COMPARATIVE ANALYSIS OF DEEPLABV3+ AND U-NET++ ON PLANT SEGMENTATION

A report submitted in partial fulfillment of the requirements for the award of the Degree of

BACHELOR OF TECHNOLOGY

in

ELECTRONICS AND COMMUNICATION ENGINEERING

by

RAJ SRI GOPAL YAMANA
ID: N200721

Under the supervision of

K.SIVALAL SIR

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES

NUZVID, INDIA

May 6, 2025

DECLARATION

I **Raj Sri Gopal Yamana**, hereby declare that the project report is submitted for partial fulfillment of the requirements for the award of the Bachelor of Technology degree in Electronics and Communication Engineering of the **Rajiv Gandhi University of Knowledge Technologies, Nuzvid** is a bonafide work done by me under the (supervision under) the guidance of **K.SivaLal**. This submission represents our ideas in our own words, and where ideas or words of others have been included, I have adequately and accurately cited and referenced the original sources. I also declare that we have adhered to the ethics of academic honesty and integrity and have not misrepresented or fabricated any data, idea, fact or source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and/or the university and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not previously been formed as the basis for the award of any degree, diploma, or similar title from any other university.

RAJ SRI GOPAL YAMANA(N200721)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES, NUZVID, INDIA

CERTIFICATE

This is to certify that the report entitled **”COMPARATIVE ANALYSIS OF DEEPLABV3+ AND U-NET++ ON PLANT SEGMENTATION”** submitted by **Raj Sri Gopal (N200721)** to the Rajiv Gandhi University of Knowledge Technologies, Nuzvid in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology in Electronics and Communication Engineering is a bonafide record of the project work carried out by him under our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.

Project Guide
Mr.K.SIVALAL Sir
Assistant Professor
Dept. of Electronics and Communication Engineering
RGUKT NUZVID CAMPUS
Nuzvid-521202

Head of the Departement
Mr.K SIVALAL Sir
Dept. of Electronics and Communication Engineering
RGUKT NUZVID CAMPUS
Nuzvid-521202

ACKNOWLEDGMENT

We express our sincere gratitude towards our guide Mr.K Sivaal Sir , Electronics and Communication Engineering, Rajiv Gandhi University of Knowledge Technologies, Nuzvid for his valuable guidance, suggestions, and supervision through- out the work. Without his patronage and guidance, the project would not have taken shape. We would also like to express our regards for his kind approval of the project, time-to-time counseling, and advice. We owe our sincere thanks to the Dept.of Electronics and Communication Engineering, Rajiv Gandhi University of Knowledge Technologies, Nuzvid.

ABSTRACT

This study focuses on improving how plant regions are segmented in agricultural fields using artificial intelligence. Accurately analyzing crop images is often difficult due to overlapping vegetation, inconsistent lighting conditions, and the visual similarity between different parts of the plants. To address these challenges, we employ two advanced deep learning models — DeepLabV3+ and U-Net++ known for their powerful semantic segmentation capabilities.

The goal of this project is to automatically segment plant regions from field images to support tasks such as monitoring plant health and optimizing field management. We trained both models using custom-generated segmentation masks from bounding box annotations provided in YOLO format, ensuring disease-wise separation during pre-processing. Each model's performance was quantitatively evaluated using standard segmentation metrics including Intersection over Union (IoU), Dice Similarity Coefficient, and Pixel Accuracy.

Experimental results demonstrate that both models achieved strong performance on the segmentation task, with DeepLabV3+ slightly outperforming U-Net++ across most metrics. These findings confirm the value of semantic segmentation in precision agriculture and its potential role in enhancing automated plant analysis systems.

Contents

List of Figures

Chapter 1

INTRODUCTION

1.1 Introduction

Plant infestations pose a significant challenge to global agricultural productivity, leading to substantial crop yield losses and higher farming costs. Early and accurate segmentation of plants is crucial for effective crop management and ensuring sustainable farming practices. Deep learning techniques have demonstrated considerable potential in advancing agricultural analysis, particularly in image segmentation tasks. In this project, we explore different deep learning architectures, specifically DeepLabV3+ and U-Net++, for segmenting agricultural images to extract plant regions. The dataset used for this study consists of field images with custom-generated segmentation masks derived from YOLO-format annotations. The goal is to evaluate the effectiveness of these models in segmenting plant regions to enhance precision agriculture applications and improve crop monitoring

1.2 Objective of the project

- To develop an AI-based system that can automatically segment plant species from agricultural images.

- To apply the DeepLabV3+ and U-Net++ models, both powerful deep learning architectures, for more accurate and efficient segmentation results in agriculture image analysis.

- To improve the identification and classification of plant diseases, enabling better disease management strategies and more precise yield predictions.

- To test and validate the models on a custom dataset, containing images of plant diseases with segmentation masks generated from YOLO-format annotations.

- To demonstrate that the advanced deep learning-based segmentation methods (DeepLabV3+ and U-Net++) outperform traditional image processing methods in segmentation accuracy and overall performance.

1.3 Scope of the project

1. To develop an AI-based method for automatically detecting and segmenting plant species in agricultural images.

2. To improve accuracy and reliability in plant disease detection by using DeepLabV3+ and U-Net++, two powerful deep learning models designed for semantic segmentation.

3. To reduce the need for manual intervention in disease identification, saving time and minimizing human errors in agricultural monitoring and management.

4. To extract crucial disease-specific information for better crop management decisions and timely interventions to control the spread of harmful weeds.

5. To build a system that can be integrated into real-world agricultural settings for faster, more accurate, and consistent disease detection in precision farming.

Chapter 2

LITERATURE REVIEW

2.1 Traditional Image Processing Methods

Early methods used basic image processing techniques like:
1. Thresholding
This method separates parts of the image based on brightness or intensity levels. If a pixel is brighter than a certain value, it's marked as part of the object (like diseased plants or weeds).
2. Edge Detection
Edge detection finds the borders of objects by looking for sudden changes in brightness. This helps outline the shape of plant structures, such as leaves or stems.
3. Region-Based Segmentation
This technique groups together neighboring pixels that look similar. It tries to find regions (like areas affected by disease) that are connected and have similar textures or colors.

2.2 Deep Learning Approaches

1. Convolutional Neural Networks (CNNs): CNNs are the foundation of most deep learning-based segmentation models. These models automatically learn hierarchical features from raw images, making them ideal for complex tasks like plant disease segmentation. CNNs can identify intricate patterns in image data, such as color or texture variations that indicate plant diseases or weed growth.

2. U-Net Architecture: U-Net is a deep learning model specifically designed for image segmentation tasks. It uses an encoder-decoder structure with skip connections to improve segmentation accuracy, even with limited data. In plant disease segmentation, U-Net helps to identify and delineate diseased regions in crop images, providing pixel-wise segmentation masks that show where diseases like Ragweed or Pigweed are located.

3. DeepLabV3+ Architecture: DeepLabV3+ is a state-of-the-art model that employs atrous convolution and spatial pyramid pooling (ASPP) to capture multi-scale context information. This enables the model to segment complex agricultural images, even when plants or diseases appear in varying sizes or orientations. DeepLabV3+ has been successfully applied to various segmentation tasks, including plant disease identification in agricultural fields.

Main Advantage: Unlike traditional methods that require manual feature selection, deep learning models like CNNs, U-Net, and DeepLabV3+ can automatically learn relevant features from data, significantly reducing the need for manual intervention and improving segmentation performance across diverse datasets.

2.3 Deep Learning Models

Deep learning has revolutionized image segmentation tasks in agriculture by providing models that can learn complex patterns and offer pixel-level accuracy. In this study, we implemented and compared two powerful models: **DeepLabV3+** and **U-Net++** for plant segmentation.

- **Convolutional Neural Networks (CNNs)**
  CNNs are the backbone of most deep learning architectures in image segmentation. They use convolutional layers to automatically extract features such as edges, textures, and shapes from images. CNNs enable efficient feature learning and are critical for tasks like identifying plant regions in agricultural fields.

- **DeepLabV3+**
  DeepLabV3+ is an advanced semantic segmentation model that extends DeepLabV3 by adding a decoder module for better object boundary recovery.
  1. It employs Atrous Spatial Pyramid Pooling (ASPP) to capture features at multiple scales, which is critical when segmenting plants of varying sizes and structures.
  2. In our study, DeepLabV3+ produced high-quality segmentation results across all disease categories, with superior performance in most evaluation metrics.

- **U-Net++**
  U-Net++ is an improved version of U-Net, designed to enhance segmentation accuracy and feature fusion.
  1. It introduces nested and dense skip pathways that bridge the semantic gap between encoder and decoder sub-networks.
  2. This architecture allows better refinement of segmentation boundaries, which is valuable in distinguishing intricate plant disease regions.

3. In our experiments, U-Net++ demonstrated strong performance and competitive results when compared to DeepLabV3+.

Advantages of Deep Learning in Agricultural Image Segmentation:

The application of deep learning techniques, particularly in the realm of agricultural image segmentation, offers numerous advantages that can significantly enhance crop monitoring and disease management in precision agriculture.

1. **High Accuracy:** Deep learning models, especially Convolutional Neural Networks (CNNs), U-Net, and Fully Convolutional Networks (FCNs), have consistently demonstrated superior performance in image segmentation tasks. These models are adept at distinguishing complex patterns within agricultural images, such as the identification and segmentation of different plant species, weeds, and crop diseases. Their accuracy makes them invaluable tools for tasks that require precise segmentation, such as automated crop monitoring and plant health assessment.

2. **Fast and Efficient:** After being trained on large datasets, deep learning models can process new agricultural images with remarkable speed. This capability is crucial for real-time applications, such as continuous crop health monitoring and automated weed control systems. The ability to quickly analyze field images enhances the efficiency of farming practices, enabling timely interventions that can minimize losses due to diseases or pest infestations.

3. **Generalization:** Deep learning models are highly versatile and, when trained on diverse datasets, can generalize across various plant species, disease types (such as the identification of common agricultural diseases), imaging angles, and environmental conditions. This generalization ability allows these models to be deployed across different agricultural environments, making them adaptable for a wide range of use cases. Whether it's varying field conditions, different plant species, or shifts in environmental factors, these models can maintain their performance and provide accurate results in diverse real-world scenarios.

4. **Scalability:** As the size of agricultural operations continues to grow, so too does the need for scalable solutions. Deep learning models are easily scalable, meaning they can handle vast amounts of data from large-scale farms. With continuous advancements in cloud computing and processing power, these models can scale to meet the demands of large agricultural datasets, making them ideal for both small farms and large commercial operations.

Chapter 3

METHODOLOGY

3.1 Overview of DEEPLABV3+ Architecture

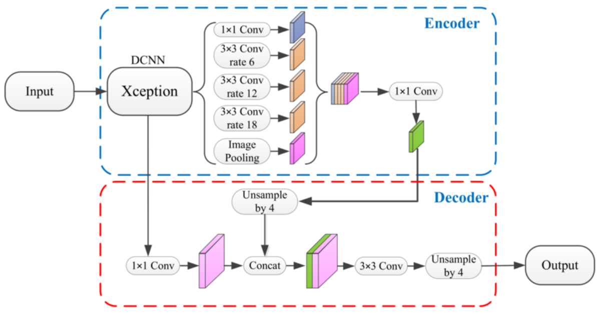

Figure 3.1: Block diagram of Gan

DeepLabV3+ is a state-of-the-art deep learning model used for semantic image segmentation, particularly effective in tasks like segmenting agricultural images. In this project, it is applied to segment different plant types and diseases, such as Ragweed, Foxtail, and other crop-related diseases.

DeepLabV3+ improves upon its predecessor (DeepLabV3) by incorporating an encoder-decoder structure and atrous spatial pyramid pooling (ASPP), which allows it to capture multi-scale context and preserve finer details in the segmentation task.

- **ASPP (Atrous Spatial Pyramid Pooling):** The ASPP module is a key part of DeepLabV3+. It uses dilated convolutions at different rates to capture multi-scale contextual information. This is particularly useful for detecting objects (such as different parts of plants or disease areas) at varying sizes and distances in the input image.

- **Decoder:** The decoder helps refine the segmentation result by recovering fine-grained details. It takes the high-level features extracted through the ASPP module and reconstructs the segmentation output at a pixel level, ensuring accurate segmentation of small and intricate regions in the image.

3.2 U-Net++ Generator Architecture

U-Net++ Generator Architecture:
U-Net++ is an advanced version of the original U-Net, designed to address the limitations of U-Net in terms of segmentation accuracy by incorporating dense skip pathways and deeper hierarchical structures. U-Net++ improves upon the basic U-Net by using a more complex architecture with nested and dense skip pathways, helping the network better handle intricate segmentation tasks, like the segmentation of different plant diseases from agricultural images.

- **Encoder (Contracting Path):**
  1. The encoder in U-Net++ is responsible for downsampling the image and learning high-level features. It captures global semantic information like plant shapes, disease regions, and their context.
  2. It uses standard convolutional layers followed by max-pooling to reduce the spatial dimensions while extracting important features.
  3. This part summarizes the essential visual features of the input image, which are crucial for understanding the plant and disease regions.

- **Decoder (Expanding Path):**
  1. The decoder upsamples the feature maps generated by the encoder, progressively rebuilding the image's resolution to produce a detailed segmentation mask.
  2. U-Net++ introduces dense skip connections, where each layer receives feature maps from not just the previous layer but from earlier stages of the encoder. This helps to refine the segmentation, ensuring that small but important features, like leaf patterns or disease spots, are accurately reconstructed.
  3. The decoder's role is to produce fine-grained segmentation, allowing the model to distinguish complex regions within the plant disease images.

- **Dense Skip Connections:**
  1. A unique feature of U-Net++ is the dense skip connections between the encoder and decoder. Unlike U-Net, which uses direct skip connections, U-Net++ employs nested and dense connections, allowing the model to preserve and utilize more information from the encoder.
  2. These skip connections help retain low-level details and prevent the loss of critical features when the image is downsampled.
  3. The dense skip connections form a comprehensive hierarchical structure, enabling the model to learn more complex representations, which is crucial for accurately segmenting the fine details in plant images and identifying disease regions.

3.3 Loss Functions

Segmentation Loss (L1 or Dice Loss)

- The segmentation loss evaluates how accurately the model's predicted segmentation matches the ground truth (manually labeled by an expert).
- This is similar to comparing two images: one drawn by a human expert, and one generated by the model. The model gets penalized if its segmentation deviates from the expected boundaries.
- The closer the match between predicted and ground truth segmentation, the smaller the loss, and the better the model's performance.

Common Segmentation Loss Types:

L1 Loss:

- Measures the average absolute difference between the predicted segmentation mask and the ground truth mask. Lower values indicate a closer match.

Dice Loss:

- Measures the overlap between the predicted segmentation area and the true area, similar to comparing puzzle pieces to see how well they fit together. A higher Dice score indicates better segmentation accuracy.

Adversarial Loss (from U-Net++)

- In the context of the U-Net++ model, adversarial loss is utilized when the network is enhanced with a GAN structure, where the generator aims to produce more realistic segmentation maps.
- The discriminator in the GAN framework distinguishes between real (ground truth) and generated (predicted by the U-Net++) segmentation maps.
- The goal is for the generator to create segmentation maps that closely resemble the real, expert-labeled maps. A lower adversarial loss means the segmentation generated by U-Net++ is indistinguishable from the ground truth, indicating better performance.

3.4 Training Procedure

- **Input Data**
  1. The model is trained on a large dataset of 2D plant disease images with their corresponding segmentation masks.
  2. Each image is paired with the correct segmentation mask, which acts as the ground truth and is used as a guide during training.

- **U-Net++ Model Makes a Prediction**
  The U-Net++ model takes an input image and generates a segmentation mask. It tries to predict which parts of the image correspond to different plant diseases (such as Ragweed, Cocklebur, Foxtail, and Pigweed).

- **Validation Loss Calculation**
  1. During training, the model's performance is evaluated using the validation set (images not seen during training).
  2. The validation loss measures how well the model's predicted segmentation matches the true segmentation mask for the validation images. This provides feedback on the model's accuracy and helps prevent overfitting.

- **Loss is Calculated**
  The primary loss function used during training is the validation loss, which can be computed using segmentation metrics such as Dice Loss or L1 Loss. The validation loss gives an indication of the model's generalization ability on unseen data.

- **Model Updates Itself**
  1. Based on the validation loss, the model adjusts its parameters (weights) to minimize the error and improve its performance on future predictions.
  2. This step involves backpropagation, where the gradients are computed and used to update the model's weights through optimization algorithms like Adam.

- **Repeat and Improve**
  1. The training process is repeated over multiple epochs, using both the training set and the validation set.
  2. Over time, the model learns to produce increasingly accurate segmentations, as evidenced by decreasing validation loss.

3.5 Dice Score

- The Dice Score, also known as the Dice Similarity Coefficient (DSC), measures how much the generated segmentation overlaps with the ground truth segmentation.

- It is computed using the formula:

$$DiceScore = \frac{2 \times |A \cap B|}{|A| + |B|}$$

  where A is the predicted segmentation and B is the ground truth segmentation.

- The Dice Score ranges from 0 to 1, where 0 indicates no overlap, and 1 represents a perfect match between the predicted and ground truth segmentations.

- A higher Dice Score indicates better model performance, as it shows a greater overlap between the predicted and actual heart boundaries in the segmentation task.

- The Dice Score is particularly useful when dealing with imbalanced datasets, as it penalizes false negatives (areas missed by the segmentation) and false positives (areas incorrectly segmented).

3.6 Intersection over Union (IoU)

- Intersection over Union (IoU) is a metric used to evaluate the performance of image segmentation models.

- It measures the overlap between the predicted segmentation and the ground truth, similar to the Dice Score but with a different formula.

- The IoU is calculated as:

$$IoU = \frac{|A \cap B|}{|A \cup B|}$$

  where A is the predicted segmentation, B is the ground truth segmentation, and $\cap$ and $\cup$ represent the intersection and union of the two sets, respectively.

- IoU ranges from 0 to 1, where a value of 1 indicates a perfect overlap (i.e., the predicted segmentation is exactly the same as the ground truth).

- In the context of heart segmentation, a higher IoU value indicates that the predicted segmentation closely matches the actual heart structure, including the myocardium, endocardium, and other key parts.

- A lower IoU value reflects a poor model performance, with less overlap between the predicted and actual regions.

- IoU is widely used in object detection and segmentation tasks, as it provides a clear measure of how well the model can delineate the target structures.

Chapter 4

Implementation

4.1 4.1 Dataset Preparation

In this project, we worked on segmenting agricultural images to identify plant diseases such as Ragweed, Cocklebur, Foxtail, and Pigweed. Each disease was treated as a separate segmentation task using YOLO-based bounding box annotations which were later converted into binary masks suitable for semantic segmentation.

- The dataset consists of 2D images in `.png` format with corresponding annotation files in `.txt` format (YOLO format).
- Annotation files include bounding box coordinates, which were converted into binary segmentation masks.
- Masks were generated separately for each disease type by processing the annotations one type at a time.
- Image and mask resizing, normalization, and data augmentation were applied to prepare the dataset for training.

4.2 4.2 Data Collection and Labelling

The dataset was manually labeled using YOLO-format annotations, indicating bounding boxes around specific plant diseases. No polygonal or per-pixel annotations were available, so we generated segmentation masks from bounding boxes.

- All data was collected from agricultural image datasets curated for crop vs. weed segmentation and land cover classification.
- Each image was manually reviewed to confirm the presence of one of the four targeted diseases.

- Annotation files were created using tools such as LabelImg, storing bounding box data in YOLO format.
- Binary masks were generated programmatically, filling in the bounding box area with white (foreground) and background with black.

4.3 4.3 Data Splitting

To ensure reliable model training and evaluation, the dataset was split into training and testing sets.

- Each disease type was processed separately — images and their corresponding masks were grouped and then split.
- A standard 80-20 split was used: 80% for training and 20% for testing.
- This splitting ensures the model is trained on a wide variety of samples and evaluated on unseen data.
- Data shuffling was used before splitting to avoid bias and ensure randomness in selection.
- The same split was maintained across both DeepLabV3+ and U-Net++ model training phases for consistent comparison.

Chapter 5

CODE IMPLEMENTATION

```
import cv2
import numpy as np
import os
image_dir = "/content/images"
txt_dir = "/content/txt"
mask_dir = "/content/masks"
os.makedirs(mask_dir, exist_ok=True)
image_filenames = [f for f in os.listdir(image_dir) if f.endswith(".jpg")]
for img_name in image_filenames:
    img_path = os.path.join(image_dir, img_name)
    txt_path = os.path.join(txt_dir, img_name.replace(".jpg", ".txt"))
    mask_path = os.path.join(mask_dir, img_name.replace(".jpg", ".png"))
    image = cv2.imread(img_path)
    if image is None:
        print(f"ERROR: Could not read {img_path}")
        continue
    height, width, _ = image.shape
    mask = np.zeros((height, width), dtype=np.uint8)
    if os.path.exists(txt_path):
        with open(txt_path, "r") as file:
            for line in file:
                values = line.split()
                if len(values) != 5:
                    print(f"Skipping invalid annotation in {txt_path})
                    continue
                x1 = int((x_center - bbox_width / 2) * width)
                y1 = int((y_center - bbox_height / 2) * height)
                x2 = int((x_center + bbox_width / 2) * width)
                y2 = int((y_center + bbox_height / 2) * height)
        cv2.imwrite(mask_path, mask)
        print(f" Mask saved: {mask_path}")

    else:
        print(f"No annotation file for {img_name}")
import cv2
import numpy as np
import os
from torch.utils.data import Dataset, DataLoader
import torch
import albumentations as A
from albumentations.pytorch import ToTensorV2

class AgricultureSegmentationDataset(Dataset):
    def __init__(self, image_dir, mask_dir, transform=None):
        self.image_dir = image_dir
        self.mask_dir = mask_dir
        self.transform = transform

    def __len__(self):
        return len(self.image_filenames)

    def __getitem__(self, idx):
        img_path = os.path.join(self.image_dir, self.image_filenames[idx])
        mask_path = os.path.join(self.mask_dir, self.image_filenames[idx])
        image = cv2.imread(img_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)
        if self.transform:
            augmented = self.transform(image=image, mask=mask)
            image, mask = augmented['image'], augmented['mask']
        mask = torch.tensor(mask, dtype=torch.long)
        return image, mask
transform = A.Compose([
    A.Resize(256, 256),
    A.RandomCrop(width=256, height=256),
    A.HorizontalFlip(p=0.5),
    A.RandomRotate90(p=0.5),
    A.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
    ToTensorV2(),
])
image_dir = "/content/images"
mask_dir = "/content/masks"
dataset = AgricultureSegmentationDataset(image_dir, mask_dir, transform=transform)
dataloader = DataLoader(dataset, batch_size=4, shuffle=True)
for images, masks in dataloader:
    print(f"Image batch shape: {images.shape}, Mask batch shape: {masks.shape}")
    break
import cv2
import numpy as np
import os
from torch.utils.data import Dataset, DataLoader
import torch
import albumentations as A
from albumentations.pytorch import ToTensorV2
from sklearn.model_selection import train_test_split
class AgricultureSegmentationDataset(Dataset):
    def __init__(self, image_filenames, image_dir, mask_dir, transform=None):
        self.image_filenames = image_filenames
        self.image_dir = image_dir
        self.mask_dir = mask_dir
        self.transform = transform
    def __len__(self):
        return len(self.image_filenames)

    def __getitem__(self, idx):
        img_path = os.path.join(self.image_dir, self.image_filenames[idx])
        mask_path = os.path.join(self.mask_dir, self.image_filenames[idx])
        image = cv2.imread(img_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)
        mask = mask.astype(np.uint8)
        mask[mask > 3] = 3
        if self.transform:
            augmented = self.transform(image=image, mask=mask)
            image, mask = augmented['image'], augmented['mask']
        if not isinstance(mask, torch.Tensor):
            mask = torch.from_numpy(mask).long()
        return image, mask
image_dir = "/content/images"
```

```
mask_dir = "/content/masks"
all_image_filenames = [f for f in os.listdir(image_dir) if f.endswith(".jpg")]
train_files, test_files = train_test_split(all_image_filenames, test_size=0.2,)
train_dataset = AgricultureSegmentationDataset(train_files, image_dir, mask_dir)
test_dataset = AgricultureSegmentationDataset(test_files, image_dir, mask_dir)

train_loader = DataLoader(train_dataset, batch_size=4, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=4, shuffle=False)
for images, masks in train_loader:
    print(f"Train Batch - Image shape: {images.shape}, Mask shape: {masks.shape}")
    break    Just check the first batch
for images, masks in test_loader:
    print(f"Test Batch - Image shape: {images.shape}, Mask shape: {masks.shape}")
    break
import torch
import numpy as np
import matplotlib.pyplot as plt
def optimize_masks(masks):
    masks = masks.float()
    masks = (masks > 0).long()
    return masks
def visualize_masks(dataloader, num_samples=4):
    images, masks = next(iter(dataloader))
    images = images.permute(0, 2, 3, 1).cpu().numpy()
    masks = optimize_masks(masks.cpu())
    fig, axes = plt.subplots(num_samples, 2, figsize=(10, num_samples * 3))
    for i in range(num_samples):
        axes[i, 0].imshow(images[i])
        axes[i, 0].set_title("Image")
        axes[i, 0].axis("off")
        axes[i, 1].imshow(masks[i], cmap="gray")
        axes[i, 1].set_title("Optimized Mask")
        axes[i, 1].axis("off")
    plt.tight_layout()
    plt.show()
visualize_masks(train_loader)
class CustomDataset(Dataset):
    def __init__(self, images_dir, masks_dir, transform=None):
        self.images_dir = images_dir
        self.masks_dir = masks_dir
        self.image_filenames = sorted(os.listdir(images_dir))
        self.mask_filenames = sorted(os.listdir(masks_dir))
        self.transform = transform
    def __len__(self):
        return len(self.image_filenames)
    def __getitem__(self, idx):
        image_path = os.path.join(self.images_dir, self.image_filenames[idx])
        mask_path = os.path.join(self.masks_dir, self.mask_filenames[idx])

        image = cv2.imread(image_path)
        mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)

          Check if image and mask are loaded correctly
        if image is None or mask is None:
            raise ValueError(f"Error loading image or mask for index {idx}")

          Ensure the same shape for image and mask
        if self.transform:
            augmented = self.transform(image=image, mask=mask)
            image, mask = augmented['image'], augmented['mask']

          Convert mask to long tensor (required for cross-entropy loss)
        mask = torch.tensor(mask, dtype=torch.long)

        return image, mask

  Define transformations with Resize to ensure image and mask have same size
transform = A.Compose([
    A.Resize(256, 256),Resize both image and mask to the same size
    A.RandomCrop(width=256,height=256),    Apply random cropping if needed
    A.RandomRotate90(p=0.5),Randomly rotate images and masks
    A.HorizontalFlip(p=0.5),Flip images and masks horizontally
    A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
    Normalize image
    ToTensorV2()
], additional_targets={'mask': 'mask'}, is_check_shapes=False)
images_dir = '/content/images'
masks_dir = '/content/masks'
dataset = CustomDataset(images_dir, masks_dir, transform=transform)
dataloader = DataLoader(dataset, batch_size=4, shuffle=True)
```

```
for images, masks in dataloader:
    print(f"Image batch shape: {images.shape}, Mask batch shape: {masks.shape}")
    break
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = models.segmentation.deeplabv3_resnet50(pretrained=True)
model.classifier[4] = nn.Conv2d(256, 2, kernel_size=(1, 1))    For binary classes
model = model.to(device)
criterion = nn.CrossEntropyLoss(ignore_index=255)
optimizer = optim.Adam(model.parameters(), lr=1e-4)
def train_model(model, train_loader, val_loader, criterion, optimizer, epochs=10):
    model.train()
    loss_history, acc_history, iou_history, dice_history = [], [], [], []
    for epoch in range(epochs):
        epoch_loss = 0.0
        correct_pixels = 0
        total_pixels = 0
        intersection = 0
        union = 0
        dice_total = 0
        for images, masks in train_loader:
            images, masks = images.to(device), masks.to(device)
            optimizer.zero_grad()
            outputs = model(images)['out']
            loss = criterion(outputs, masks)
            loss.backward()
            optimizer.step()
            epoch_loss += loss.item()
            preds = torch.argmax(outputs, dim=1)
            correct_pixels += (preds == masks).sum().item()
            total_pixels += masks.numel()
            inter = ((preds == 1) & (masks == 1)).sum().item()
            union_ = ((preds == 1) | (masks == 1)).sum().item()
            dice = (2 * inter) / (preds.eq(1).sum().item() + masks.eq(1).sum().item()
            intersection += inter
            union += union_
            dice_total += dice
        avg_loss = epoch_loss / len(train_loader)
        accuracy = correct_pixels / total_pixels
        iou = intersection / (union + 1e-6)
        dice_score = dice_total / len(train_loader)
        loss_history.append(avg_loss)
        acc_history.append(accuracy)
        iou_history.append(iou)
        dice_history.append(dice_score)
    return model, loss_history, acc_history, iou_history, dice_history
plt.figure(figsize=(15, 4))
plt.subplot(1, 4, 1)
plt.plot(loss_hist, label="Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Training Loss")
plt.legend()
plt.subplot(1, 4, 2)
plt.plot(acc_hist, label="Accuracy", color='green')
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.title("Training Accuracy")
plt.legend()
plt.subplot(1, 4, 3)
plt.plot(iou_hist, label="IoU", color='orange')
plt.xlabel("Epoch")
plt.ylabel("IoU")
plt.title("Intersection over Union")
plt.legend()
plt.subplot(1, 4, 4)
plt.plot(dice_hist, label="Dice", color='red')
plt.xlabel("Epoch")
plt.ylabel("Dice Coefficient")
plt.title("Dice Score")
plt.legend()
plt.tight_layout()
plt.show()
import os
import cv2
import torch
import random
import numpy as np
from torch import nn, optim
from torch.utils.data import Dataset, DataLoader, random_split
import torchvision.transforms as T
import albumentations as A
from albumentations.pytorch import ToTensorV2
from tqdm import tqdm
import matplotlib.pyplot as plt
image_dir = '/content/images'
mask_dir = '/content/masks'
image_size = 256
batch_size = 4
epochs = 6
lr = 1e-3
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
class SegmentationDataset(Dataset):
    def __init__(self, image_dir, mask_dir, transform=None):
        self.image_dir = image_dir
        self.mask_dir = mask_dir
        self.transform = transform
        self.images = sorted(os.listdir(image_dir))
        self.masks = sorted(os.listdir(mask_dir))
    def __len__(self):
        return len(self.images)
    def __getitem__(self, idx):
        img_path = os.path.join(self.image_dir, self.images[idx])
        mask_path = os.path.join(self.mask_dir, self.masks[idx])
        image = cv2.imread(img_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)
        if self.transform:
            augmented = self.transform(image=image, mask=mask)
            image = augmented['image']
            mask = augmented['mask']
        else:
            image = T.ToTensor()(image)
            mask = torch.tensor(mask, dtype=torch.float32)
        mask = (mask > 127).float()  # Binarize
        return image, mask
train_transform = A.Compose([
    A.Resize(image_size, image_size),
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.2),
    A.Normalize(),
    ToTensorV2(),
])
val_transform = A.Compose([
    A.Resize(image_size, image_size),
    A.Normalize(),
    ToTensorV2(),
])
full_dataset = SegmentationDataset(image_dir, mask_dir, transform=train_transform)
train_size = int(0.8 * len(full_dataset))
val_size = len(full_dataset) - train_size
train_dataset, val_dataset = random_split(full_dataset, [train_size, val_size])
val_dataset.dataset.transform = val_transform
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size)
class ConvBlock(nn.Module):
    def __init__(self, in_ch, out_ch):
        super().__init__()
        self.block = nn.Sequential(
            nn.Conv2d(in_ch, out_ch, 3, padding=1),
            nn.BatchNorm2d(out_ch),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_ch, out_ch, 3, padding=1),
            nn.BatchNorm2d(out_ch),
            nn.ReLU(inplace=True),
        )
    def forward(self, x):
        return self.block(x)
class NestedUNet(nn.Module):
    def __init__(self, in_channels=3, out_channels=1):
        super().__init__()
        nb_filter = [64, 128, 256, 512, 1024]
        self.pool = nn.MaxPool2d(2, 2)
        self.up = lambda x: nn.functional.interpolate(x, scale_factor=2,)
        self.conv0_0 = ConvBlock(in_channels, nb_filter[0])
        self.conv1_0 = ConvBlock(nb_filter[0], nb_filter[1])
        self.conv2_0 = ConvBlock(nb_filter[1], nb_filter[2])
        self.conv3_0 = ConvBlock(nb_filter[2], nb_filter[3])
        self.conv4_0 = ConvBlock(nb_filter[3], nb_filter[4])
        self.conv0_1 = ConvBlock(nb_filter[0]+nb_filter[1], nb_filter[0])
        self.conv1_1 = ConvBlock(nb_filter[1]+nb_filter[2], nb_filter[1])
        self.conv2_1 = ConvBlock(nb_filter[2]+nb_filter[3], nb_filter[2])
        self.conv3_1 = ConvBlock(nb_filter[3]+nb_filter[4], nb_filter[3])
        self.conv0_2 = ConvBlock(nb_filter[0]*2+nb_filter[1], nb_filter[0])
        self.conv1_2 = ConvBlock(nb_filter[1]*2+nb_filter[2], nb_filter[1])
        self.conv2_2 = ConvBlock(nb_filter[2]*2+nb_filter[3], nb_filter[2])
        self.conv0_3 = ConvBlock(nb_filter[0]*3+nb_filter[1], nb_filter[0])
        self.conv1_3 = ConvBlock(nb_filter[1]*3+nb_filter[2], nb_filter[1])
        self.conv0_4 = ConvBlock(nb_filter[0]*4+nb_filter[1], nb_filter[0])
        self.final = nn.Conv2d(nb_filter[0], out_channels, kernel_size=1)
    def forward(self, x):
        x0_0 = self.conv0_0(x)
        x1_0 = self.conv1_0(self.pool(x0_0))
        x0_1 = self.conv0_1(torch.cat([x0_0, self.up(x1_0)], 1))
        x2_0 = self.conv2_0(self.pool(x1_0))
        x1_1 = self.conv1_1(torch.cat([x1_0, self.up(x2_0)], 1))
        x0_2 = self.conv0_2(torch.cat([x0_0, x0_1, self.up(x1_1)], 1))
        x3_0 = self.conv3_0(self.pool(x2_0))
        x2_1 = self.conv2_1(torch.cat([x2_0, self.up(x3_0)], 1))
        x1_2 = self.conv1_2(torch.cat([x1_0, x1_1, self.up(x2_1)], 1))
        x0_3 = self.conv0_3(torch.cat([x0_0, x0_1, x0_2, self.up(x1_2)], 1))
        x4_0 = self.conv4_0(self.pool(x3_0))
        x3_1 = self.conv3_1(torch.cat([x3_0, self.up(x4_0)], 1))
        x2_2 = self.conv2_2(torch.cat([x2_0, x2_1, self.up(x3_1)], 1))
        x1_3 = self.conv1_3(torch.cat([x1_0, x1_1, x1_2, self.up(x2_2)], 1))
        x0_4 = self.conv0_4(torch.cat([x0_0, x0_1, x0_2, x0_3, self.up(x1_3)], 1))
        return self.final(x0_4)
def pixel_accuracy(preds, masks):
    preds = (preds > 0.5).float()
    correct = (preds == masks).float()
    return correct.mean().item()
def iou_score(preds, masks):
    preds = (preds > 0.5).float()
    intersection = (preds * masks).sum()
    union = preds.sum() + masks.sum() - intersection
    return (intersection / (union + 1e-7)).item()
def dice_score(preds, masks):
    preds = (preds > 0.5).float()
    intersection = (preds * masks).sum()
    return (2. * intersection / (preds.sum() + masks.sum() + 1e-7)).item()
def train_model(model, train_loader, val_loader, epochs=10, lr=1e-3):
```

```
model.to(device)
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=lr)
history = {'loss': [], 'val_loss': [], 'acc': [], 'val_acc': [],
           'iou': [], 'val_iou': [], 'dice': [], 'val_dice': []}
for epoch in range(epochs):
    model.train()
    train_loss, train_acc, train_iou, train_dice = 0, 0, 0, 0
    for images, masks in tqdm(train_loader, desc=f"Epoch {epoch+1}/{epochs}):
    images, masks = images.to(device), masks.to(device).unsqueeze(1)
    optimizer.zero_grad()
    outputs = model(images)
    loss = criterion(outputs, masks)
    loss.backward()
    optimizer.step()
    preds = torch.sigmoid(outputs)
    train_loss += loss.item()
    train_acc += pixel_accuracy(preds, masks)
        train_iou += iou_score(preds, masks)
        train_dice += dice_score(preds, masks)
    model.eval()
    val_loss, val_acc, val_iou, val_dice = 0, 0, 0, 0
    with torch.no_grad():
        for images, masks in tqdm(val_loader,desc=f"Epoch {epoch+1}/{epochs})
            images, masks = images.to(device), masks.to(device).unsqueeze(1)
            outputs = model(images)
            loss = criterion(outputs, masks)
            preds = torch.sigmoid(outputs)
            val_loss += loss.item()
            val_acc += pixel_accuracy(preds, masks)
            val_iou += iou_score(preds, masks)
            val_dice += dice_score(preds, masks)
    n_train = len(train_loader)
    n_val = len(val_loader)
    history['loss'].append(train_loss / n_train)
    history['acc'].append(train_acc / n_train)
    history['iou'].append(train_iou / n_train)
    history['dice'].append(train_dice / n_train)
    history['val_loss'].append(val_loss / n_val)
    history['val_acc'].append(val_acc / n_val)
```

```
        history['val_iou'].append(val_iou / n_val)
        history['val_dice'].append(val_dice / n_val)
        print(f"Epoch {epoch+1}: Train Loss={train_loss/n_train:.4f},
                Val Loss={val_loss/n_val:.4f}, "
                f"IoU={val_iou/n_val:.4f}, Dice={val_dice/n_val:.4f}")
    return model, history
model = NestedUNet()
model, history = train_model(model, train_loader, val_loader, epochs=epochs, lr=lr)
import matplotlib.pyplot as plt
def plot_metrics(history):
    epochs = range(1, len(history['loss']) + 1)
    plt.figure(figsize=(16, 8))
    plt.subplot(2, 2, 1)
    plt.plot(epochs, history['loss'], 'b-', label='Train Loss')
    plt.plot(epochs, history['val_loss'], 'r-', label='Val Loss')
    plt.title('Loss')
    plt.xlabel('Epochs')
    plt.plot(epochs, history['acc'], 'b-', label='Train Accuracy')
    plt.plot(epochs, history['val_acc'], 'r-', label='Val Accuracy')
    plt.title('Pixel Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.subplot(2, 2, 3)
    plt.plot(epochs, history['iou'], 'b-', label='Train IoU')
    plt.plot(epochs, history['val_iou'], 'r-', label='Val IoU')
    plt.title('IoU Score')
    plt.xlabel('Epochs')
    plt.ylabel('IoU')
    plt.legend()
    plt.subplot(2, 2, 4)
    plt.plot(epochs, history['dice'], 'b-', label='Train Dice')
    plt.plot(epochs, history['val_dice'], 'r-', label='Val Dice')
    plt.title('Dice Coefficient')
    plt.xlabel('Epochs')
    plt.ylabel('Dice')
    plt.legend()
    plt.tight_layout()
    plt.show()
plot_metrics(history)
```

Chapter 6

RESULTS

6.1 Metric Outputs

In this section, we present the results of the model evaluations using various performance metrics. The following table summarizes the comparison between **DeepLabV3+** and **U-Net++** models across key metrics.

Metric	**DeepLabV3+**	**U-Net++**
IoU (Intersection over Union)	0.8096	0.6116
Dice Score	0.8885	0.7537
Pixel Accuracy	0.9557	0.9137
Validation Loss	0.1350	0.2531

Table 6.1: Comparison of performance metrics for DeepLabV3+ and U-Net++ models.

The table above provides a comparison between the two models based on commonly used segmentation evaluation metrics. These results help in understanding the performance of each model in terms of segmentation accuracy and efficiency.

6.2 Output Plots

The following plots represent the training and validation performance of both models. These visualizations help in understanding the models' learning curves and how effectively they generalize on unseen data.

These plots visually demonstrate the training and validation progress for each model, providing insights into their convergence and performance over time.

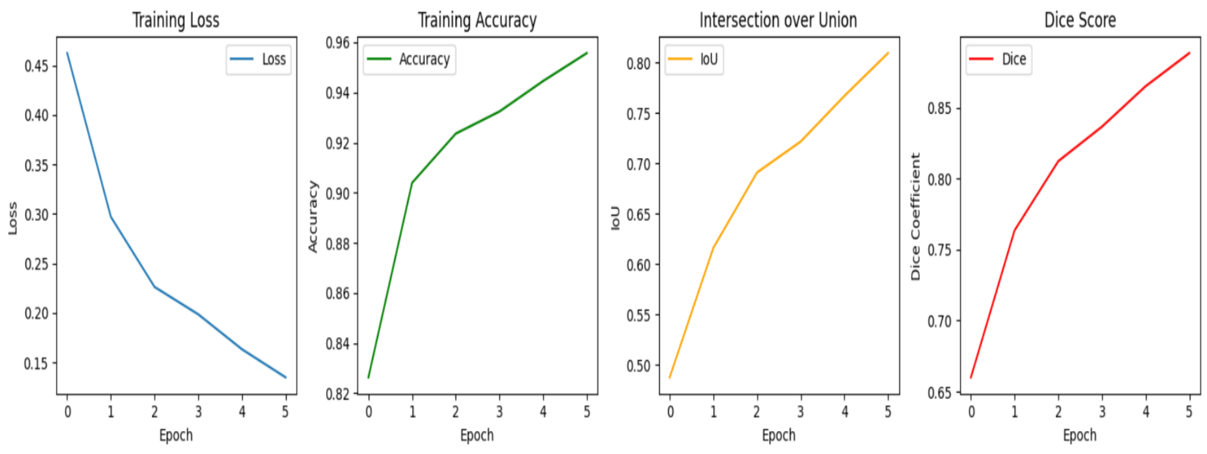

Figure 6.1: Evaluation Metric Plots of DeepLabV3+

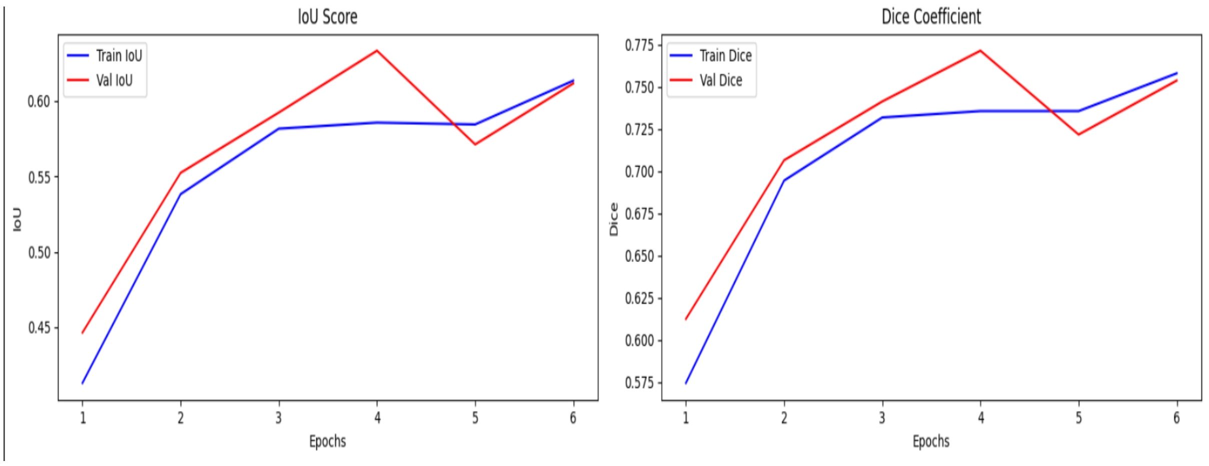

Figure 6.2: IoU and Dice Coefficient of U-Net++

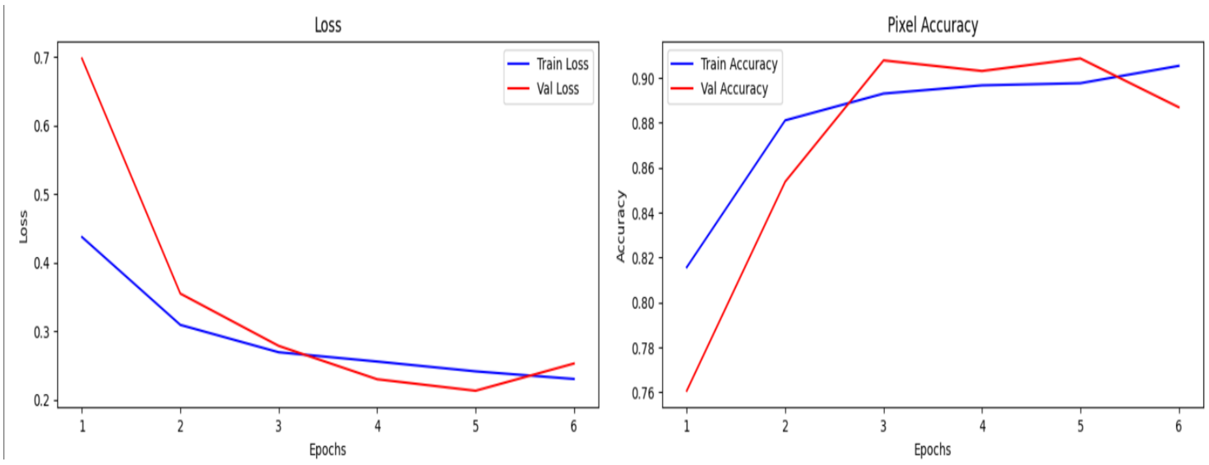

Figure 6.3: Loss and Pixel Accuracy of U-Net++

6.3 Output Images

In this section, we show the segmentation results obtained from both **DeepLabV3+** and **U-Net++** models.

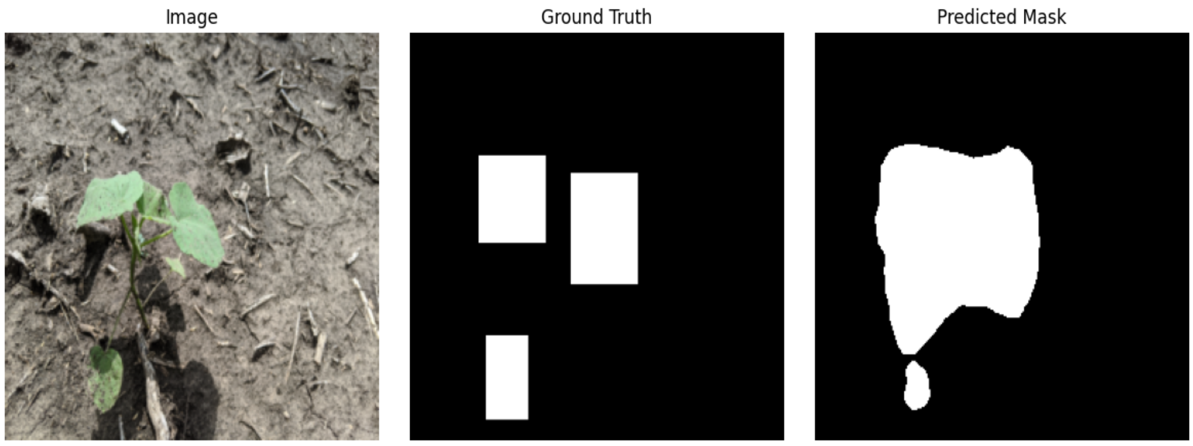

Figure 6.4: Segmentation Output-1 by DeepLabV3+

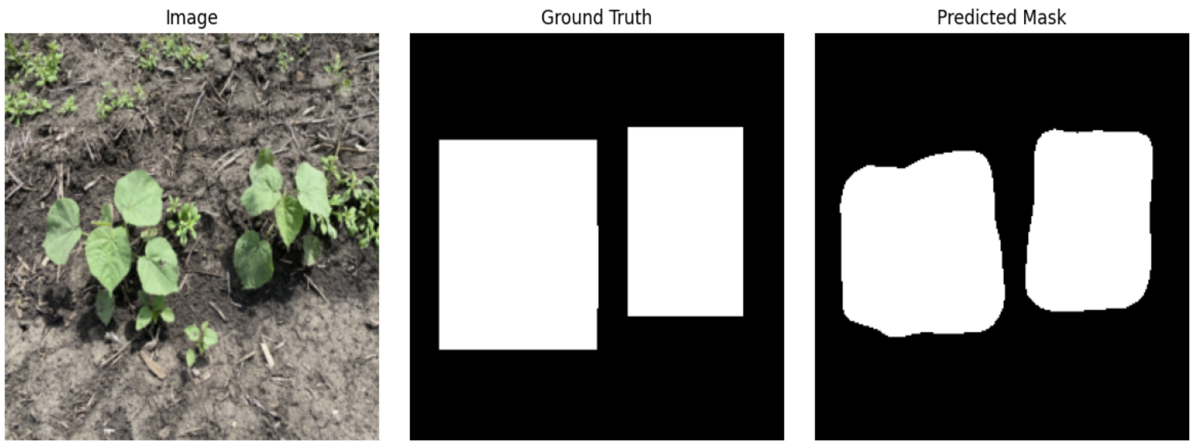

Figure 6.5: Segmentation Output-2 by DeepLabV3+

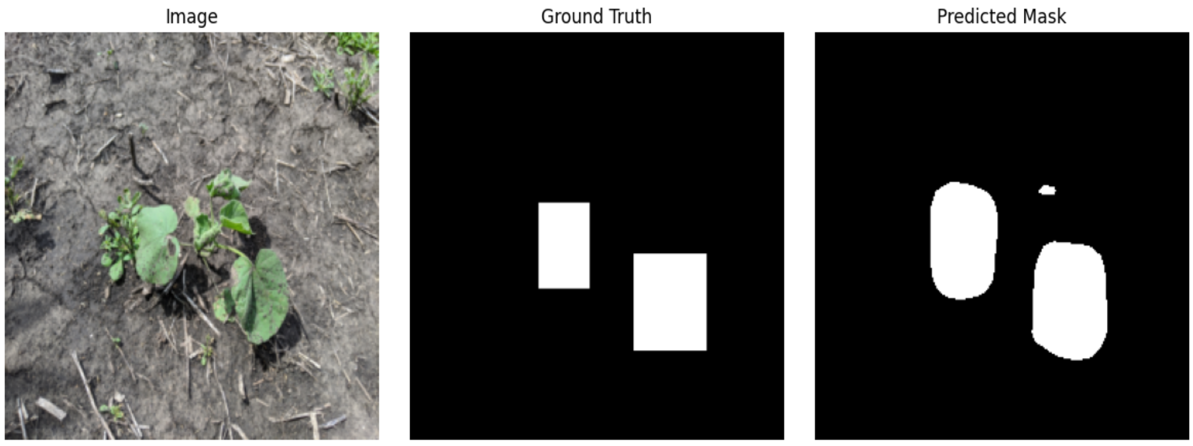

Figure 6.6: Segmentation Output-1 by U-Net++

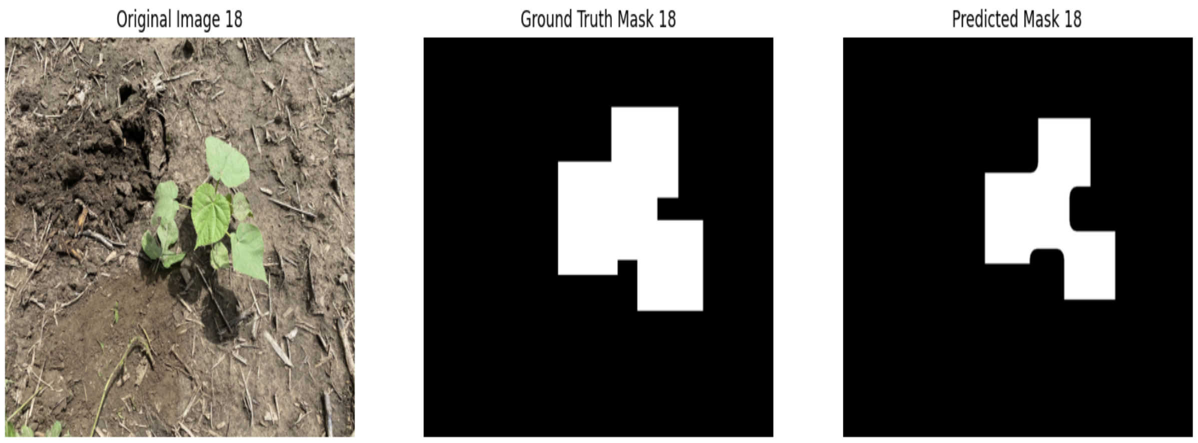

Figure 6.7: Segmentation Output-2 by U-Net++

- **Summary of Results:** The DeepLabV3+ and U-Net++ models demonstrated strong performance in segmenting plants from their backgrounds in agricultural field images. The segmentation outputs were clear and accurate, effectively identifying plant regions and providing precise boundary separation.
- **Model Training on Specific Plant Categories:** Separate training was conducted for individual plant categories, allowing the models to focus on learning the unique features of each plant type. This approach helped improve the quality and consistency of the segmentation results.
- **Clear and Precise Segmentation Outputs:** U-Net++, with its nested architecture and dense skip connections, produced smooth and detailed segmentation maps. DeepLabV3+ delivered sharp and well-localized segmentations due to its multi-scale feature extraction. Both models generated outputs closely resembling the provided ground truth masks.
- **Performance Based on Segmentation Metrics:** Metrics such as Dice Score, IoU (Intersection over Union), and Pixel Accuracy were used to evaluate segmentation quality. Both models achieved high scores across these metrics, validating their reliability in segmenting plant regions under different image conditions.
- **Impact of Dataset Size and Diversity:** Increasing the number and variety of training images led to significant improvements in model performance. A larger and more diverse dataset helped the models generalize better and handle different plant shapes and lighting conditions.
- **Model Comparison and Insights:** While both DeepLabV3+ and U-Net++ performed effectively, DeepLabV3+ showed slightly better results in IoU and pixel accuracy. DeepLabV3+ was faster to train and easier to modify, making it suitable for scenarios where flexibility and interpretability are important.

Chapter 7

COMPARISON

7.1 Comparison of Evaluation Metric Plots of Two Models

This section presents a visual comparison of the performance metrics for DeepLabV3+ and U-Net++ models. The plots show how each model performed in terms of training and validation accuracy, IoU, Dice score, and loss over epochs. This comparison helps in analyzing which model was more stable, accurate, and effective for plant segmentation.

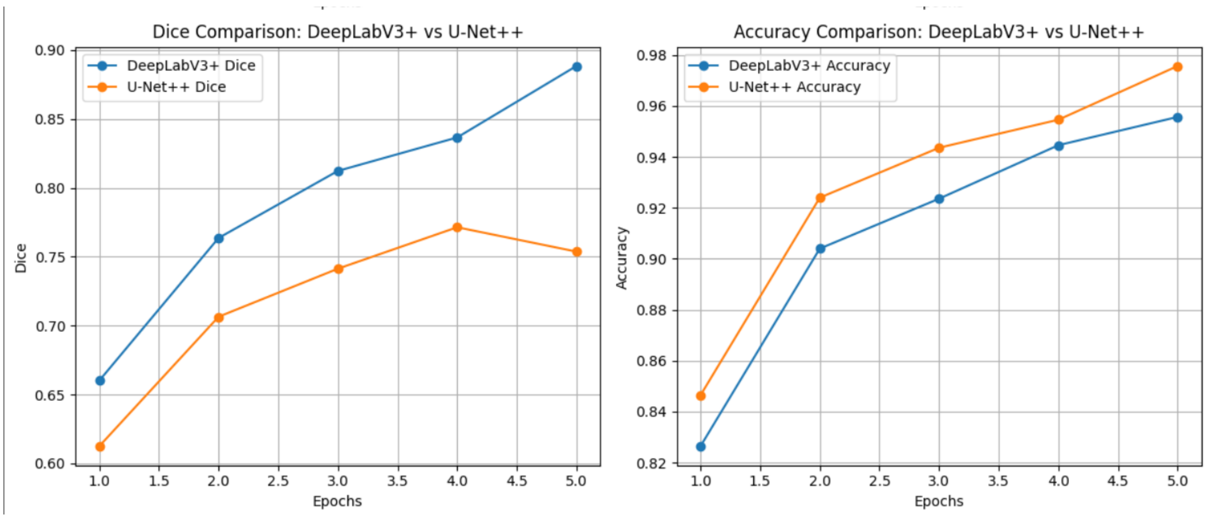

Figure 7.1: Dice and Accuracy Comparison Plots of DeepLabV3+ and U-Net++

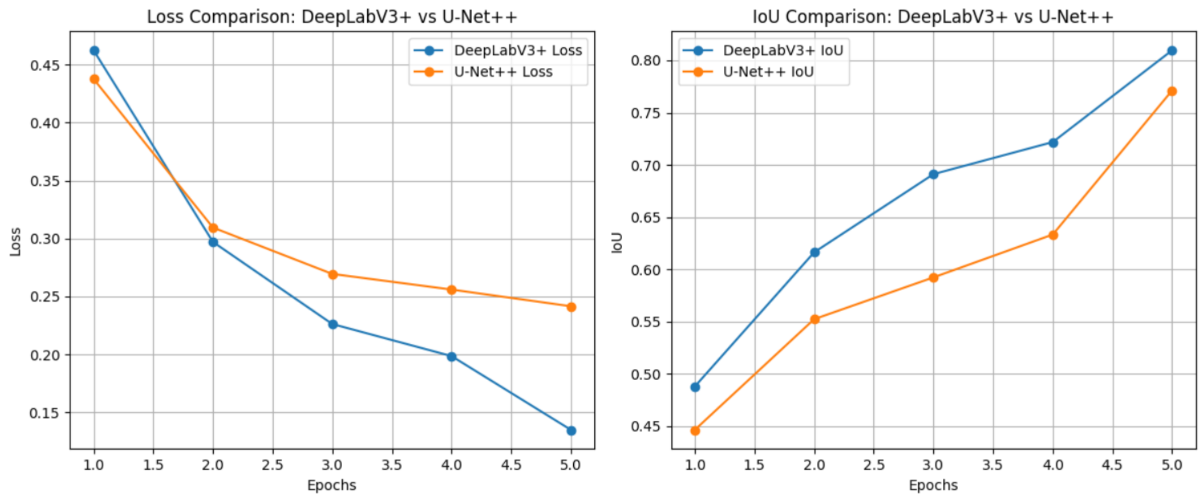

Figure 7.2: Loss and IoU Comparison Between DeepLabV3+ and U-Net++

Model Performance Comparison Summary

In this project, both **DeepLabV3+** and **U-Net++** were implemented and evaluated for the task of plant segmentation using standard evaluation metrics such as Intersection over Union (IoU), Dice Similarity Coefficient, and Pixel Accuracy.

- **Segmentation Quality:** DeepLabV3+ produced more accurate and well-defined segmentation masks. It was particularly effective in capturing the overall plant structure and minimizing background noise. U-Net++ showed decent results but occasionally missed finer boundaries.

- **Model Complexity and Training:** While U-Net++ has a nested skip connection design for feature refinement, DeepLabV3+ leveraged its Atrous Spatial Pyramid Pooling (ASPP) to capture multi-scale contextual information more effectively, resulting in better segmentation performance despite longer training time.

- **Overall Accuracy:** DeepLabV3+ consistently achieved higher scores across all evaluation metrics compared to U-Net++. The improvements in Dice and IoU values clearly indicated superior segmentation accuracy.

- **Visual Output Quality:** Visual inspection of segmentation masks confirmed that DeepLabV3+ provided more reliable and visually consistent outputs, even in complex plant structures.

These results suggest that while both architectures are suitable for plant segmentation, **DeepLabV3+ offers a slight edge in fine-grained segmentation quality and model efficiency**.

Chapter 8

CONCLUSION

This project focused on developing deep learning models for plant segmentation and comparing their performance. We trained and evaluated two models, DeepLabV3+ and U-Net++, using a custom-labeled dataset containing plant images with corresponding segmentation masks.

The results of the evaluation revealed that DeepLabV3+ outperformed U-Net++ across all evaluation metrics, including Intersection over Union (IoU), Dice Similarity Coefficient, and Pixel Accuracy. The superior performance of DeepLabV3+ can be attributed to its Atrous Spatial Pyramid Pooling (ASPP) component, which enables the model to capture multi-scale contextual information effectively. This structure played a crucial role in its ability to extract detailed plant structures from complex backgrounds and handle the spatial relationships within the images.

While **U-Net++** showed good performance, it was slightly less accurate compared to DeepLabV3+ and exhibited more sensitivity to variations in the dataset. The segmentation masks produced by U-Net++ were not as clean and detailed as those generated by DeepLabV3+, leading to less precise plant boundary delineation.

Chapter 9

REFERENCES

1. L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. *DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs.* arXiv preprint arXiv:1606.00915, 2016.

2. R. Gao, K. Sapra, T. Shi, J. You, S. Ermon, and L. Fei-Fei. *Agricultural-Vision: A Large Aerial Image Database for Agricultural Pattern Analysis.* Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020. https://www.agricultural-vision.com/

3. A. Olsen, D. Konovalov, R. Wood, D. R. McCool, and B. M. Haysom. *DeepWeeds: A Multiclass Weed Species Image Dataset for Deep Learning.* Scientific Reports, 2019.

4. Y. Chen, J. Zhou, Y. Hu, and C. Gong. *Plant Phenotyping Using Deep Learning and Image Segmentation: A Review.* Computers and Electronics in Agriculture, 2023.