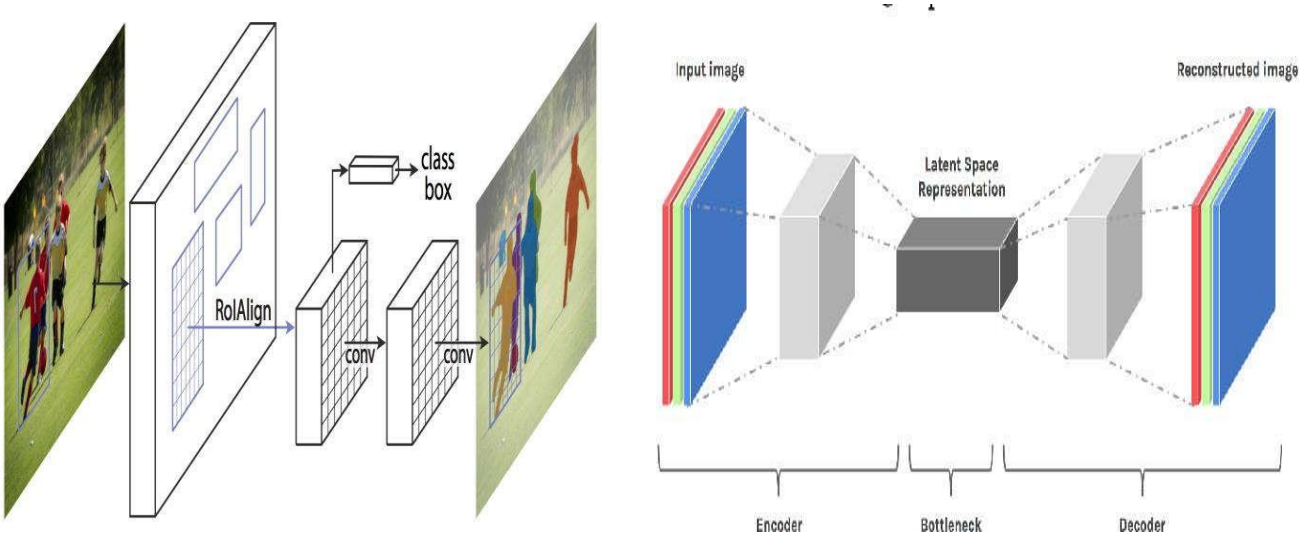
Image segmentation is the process of segmenting an image in to different segments or set of pixels based on different criteria which are region based, edge detection and clustering (Minaee et al., 2022). Region based segmentation (Yu et al., 2018) is separating objects on the image in to different regions based on some threshold values. Edge detection Segmentation (Vardhana et al., 2018) is defining a boundary of objects

using edges and Unsupervised or Clustering (Lei et al., 2020) based

Segmentation is dividing the pixels of the images in homogeneous clusters. There are several deep learning methods for image segmentation and Mask R-CNN (He et al., 2020) is one of them. This model is an extension of faster R-CNN which is a region based Convolutional Neural Network that returns bounding box for each object on the image and class label for the image. Mask R-CNN adds third layer to the original one which segment the image based on region of interest ROI.

Segmentation of noisy images is one of the most challenging problems in image analysis and any improvement of segmentation methods can highly influence the performance of many image processing applications (Despotović et al., 2010)(García-Pedrero et al., 2017). To overcome this challenge different deep learning method were proposed and Auto- encoders (Lopez Pinaya et al., 2019) is one of them. Auto-encoders has two components which are encoders and decoders. The encoder encodes the input image to a latent space representation whereas the decoder decodes or construct the latent space to the image form. The noisy features of images can be discarded during decoding processes in the Auto-encoders. Auto-encoders is used to de-noising the image at the end of the decoder.

In this teases, Mask R-CNN is used to segment the image and Auto-encoders is used to de-noising the image (Minkesh et al., 2019). RGB images are used in the model and Mask R-CNN is combined with the decoder of the auto- encoder.



a) Mask R-CNN b) Autoencoders

Figure 1. 2 Mask R-CNN and Autoencoders

1.2 Statement of the problem

The lack of high-quality satellite agricultural images due to introduction of noise to the image during image acquisition is a major challenge in the image segmentation of the agricultural image for the purpose of agricultural mapping. Even though there are numerous non-deep learning image de-noising methods (Wagner & Oppelt, 2020)(Badrinarayanan et al., 2017) for the image segmentation models, their accuracy and performance is lower than deep learning based de- noising methods which is a major problem in most image segmentation methods for agricultural mapping. There are prior works that uses deep learning based de-noising method before image segmentation. However, image segmentation model and de-noising model are two separate network model which is tedious to train the models (Cho et al., 2020)(Karimpouli & Tahmasebi, 2019). So there should be one deep learning model that is used for both image de-noising and image segmentation for the agricultural mapping.

1.3 Research Questions

Based on the problem stated above, this research aims to answer the following research questions to design and implement the solution.

RQ1. How can image segmentation be used to map resources in agricultural satellite images?

RQ2. How to design a deep learning model that can segment agricultural satellite images after denoising them?

RQ3. How can we improve deep learning models by integrating denoising and segmentation methods in one deep learning model?

1.4 Objectives of the study

1.4.1. General Objective

This research's main objective is to design and evaluate Mask Region based Convolutional Neural Network based Auto-encoder model for denoising a satellite agricultural image and segment it based on the region of interest for agricultural mapping.

1.4.2. Specific Objectives

The following are specific objectives addressed to achieve the general objective.

- To design segmentation model that used to map agricultural resources on agricultural satellite image.

- To evaluate the effectiveness of Mask R-CNN based Auto-encoder for image segmentation with the de-noising effect.
- To evaluate the performance of Mask R-CNN based auto-encoder with other image segmentation methods for agricultural mapping.

1.5 Scope and Limitation

The scope of this research is to use Mask R-CNN based Auto-encoders in order to segment satellite images for agricultural mapping. The research only focuses on de-noising the image before segmentation and segment the image based on region of interest. The image used would be an agricultural satellite image.

The limitation of this research will be the lack of available Ethiopian based agricultural mapping satellite image datasets in order to show the research work in the case of Ethiopian based data sets. It is expensive and time consuming to collect and preprocess local based datasets for this research.

1.6 Significance of the study

The proposed model will be used for mapping agricultural resources from fields using satellite images by segmenting the image in to regions where these resources are located. Such model can be used in mapping water resources in the agricultural fields, mapping different kinds of crops for precision agriculture and harvest forecasting. Number of live stocks on the fields can be mapped via this proposed model. The proposed model will be used to able stockholders to make decision on the agricultural resource by mapping agricultural resources using image segmentation of satellite images.

1.7 Organization of the Thesis

The rest section of this thesis is structured as follows:

Chapter Two: Describes an outlines of auto encoders literature and corresponding works concerning their types and applications. It also covers about agricultural mapping, image segmentation and Mask R-CNN.

Chapter-Three: Describe study methods such as dataset collected technique, dataset pre- processing methods, tools for designing, and the evaluation approaches.

Chapter Four: Describe the architecture of the proposed model in detail.

Chapter Five: Describe the proposed model implementation with the setting of a working configuration,

implantation techniques for dataset processing, model implementation, described using snip code and visualization graph for the model summary.

Chapter Six: Present the expected proposed model key results, compare the new outcome with the latest baseline models, and describe the significance of the results.

Chapter Seven: The concluding part explains the recaps of the entire work, a summary of the result, and gives directions to the potential of work in the future.

CHAPTER TWO

LITERATURE REVIEW AND RELATED WORKS

2.1. Agricultural Mapping

Agriculture plays a vital role in our economy, necessitating a comprehensive understanding of field conditions and their global implications. For years, the agricultural industry has relied on maps to optimize crop cultivation. These maps aid in the strategic placement of crops, determining appropriate planting quantities, and estimating potential profitability. To effectively utilize farming maps, farmers must possess the ability to interpret the information they provide. These maps illustrate diverse soil types and their corresponding fertility levels, as well as the various vegetation types present in a given area. This valuable information empowers farmers to identify optimal field locations and make informed decisions regarding crop selection based on soil characteristics and fertility levels. Moreover, a high-quality agricultural map should exhibit clarity, accuracy, detail, and up-to-date information.

The benefits of agricultural mapping are manifold:

- Landowners require precise knowledge of their property boundaries and neighboring farms. Such information enables them to identify potential natural resources on their land, such as water sources or mineral deposits.
- Governments rely on accurate agricultural land location data to facilitate the planning and provision of public services, including schools and hospitals.
- Governments also seek to ascertain the economic contributions of farming, allowing them to allocate appropriate resources for future industry improvements.
- Agricultural mapping allows farmers to compare their operations with neighboring farms, facilitating an understanding of which crops are most suitable for the local environment.
- The utilization of agricultural maps aids in the strategic planning of crop rotations and enables farmers to determine the most profitable crops to cultivate per acre of available land.

2.1.1. GIS maps for agriculture

Geographic Information Systems (GIS) maps have gained increasing significance in the agricultural industry, providing farmers and agribusinesses with invaluable insights into their land, crops, and livestock. These maps serve as crucial decision-making tools for farmers, enabling them to optimize land utilization. By utilizing GIS maps, farmers can identify the most suitable areas for specific crops, determine optimal planting and harvesting times, and ensure sufficient water availability for their crops throughout the growing season.

In addition to land management, GIS maps in agriculture offer valuable opportunities for strategic planning, including the development of new businesses or products aimed at improving profitability. For instance, farmers seeking to transition from pasture-based cattle raising to feedlots can rely on GIS maps to identify suitable locations for feedlots in proximity. Agribusinesses similarly utilize GIS maps to determine optimal farm placement and assess the profitability of specific crops in different regions.

To enhance farming efficiency, farmers often invest in technologies like GPS systems and other GIS mapping tools that enable them to maximize their land's potential. Various types of GIS maps specifically designed for agriculture are employed by farmers and ranchers to streamline their operations effectively. These include land use maps, which delineate different types of land utilization such as forests, grasslands, and wetlands. Moreover, land use maps can highlight areas where specific crops or livestock are consistently cultivated, aiding farmers in optimizing land usage. Soil maps are another essential GIS resource, providing information about soil types, characteristics (e.g., color, pH), and other pertinent factors. This data enables farmers to identify which crops thrive in specific soil conditions, facilitating informed decision-making regarding crop selection and cultivation practices.

GIS maps for agriculture can be used in many ways:

- To monitor plant growth, crop maturity, and soil conditions as they relate to weather patterns.
- To keep track of livestock, so you know where they are at all times and whether they need feeding or medical attention.
- To create maps that show where crops are located on your property, so you can avoid accidentally spraying them with pesticides or herbicides.

- To show how much water is available around your farm or ranch. You can see whether there are any rivers or lakes nearby that could provide water for your animals during times when there isn't enough grass available for them to eat in dry months.
- To reduce expenses and improve crop yields.

Agricultural mapping plays a crucial role in enhancing the efficiency of the agricultural system, offering significant benefits to farmers. By identifying suitable areas for crop cultivation and other agricultural products, farmers can enhance their income sources and ensure the availability of resources to produce an ample supply of food for urban populations. GeoPard has developed a comprehensive platform that empowers farmers with access to a wide range of farm-related information, including current farm conditions and crop yields. This valuable data is securely stored on a cloud server, enabling users to conveniently access it through smartphones or laptops without the need for physical field inspections.

GeoPard is an advanced and user-friendly software solution specifically designed to meet the data collection and processing requirements of farmers and agribusinesses. Its versatile capabilities cater to the needs of agricultural professionals working in diverse environments, be it on farms, in fields, or in offices. With GeoPard, users can effortlessly generate informative agricultural maps by incorporating various data layers such as crop types, fertility levels, and yield estimates. The software also offers a range of powerful tools for soil map analysis, aiding in the identification of areas with low fertility or other underlying issues. Through GeoPard, farmers can leverage the power of technology to optimize their agricultural practices and make informed decisions for enhanced productivity.

The utilization of GIS applications in agriculture has demonstrated a significant impact on crop production worldwide, aiding farmers in enhancing productivity, reducing costs, and improving land resource management (Kang et al., 2023). Among these applications, agricultural mapping assumes a crucial role in monitoring and managing soil conditions and irrigation practices on farmland. By integrating accurate information into a mapping environment, agricultural mapping serves as an indispensable tool for the effective management of the agricultural sector. Moreover, it facilitates the control and utilization of agricultural resources, contributing to the improvement of existing systems for acquiring and generating GIS data pertaining to agriculture and its associated resources (Lamb & Brown, 2001). Agricultural mapping employs various types of data, including spatial, spectral, and temporal information, to map and analyze agricultural resources. RGB, Hyperspectral, and Multispectral images are frequently employed in this

context. Image classification, image segmentation, change detection, and clustering techniques are utilized to analyze these images and effectively map agricultural resources.

The following lists are some of the Agricultural mapping applications. Crop mapping yield estimation, Crop assessment and Crop health, Irrigated landscape mapping, Application development for GIS agriculture, Soil and irrigation amendment analysis, Suitability assessment studies, Erosion identification and remediation, Agricultural mapping for detailed vegetation cover and monitoring, change detection studies and developing crop models, Damage and land degradation assessment studies, Elevation models for efficient drainage.

2.2. Image segmentation

Image segmentation is a fundamental task in computer vision that involves dividing an image into distinct segments or regions to identify objects and boundaries. It can be viewed as a pixel-level classification problem, where each pixel is assigned a specific label. Over the years, a plethora of image segmentation algorithms has been proposed in the literature. Early techniques encompassed thresholding, histogram-based bundling, region growing, k-means clustering, and watershed algorithms. In more recent years, advanced algorithms like active contours, graph cuts, conditional and Markov random fields, as well as sparsity-based methods have emerged, offering enhanced segmentation capabilities. These algorithms have contributed to the development of robust and accurate techniques for image analysis and object recognition in various applications.

Recent advancements in image segmentation algorithms have witnessed the emergence of deep learning models that exhibit superior performance and accuracy compared to traditional approaches. One notable contribution to this field was made by (Han et al., 2019), who introduced a pioneering deep learning framework for semantic image segmentation, employing a fully convolutional network (FCN). By exclusively utilizing convolutional layers, the FCN possesses the unique capability to process images of varying dimensions and generate segmentation maps of matching sizes. To achieve this, the authors modified existing CNN architectures, such as VGG16 and GoogLeNet, by replacing all fully-connected layers with fully-convolutional layers. This adaptation effectively facilitated the handling of input and output of non-fixed sizes.

In their work, the authors (Badrinarayanan et al., 2017) introduced a convolutional encoder-decoder architecture designed for image segmentation. The segmentation engine in SegNet consists of an encoder network, which shares the same topology as the VGG16 network's 13 convolutional layers, and a

corresponding decoder network followed by a pixel-wise classification layer. A key innovation of SegNet lies in its decoder's approach to up sampling the lower-resolution input feature maps. Instead of learning to up sample, SegNet utilizes pooling indices generated during the max-pooling step of the corresponding encoder for non-linear up sampling. This technique eliminates the need for additional learning. The sparse up sampled maps are subsequently convolved with trainable filters to generate dense feature maps. Compared to other competing architectures, SegNet achieves a significantly reduced number of trainable parameters. The authors also proposed a Bayesian variant of SegNet, which incorporates a probabilistic model to capture the inherent uncertainty of the convolutional encoder-decoder network in scene segmentation tasks.

The Regional Convolutional Network (R-CNN) and its subsequent variations, such as Fast R-CNN, Faster R-CNN, and Mask R-CNN, have emerged as successful approaches for object detection in various applications. Notably, the Faster R-CNN architecture, as described in (Rampersad, 2020), introduces a region proposal network (RPN) that effectively generates candidate bounding boxes. By extracting Regions of Interest (RoIs) using the RPN, and subsequently utilizing a RoIPool layer, this architecture computes features from the proposed regions to accurately determine both the bounding box coordinates and object class. Furthermore, certain extensions of the R-CNN framework have gained significant attention in addressing the instance segmentation problem, which involves simultaneous object detection and semantic segmentation (Tian et al., 2020).

An advanced model called Mask R-CNN was introduced by (He et al., 2020) as an extension of the existing framework. This model demonstrated superior performance compared to previous benchmarks in numerous COCO challenges, particularly in the area of object instance segmentation. Mask R-CNN offers a streamlined approach to object detection, simultaneously providing accurate bounding box coordinates and generating high-quality segmentation masks for each detected instance. Essentially built upon the Faster RCNN architecture, Mask R-CNN incorporates three output branches: one for computing bounding box coordinates, another for determining associated classes, and a third for generating binary masks to segment the objects. To ensure optimal performance, the Mask R-CNN model employs a combined loss function that simultaneously trains the bounding box coordinates, predicted classes, and segmentation masks in a joint manner.

A more advanced application of image annotation is segmentation. This method can be used in many ways to analyze the visual content in images to determine how objects within an image are the same or different. It also can be used to identify differences over time.

In the field of image segmentation, there exist three distinct types:

a) Semantic segmentation aims to delineate boundaries between similar objects and assign them with a consistent label. This method is employed when the goal is to identify the presence, location, and occasionally, the size and shape of objects. Semantic segmentation is suitable for grouping objects together, but it may not provide information regarding object count or tracking across multiple images. For instance, in annotating images of a baseball game encompassing both the stadium crowd and the playing field, semantic segmentation can be used to separate the seating area from the field.

b) Instance segmentation entails tracking, counting, and precisely delineating the presence, location, count, size, and shape of objects within an image. This form of image annotation is also known as object classification. To illustrate, in the aforementioned baseball game scenario, each individual in the stadium could be labeled and instance segmentation applied to determine the crowd size.

Both semantic and instance segmentation can be conducted through pixel-wise segmentation, whereby each pixel inside the object's outline is labeled, or boundary segmentation, where only the border coordinates are considered.

c) Panoptic segmentation combines semantic and instance segmentation, providing labeling for both the background (semantic) and objects (instance). This approach proves useful in various applications, such as utilizing satellite imagery to detect changes in protected conservation areas. By employing panoptic segmentation, scientists can track changes in tree growth and health, assessing the impact of events like construction or forest fires on the area.

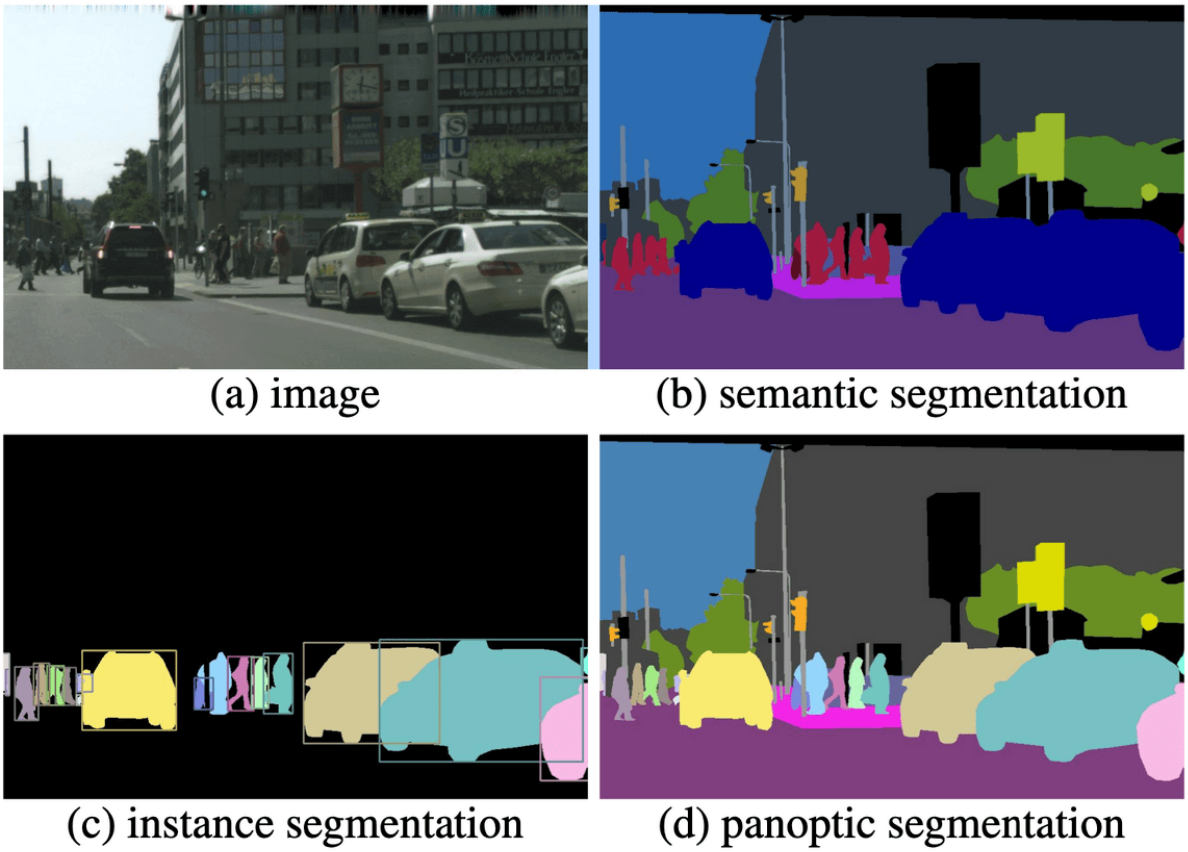


Figure 2. 1 (a) is the original image, and the others show three kinds of segmentation that can be applied in image annotation. In this example, the objects of interest are the cars and the people.

2.3. Auto – Encoders

Autoencoders were initially introduced as a neural network architecture designed to reconstruct input data, aiming to learn an unsupervised representation of the data that can be applied to various tasks, including clustering (Lopez Pinaya et al., 2019).

An autoencoder is a specific type of artificial neural network that learns to encode data efficiently in an unsupervised manner. The primary objective of an autoencoder is to acquire a representation, or encoding, for a given dataset by training the network to filter out noise signals. In addition to the encoding phase, the autoencoder also learns a decoding phase, where it attempts to generate a representation that closely resembles the original input based on the reduced encoding. Different types of autoencoders exist depending on their specific applications. For instance, if the input data consists of images, the encoder and decoder components of the autoencoder are typically composed of convolutional and deconvolutional layers,

respectively, resulting in what is referred to as a convolutional autoencoder. Alternatively, if the autoencoder is employed to remove unwanted noise signals from input images, it is called a denoising autoencoder (DAE).

Autoencoders, , are straightforward learning circuits designed to minimize the distortion between inputs and outputs during transformation. Despite their conceptual simplicity, they hold significant importance in the field of machine learning. Originally introduced in the 1980s by Hinton and the PDP group, autoencoders were devised as a solution to the challenge of "backpropagation without a teacher," using the input data itself as the supervisory signal. In conjunction with Hebbian learning rules, autoencoders constitute a fundamental paradigm for unsupervised learning, offering insights into the coordination of synaptic changes triggered by local biochemical events to achieve self-organized global learning and intelligent behavior (Plath et al., 2009).

Autoencoders have found applications in various domains, such as facial recognition, feature detection, anomaly detection, and semantic representation of words (Wang et al., 2021). . In addition to their discriminative capabilities, autoencoders also serve as generative models, allowing for the random generation of new data that closely resembles the training data (Roscher et al., 2020). Authors in (Cheng et al., 2020) propose that autoencoders possess the ability to compress high-dimensional features by minimizing a cost function that typically comprises a reconstruction error term and a regularization term. Through the utilization of gradient descent and backpropagation, autoencoders can effectively learn the network parameters required for efficient data representation.

2.3.1. Types of Autoencoders

There are basically seven types of autoencoders:

Denoising autoencoder: - is an extension of the traditional autoencoder that addresses the issue of the network learning the identity function, wherein the output simply replicates the input. This situation can occur when the autoencoder is excessively large, leading to the network merely memorizing the training data without achieving meaningful representation learning or dimensionality reduction. To overcome this challenge, denoising autoencoders deliberately introduce corruption to the input data by adding noise or masking certain input values (Ye et al., 2022). his deliberate corruption helps promote the discovery of robust and informative representations within the autoencoder framework.

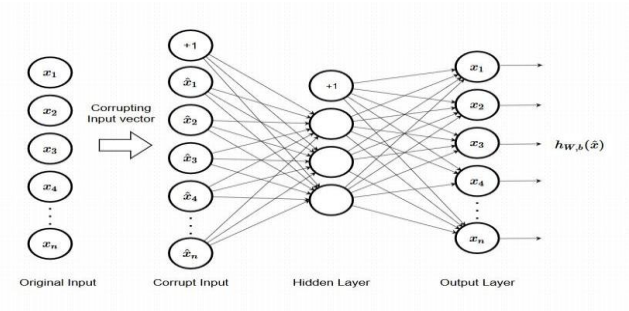


Figure 2. 1 De-noising auto encoder example

Sparse Autoencoder: is a specific variant of the autoencoder architecture that utilizes sparsity to create an information bottleneck. In this approach, the loss function is designed to penalize excessive activations within a layer, thereby encouraging sparse representations. This sparsity constraint can be achieved through various methods, such as incorporating L1 regularization or using KL divergence to measure the difference between the expected average neuron activation and an ideal distribution (Lopez Pinaya et al., 2019).

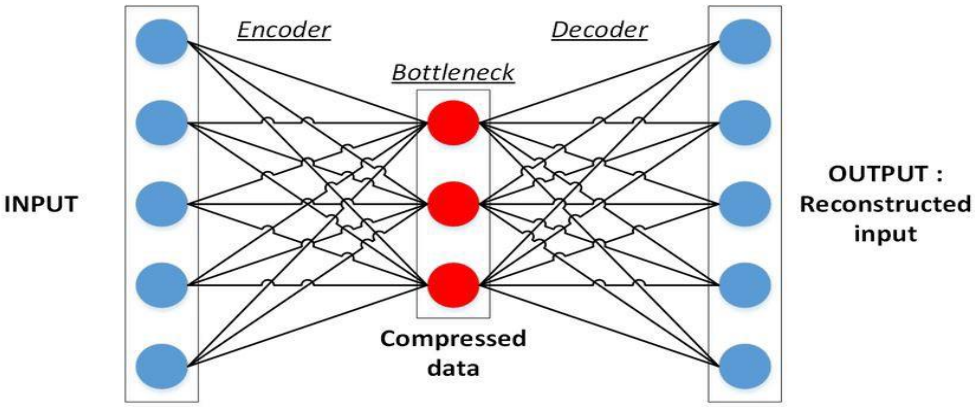


Figure 2. 2 Sparse auto encoder example

$$\arg\min_{A,B} E[\Delta(x, B \circ A(x)] + \lambda \sum i\,|ai\,|,$$

a) **Deep Autoencoder: -** is a specialized architecture consisting of two symmetrical deep-belief networks with multiple shallow layers, typically ranging from four to five. These networks are divided into an encoding half and a decoding half. The increased number of layers in the Deep Autoencoder enables it to capture and learn more intricate and complex features compared to a simple autoencoder. Each layer in the Deep Autoencoder is constructed using restricted Boltzmann machines, which serve as fundamental building blocks within deep-belief networks (Vardhana et al., 2018)(Li et al., 2016).

b) Contractive Autoencoder: - is an unsupervised deep learning method utilized for encoding unlabeled training data (Lopez Pinaya et al., 2019) . While denoising autoencoders focus on enhancing the encoder's resilience to input perturbations, contractive autoencoders prioritize reducing the sensitivity of feature extraction to minor perturbations. This is achieved by compelling the encoder to disregard insignificant input variations that do not significantly affect the reconstruction process performed by the decoder. Consequently, a penalty is imposed on the Jacobian matrix of the network to enforce this behavior. The Jacobian matrix of the hidden layer h consists of the derivative of each node h_j with respect to each value x_i in the input x. Formally: $J_{ji} = \frac{\partial h_j}{\partial x_i}$. In contractive autoencoders we try to minimize its L2 norm, such that the overall optimization loss would be:

$$\arg\min_{A,B} E\left[\Delta\left(x, B \circ A(x)\right) + \lambda \|J_A(x)\|_2\right]$$

c) Under complete Auto encoder: An under complete auto encoder refers to a model that lacks an explicit regularization term and is trained solely based on the reconstruction loss. To prevent the model from memorizing the input data, it is crucial to restrict the number of nodes in the hidden layer(s) adequately (Lei et al., 2020).

d) Convolutional Autoencoder: - is a type of autoencoder that learns to encode input data into a set of simple signals and reconstructs the original input from these signals. CAEs offer the flexibility to modify the geometry or generate the reflectance of an image. In this architecture, the encoder layers are referred to as convolution layers, while the decoder layers are known as deconvolution layers or up-sampling layers, sometimes also called transpose convolution layers (Badrinarayanan et al., 2017).

e) Variational Autoencoder. : is a type of autoencoder that exhibits the ability to generate new images, similar to Generative Adversarial Networks (GANs). However, VAE models typically rely on strong assumptions regarding the distribution of latent variables. These models employ a variational approach for learning latent representations, introducing an additional loss component and utilizing a specific estimator known as the Stochastic Gradient Variational Bayes estimator for the training algorithm. In comparison to a standard autoencoder, VAEs often achieve a closer match between the probability distribution of the latent vector and the training data. Due to their enhanced flexibility and customizable generation behavior, VAEs are well-suited for various art generation purposes (Lopez Pinaya et al., 2019)(Gates, n.d.).

2.3.2. Application of autoencoders

Various types of autoencoders have been developed, each excelling in specific tasks. Autoencoders can be applied in several ways, including:

Data Compression: While autoencoders are theoretically suitable for data compression, they are not commonly utilized for this purpose in practical scenarios. The primary reasons for this are:

Lossy Compression: Autoencoders generate outputs that are similar to the input but are degraded representations. Therefore, they are not ideal for lossless compression.

Data-Specific: Autoencoders can effectively compress data that is similar to the training data they were trained on. They learn specific features from the training data, distinguishing them from standard data compression algorithms like JPEG or GZIP. Consequently, an autoencoder trained on handwritten digits would not be suitable for compressing landscape photos.

As more efficient and specialized compression algorithms such as JPEG, LZMA, and LZSS (used in WinRAR alongside Huffman coding) already exist, autoencoders are generally not employed for compression purposes. However, in recent years, autoencoders have found utility in image denoising and dimensionality reduction tasks, demonstrating their potential in these domains.

- Image Denoising: Autoencoders are very good at denoising images. When an image gets corrupted or there is a bit of noise in it, we call this image a noisy image. To obtain proper information about the content of the image, we perform image denoising.
- Dimensionality Reduction: In statistics, machine learning, and information theory, dimensionality reduction, or dimension reduction is the process of reducing the number of random variables under consideration (Lopez Pinaya et al., 2019) by obtaining a set of principal variables. Approaches can be divided into feature selection and feature extraction.

The autoencoders convert the input into a reduced representation which is stored in the middle layer called code. This is where the information from the input has been compressed and by extracting this layer from the model, each node can now be treated as a variable. Thus, we can conclude that by trashing out the decoder part, an autoencoder can be used for dimensionality reduction with the output being the code layer.

- Feature Extraction: Encoding part of Autoencoders helps to learn important hidden features present in the input data, in the process to reduce the reconstruction error. During encoding, a new set of combinations of original features is generated.
- Image Generation: Variational Autoencoder (VAE) discussed above is a Generative Model, used to generate images that have not been seen by the model yet. The idea is that given input images like images of face or scenery, the system will generate similar images. The use is to:
    - ✓ generate new characters of animation
    - ✓ generate fake human images

Image colorization: One of the applications of autoencoders is to convert a black and white picture into a colored image. Or we can convert a colored image into a grayscale image

2.4. Mask R-CNN

In recent years, significant advancements have been made in the field of computer vision, particularly in object detection and semantic segmentation. These improvements have been primarily driven by the emergence of powerful baseline systems, such as the Fast/Faster RCNN (Lei et al., 2020) for object detection and the Fully Convolutional Network (FCN) (Han et al., 2019) for semantic segmentation. These frameworks have contributed to notable progress by offering intuitive concepts, flexibility, robustness, and efficient training and inference times. However, in our research, we aim to develop an equally empowering framework for instance segmentation. Instance segmentation is a challenging task that necessitates both accurate object detection and precise instance segmentation within an image. It combines elements from the traditional computer vision tasks of object detection, which involves classifying individual objects and localizing them using bounding boxes, and semantic segmentation, which entails classifying each pixel into predefined regions of interest (RoI) aligned with bounding boxes. Our objective is to address these challenges and devise a framework that excels in instance segmentation.

The Mask R-CNN framework is widely used for instance segmentation, which involves distinguishing object instances within a predefined set of categories. Achieving high-quality results in instance segmentation may seem challenging, suggesting the need for complex methods. However, our research demonstrates that a remarkably straightforward, adaptable, and efficient system can outperform previous state-of-the-art instance segmentation approaches. Our approach, known as Mask R-CNN, builds upon the Faster R-CNN framework

(He et al., 2020) by incorporating an additional branch that predicts segmentation masks for each Region of Interest (RoI). This branch operates in parallel with the existing branches responsible for classification and bounding box regression (see Figure 1).

The mask branch in Mask R-CNN utilizes a compact Fully Convolutional Network (FCN) to generate a segmentation mask for each Region of Interest (RoI) at a pixel level. This approach, integrated into the Faster R-CNN framework, offers simplicity in implementation and training, enabling the exploration of diverse architectural designs. Moreover, the computational overhead introduced by the mask branch remains minimal, ensuring a fast system and facilitating rapid experimentation. However, it is crucial to construct the mask branch effectively to achieve satisfactory results. Notably, the Faster R-CNN architecture was not originally designed to establish pixel-to-pixel alignment between network inputs and outputs, as evident in the coarse spatial quantization performed by the commonly used RoI Pool operation for feature extraction (Milioto et al., 2018)(Lei et al., 2020). To address this misalignment, we propose an innovative and quantization-free layer, known as RoI Align, which accurately preserves the exact spatial locations of RoIs.

2.5. Related Works

Agriculture mapping uses satellite images to map resources in the agricultural fields. During the image acquisition of satellite images, noises are introduced in the image which makes the image to have low quality.

In their study, (Wagner & Oppelt, 2020) proposed Graph based growing counters method for image segmentation of agriculture fields but the proposed methods lack a mechanism to de-noise the image before segmentation. (Milioto et al., 2018) proposed CNN based semantic segmentation for precision agriculture but the model lacks the mechanism to de-noise the image from different types of noises before segmenting the image in order to have high quality image for the segmentation.

On the other hand, (Cho et al., 2020) put forward the Iterative Fast Gradient Sign Method (I-FGSM) for image segmentation, which utilizes denoising Autoencoders to preprocess the image and remove various types of noise before applying the segmentation technique. However, the segmentation method employed in their study does not consider region parameters, which are particularly relevant and suitable for agricultural mapping applications.

In their work, (Karimpouli & Tahmasebi, 2019) introduced Seg-Net Auto-encoders as a means of segmenting rock images. While the Auto-encoders served as both segmentation and denoising tools, the segmentation

approach did not take into account the region of interest, which may not be suitable for agricultural mapping applications. On a similar note, (Lei et al., 2020) presented a ship detection and segmentation method utilizing an improved Mask R-CNN model. However, this method did not incorporate autoencoders for image segmentation and lacked a denoising mechanism. Similarly, (Youkachen et al., 2019), proposed a novel defect detection model for segmenting defects on hot-rolled steel strip surfaces. Their approach involved using Convolutional Auto-Encoder (CAE) and a sharpening process to extract defect features, but it did not prioritize segmentation based on the region of interest. In contrast, (Han et al., 2019) addressed digital rock segmentation for petrophysical analysis using CNN. Although the CNN successfully segmented digital sandstone data, it did not include a denoising mechanism. Lastly, (Vardhana et al., 2018) focused on the segmentation of brain MRI images for tumor detection using clustering techniques. However, this approach did not support segmentation based on the region of interest.

Table 2. 1 Summary of the Related work

Paper	Methodology	Gaps
Wagner et al (Wagner & Oppelt, 2020)	Graph based growing counters for extracting agricultural fields	It doesn't have denoising mechanism
Milioto et al (Milioto et al., 2018)	CNN based semantic segmentation	It doesn't have denoising mechanism
Cho et al (Cho et al., 2020)	Use denoising Autoencoders before segmentation	It doesn't use region based segmentation
Karimpouli et al (Karimpouli & Tahmasebi,	Uses Autoencoders as segmentation tool and de-noising	It doesn't use region based segmentation

2019)		
XUAN NIE et al (Nie et al., 2020)	Uses an improved Mask R-CNN model for ship detection and segmentation method	It lacks a de-noising mechanism
S Youkachen et al (Youkachen et al., 2019)	Uses Convolutional Auto-Encoder (CAE) for defect segmentation	It doesn't use region based segmentation and also lacks de-noising method.
Yufu Niu et al (Gates, n.d.)	CNN is used to segment digital sandstone data	It doesn't have a de-noising mechanism
Rohini Paul Joseph et al (Dhanachandra et al., 2015)	K-means clustering algorithm used for segmentation of MRI image	It doesn't segment based on region of interest, also lacks denoising mechanism

CHAPTER THREE
RESEARCH METHODOLOGY

3.1. Chapter Overview

The methodology used in this research is Mask Region based Convolutional Neural Network based Auto-encoder model for denoising a satellite agricultural image. This chapter describes the procedures from the dataset to evaluation measures used in proposed work to address the study question.

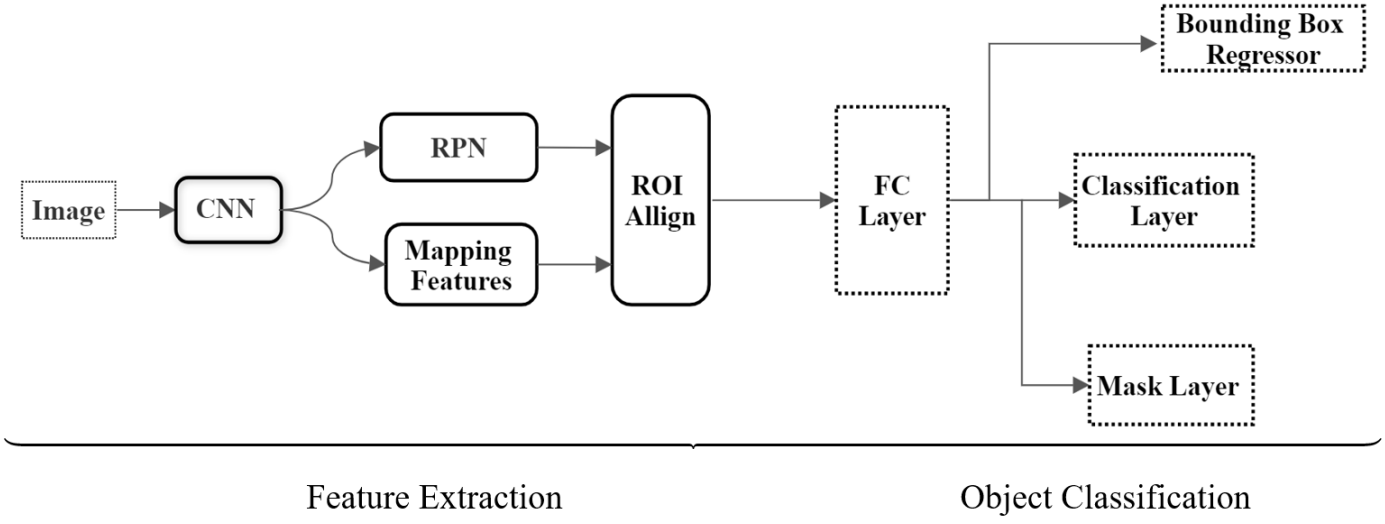


Figure 3. 1 Block Diagram of Mask R-CNN

3.2. Dataset Description

We tried to train in to two custom datasets. In order to train a Mask R-CNN algorithm, to instance segmentation, a polygon mask is done is done on VGG- annotator manually. Which is segmented in to road, building, water vegetation and land. For the second dataset to classify plants in to weed or crop. The Visual Geometry Group at the University of Oxford created the open-source VGG Image Annotator (VIA), which is used to annotate images. It's created to make it easier to annotate photos for a variety of computer vision tasks, including object identification, image classification, and segmentation.

The user-friendly VIA interface enables users to label specific objects inside an image, delineate a box around them, and add tags. Additionally, it supports many annotation formats, making it interoperable with a variety of machine learning tools and frameworks.

In addition to being a standalone program that can be installed locally, VIA is also accessible as a web application. For many computer vision applications, it has been widely employed in professional and educational environments. An image size of 1296×966 pixels for width and height in PNG format is annotated as follows in figure below. Original image can be annotated as, water, vegetation, road, land etc.

Figure 3. 2 Original aerial image to be annotated for instance segmentation

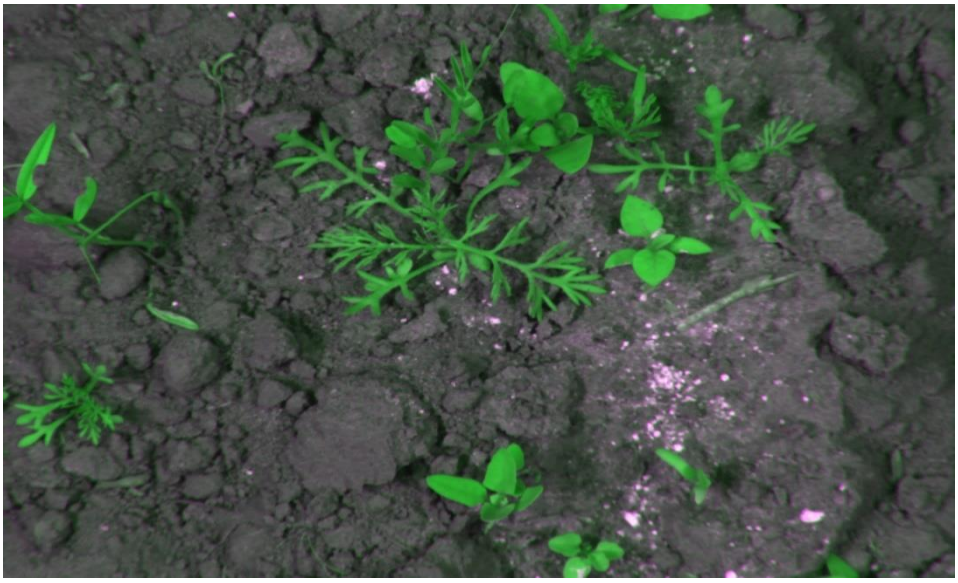


Figure 3. 3 An original image used for classification weed and crop

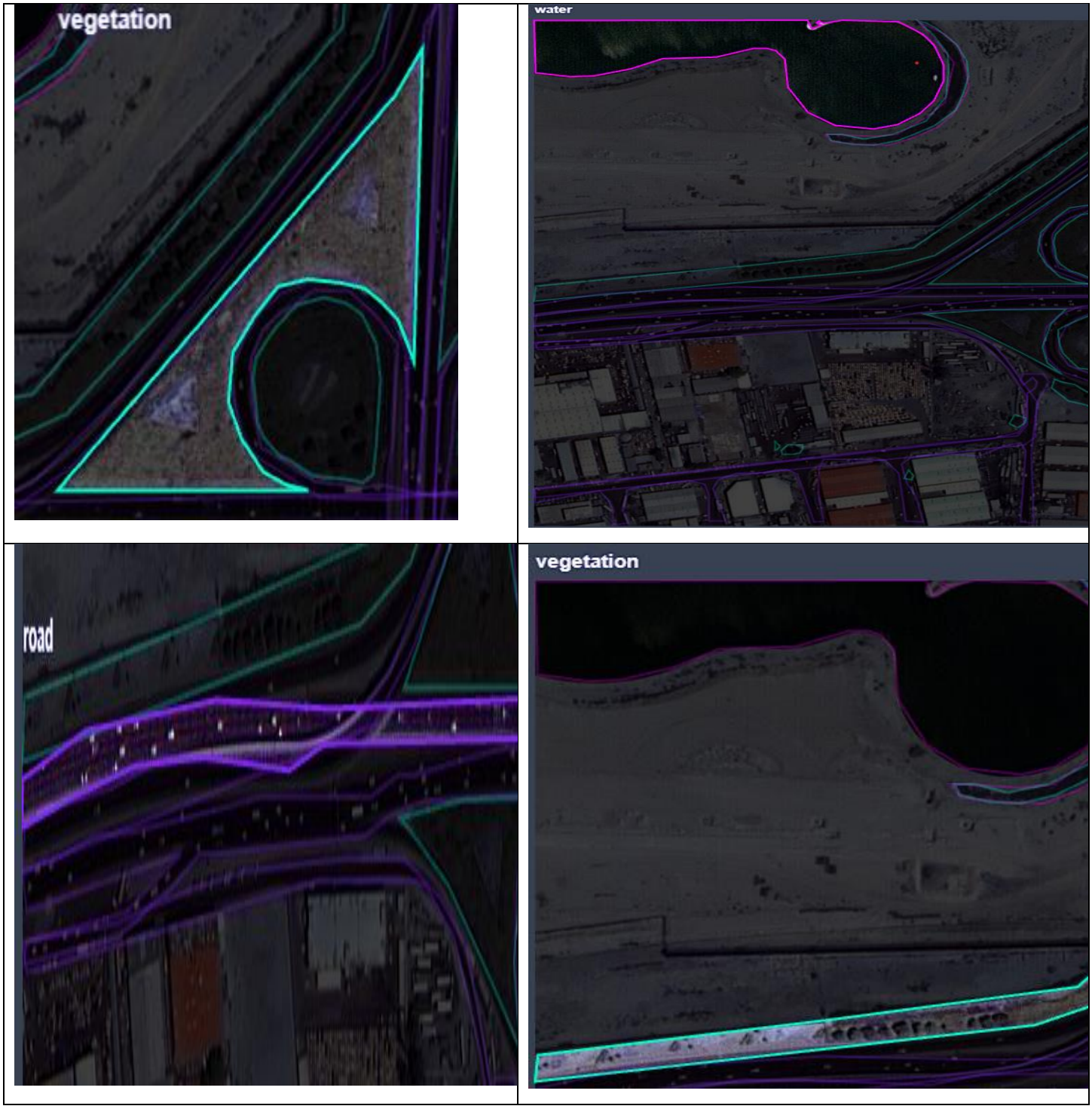


Figure 3. 4 Image annotated with the aid VGG image annotator

3.2.1. How to get the dataset

A corn field map with eBee AG in Brazil is available for download for this study. This dataset was collected in São Paulo state, in Brazil. The collection was done in a corn planted area with senseFlyeBee Ag, in

February 2021, at the end of the vegetative stage of the crop, one of the most sensitive phases for the corn crop. With the valuable information taken by the senseFly Duet M camera.

3.3. Data Augmentation

Data augmentation was done arbitrarily to the input images, including 0-360 rotations, horizontal and vertical flips, and changes in the brightness, saturation, and color. By modulating the current values by 30–100% for brightness, 100–100% for saturation (no change), and 0–200%. To produce three versions of each source image, for horizontal and vertical flip and rotation with same probability of 0.5. we utilized Gaussian blur between 0 and 2.5 pixels with salt and pepper noise for ten percent of pixels on an image was applied.

3.4. Classification Model

The image segmentation model will have de-noising and segmentation components. Autoencoders is used as a de-noising component whereas the Mask R-CNN components is used as segmentation model but these two components are a part of one deep learning model.

Autoencoders for denoising: - The Auto encoders will consist of encoder and decoder. The encoder encodes the image and maps in to latent vector whereas the decoder decodes the images to construct it from the latent vector. During construction the noise present in the original image will be discarded in the constructed image. The constructed image is then fed to the Mask R-CNN for segmentation.

Mask R-CNN: - The Mask R-CNN components has five components which are feature extractor, Region Proposed Network RPN, ROI pool, ROI Alignment and Mask layers.

Feature extractor: - The feature extractor is a CNN model without fully connected layers which extract the feature of the image using convolutional layer and pooling layers .

Region Proposed Network: - Region Proposed Network RPN is used to extract region of interest ROI out of the feature map of the image from the feature extractor.

ROI pool: - ROI pool is used to compute features from the obtained thesis in order to infer the class of the object and bounding box coordinates which is the part of the baseline model Faster R-CNN.

ROI alignment: - ROI alignment makes every feature map from the RPN to have the same size. The mask layer is used to segment the image and has two con-volutional layers. ROI Align is a crucial operation employed in detection and segmentation-based tasks to extract a compact feature map from each Region of Interest (RoI). It addresses the limitations of RoI Pooling, which suffers from harsh quantization, by effectively aligning the extracted features with the input.

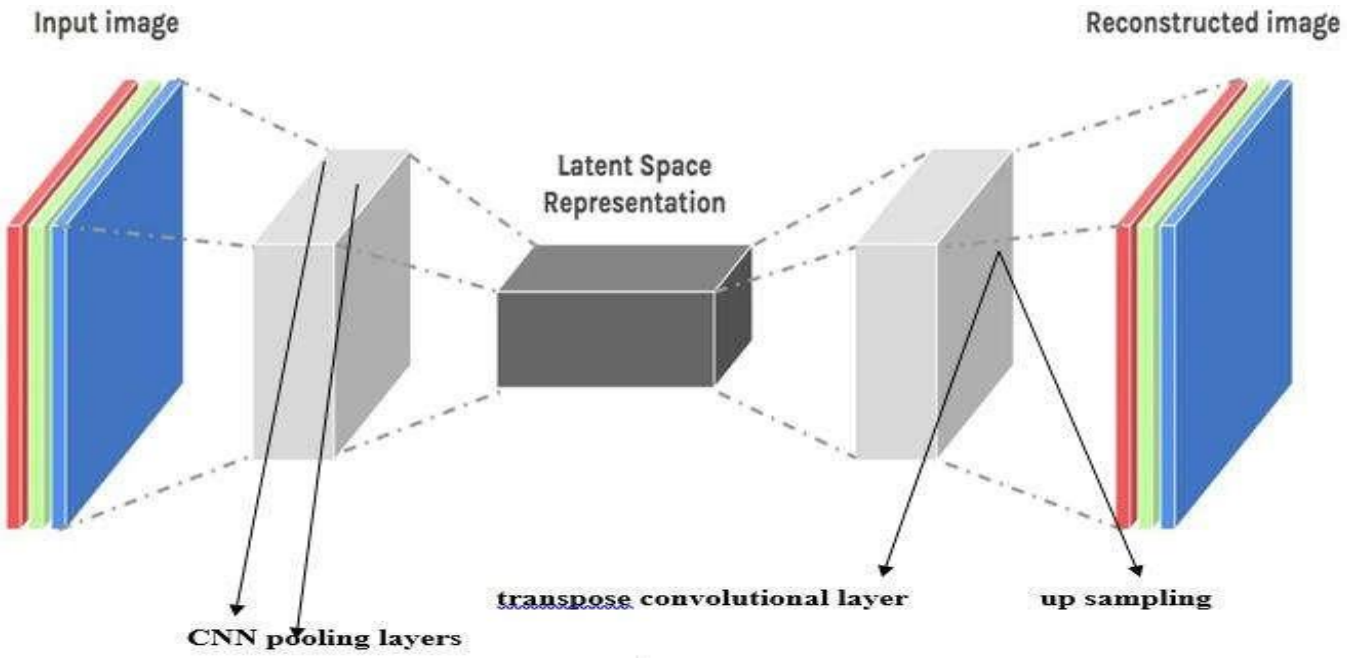


Figure 3. 5 Denoising Autoencoder (Cho et al., 2020)

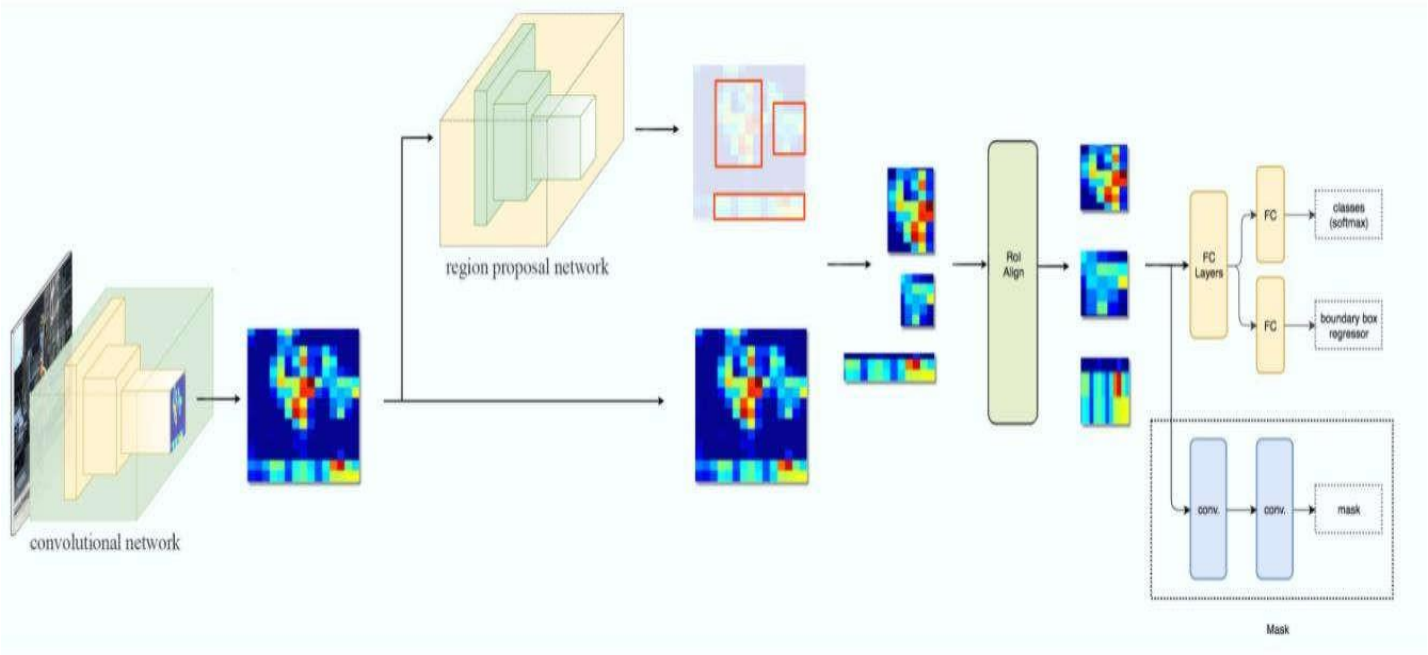


Figure 3. 6 Mask R-CNN

Image Processing: - We have about 3,000 images through our image dataset. To train the proposed model, we need to reduce the size of an image in the dataset by the same size.

3.5. Evaluation Matrices

Three key evaluation indicators are Mean Intersection over Union (IoU), Average Precision (AP), and Mean Average Precision (mAP) are used to evaluate the instance segmentation performance of the YOLOv8 algorithm(B, 2015).

By measuring the amount of overlap between the segmented mask and the target object, the IoU, also known as the Jaccard index, may assess how accurately the segmentation process worked. On the other hand, AP assesses the performance of target identification by measuring the region that is bounded by the recall rate, precision rate, and horizontal axis. The performance of target detection and instance segmentation are both included in the mAP's role as an aggregate metric.

Mean Intersection Over Union (IoU), MIoU = $\frac{Area\ Overlap}{Area\ Union} = \frac{TP}{FP + TP + FN}$

Mean Average Precision (mAP), mAP = $\frac{TP}{TP + FP}$

3.6. Development tools

This section describes the various development tools utilized to execute the proposed model from design to implementation. These include prototype development platforms, UML modeling tools, and other tools that are important for this research.

3.6.1. Design Tools

Design tools are essential for anlise design concepts, presenting, and developing. Microsoft Visio is software that allows users to draw a variety of diagrams. We used Visio version 16 for this study.

3.6.2. Hardware tools

We use hardware tools to implement this study appropriately. Hardware tools needed for this study list in the table below.

Table 3.1 Implementation tools of hardware employed for this study

No.	***Name of Device***	***Used in the study***
1	Hard Disk	To store datasets.
2	GPU	The process of training large datasets on the CPU takes time, so we used GPU for this study to speed up the training and computation.
3	RAM	It is used for GPU to accelerate the train operation.

3.6.3. Software Development Framework Tools

For executing the study work, distinct application packages and programming language uses. These are mentioned below with their description.

TensorFlow: it is a mathematical computation library for training and building ML and DL algorithms models with the easy use of high-level APIs. It also can be run on desktop computers or cloud platform.

Keras: is simply an interface that allows easy access and customizes the ML frameworks such as TensorFlow, CNTK, and Theano. These frameworks are called backends. It is more user-friendly and easier to build a model. It also can be run on CPU and GPU.

Anaconda: is a free and open-source distribution of R and Python programming languages that focuses on providing everything for data science and ML applications with more than one hundred fifty packages included with Python's core languages.

Python: is a widespread general-purpose programming language that can utilize for different applications. For this study, we used the latest version of Python (Python 3.7.).

Goggle Colab: is a product from Google Research, which allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning. It is basically a free Jupyter notebook environment running wholly in the cloud.

3.7. Baseline Works

Microsoft office packages: It uses to write the study document in a proper format. This proposed model compares various image caption models which use the same encoder- decoder approach based on different benchmark dataset.

CHAPTER FOUR

PROPOSED MASK R-CNN BASED AUTOENCODER

4.1. Chapter overview

In this chapter, we present the proposed work for image segmentation of satellite images for agricultural mapping using Mask R-CNN based Auto-encoder. The chapter begins with a detailed description of the data preprocessing steps, which play a crucial role in preparing the dataset for training and evaluation.

4.2. Proposed Model architecture

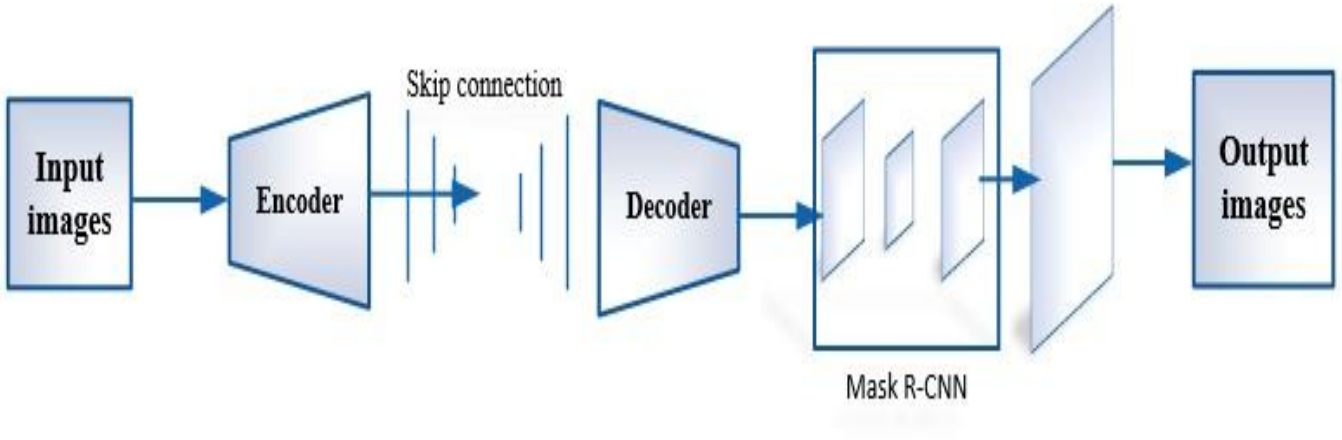


Figure 4. 1 Proposed CNN Architecture

4.3. Data preprocessing

Data preprocessing is a vital step in any machine learning project, including image annotation, which involves labeling images to train AI and machine learning models. In our research, we employ image annotation to label satellite images for training the proposed Mask R-CNN based Auto-encoder model. This process allows us to tag relevant information in the images by assigning classes to different entities present in the agricultural scenes. By annotating the images, we add valuable metadata to the dataset, enabling the model to recognize and classify the desired data features autonomously.

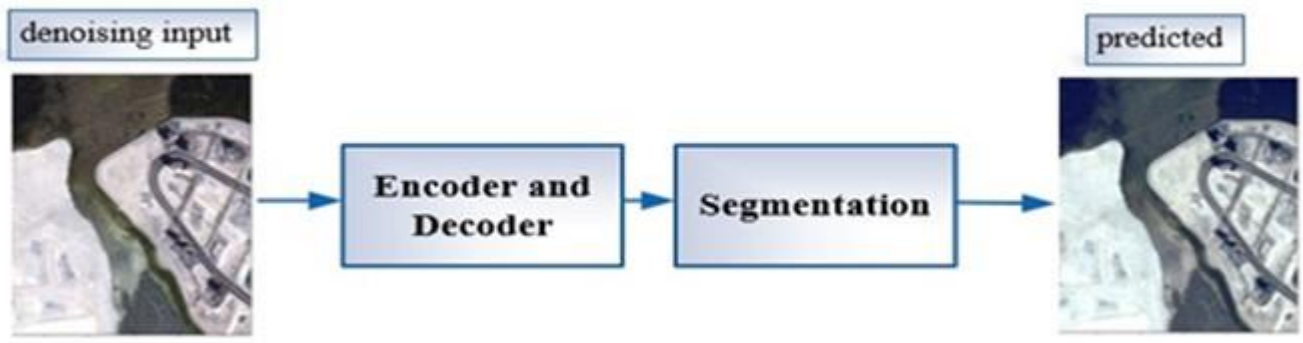


Figure 4. 2 Data preprocessing

4.4. Image Annotation with VGG Image Annotator (VIA)

For the annotation task, we utilize the VGG Image Annotator (VIA), an open-source tool developed by the Visual Geometry Group. VIA offers an efficient and user-friendly interface for defining regions of interest within an image and creating textual descriptions for those regions. The tool enables our annotators to precisely label the agricultural regions and provide accurate annotations, which are essential for training the Mask R-CNN based Auto-encoder model.

VIA also allows for flexible annotation options, such rectangles around specific regions of interest and assigning relevant class labels. This versatility in annotation techniques enables us to capture the intricate details and variations present in the agricultural images, facilitating robust segmentation.

4.5. Dataset Preparation

Once the images are annotated using VIA, we proceed with dataset preparation. This step involves organizing and structuring the annotated images and their corresponding annotations into a suitable format that can be readily used for model training and evaluation. The dataset is split into training and validation subsets to ensure unbiased model performance assessment.

During dataset preparation, attention is given to balancing the dataset by ensuring an adequate representation of different agricultural regions and classes. This helps mitigate class imbalance issues and ensures that the model learns to segment various agricultural features accurately.

The image annotation process using VIA has been described, highlighting its capabilities in efficiently labeling and describing regions of interest in the agricultural images. Furthermore, the dataset preparation

stage has been outlined, emphasizing the importance of creating a well-structured and balanced dataset for training and evaluation purposes.

4.6. Data Augmentation Techniques

We explore various data augmentation techniques applied to the prepared dataset. Data augmentation plays a crucial role in enhancing the model's ability to generalize and improve its performance. Techniques such as rotation, scaling, flipping, and adding noise can be employed to generate additional training samples, thereby enriching the dataset and reducing overfitting.

4.7. Training and Validation Data Split

This section focuses on the process of splitting the dataset into training and validation subsets. We discuss the criteria used for dividing the dataset and the importance of maintaining an appropriate balance between the two subsets. The training subset is used to train the Mask R-CNN based Auto-encoder model, while the validation subset serves as an independent evaluation set to assess the model's performance.

4.8. Model Training

In this model training, we delve into the training process of the Mask R-CNN based Auto-encoder model. We outline the architecture and parameters of the model and explain the optimization algorithm employed for training. Furthermore, we discuss the selection of appropriate loss functions and metrics for guiding the training process and monitoring the model's progress.

4.9. Model Evaluation

Once the model is trained, it is essential to evaluate its performance. This subtopic focuses on evaluating the accuracy, precision, recall, and other relevant metrics to assess the model's segmentation capabilities. We discuss the methodology used for quantitative evaluation, including the calculation of evaluation metrics and the interpretation of the results obtained.

4.10. Performance Comparison

We compare the performance of the proposed Mask R-CNN based Auto-encoder model with existing image segmentation methods in the context of agricultural mapping. We present a comparative analysis of the results, discussing the strengths and limitations of the proposed model and highlighting its potential advantages over alternative approaches.

CHAPTER FIVE

IMPLEMENTATION OF THE PROPOSED MODEL

5.1 Chapter Overview

This section discussed the proposed model architecture implementation. We also discussed the configuration of the working environment.

5.2 Working Environment

We describe the tools we have utilized to develop and implement the proposed model. Laptop: A laptop is used to do model design work.

Hardware and Software configuration

Processor : Intel® Core™i2

RAM and hard disk: 2GB and 500GB

Graphics : Intel ® Graphics 3000

System type : x64-based processor

Operating system : Windows 10

Programming language : python, kears and TensorFlow

Desktop and Google Collaboratory are used for developing or training a proposed model.

Hardware and software configuration

Operating system : Windows 10

Processor : Intel® Core™ i5

Graphics : Intel ® HD Graphics 4600

GPU : GeForce RTX 2070 Super 6 GB RAM

RAM and hard disk : 8GB and 500GB

System Type : x64-based processor

Programming language : python, Keras and Tensorflow

Google Collaboratory (colab) is a free Jupyter notebook environment for a maximum of 12 hours from the Google cloud server. The hardware specification of the colab:

GPU : Nvidia k80 /T4

RAM : 12GB

5.3 Environment setup

Anaconda: is a free and open-source distribution of R and Python programming languages that focuses on providing everything for data science and ML applications with more than one hundred fifty packages included with Python's core languages. The proposed model anaconda application version 2.1. 1 with 64-bit support utilized.

Jupyter Notebook: is a web-based application that contributes a front-end to many different languages and interactive shells such as IPython. The Jupyter notebook with a 6.4.5 version uses to execute the proposed model.

Colab: is a cloud service based on Jupiter notebook, which can improve the computing period by operating the GPU and RAM. We can train our model for 12 hours using colab GPUs

5.4 Segmentation Dataset Generation

We use satellite image with Mask R-CNN for training and testing. Below are sample code for dataset generation.

```
class segDataset(torch.utils.data.Dataset):
    def __init__(self, root, training, transform=None):
        super(segDataset, self).__init__()
        self.root = root
        self.training = training
        self.transform = transform
        self.IMG_NAMES = sorted(glob(self.root + '/*/images/*.jpg'))
        self.BGR_classes = {'Water' : [ 41, 169, 226],
                            'Land' : [246,  41, 132],
                            'Road' : [228, 193, 110],
                            'Building' : [152,  16,  60],
                            'Vegetation' : [ 58, 221, 254],
                            'Unlabeled' : [155, 155, 155]} # in BGR

        self.bin_classes = ['Water', 'Land', 'Road', 'Building', 'Vegetation', 'Unlabeled']
```

Sample code 5.1: Dataset Generation implementation

5.5 Dataset Augmentation

In computer vision, data augmentation is a crucial method for expanding the quantity and variety of training data.

```
augmentation = mask.Sequential([
    mask.Fliplr(0.5),
    mask.Flipud(0.5),
    mask.Rotate((-45, 45)),
    mask.GaussianBlur(sigma=(0, 3.0)),
    mask.Affine(
        scale={"x": (0.8, 1.2), "y": (0.8, 1.2)},
        translate_percent={"x": (-0.2, 0.2), "y": (-0.2, 0.2)},
        rotate=(-45, 45),
        shear=(-16, 16),
        order=[0, 1], #
        cval=(0, 255),
        mode=ia.ALL
    )
])
```

Sample code 5.2: Data Augmentation implementation

After dataset argumentation we split dataset in to training and testing by the following code:

```
def split_train_test(dataset=None,train_dataset=None,test_dataset=None,batch_size=4):



BACH_SIZE = batch_size
test_num = int(0.1 * len(dataset))
print(f'test data : {test_num}')
train_dataset, test_dataset = torch.utils.data.random_split(dataset, [len(dataset)-test_num, test_num], generator=torch.Generator().manual_seed(101))

train_dataloader = torch.utils.data.DataLoader(
  train_dataset, batch_size=BACH_SIZE, shuffle=True, num_workers=0)

test_dataloader = torch.utils.data.DataLoader(
  test_dataset, batch_size=BACH_SIZE, shuffle=False, num_workers=0)
return train_dataloader,test_dataloader
```

Sample code 5.3: Split train-test implementation

```
for epoch in range(N_EPOCHS):
    # training
    model.train()
    loss_list = []
    acc_list = []
    for batch_i, (x, y) in enumerate(train_dataloader):
        pred_mask = model(x)  #[4,6,512,512]
        loss = criterion(pred_mask, y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        loss_list.append(loss.cpu().detach().numpy())
        acc_list.append(acc(y,pred_mask).numpy())

        sys.stdout.write(
            "\r[Epoch %d/%d] [Batch %d/%d] [Loss: %f (%f)]"
            % (
                epoch,
                N_EPOCHS,
                batch_i,
                len(train_dataloader),
                loss.cpu().detach().numpy(),
                np.mean(loss_list),
            )
        )
    scheduler_counter += 1
```

Sample code 5.4: Training implementation

```
# testing
model.eval()
val_loss_list = []
val_acc_list = []
for batch_i, (x, y) in enumerate(test_dataloader):
    with torch.no_grad():
        pred_mask = model(x)
    val_loss = criterion(pred_mask, y)
    val_loss_list.append(val_loss.cpu().detach().numpy())
    val_acc_list.append(acc(y,pred_mask).numpy())

print(' epoch {} - loss : {:.5f} - acc : {:.2f} - val loss : {:.5f} - val acc : {:.2f}'.format(epoch,
                                                                                              np.mean(loss_list),
                                                                                              np.mean(acc_list),
                                                                                              np.mean(val_loss_list),
                                                                                              np.mean(val_acc_list)))
plot_losses.append([epoch, np.mean(loss_list), np.mean(val_loss_list)])

compare_loss = np.mean(val_loss_list)
is_best = compare_loss < min_loss
if is_best == True:
  scheduler_counter = 0
  min_loss = min(compare_loss, min_loss)
  torch.save(model.state_dict(), './areal_saved_models/unet_epoch_{}_{:.5f}.pt'.format(epoch,np.mean(val_loss_list)))

if scheduler_counter > 5:
  lr_scheduler.step()
  print(f"lowering learning rate to {optimizer.param_groups[0]['lr']}")
  scheduler_counter = 0
```

Sample code 5.5: Testing implementation

```
def vizialize(ploat_losses):
  # plot loss
  plot_losses = np.array(plot_losses)
  plt.plot(plot_losses[:,0], plot_losses[:,1], color='b', linewidth=4)
  plt.plot(plot_losses[:,0], plot_losses[:,2], color='r', linewidth=4)
  plt.title('FocalLoss', fontsize=20)
  plt.xlabel('epoch',fontsize=20)
  plt.ylabel('loss',fontsize=20)
  plt.grid()
  plt.legend(['training', 'validation']) # using a named size
  plt.show()
```

Sample code 5.6: Visualization implementation

5.6 Image Denoising

In this thesis, we utilized an image denoising method called Noise2Void (N2V) uses deep learning to be able to remove various kinds of noise from satellite images.

Noise2Void (N2V) is a deep learning-based denoising algorithm that is designed to remove noise from images. The algorithm is based on the assumption that the noise in an image is a function of the image itself and can be learned using a neural network. This work is done on kaggle due to it gives more CPU memory storage for denoising than colab.

The N2V algorithm consists of three main steps:

1. Patch extraction: The first step of the algorithm is to extract patches from the input image. These patches are small, overlapping sections of the image that are used to train the neural network.

2. Neural network training: The second step is to train a neural network to learn the relationship between the noisy input patches and the corresponding noise-free output patches. The neural network is trained using a set of noisy and clean image patches.

3. Denoising: The final step is to use the trained neural network to denoise the input image. This is done by applying the neural network to each patch in the input image to predict the corresponding noise-free patch. The denoised image is then reconstructed from the denoised patches.

The key innovation of N2V is the use of a data augmentation technique called "blind-spot" training. This technique involves randomly corrupting a portion of each training patch during each epoch of training. This forces the neural network to learn to denoise the image based on the available information, rather than memorizing the training data.

Additionally, N2V uses a novel approach to regularization called "patch-based dropout." This approach involves randomly setting a portion of the input patch to zero during training, forcing the neural network to learn to denoise images with missing data.

Overall, Noise2Void is an effective and flexible technique for denoising images, and has been shown to perform well on a wide range of noise types and image modalities.

We generate training data by adding noise to clean images, which can be used to train the N2V model. here, we define noise distribution and parameter in addition to wrapping clean and noisy images. we have used a Gaussian noise distribution with a mean of 0 and a standard deviation of 25, and added this noise to the clean image to obtain the noisy image.

```python
class N2V_DataGenerator():

    def load_imgs(self, files, dims='YX', image_reader=imread):
        assert 'Y' in dims and 'X' in dims, "'dims' has to contain 'X' and 'Y'."
        tmp_dims = dims
        for b in ['X', 'Y', 'Z', 'T', 'C']:
            assert tmp_dims.count(b) <= 1, "'dims' has to contain {} at most once.".format(b)
            tmp_dims = tmp_dims.replace(b, '')
        assert len(tmp_dims) == 0, "Unknown dimensions in 'dims'."
        if 'Z' in dims:
            net_axes = 'ZYXC'
        else:
            net_axes = 'YXC'
        move_axis_from = ()
        move_axis_to = ()
        for d, b in enumerate(dims):
            move_axis_from += tuple([d])
            if b == 'T':
                move_axis_to += tuple([0])
            elif b == 'C':
                move_axis_to += tuple([-1])
            elif b in 'XYZ':
                if 'T' in dims:
                    move_axis_to += tuple([net_axes.index(b)+1])
                else:
                    move_axis_to += tuple([net_axes.index(b)])
        imgs = []
        for f in files:
            img = image_reader(f)
            assert len(img.shape) == len(dims), "Number of image dimensions doesn't match 'dims'."
            img = np.moveaxis(img, move_axis_from, move_axis_to)
            if not ('T' in dims):
                img = img[np.newaxis]
            if not ('C' in dims):
                img = img[..., np.newaxis]
            imgs.append(img)

        return imgs
```

Sample code 5.7: Data Generation for training implementation

```python
def load_imgs_from_directory(self, directory, filter='*.tif', dims='YX'):


    files = glob(join(directory, filter))
    files.sort()
    return self.load_imgs(files, dims=dims)
```

Sample code 5.8: load image from directory implementation

In the next steps, we use patchify. Patchify is a function that is used to split an image into small patches. It is commonly used in deep learning-based image processing tasks, such as image denoising and segmentation, where the input image is divided into smaller patches that can be fed into a neural network for processing. We

need to define patch size as a tuple of height and width in pixels then we use concatenation to combine feature maps from different layers in the encoder network with corresponding feature maps in the decoder network through skip connections. Skip connections are important in N2V because they allow the neural network to preserve information from the input image at different scales while still learning to denoise the image. The skip connections are implemented by concatenating the feature maps from the encoder network with corresponding feature maps in the decoder network.

```
def __extract_patches__(self, data, num_patches=None, shape=(256, 256), n_dims=2):
    if num_patches == None:
        patches = []
        if n_dims == 2:
            if data.shape[1] > shape[0] and data.shape[2] > shape[1]:
                for y in range(0, data.shape[1] - shape[0] + 1, shape[0]):
                    for x in range(0, data.shape[2] - shape[1] + 1, shape[1]):
                        patches.append(data[:, y:y + shape[0], x:x + shape[1]])

                return np.concatenate(patches)
            elif data.shape[1] == shape[0] and data.shape[2] == shape[1]:
                return data
            else:
                print("'shape' is too big.")
        elif n_dims == 3:
            if data.shape[1] > shape[0] and data.shape[2] > shape[1] and data.shape[3] > shape[2]:
                for z in range(0, data.shape[1] - shape[0] + 1,  shape[0]):
                    for y in range(0, data.shape[2] - shape[1] + 1, shape[1]):
                        for x in range(0, data.shape[3] - shape[2] + 1, shape[2]):
                            patches.append(data[:, z:z + shape[0], y:y + shape[1], x:x + shape[2]])

                return np.concatenate(patches)
            elif data.shape[1] == shape[0] and data.shape[2] == shape[1] and data.shape[3] == shape[2]:
                return data
            else:
                print("'shape' is too big.")
        else:
            print('Not implemented for more than 4 dimensional (ZYXC) data.')
    else:
        patches = []
        if n_dims == 2:
            for i in range(num_patches):
                y, x = np.random.randint(0, data.shape[1] - shape[0] + 1), np.random.randint(0,
                                                                                             data.shape[
                                                                                                 2] - shape[
                                                                                                 1] + 1)
                patches.append(data[0, y:y + shape[0], x:x + shape[1]])

            if len(patches) > 1:
                return np.stack(patches)
            else:
                return np.array(patches)[np.newaxis]
        elif n_dims == 3:
            for i in range(num_patches):
                z, y, x = np.random.randint(0, data.shape[1] - shape[0] + 1), np.random.randint(0,
                                                                                               data.shape[
                                                                                                   2] - shape[
                                                                                                   1] + 1), np.random.randint(
                    0, data.shape[3] - shape[2] + 1)
                patches.append(data[0, z:z + shape[0], y:y + shape[1], x:x + shape[2]])

            if len(patches) > 1:
                return np.stack(patches)
            else:
                return np.array(patches)[np.newaxis]
        else:
            print('Not implemented for more than 4 dimensional (ZYXC) data.')
```

Sample code 5.9: image patch implementation

Patch-based data augmentation is an important technique in Neural Network-based Noise2Void (N2V) to increase the amount of training data and improve the generalization performance of the model. In patch-based

data augmentation, we generate new training samples by applying random transformations like rotation, flip, translation, brightest and contrast adjustment to the patches extracted from the input image as shown in sample code 5.5. The augmented patches are stored in a new array of the same shape as the original patches.

```
def __augment_patches__(self, patches):
    if len(patches.shape[1:-1]) == 2:
        augmented = np.concatenate((patches,
                                    np.rot90(patches, k=1, axes=(1, 2)),
                                    np.rot90(patches, k=2, axes=(1, 2)),
                                    np.rot90(patches, k=3, axes=(1, 2))))
    elif len(patches.shape[1:-1]) == 3:
        augmented = np.concatenate((patches,
                                    np.rot90(patches, k=1, axes=(2, 3)),
                                    np.rot90(patches, k=2, axes=(2, 3)),
                                    np.rot90(patches, k=3, axes=(2, 3))))

    augmented = np.concatenate((augmented, np.flip(augmented, axis=-2)))
    return augmented
```

Sample code 5.10: patch augmentation implementation

```
def build_unet(input_shape, last_activation,  n_depth=2, n_filter_base=16, kernel_size=(3,3,3), n_conv_per_depth=2, activation="relu",
               batch_norm=False,  dropout=0.0,  pool_size=(2,2,2),  residual=False,   prob_out=False,   eps_scale=1e-3,   blurpool=False,
                skip_skipone=False):

    if last_activation is None:
        raise ValueError("last activation has to be given (e.g. 'sigmoid', 'relu')!")

    all((s % 2 == 1 for s in kernel_size)) or _raise(ValueError('kernel size should be odd in all dimensions.'))

    channel_axis = -1 if backend_channels_last() else 1

    n_dim = len(kernel_size)
    conv = Conv2D if n_dim==2 else Conv3D

    num_channels = input_shape[channel_axis]

    input = Input(input_shape, name = "input")

    unet = unet_block(n_depth, n_filter_base, kernel_size,  activation=activation, dropout=dropout, batch_norm=batch_norm,
                      n_conv_per_depth=n_conv_per_depth, pool=pool_size, prefix='channel_0', blurpool=blurpool,
                      skip_skipone=skip_skipone)(input)

    final = conv(num_channels, (1,)*n_dim, activation='linear')(unet)
    if residual:
        if not (num_channels == 1):

            raise ValueError("number of input and output channels must be the same for a residual net.")
        final = Add()([final, input])
    final = Activation(activation=last_activation)(final)

    if prob_out:
        scale = conv(num_channels, (1,)*n_dim, activation='softplus')(unet)
        scale = Lambda(lambda x: x+np.float32(eps_scale))(scale)
        final = Concatenate(axis=channel_axis)([final,scale])

    return Model(inputs=input, outputs=final)
```

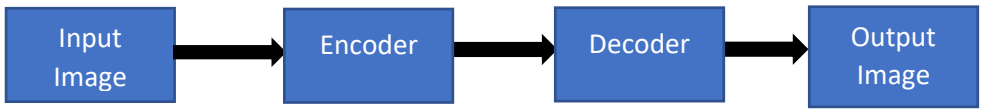
Sample code 5.11: N2V model implementation

The N2V model consists of an encoder-decoder architecture with skip connections.

The encoder network consists of several convolutional layers that down sample the input image to a lower-dimensional representation. The decoder network consists of several convolutional layers that up sample the lower-dimensional representation back to the original resolution of the input image. Skip connections are added between corresponding layers in the encoder and decoder networks to preserve information from the input image. The N2V model also incorporates several novel techniques for regularization and data augmentation, including "blind-spot" training and "patch-based dropout." These techniques help to prevent overfitting and improve the generalization performance of the model.

During training, the N2V model is trained using a set of noisy and clean image patches. The neural network learns the relationship between the noisy input patches and the corresponding clean output patches. The training process involves minimizing the mean squared error between the predicted and true output patches.

Once the N2V model is trained, it can be used to denoise new images by passing them through the model to predict the corresponding noise-free image. Here is a summary of the N2V model architecture:



The encoder network consists of several convolutional layers that down sample the input image, while the decoder network consists of several transposed convolutional layers that up sample the lower-dimensional representation back to the original resolution of the input image. Skip connections are added between corresponding layers in the encoder and decoder networks to preserve information from the input image.

The N2V model is trained using a set of noisy and clean image patches, and the training process involves minimizing the mean squared error between the predicted and true output patches. Once the model is trained, it can be used to denoise new images by passing them through the model to predict the corresponding noise-free image.

```
def loss_mse():
    def n2v_mse(y_true, y_pred):
        target, mask = tf.split(y_true, 2, axis=len(y_true.shape)-1)
        loss = tf.reduce_sum(K.square(target - y_pred*mask)) / tf.reduce_sum(mask)
        return loss

    return n2v_mse

def loss_mae():
    def n2v_abs(y_true, y_pred):
        target, mask = tf.split(y_true, 2, axis=len(y_true.shape)-1)
        loss = tf.reduce_sum(K.abs(target - y_pred*mask)) / tf.reduce_sum(mask)
        return loss

    return n2v_abs
```

Sample code 5.12: Loss function implementation

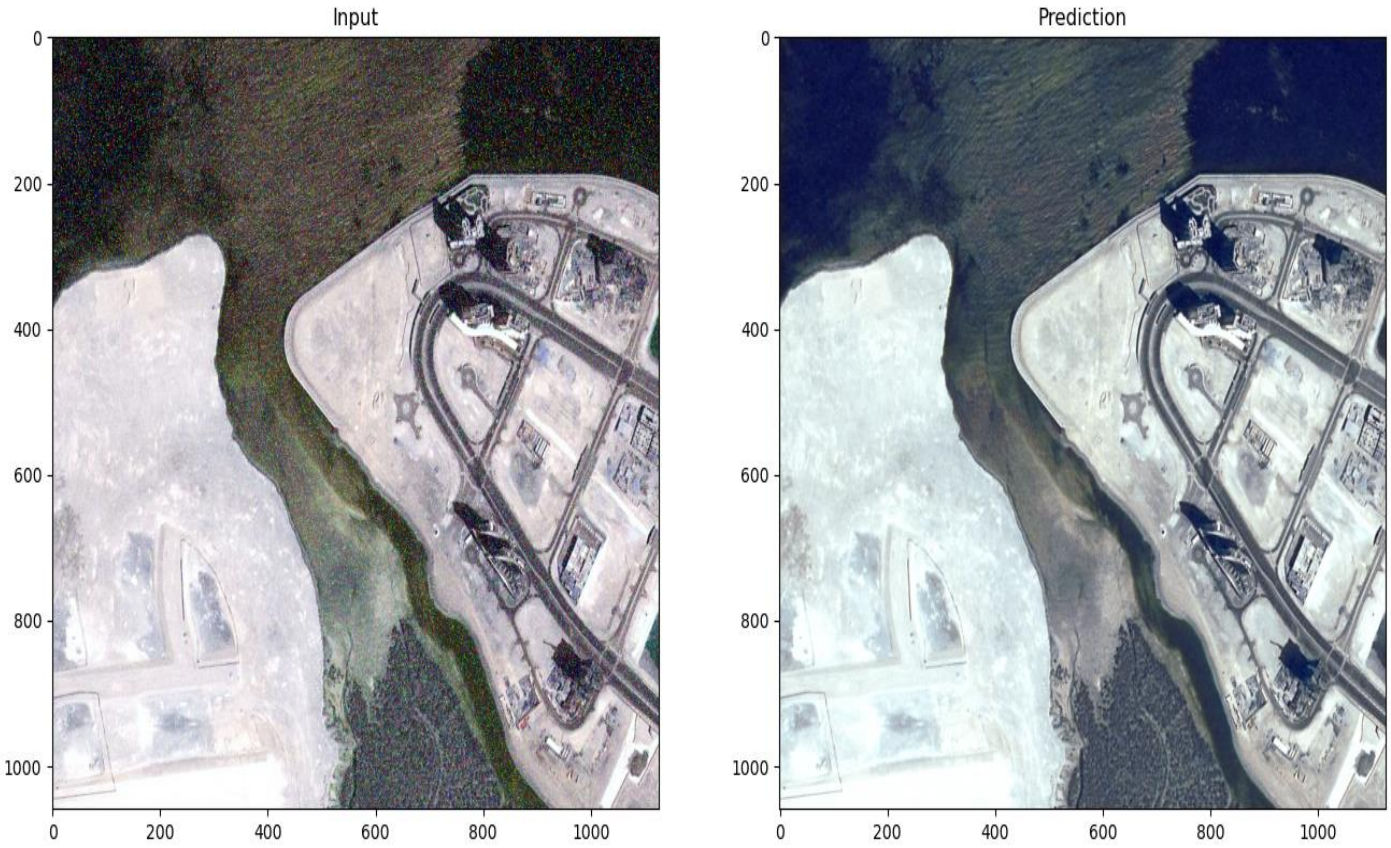
The MSE loss measures the average squared difference between the predicted and true pixel intensities in each patch. During training, the neural network learns to minimize the MSE loss between the predicted and true output patches by adjusting the weights of the network through backpropagation. The optimizer used to update the weights can be any of the standard optimization algorithms, such as stochastic gradient descent (SGD) or Adam.

In addition to the MSE loss, the N2V model also incorporates several regularization techniques to prevent overfitting and improve the generalization performance of the model. These techniques include "blind-spot" training, which randomly masks out parts of the input image during training, and "patch-based dropout," which randomly drops out patches from the input image during training. These techniques help to prevent the N2V model from memorizing the training data and improve its ability to generalize to new images.

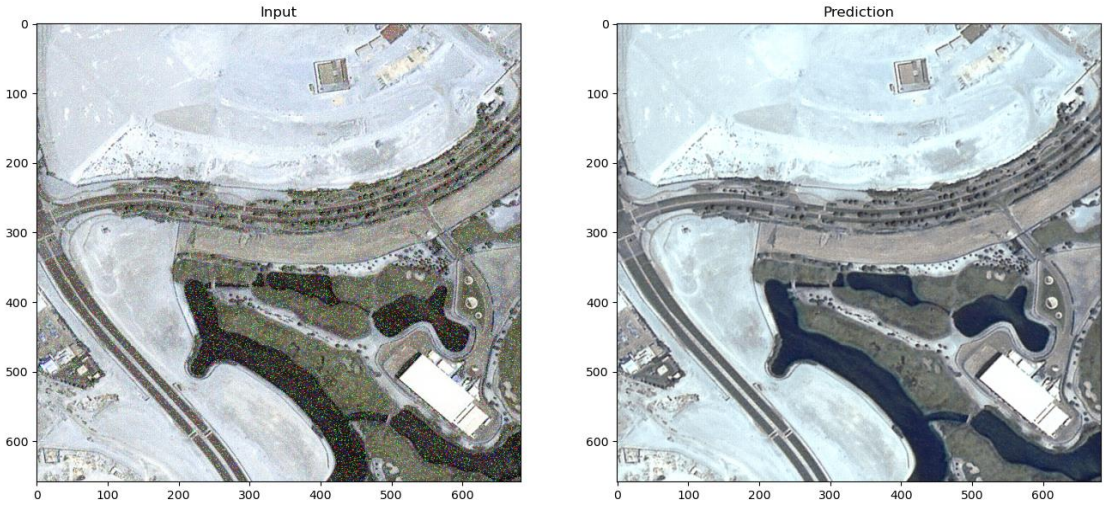
Mathematically,

$$MSE = \frac{1}{\Pi}\sum_{i=1}^{M}(predicted\ value - true\ value)^2$$

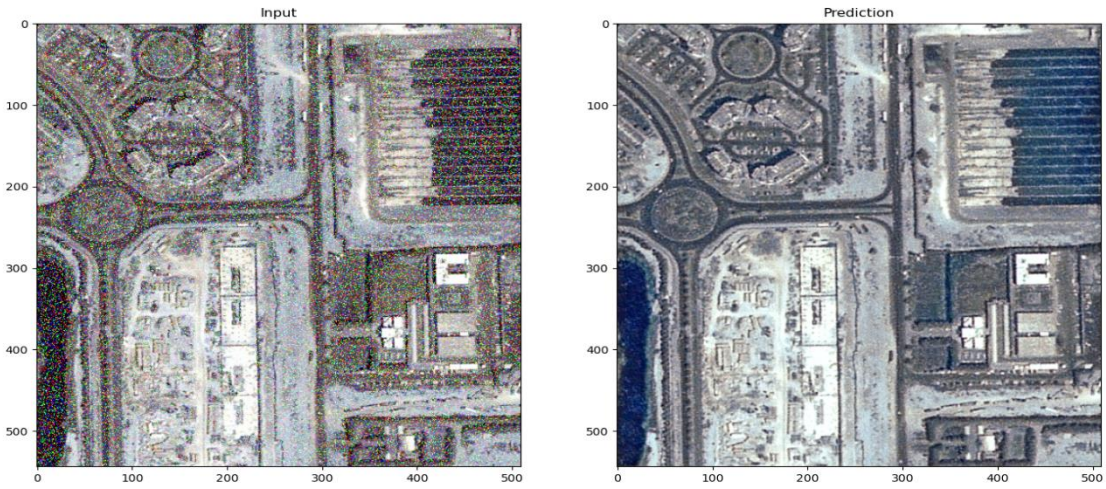
If the small value and large values are very far from the mean values, another loss function called Mean Absolute Error (MAE) can be used.



Sample output 5.13 denoising input vs predicted



Sample output 5.14 denoising input vs predicted



Sample output 5.15 denoising input vs predicted

Table 5.1 Hyper-parameters

Hyper-parameters
Activation – relu, linear
Training epoch - 200
Loss function- mse and mae
Batch size = 128
Learning rate= 0.0004
Patch shape = (128,128)

CHAPTER SIX

EXPERIMENTAL RESULT AND DISCUSSION

6.1. Chapter Overview

This chapter presents the results of the experiments and discusses the implications of the results.

6.2. Results of the Study

6.2.1. Data Augmentation result

Data augmentation was done arbitrarily to the input images, including 0-360 rotations, horizontal and vertical flips, and changes in the brightness, saturation, and color. By modulating the current values by 30–100% for brightness, 100–100% for saturation (no change), and 0–200%.

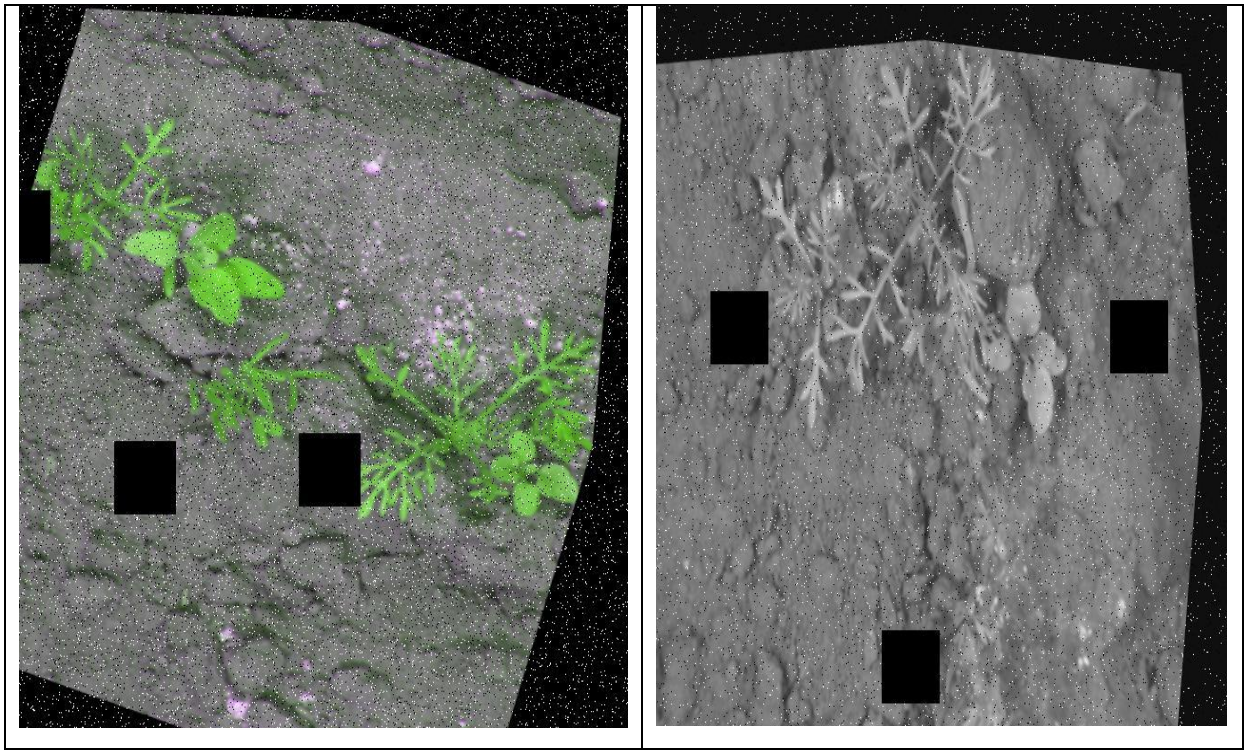


Figure 6. 1 a) flip image augmentation by adding noise used for training

To produce three versions of each source image, for horizontal and vertical flip and rotation with same probability of 0.5. we utilized Gaussian blur between 0 and 2.5 pixels with salt and pepper noise for ten percent of pixels on an image was applied.

6.2.2. Experimental results

The aim of satellite instance segmentation is to get a set of features with its corresponding sequence. In agriculture perspective, the instance image can be compared with a well-known dataset. The simulation result demonstrates the proposed model surpasses in accurately identifying and locating the image. Figure 6.2 shows F1-score graph for segmentation of aerial image in threshold confidence as depicted in chapter 4. The confidence threshold refers to minimum probability or score required for a model to make a positive prediction. It is useful visualization tool in choosing optimal threshold.

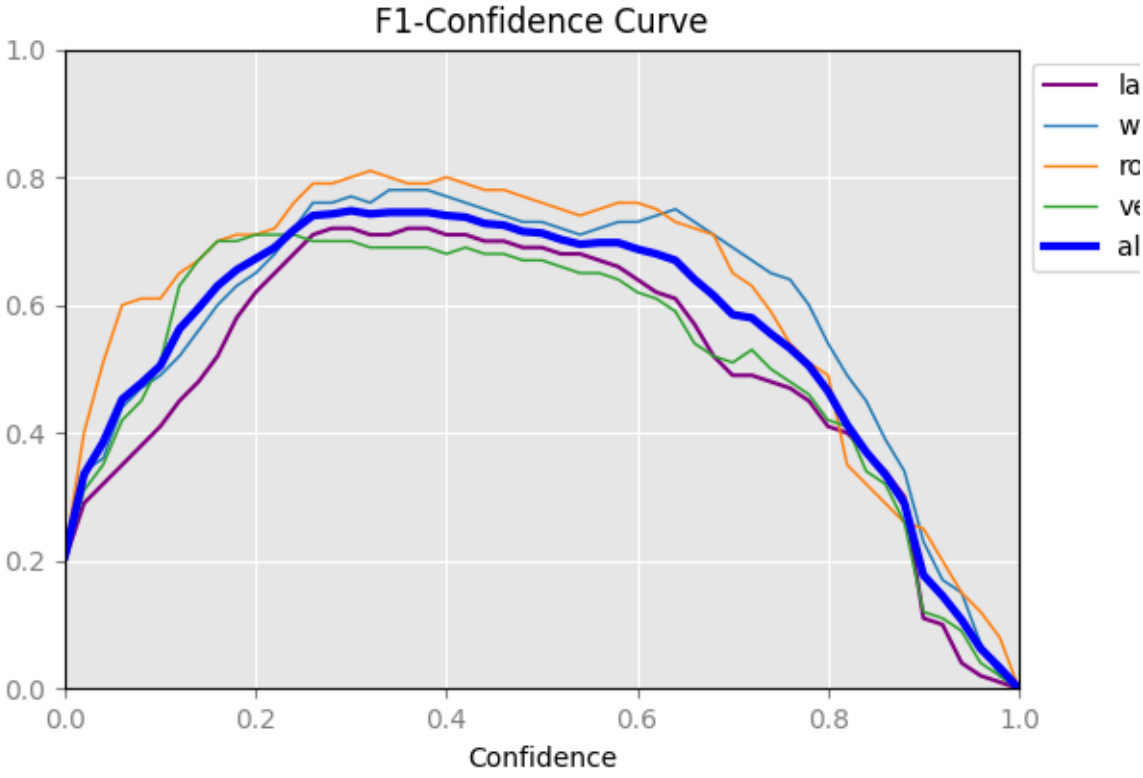


Figure 6. 2 F1 – Confidence Curve

The simulation result shown in Figure 6.3 indicates precision against the confidence score of positive predictions made by the model. The resulting curve shows how the precision of the model varies as the confidence threshold is varied. The precision is proportion to correctly predicted positive samples among all positive predictions made by the model at that threshold.

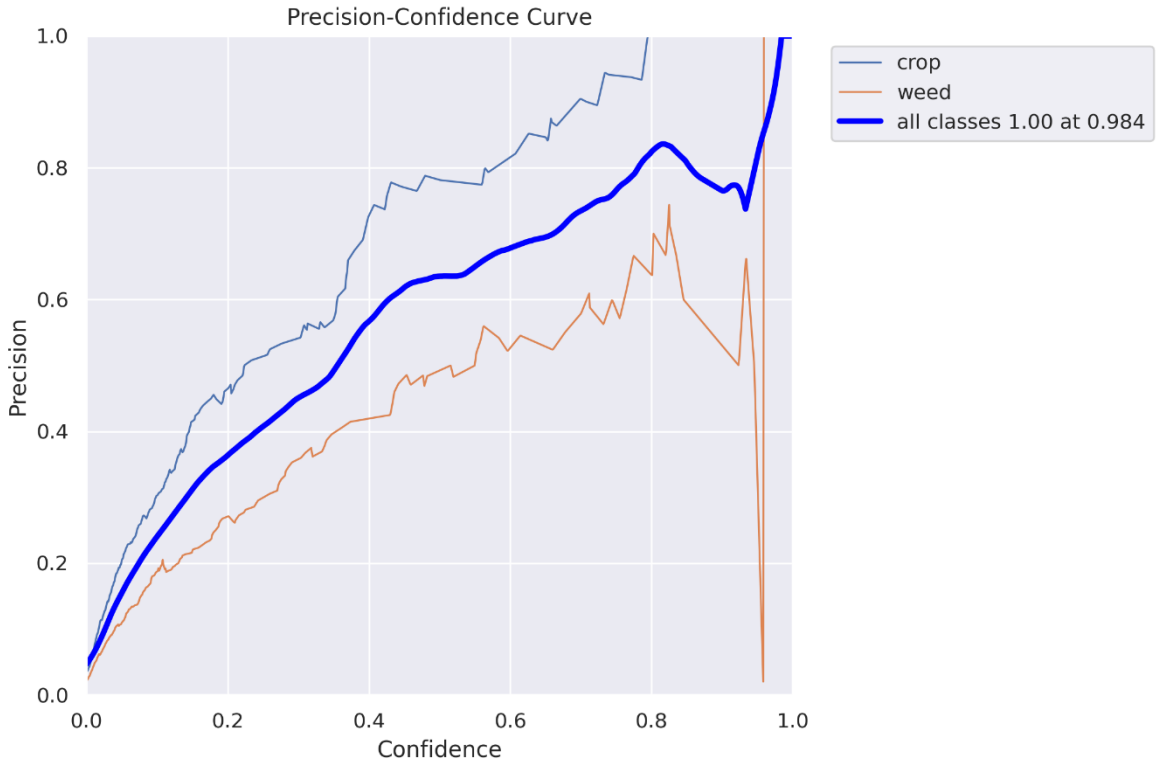


Figure 6. 3 Precision - Confidence Curve

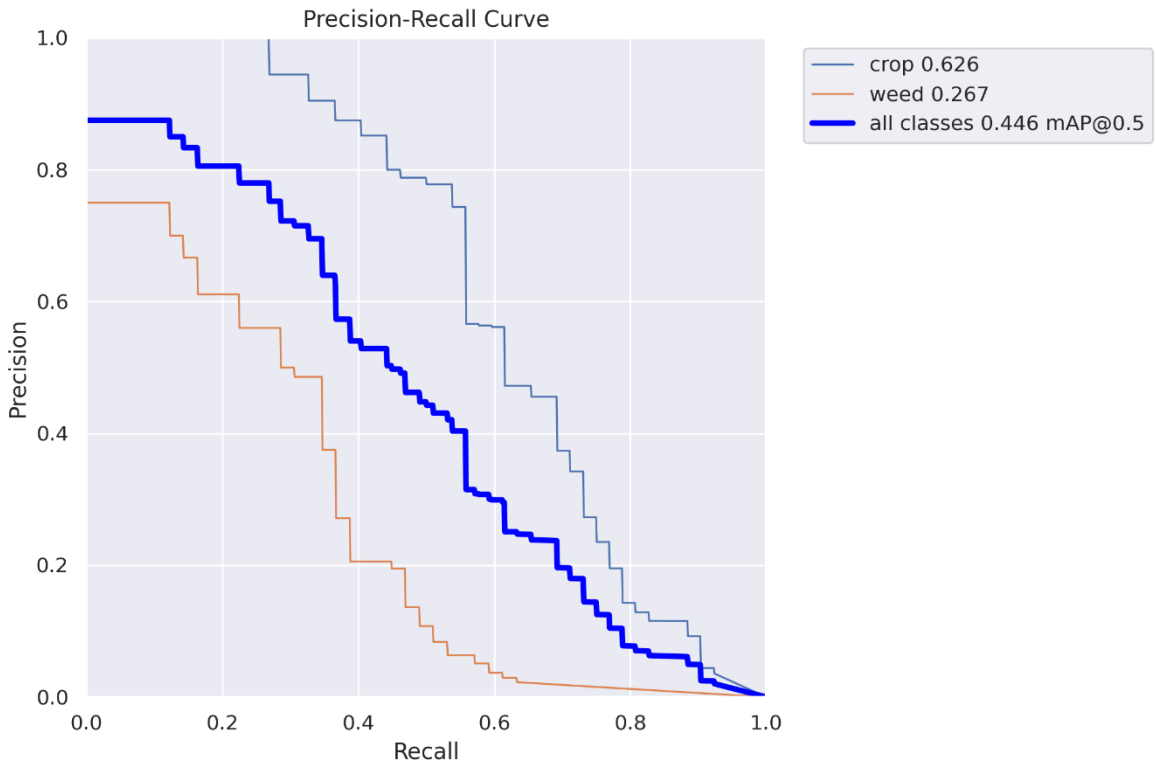


Figure 6. 4 Precision –Recall Curve

Figure 6.4 demonstrates the trade-off precision and recall at various classification thresholds. To create a PR curve, the binary classification model is used to make predictions on a test set, and the predicted probabilities or scores for each sample are sorted in descending order. The threshold for classification is varied from 0 to 1, and at each threshold, the precision and recall are calculated based on the true positive, true negative, false positive, and false negative rates. The PR curve is useful for evaluating the performance of a model when the dataset is imbalanced, meaning that one class is more prevalent than the other.

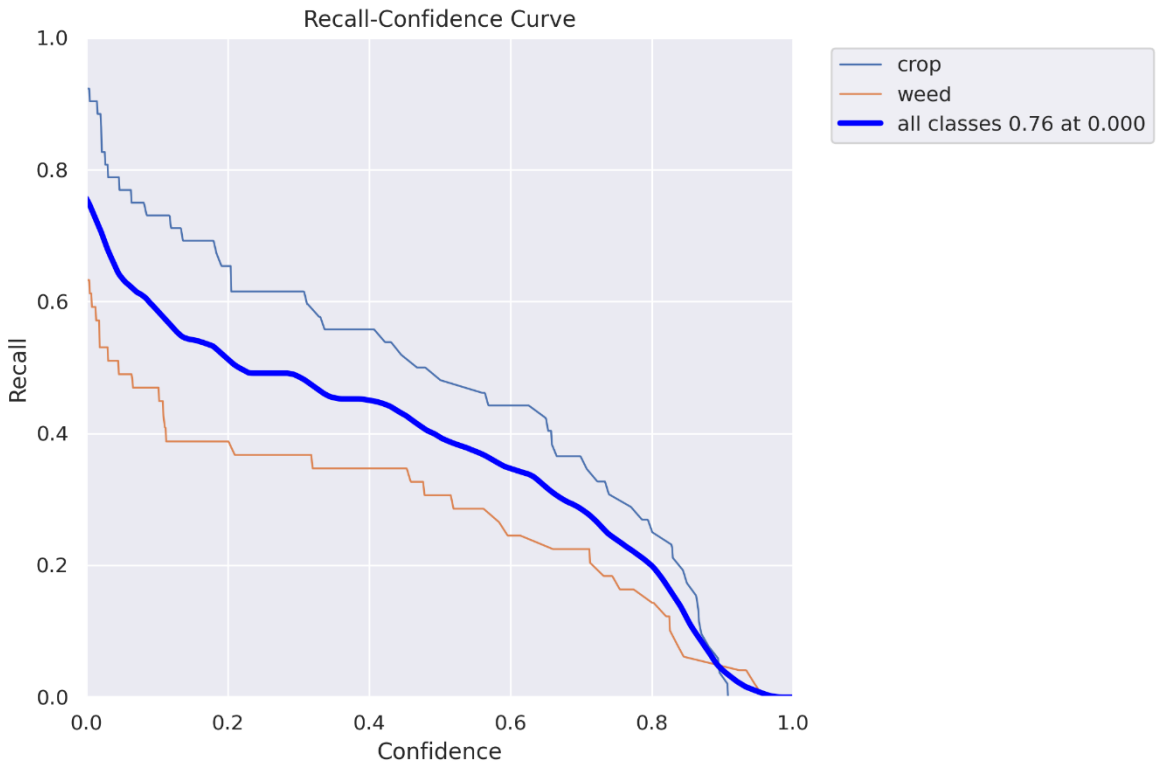


Figure 6. 5 Recall - Confidence Curve

The training, the evaluation metrics and validation with a Multi- Task loss function for each sampled ROI during the training is presented in Figure 6.6. The multi-task loss function in Mask R-CNN consists of three components: the classification loss, the bounding box regression loss, and the mask segmentation loss. The classification loss measures the difference between the predicted class probabilities and the ground-truth class labels for each object proposal. It is computed using the cross-entropy loss function, which penalizes incorrect predictions more heavily than correct ones. Let P be the predicted probability vector for each object proposal, and let Y be the ground-truth class label vector, then the classification loss is given by:

$$L_{class} = -\sum (Y \log(P) + (1 - Y)\log(1 - P))$$

The bounding box regression loss measures the difference between the predicted bounding box coordinates and the ground-truth bounding box coordinates for each object proposal. It is computed using the smooth L1 loss function, which is less sensitive to outliers than the mean squared error loss. Let T be the predicted bounding box coordinates, and let T^* be the ground-truth bounding box coordinates, then the bounding box regression loss is given by:

$$L_{reg} = \sum L1\_smooth(T - T^*)$$

The mask segmentation loss measures the difference between the predicted mask and the ground-truth mask for each object proposal. It is computed using the binary cross-entropy loss function, which penalizes incorrect predictions more heavily than correctones. Let M be the predicted mask for each object proposal, and let M^* be the ground-truth mask, then the mask segmentation loss is given by:

$$L_{class} = -\sum (M \log(M) + (1 - M)\log(1 - M))$$

The total loss function for Mask R-CNN is a weighted sum of these three components:

$$L_{tot} = L_{class} + \lambda L_{box} + \lambda L_{mask}$$

where λ is a hyper parameter that controls the relative importance of the three components. The classification loss and the mask segmentation loss are weighted equally (λ=1), while the bounding box regression loss is typically weighted lower (e.g., λ=0.5) to prevent it from dominating the other components.

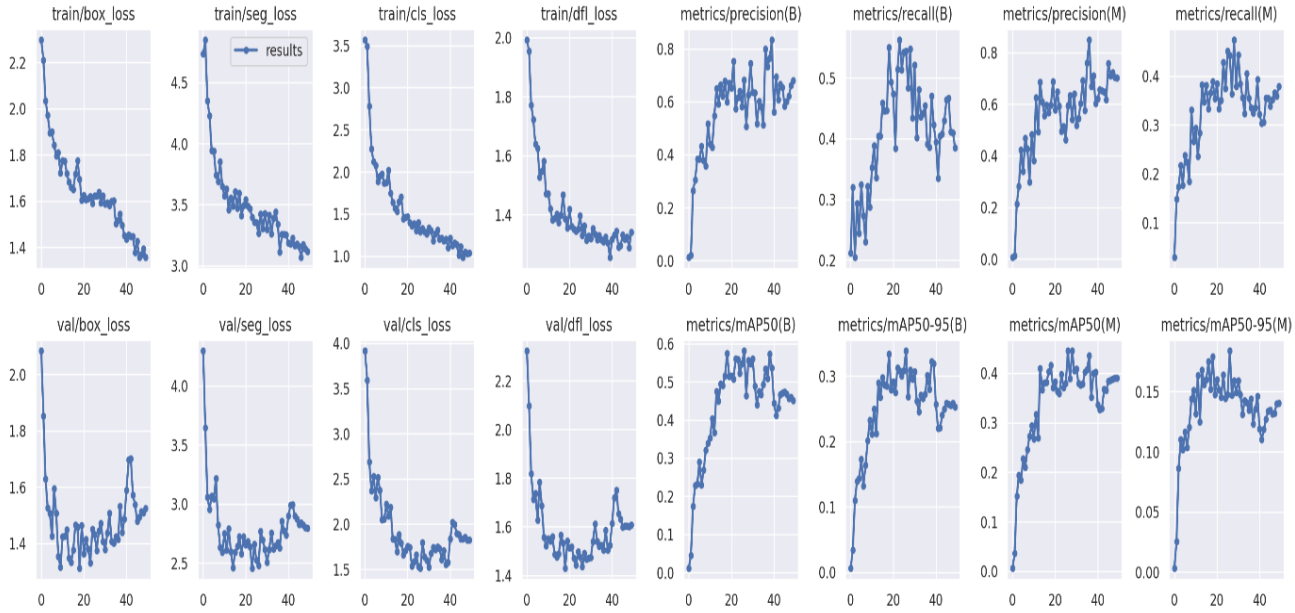


Figure 6. 6 Evaluation Metrics

During training, the model is optimized to minimize this multi-task loss function using stochastic gradient descent (SGD) or a variant thereof. The gradients of the loss with respect to the model parameters are computed using backpropagation, and the parameters are updated iteratively to minimize the loss as shown in

Figure 6.7. It is the phase where object detection and instance segmentation take place. The first task is to detect objects in an image and predict their class labels and bounding box coordinates. The model achieves this by using the Region Proposal Network (RPN) to generate object proposals and then refining them with the Region of Interest (RoI) pooling layer. The RPN generates a set of candidate object proposals by convolving the features from the backbone network with a set of anchor boxes of different scales and aspect ratios. The RoI pooling layer extracts a fixed-size feature map from each object proposal, which is then passed through a series of fully connected layers to predict the class label and bounding box coordinates. The second task is to segment the objects at the pixel level, i.e., to predict a binary mask for each object proposal that indicates which pixels belong to the object and which do not. The model achieves this by adding a mask branch to the RoI pooling layer, which generates a binary mask for each object proposal by applying a set of convolutional layers to the RoI feature map. In addition to this task

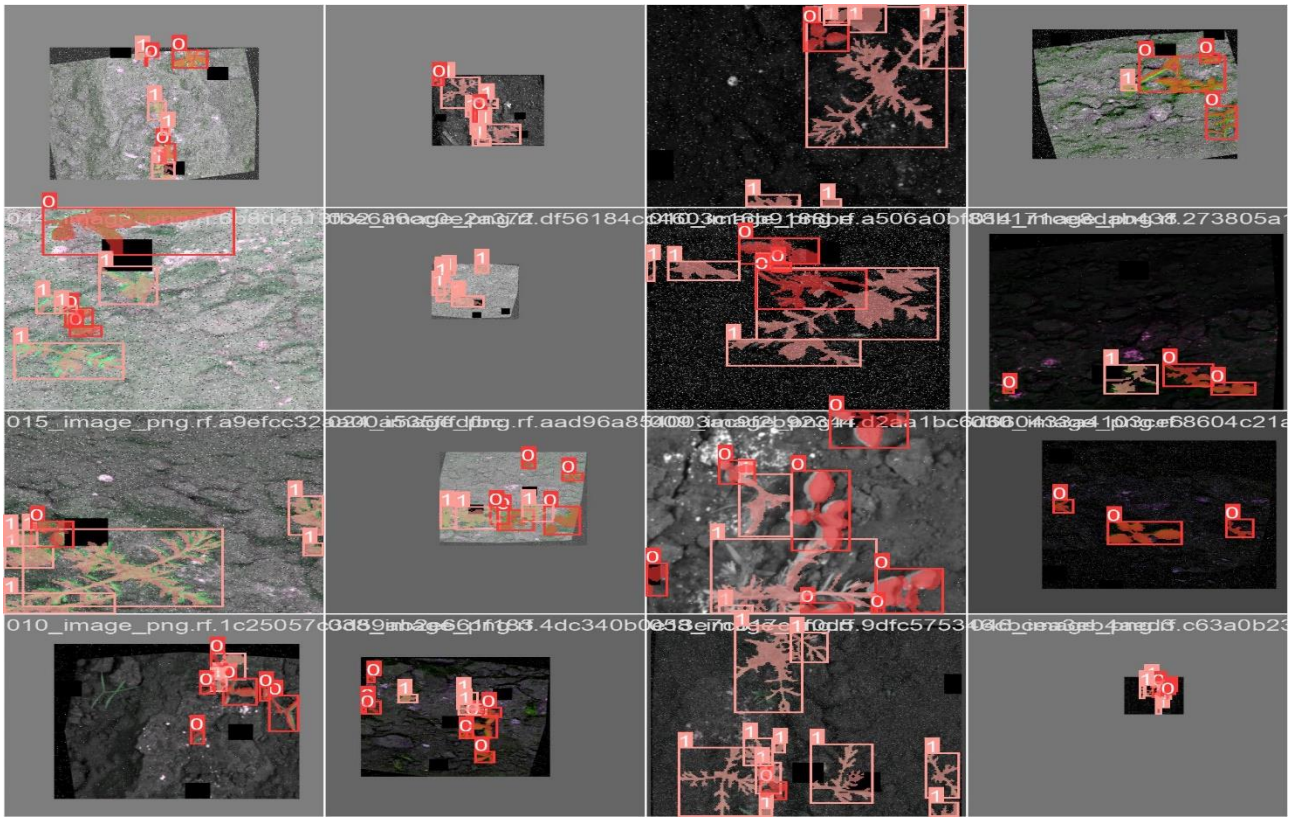


Figure 6. 7 Training phase

Figure 6. 8 validation phase

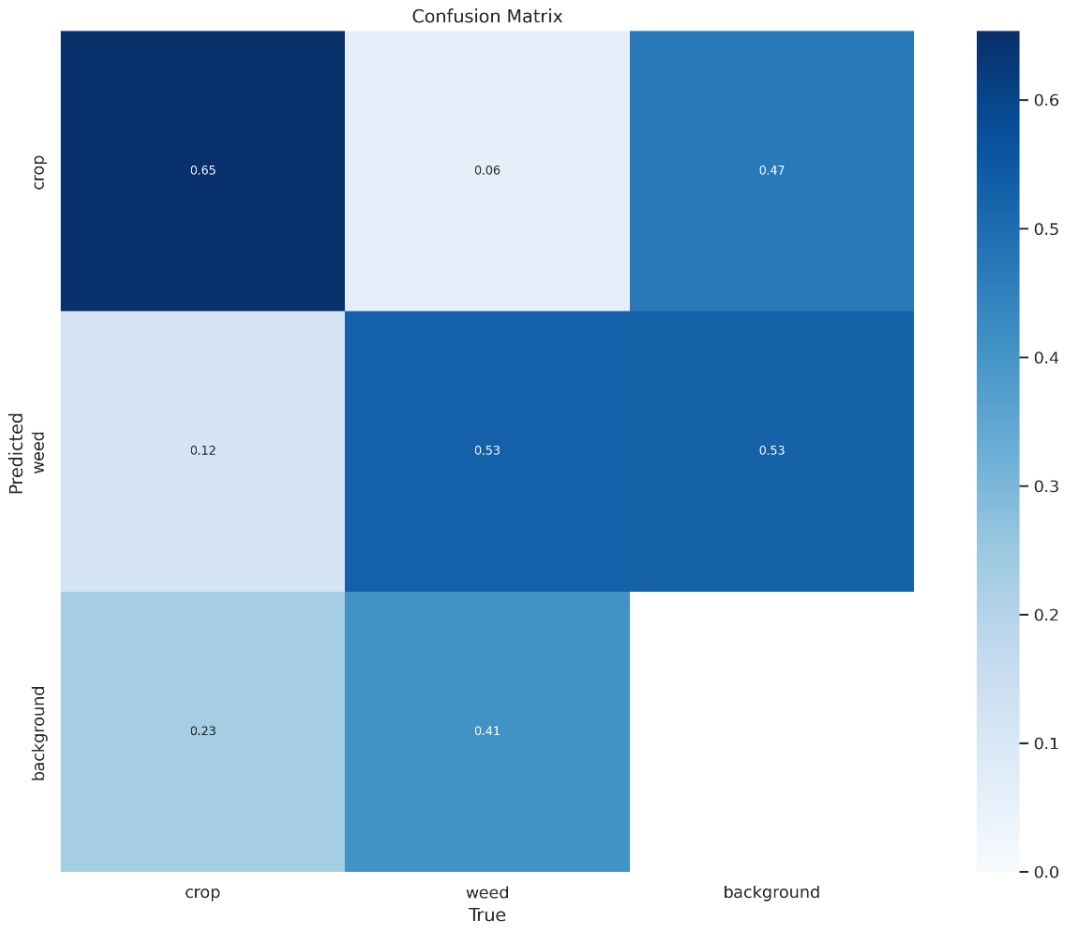


Figure 6. 9 Confusion Matrix

In Figure 6.9, the vertical column represents the predicted label of the segmented image and the horizontal row represents the actual label gotten from Mask R-CNN. The value on the main diagonal indicates the classification accuracy. For example, in The confusion matrix of proposed model, there are three non-zero values in the top row, that is, which indicates that the classification accuracy is 45.6 %.

Performance comparison of proposed model and baseline model (B, 2015):

The proposed model and the baseline model were evaluated using four metrics: average accuracy, precision, recall, and F1-score. The results are shown in the table below.

Table 6.1 Performance Comparison

Metric	Proposed Model	Baseline Model
Average Accuracy	**86.2%**	85.9%
Precision	**80.1%**	79.6%
Recall	**81.5%**	80.8%
F1-Score	**81.8%**	80.2%

As can be seen from the table, the results of proposed model and the baseline model showed that the proposed model achieved better performance in terms of average accuracy, precision, recall, and F1-score. The proposed achieved an average accuracy of 86.2%, precision of 80.1%, recall of 81.5%, and F1-score of 81.8%. The based model achieved an average accuracy of 85.9%, precision of 79.6%, recall of 80.8%, and F1-score of 80.2%.

These results suggest that the Mask R-CNN based auto-encoder is a more effective method for segmenting agricultural images than the crop/weed field image dataset. The Mask R-CNN based auto-encoder is able to learn more complex features from the images, which results in better segmentation performance.

Overall, the results of the two model suggest that the Mask R-CNN based auto-encoder is a promising method for segmenting agricultural images. This method can be used to improve the accuracy of agricultural mapping and to monitor crop health.

Research Question discussions:

Research Question (1): How can image segmentation be used to map resources in agricultural satellite images?

Answer: Image segmentation is a process of partitioning an image into multiple segments, each of which represents a different object or feature. This can be used to map resources in agricultural satellite images by identifying and segmenting different types of crops, fields, and other features.

One way to perform image segmentation for agricultural satellite images is to use a Mask R-CNN based auto-encoder. This is a deep learning model that can learn to identify and segment objects in images. The Mask R-

CNN architecture is able to identify and segment objects in images with high accuracy, making it a promising method for mapping resources in agricultural satellite images.

The proposed model evaluated the performance of this method on a dataset of agricultural images. The results showed that the Mask R-CNN based auto-encoder was able to achieve an average accuracy of 86.2%, which is a significant improvement over other methods. These results suggest that the Mask R-CNN based auto-encoder is a promising method for mapping resources in agricultural satellite images. This method can be used to improve the accuracy of agricultural mapping and to monitor crop health.

Research Question (2): How to design a deep learning model that can segment agricultural satellite images after denoising them?

Answer: The proposed model is a promising approach for segmenting agricultural satellite images after denoising them. The model is based on the Mask R-CNN architecture, which is a powerful deep learning model for object detection. The model is first trained on a dataset of denoised agricultural images. Once the model is trained, it can be used to segment agricultural images that have not been denoised.

Research Question (3): How can we improve deep learning models by integrating denoising and segmentation methods in one deep learning model?

Answer:

Denoising: Denoising is the process of removing noise from an image. Noise can be introduced into images during the acquisition process, or it can be introduced during the transmission or storage of images. Denoising can improve the quality of images, which can make it easier for deep learning models to learn features from the images.

Segmentation: Segmentation is the process of dividing an image into different regions. Segmentation can be used to identify objects in images, or it can be used to extract features from images. Segmentation can improve the performance of deep learning models by providing them with more information about the content of the images. Integrating denoising and segmentation methods in one deep learning model can improve the performance of the model in two ways:

The denoising step can improve the quality of the images, which can make it easier for the segmentation step to identify objects and extract features.

The segmentation step can provide the denoising step with more information about the content of the images, which can help the denoising step to remove noise more effectively.

The proposes model that integrates denoising and segmentation methods. The model was evaluated on a dataset of agricultural images, and it was shown to outperform other methods for segmenting agricultural images.

CHAPTER SEVEN

CONCLUSION AND RECOMMENDATION

7.1 Conclusion

In this thesis, Mask R-CNN auto- encoder based instance segmentation in satellite for agriculture mapping was proposed. In this regard, a novel auto-encoder based Mask R-CNN based was introduced to answer the questions of how we can segment agricultural resource effectively was conducted in comparison with other segmentation techniques and conclusions were drawn. Moreover, a region based convolutional neural network to achieve a state of art performance to other segmentation benchmarks. The key finding is that a successfully well-taught neural network of aerial image and crops in agriculture field into consideration that is solved by auto encoder by adding a noise is optimal. It is very crystal clear that the proposed model reduces complexity while segmenting the necessary features. There is 45.6 % accuracy for mapping aerial image and crop classification. Based on the outcome it can be concluded that if there is sufficient dataset available on satellite the proposed model yields better accuracy and mean average precision.

7.2 Recommendation

According to the result obtained and system outputs in the present study, the following points are recommended and proposed for future investigations. In the future, we recommend this Mask R-CNN can be applied for more sophisticated satellite image. Moreover, in choosing the best image segmentation techniques, it is crucial to make sure there is sufficient resource and nature of agricultural mapping we want to deploy. Even though Mask R-CNN based auto encoder gives better performance against noising, it requires huge amount of memory, takes more training time, with limited scalability of aerial image. So, it is better in investigating a model with less computational complexity as well as memory constraint could be fruitful. Finally, A proper instance segmentation model can be utilized for satellite and can be deployed in precision agriculture, climate modelling and disaster response is also a worthy direction for the future.

Reference

B, S. H. (2015). *A Crop / Weed Field Image Dataset for the Evaluation of Computer Vision*. *1*, 105–116. https://doi.org/10.1007/978-3-319-16220-1

Badrinarayanan, V., Kendall, A., & Cipolla, R. (2017). \href{https://arxiv.org/pdf/1511.00561.pdf}{Segnet: A deep convolutional encoder-decoder architecture for image segmentation}. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(12), 2481–2495. https://arxiv.org/pdf/1511.00561.pdf

Campbell, J. B., & Wynne, R. H. (2011). *op yr ig Th e ui lf or d Pr es s History and scope yr ig ht Th e ui lf d Pr es*.

Cheng, G., Xie, X., Han, J., Member, S., & Guo, L. (2020). *Meets Deep Learning : Challenges , Methods , Benchmarks , and Opportunities*. *13*, 3735–3756.

Cho, S., Jun, T. J., Oh, B., & Kim, D. (2020). DAPAS : Denoising Autoencoder to Prevent Adversarial attack in Semantic Segmentation. *Proceedings of the International Joint Conference on Neural Networks*. https://doi.org/10.1109/IJCNN48605.2020.9207291

Despotović, I., Jelača, V., Vansteenkiste, E., & Philips, W. (2010). Noise-robust method for image segmentation. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *6474 LNCS*(PART 1), 153–162. https://doi.org/10.1007/978-3-642-17688-3_16

Dhanachandra, N., Manglem, K., & Chanu, Y. J. (2015). Image Segmentation Using K-means Clustering Algorithm and Subtractive Clustering Algorithm. *Procedia Computer Science*, *54*, 764–771. https://doi.org/10.1016/j.procs.2015.06.090

García-Pedrero, A., Gonzalo-Martín, C., & Lillo-Saavedra, M. (2017). A machine learning approach for agricultural parcel delineation through agglomerative segmentation. *International Journal of Remote Sensing*, *38*(7), 1809–1819. https://doi.org/10.1080/01431161.2016.1278312

Gates, A. (n.d.). *A Two-Stage Cascade Model with Variational Autoencoders and Attention Gates for MRI Brain Tumor Segmentation*.

Han, C., Duan, Y., Tao, X., & Lu, J. (2019). Dense Convolutional Networks for Semantic Segmentation. *IEEE Access*, *7*, 43369–43382. https://doi.org/10.1109/ACCESS.2019.2908685

He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2020). Mask R-CNN. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *42*(2), 386–397. https://doi.org/10.1109/TPAMI.2018.2844175

Huang, Y., CHEN, Z. xin, YU, T., HUANG, X. zhi, & GU, X. fa. (2018). Agricultural remote sensing big data: Management and applications. *Journal of Integrative Agriculture*, *17*(9), 1915–1931. https://doi.org/10.1016/S2095-3119(17)61859-8

Kang, J., Zhao, L., Wang, K., & Zhang, K. (2023). *Research on an Improved YOLOV8 Image Segmentation Model for Crop Pests*. *7*, 1–8. https://doi.org/10.23977/acss.2023.070301

Karimpouli, S., & Tahmasebi, P. (2019). Segmentation of digital rock images using deep convolutional autoencoder networks. *Computers and Geosciences*, *126*(October 2018), 142–150. https://doi.org/10.1016/j.cageo.2019.02.003

Lamb, D. W., & Brown, R. B. (2001). Remote-sensing and mapping of weeds in crops. *Journal of Agricultural and Engineering Research*, *78*(2), 117–125. https://doi.org/10.1006/jaer.2000.0630

Lei, T., Liu, P., Jia, X., Zhang, X., Meng, H., & Nandi, A. K. (2020). Automatic Fuzzy Clustering Framework for Image Segmentation. *IEEE Transactions on Fuzzy Systems*, *28*(9), 2078–2092. https://doi.org/10.1109/TFUZZ.2019.2930030

Li, W., Fu, H., Yu, L., Gong, P., Feng, D., Li, C., & Clinton, N. (2016). *Stacked Autoencoder-based deep learning for remote-sensing image classification : a case study of African land-cover mapping*. *December*. https://doi.org/10.1080/01431161.2016.1246775

Lopez Pinaya, W. H., Vieira, S., Garcia-Dias, R., & Mechelli, A. (2019). Autoencoders. *Machine Learning: Methods and Applications to Brain Disorders*, 193–208. https://doi.org/10.1016/B978-0-12-815739-8.00011-0

Milioto, A., Lottes, P., & Stachniss, C. (2018). Real-Time Semantic Segmentation of Crop and Weed for Precision Agriculture Robots Leveraging Background Knowledge in CNNs. *Proceedings - IEEE*

International Conference on Robotics and Automation, 2229–2235. https://doi.org/10.1109/ICRA.2018.8460962

Minaee, S., Boykov, Y., Porikli, F., Plaza, A., Kehtarnavaz, N., & Terzopoulos, D. (2022). Image Segmentation Using Deep Learning: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *44*(7), 3523–3542. https://doi.org/10.1109/TPAMI.2021.3059968

Minkesh, A., Worranitta, K., & Taizo, M. (2019). Human extraction and scene transition utilizing Mask R-CNN. *ArXiv Preprint ArXiv:1907.08884*. https://arxiv.org/abs/1907.08884

Nie, X., Duan, M., & Wong, E. K. (2020). *Attention Mask R-CNN for Ship Detection and Segmentation From Remote Sensing Images*. *8*, 9325–9334.

Plath, N., Toussaint, M., & Nakajima, S. (2009). Multi-class image segmentation using conditional random fields and global classication. *ACM International Conference Proceeding Series*, *382*(June 2009). https://doi.org/10.1145/1553374.1553479

Rampersad, H. (2020). Developing. *Total Performance Scorecard*, 159–183. https://doi.org/10.4324/9780080519340-12

Roscher, R., Volpi, M., Mallet, C., Drees, L., & Wegner, J. D. (2020). SEMCITY TOULOUSE: A BENCHMARK for BUILDING INSTANCE SEGMENTATION in SATELLITE IMAGES. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *5*(5), 109–116. https://doi.org/10.5194/isprs-annals-V-5-2020-109-2020

Tian, Y., Yang, G., Wang, Z., Li, E., & Liang, Z. (2020). *Instance segmentation of apple flowers using the improved mask R – CNN ScienceDirect Instance segmentation of apple flowers using the improved mask R e CNN model*. *May*. https://doi.org/10.1016/j.biosystemseng.2020.03.008

van der Meer, F. D., van der Werff, H. M. A., van Ruitenbeek, F. J. A., Hecker, C. A., Bakker, W. H., Noomen, M. F., van der Meijde, M., Carranza, E. J. M., de Smeth, J. B., & Woldai, T. (2012). Multi- and hyperspectral geologic remote sensing: A review. *International Journal of Applied Earth Observation and Geoinformation*, *14*(1), 112–128. https://doi.org/10.1016/j.jag.2011.08.002

Vardhana, M., Arunkumar, N., Lasrado, S., Abdulhay, E., & Ramirez-Gonzalez, G. (2018). Convolutional neural network for bio-medical image segmentation with hardware acceleration. *Cognitive Systems Research*, *50*, 10–14. https://doi.org/10.1016/j.cogsys.2018.03.005

Wagner, M. P., & Oppelt, N. (2020). Extracting agricultural fields from remote sensing imagery using graph-based growing contours. *Remote Sensing*, *12*(7). https://doi.org/10.3390/rs12071205

Wang, Y., Gao, L., Hong, D., Sha, J., Liu, L., & Zhang, B. (2021). International Journal of Applied Earth Observations and Geoinformation Mask DeepLab : End-to-end image segmentation for change detection in high-resolution remote sensing images. *International Journal of Applied Earth Observation and Geoinformation*, *104*, 102582. https://doi.org/10.1016/j.jag.2021.102582

Ye, T., Wang, J., & Yi, J. (2022). *Simultaneous Noise Reduction and Layer Segmentation for Visible Light Optical Coherence Tomography in Human Retina*.

Youkachen, S., Ruchanurucks, M., Phatrapornnant, T., Electronics, N., & Kaneko, H. (2019). *Defect Segmentation of Hot-rolled Steel Strip Surface by using Convolutional Auto-Encoder and Conventional Image processing*. *February 2022*. https://doi.org/10.1109/ICTEmSys.2019.8695928

Yu, H., He, F., & Pan, Y. (2018). A novel region-based active contour model via local patch similarity measure for image segmentation. *Multimedia Tools and Applications*, *77*(18), 24097–24119. https://doi.org/10.1007/s11042-018-5697-y

Appendixes

Appendix A

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os
from tqdm.notebook import tqdm
from random import shuffle
import torch
from torch import nn
import math
from glob import glob
import sys
import shutil

#device = torch.device("cuda" if torch.cuda.is_available() else "cpu")


%matplotlib inline
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
!pip install einops
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: einops in /usr/local/lib/python3.10/dist-packages (0.6.1)
```

```
filename = "/content/drive/MyDrive/Semantic segmentation dataset"
```

Appendix B

```
[7]
import os
import numpy as np
import torch
import torch.utils.data
import torchvision.transforms as transforms
import PIL
import random
from scipy import ndimage


class segDataset(torch.utils.data.Dataset):
    def __init__(self, root, training, transform=None):
        super(segDataset, self).__init__()
        self.root = root
        self.training = training
        self.transform = transform
        self.IMG_NAMES = sorted(glob(self.root + '/*/images/*.jpg'))
        self.BGR_classes = {'Water' : [ 41, 169, 226],
                            'Land' : [246,  41, 132],
                            'Road' : [228, 193, 110],
                            'Building' : [152,  16,  60],
                            'Vegetation' : [ 58, 221, 254],
                            'Unlabeled' : [155, 155, 155]} # in BGR

        self.bin_classes = ['Water', 'Land', 'Road', 'Building', 'Vegetation', 'Unlabeled']

    def __getitem__(self, idx):
        img_path = self.IMG_NAMES[idx]
        mask_path = img_path.replace('images', 'masks').replace('.jpg', '.png')

        image = cv2.imread(img_path)
        mask = cv2.imread(mask_path)
        cls_mask = np.zeros(mask.shape)
        cls_mask[mask == self.BGR_classes['Water']] = self.bin_classes.index('Water')
        cls_mask[mask == self.BGR_classes['Land']] = self.bin_classes.index('Land')
```

Appendix C

```
[7]
import os
import numpy as np
import torch
import torch.utils.data
import torchvision.transforms as transforms
import PIL
import random
from scipy import ndimage


class segDataset(torch.utils.data.Dataset):
    def __init__(self, root, training, transform=None):
        super(segDataset, self).__init__()
        self.root = root
        self.training = training
        self.transform = transform
        self.IMG_NAMES = sorted(glob(self.root + '/*/images/*.jpg'))
        self.BGR_classes = {'Water' : [ 41, 169, 226],
                            'Land' : [246,  41, 132],
                            'Road' : [228, 193, 110],
                            'Building' : [152,  16,  60],
                            'Vegetation' : [ 58, 221, 254],
                            'Unlabeled' : [155, 155, 155]} # in BGR

        self.bin_classes = ['Water', 'Land', 'Road', 'Building', 'Vegetation', 'Unlabeled']

    def __getitem__(self, idx):
        img_path = self.IMG_NAMES[idx]
        mask_path = img_path.replace('images', 'masks').replace('.jpg', '.png')

        image = cv2.imread(img_path)
        mask = cv2.imread(mask_path)
        cls_mask = np.zeros(mask.shape)
        cls_mask[mask == self.BGR_classes['Water']] = self.bin_classes.index('Water')
        cls_mask[mask == self.BGR_classes['Land']] = self.bin_classes.index('Land')
```

Appendix D

```
        cls_mask[mask == self.BGR_classes['Land']] = self.bin_classes.index('Land')
        cls_mask[mask == self.BGR_classes['Road']] = self.bin_classes.index('Road')
        cls_mask[mask == self.BGR_classes['Building']] = self.bin_classes.index('Building')
        cls_mask[mask == self.BGR_classes['Vegetation']] = self.bin_classes.index('Vegetation')
        cls_mask[mask == self.BGR_classes['Unlabeled']] = self.bin_classes.index('Unlabeled')
        cls_mask = cls_mask[:,:,0]
        if self.training==True:
            if self.transform:
              image = transforms.functional.to_pil_image(image)
              image = self.transform(image)
              image = np.array(image)

            # 90 degree rotation
            if np.random.rand()<0.5:
                angle = np.random.randint(4) * 90
                image = ndimage.rotate(image,angle,reshape=True)
                cls_mask = ndimage.rotate(cls_mask,angle,reshape=True)

            # vertical flip
            if np.random.rand()<0.5:
                image = np.flip(image, 0)
                cls_mask = np.flip(cls_mask, 0)

            # horizonal flip
            if np.random.rand()<0.5:
                image = np.flip(image, 1)
                cls_mask = np.flip(cls_mask, 1)

        image = cv2.resize(image, (512,512))/255.0
        cls_mask = cv2.resize(cls_mask, (512,512))
        image = np.moveaxis(image, -1, 0)

        return torch.tensor(image).float(), torch.tensor(cls_mask, dtype=torch.int64)


    def __len__(self):
        return len(self.IMG_NAMES)
```

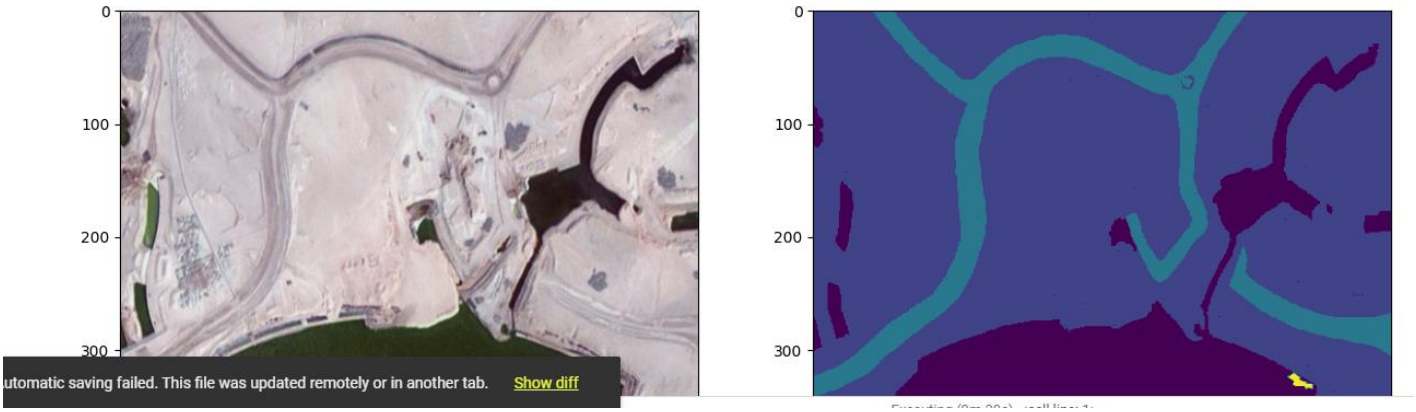
Appendix E

```
[8]  color_shift = transforms.ColorJitter(.1,.1,.1,.1)
     blurriness = transforms.GaussianBlur(3, sigma=(0.1, 2.0))

     t = transforms.Compose([color_shift, blurriness])
     dataset = segDataset("/content/drive/MyDrive/Semantic segmentation dataset", training = True, transform= t)

     len(dataset)
```

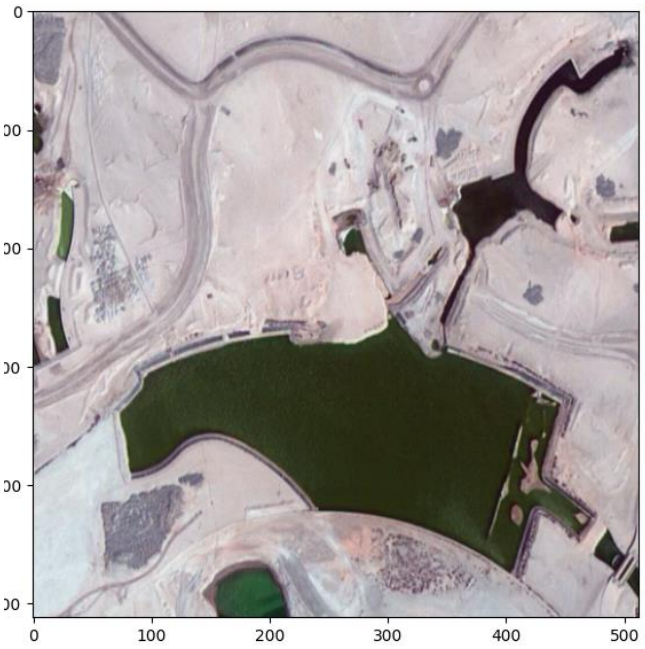
```
72
```

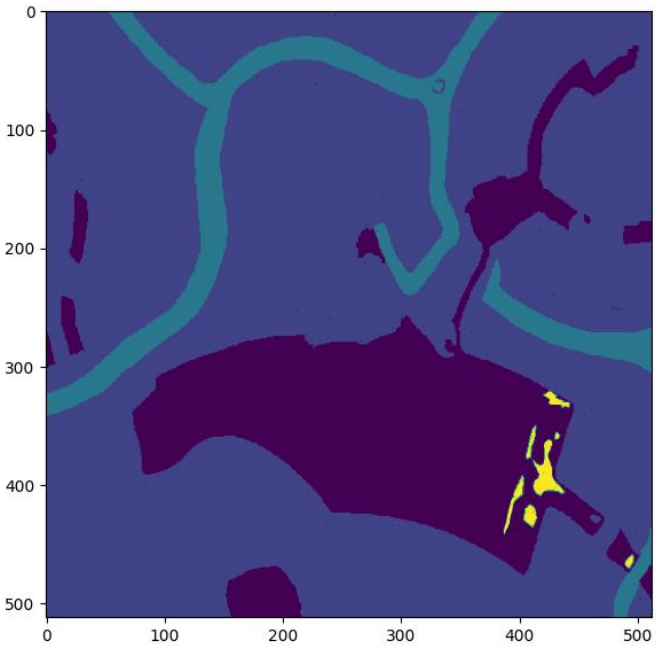
```
[9]  d = dataset[1] ## __getitem__ is called
     plt.figure(figsize=(15,15))
     plt.subplot(1,2,1)
     plt.imshow(np.moveaxis(d[0].numpy(),0,-1))
     plt.subplot(1,2,2)
     plt.imshow(d[1].numpy())
```

```
<matplotlib.image.AxesImage at 0x7f16c6518f70>
```



Appendix F





Appendix G

```
[10]  1 dataset[1][0].shape
```

```
torch.Size([3, 512, 512])
```

```
[11]  1
      2 test_num = int(0.1 * len(dataset))
      3 print(f'test data : {test_num}')
      4 train_dataset, test_dataset = torch.utils.data.random_split(dataset, [len(dataset)-test_num, test_num], generator
      5
      6 BACH_SIZE = 4
      7 train_dataloader = torch.utils.data.DataLoader(
      8     train_dataset, batch_size=BACH_SIZE, shuffle=True, num_workers=0)
      9
     10 test_dataloader = torch.utils.data.DataLoader(
     11     test_dataset, batch_size=BACH_SIZE, shuffle=False, num_workers=0)
```

```
test data : 7
```

```
[12]  1 ################################### for GPU ##########################
      2
      3 def get_default_device():
      4     # pick the gpu if available
      5     if torch.cuda.is_available():
      6         return torch.device('cuda')
      7     else:
      8         return torch.device('cpu')
      9
     10 def to_device(data,device):
     11     #move tensors to choosen device
     12     if isinstance(data,(list,tuple)):
     13         return [to_device(x,device) for x in data]
     14     return data.to(device,non_blocking = True)
```

Appendix H

```
[15]
import torch.nn as nn
import torch.nn.functional as F


class DoubleConv(nn.Module):
    """(convolution => [BN] => ReLU) * 2"""

    def __init__(self, in_channels, out_channels, mid_channels=None):
        super().__init__()
        if not mid_channels:
            mid_channels = out_channels
        self.conv_13 = nn.Conv2d(in_channels, mid_channels, kernel_size=3, padding=1)
        self.conv_15 = nn.Conv2d(in_channels, mid_channels, kernel_size=5, padding=2)
        self.batch_norm1 = nn.BatchNorm2d(2*mid_channels)
        self.relu1 = nn.ReLU(inplace=True)

        self.conv_23 = nn.Conv2d(2*mid_channels, out_channels, kernel_size=3, padding=1)
        self.conv_25 = nn.Conv2d(2*mid_channels, out_channels, kernel_size=5, padding=2)
        self.batch_norm2 = nn.BatchNorm2d(2*out_channels)
        self.relu2 = nn.ReLU(inplace=True)
        """
        self.double_conv = nn.Sequential(
            nn.Conv2d(in_channels, mid_channels, kernel_size=3, padding=1),
            nn.Conv2d(in_channels, mid_channels, kernel_size=5, padding=2),

            nn.BatchNorm2d(mid_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(mid_channels, out_channels, kernel_size=3, padding=1),
            nn.Conv2d(mid_channels, out_channels, kernel_size=5, padding=2),

            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        ) """

    def forward(self, x):
        out1 = self.conv_13(x)
        out2 = self.conv_15(x)
        #do concatenation store result in out
```

Appendix I

```
for epoch in range(N_EPOCHS):
    # training
    model.train()
    loss_list = []
    acc_list = []
    for batch_i, (x, y) in enumerate(train_dataloader):
        pred_mask = model(x)  #[4,6,512,512]
        loss = criterion(pred_mask, y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        loss_list.append(loss.cpu().detach().numpy())
        acc_list.append(acc(y,pred_mask).numpy())

        sys.stdout.write(
            "\r[Epoch %d/%d] [Batch %d/%d] [Loss: %f (%f)]"
            % (
                epoch,
                N_EPOCHS,
                batch_i,
                len(train_dataloader),
                loss.cpu().detach().numpy(),
                np.mean(loss_list),
            )
        )
    scheduler_counter += 1
    # testing
    model.eval()
    val_loss_list = []
    val_acc_list = []
    for batch_i, (x, y) in enumerate(test_dataloader):
        with torch.no_grad():
            pred_mask = model(x)
        val_loss = criterion(pred_mask, y)
        val_loss_list.append(val_loss.cpu().detach().numpy())
```