Comparative Study of DeepLabV3, U-Net and SegFormer for Semantic Segmentation in FoodSeg103

Mahendran Jinachandran

24088951

Supervised by J.J. Collins

Submitted to the University of Limerick for the degree of

MSc in Artificial Intelligence and Machine Learning

August 27, 2025

Supervisor: J.J. Collins
Department of Computer Science and Information Systems
University *of* Limerick
Ireland

Abstract

We present a comparative study of three leading semantic segmentation models - DeepLabV3, U-Net and SegFormer with the goal of evaluating their effectiveness in FoodSeg103 dataset. Semantic segmentation decomposes an image into cohesive entities through classification and labeling at the pixel level. SegFormer is a recent transformer-based segmentation model that combines a hierarchically structured encoder with a lightweight multilayer perceptron (MLP) decoder. DeepLabV3 is a Convolutional Neural Network (CNN) model that improves segmentation performance through a deep encoder-decoder architecture with skip connections by L.-C. Chen et al. (2017). U-Net which was originally designed for biomedical image segmentation, follows a symmetric encoder-decoder structure with skip connections by Ronneberger et al. (2015). In this thesis, we extend the comparison originally presented by Xie et al. (2021) using Cityscapes dataset by evaluating SegFormer against DeepLabV3 and U-Net on FoodSeg103 dataset. This introduces new challenges such as overlapping objects, fine-grained class boundaries and domain-specific variations. All models are trained under consistent conditions and evaluated using standard metrics such as mean Intersection over Union (mIoU), Pixel Accuracy. The motivation behind this comparison is to explore how segmentation models behave where object appearance and arrangement are more variable and visually ambiguous (For example, food trays). This study not only compares model performance across these distinct environments but also explores how certain things such as the use of attention mechanisms or decoder complexity impacts the generalization. If time permits, further optimization through hyperparameter tuning will be conducted to study potential performance gains.

Declaration

I herewith declare that I have produced this thesis without the prohibited assistance of third parties and without making use of aids other than those specified; notions taken over directly or indirectly from other sources have been identified as such. This paper has not previously been presented in identical or similar form to any other Irish or foreign examination board.

The thesis work was conducted from May 2025 to August 2025 under the supervision of *J.J. Collins* at University of Limerick.

Limerick, 2025

Acknowledgments

I want to sincerely thank my professor, J.J. Collins for guiding me throughout the thesis. His constant support, patience and feedback in every week Wednesday meeting helped me learn and grow during this journey. I am truly grateful for all the time and effort he spent helping me improve.

Moving to a new country and I am someone who never left my home, adjusting to a new environment and balancing academics with personal challenges was not easy. I want to thank the University of Limerick for giving me the chance to study in such a supportive and lovely environment.

Lastly and most importantly, I want to thank my family and my friends. Their constant support and faith in me and words of encouragement gave me strength when I felt low. I dedicate this work to them with all my heart.

Data Source and Use of Generative AI

Data Source

For this research, the FoodSeg103 dataset was used. This is a publicly available dataset designed specifically for food image segmentation tasks and is hosted on Hugging Face. Click here to view the dataset. It consists of about 7,100 real-world food images, each annotated at the pixel level across 104 classes (103 food ingredients and 1 background class). The dataset presents real-world challenges such as overlapping food items, class imbalance and diversity. The dataset is intended for research and academic use. It belongs to the original dataset creators and contributors who have made this resource openly accessible for the purpose of advancing research in computer vision and dietary analysis.

Generative AI

I herewith declare that I have used Gen AI tool to asisst in understanding certain technical concepts and codes such as the underlying working of models and architectures, also for summarizing research papers and for formatting purposes within Overleaf including bibliography, table creation and general layout adjustments. No generative AI was used to produce or generate the actual written content of this thesis.

Limerick, 2025

Contents

1 Introduction **1**

1.1 Overview 1
1.2 Research Question 3
1.3 Contribution 4
1.4 Methodology 4
1.5 Overview of the dissertation 6
1.6 Motivation 6

2 Literature Review **7**

2.1 Intro to Artificial Intelligence and Machine Learning 7
2.2 Artificial Neural Networks 10
2.2.1 The Perceptron 11
2.2.2 The Multilayer Perceptron 12
2.2.3 Backpropagation 13
2.3 Convolutional Neural Networks 14
2.3.1 History of CNN 15
2.3.2 Convolution Layer 16
2.3.3 Pooling Layer 17
2.3.4 Activation Layer 18
2.3.5 Fully Connected Layer 19
2.3.6 Dropout 20
2.3.7 Batch Normalisation 21

2.4 CNNs for Semantic Segmentation . . . 22
2.4.1 History of Semantic Segmentation . . . 22
2.4.2 Working of Semantic Segmentation . . . 23
2.4.3 Semantic Segmentation Architectures . . . 25
2.4.4 Evaluation Metrics . . . 29
2.5 Datasets and Published results . . . 31
2.5.1 FoodSeg103 . . . 32
2.5.2 UNIMIB2016 . . . 33
2.6 Summary . . . 34

3 Prototypes **36**
3.1 Introduction . . . 36
3.2 Development Environment . . . 37
3.3 Architecture Details . . . 37
3.3.1 DeepLabV3 . . . 37
3.3.2 U-Net . . . 38
3.3.3 SegFormer . . . 39
3.4 Implementation . . . 40
3.4.1 Transfer Learning . . . 40
3.4.2 WarmupPolyLR . . . 42
3.4.3 Focal and Dice Loss . . . 44
3.5 Training . . . 47
3.6 Summary . . . 48

4 Empirical Studies **49**
4.1 Introduction . . . 49
4.1.1 Dataset . . . 49
4.1.2 Data Transformations . . . 50
4.2 Experiments . . . 50
4.2.1 Experiment 1 . . . 50
4.2.2 Experiment 2 . . . 53

4.2.3 Experiment 3 55
4.2.4 Experiment 4 58
4.2.5 Experiment 5 60
4.2.6 Experiment 6 62
4.2.7 Experiment 7 64
4.2.8 Experiment 8 67
4.2.9 Experiment 9 69
4.2.10 Experiment 10 71
4.3 Final Evaluation 74
4.4 The Research Questions Revisited 76
4.5 Cityscapes Revisited 77
4.6 Threats to Validity 80

5 Discussion and Conclusions **82**

List of Figures

2.1 A Single layer perceptron 12
2.2 A Multi layer perceptron 12
2.3 Diagram of a simple CNN Architecture 15
2.4 Visual representation of convolution operation 17
2.5 An Example of Max Pooling 18
2.6 An Example of Dropout (Srivastava et al. 2014) 21
2.7 Semantic Segmentation (Géron 2023) 22
2.8 U-Net Architecture 26
2.9 SegNet predictions on road scenes and indoor scenes. 27
2.10 DeepLabv3 Architecture with labeled blocks 29
2.11 Performance of BEiT v2 and InternImage-B model on Food-Seg103 (Sinha et al. 2023). 32
2.12 Examples of segmentation results of some UNIMIB2016 images. 34

3.1 DeepLabV3 model 37
3.2 U-Net model 39
3.3 DeepLabV3 model 39
3.4 A custom function for freezing/unfreezing the encoders in a model 41
3.5 Freezing / Unfreezing in DeepLabV3 Implementation 42
3.6 A custom class for modifying the learning rate 43
3.7 Focal Loss Code 45
3.8 Dice Loss Code 46

3.9 Combination of both Focal and Dice Loss 47
3.10 SegFormer output interpolation code 47

4.1 Loss progression for DeepLabV3, U-Net and SegFormer during training for Experiment 1 52
4.2 Original Image, DeepLabV3, U-Net, SegFormer (Left to Right) for Experiment 1 52
4.3 Loss progression for DeepLabV3, U-Net and SegFormer during training for Experiment 2 54
4.4 Original Image, DeepLabV3, U-Net, SegFormer (Left to Right) for Experiment 2 54
4.5 Loss progression for DeepLabV3, U-Net, and SegFormer during training for Experiment 3 57
4.6 Original Image, DeepLabV3, U-Net, SegFormer (Left to Right) for Experiment 3 57
4.7 Loss progression for SegFormer during training for Experiment 4 59
4.8 Original Image and SegFormer prediction for Experiment 4 . . 59
4.9 Loss progression for SegFormer during training for Experiment 5 61
4.10 Original Image and SegFormer prediction for Experiment 5 . . 62
4.11 Loss progression for SegFormer during training for Experiment 6 63
4.12 Original Image and SegFormer prediction for Experiment 6 . . 64
4.13 Loss progression for DeepLabV3, U-Net, and SegFormer during training for Experiment 7 66
4.14 Original Image, DeepLabV3, U-Net, SegFormer (Left to Right) for Experiment 7 66
4.15 Loss progression for DeepLabV3, U-Net, and SegFormer during training for Experiment 8 68
4.16 Original Image, DeepLabV3, U-Net, SegFormer (Left to Right) for Experiment 8 68
4.17 Loss progression for DeepLabV3, U-Net, and SegFormer during training for Experiment 9 70

4.18 Original Image, DeepLabV3, U-Net, SegFormer (Left to Right) for Experiment 9 70
4.19 Loss progression for U-Net++ during training for Experiment 10 72
4.20 Original Image and U-Net++ for Experiment 10 73
4.21 Loss progression for DeepLabV3, U-Net, and SegFormer during training for Cityscapes 79
4.22 Original Image, DeepLabV3, U-Net, SegFormer (Left to Right) for Cityscapes 79

List of Tables

4.1 Hyperparameter Configuration for Experiment 1 51
4.2 Evaluation Metrics for Experiment 1 51
4.3 Hyperparameter Configuration for Experiment 2 53
4.4 Evaluation Metrics for Experiment 2 54
4.5 Hyperparameter Configuration for Experiment 3 56
4.6 Evaluation Metrics for Experiment 3 56
4.7 Hyperparameter Configuration for Experiment 4 58
4.8 Evaluation Metrics for Experiment 4 59
4.9 Hyperparameter Configuration for Experiment 5 61
4.10 Evaluation Metrics for Experiment 5 61
4.11 Hyperparameter Configuration for Experiment 6 63
4.12 Evaluation Metrics for Experiment 6 63
4.13 Hyperparameter Configuration for Experiment 7 65
4.14 Evaluation Metrics for Experiment 7 65
4.15 Hyperparameter Configuration for Experiment 8 67
4.16 Evaluation Metrics for Experiment 8 67
4.17 Hyperparameter Configuration for Experiment 9 69
4.18 Evaluation Metrics for Experiment 9 70
4.19 Hyperparameter Configuration for Experiment 10 72
4.20 Evaluation Metrics for Experiment 10 72
4.21 Accuracy and mIoU Scores Across 10 Experiments 74
4.22 mIoU comparison against existing FoodSeg103 models 76

4.23 Hyperparameter Configuration for Cityscapes 78
4.24 Evaluation Metrics for for Cityscapes 78
4.25 mIoU comparison against existing Cityscapes models 80

Chapter 1

Introduction

1.1 Overview

In the field of dietetics, understanding what people eat is crucial for improving health, preventing diseases and also for promoting good nutrition. Traditionally, dietitians relied on food journals, questionnaires or direct consultations to track what a person eats in a day. However, these methods were often inaccurate due to human error (Sinha et al. 2023). To overcome this issue, computer vision was used to automatically analyze food from photos taken by individuals, especially using smartphones. The biggest success in this, came with the help of *semantic segmentation* (Long et al. 2015), a technique that allows a computer to look at an image and label every single pixel, identifying which part belongs to which food item. For example, in a plate of rice and curry, semantic segmentation helps the system understand which exact pixels belong to the rice and which belong to the curry. This is very important in dietetics, because it can be used for portion estimation, nutrient tracking, calorie estimation (Wu et al. 2021).

In recent years, many powerful machine learning model architectures have been designed specifically for semantic segmentation. Among them, three

models stand out due to their strong performance and popularity: DeepLabV3, U-Net and SegFormer. U-Net was one of the first segmentation architectures and it was originally built for medical imaging (Ronneberger et al. 2015). Its encoder-decoder structure allows it to learn both low-level and high-level features. It is lightweight and performs well on smaller datasets. DeepLabV3 is a more advanced model that uses atrous convolutions and spatial pyramid pooling. It is designed to understand objects at different scales, which is especially useful in food images where items can vary in size (L.-C. Chen et al. 2017). SegFormer is a newer architecture based on Transformers instead of Convolutional Neural Networks (CNNs). It has gained attention for its ability to obtain a strong understanding of overall context in an image. (Xie et al. 2021).

In this dissertation, we compare and analyze the performance of these three models on a domain specific dataset called FoodSeg103 (Wu et al. 2021). This dataset consists of real-world images containing various food items, each annotated with precise pixel-level labels across 104 classes (103 Food ingredients and 1 background class). Unlike other datasets which can be solved by general segmentation, FoodSeg103 presents unique challenges such as overlapping food items, diverse appearances, long tailed distribution i.e class imbalance. In this dissertation, each model was initialised with pretrained weights, followed by the fine-tuning on FoodSeg103. In some of the experiments, the encoders (the backbone) were frozen to keep the pretrained features that was learned during the pretraining. After a few epochs, the encoders were unfrozen to adapt to the FoodSeg103 dataset. Different loss functions such as CrossEntropyLoss, Focal and Dice Loss separately and combined were used. Focal Loss helped the models focus more on pixels which are difficult to classify, especially useful for handling class imbalance (Wizard 2020). Dice Loss, on the other hand, improve the overlap between the predicted and actual masks which is a crucial factor for accurate segmentation (Diary 2021). Another important strategy was how the learning rate

was scheduled during training. Rather than using a fixed or standard decay schedule, we implemented a WarmupPolyLR strategy (Xie et al. 2021). This means the learning rate starts low and gradually increases (warmup) before slowly decaying using a polynomial formula. This approach helped stabilize the training and prevent early divergence.

After training the models, we evaluated them using two standard semantic segmentation metrics: Pixel Accuracy (PA) and mean Intersection over Union (mIoU)(Huynh 2023). We also created visual comparisons by overlaying predicted segmentation masks on the original images to see how well the models performed visually. This comparison is not only valuable for academic purposes but also has practical implications for building real-world applications in healthcare, nutrition analysis and fitness monitoring.

1.2 Research Question

This research investigates the effectiveness of three widely used semantic segmentation models DeepLabV3, U-Net and SegFormer with a specific focus on FoodSeg103 dataset. The primary goal of this research is to understand how these three models perform under those conditions and determine which architecture offers the most accurate and efficient segmentation. The study also aims to guide future work in food image analysis and provide insights for semantic segmentation.

The Research Questions in the context of semantic segmentation are:
RQ1: Comparative Study of DeepLabV3, U-Net and SegFormer for Semantic Segmentation in FoodSeg103.
RQ2: Which of the following loss functions performs better - CrossEntropy, Focal, and Dice Loss separately or combinations thereof.
RQ3: Does WarmupPolyLR improve training stability compared to standard decay schedules?

1.3 Contribution

This study makes key contribution to the field of semantic segmentation, especially in the field of dietetics and food image analysis. The performance of these models are evaluated using the segmentation metrics such as mIoU and Pixel Accuracy. The following are the contributions

- A experimental comparison between three state of the art segmentation models - DeepLabV3, U-Net and SegFormer on the FoodSeg103 Dataset. It poses unique challenges due to its complex food structures such as overlapping ingredients, very small objects and class imbalance.
- It also explores the impact of training strategies such as transfer learning, layer freezing and unfreezing, and advanced loss functions such as FocalLoss, Dice Loss and combination of those two losses.

1.4 Methodology

This research followed a structural approach to compare the performance of three semantic segmentation models DeepLabV3, U-Net and SegFormer on the FoodSeg103 dataset (Wu et al. 2021). This study was conducted in several stages making sure that each step can be replicated by future researchers. The work began with a review of literature to understand deeper about the challenges of food image segmentation and how various semantic segmentation models have performed in similar tasks. Based on the research, three models were selected where each one represents a different approach: DeepLabV3 is convolutional based model (L.-C. Chen et al. 2017), U-Net is a encoder-decoder based model (Ronneberger et al. 2015) and SegFormer is a transformer based model (Xie et al. 2021). This different selection allowed to explore how different architectural designs impact segmentation performance in the food domain.

The FoodSeg103 which consists of 103 classes of food items was chosen for its diversity and detailed annotations. The dataset was divided into training and validation sets for fair evaluation. Before training, transformations such as resizing to a consistent dimension, normalisation and other basic augmentations steps such as random flipping were done to improve the models ability to generalise to new images.

For training, pretrained versions of each model were used to learn from knowledge learned from ImageNet dataset. In some experiments, we fine-tuned the entire model from the start, while in others, we initially froze the encoder in the model before gradually unfreezing them to see if this improved stability and performance. Different training strategies were explored such as varying the learning rate schedule and different loss functions such as Cross-Entropy Loss, Dice Loss and Focal Loss, as well as combinations of these. The models were trained for a set number of epochs on University of Limerick's GPU using an "NVIDIA A100 80GB PCIe". Performance was evaluated using two widely accepted metrics for segmentation: Pixel Accuracy and Mean Intersection over Union (mIoU) (Huynh 2023). Additionally, segmentation outputs were visually compared to assess how well each model identified food items, preserved boundaries and handled challenging cases such as overlapping ingredients or small objects.

By following this structured methodology starting from model selection and dataset preparation to training strategies and evaluation, this study provides a clear basis for understanding the comparative strengths and weaknesses of the different segmentation models in FoodSeg103 dataset.

1.5 Overview of the dissertation

This thesis is structured as follows:
Chapter 2 presents a detailed literature review of semantic segmentation techniques, including CNN and transformer-based models.
Chapter 3 presents further deeper understanding of the architecture and implementation of DeepLabV3, U-Net and SegFormer and strategies such as Schedulers, Loss functions and layer freezing.
Chapter 4 presents a outline of the experimental setup and processes and also discussion of the evaluation and results.
Chapter 5 concludes the thesis with key takeaways and directions for future work.

1.6 Motivation

In the current modern world, dietary tracking has gained popularity because of the development of advanced mobile applications and image recognition. However, segmenting food items in real-world images still remains a challenging task due to the visual complexity, overlapping objects and class imbalance. While CNN models like U-Net and DeepLabV3 have shown success in medical and natural images, their effectiveness on food specific datasets has not been thoroughly evaluated. On the other hand, transformer based models like SegFormer offers a promising alternative with their global attention mechanisms.

This research was driven by the curiosity to understand how well these models adapt to food segmentation and whether the new transformer based approaches can truly outperform traditional convolutional architectures in this food domain.

Chapter 2

Literature Review

Understanding the existing research and the current developments in the field of Artificial Intelligence (AI) and Machine Learning (ML) and their applications is essential in framing this study. This chapter covers a wide range of topic starting from the basics of AI and ML to the state-of-the-art technologies. This foundational knowledge provides the necessary context for understanding the design of semantic segmentation in CNNs and evaluating their effectiveness.

2.1 Intro to Artificial Intelligence and Machine Learning

Artificial Intelligence means making computers or machines think and act like we humans do (McCarthy 1955). A more engineering oriented one would be "A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E" (Mitchell 1997). We can find AI in almost everything we do in our current day to day life activities like speech recognition in smartphones (Apple's Siri), understanding language,

online shopping websites and even the self driving cars. Intelligence can be defined as the ability to learn and perform suitable actions to solve problems and achieve a particular goal.

AI has been around for several decades starting in 1950s, but its progress was limited by the technology and data (Russell et al. 2021). But in the recent years, it has grown very rapidly. This is because of two reasons: very powerful computers, thanks to semi-conductor industries and lots of data. Today, we gather huge amounts of data through smartphones, emails, photos, social media and many other sources. AI uses this data to learn, adapt and improve its performance over time (Géron 2023).

However, this also raises important ethical concerns around data privacy, consent and responsible use of personal information. How is that information being used and is it secure enough in such a way others can't access it? Questions around privacy, bias and accountability are becoming more important as AI continues to expand. While AI has the power to bring great benefits in our lives, it must be developed and used responsibly to protect people's rights and trust.

Machine Learning is a subset of AI, studying how computers can improve their knowledge, thinking process and problem-solving capabilities based on the dataset or previous experiences. In simple words, it is a way for computers to learn from the data without being explicitly told what to do. Let's look at a simple example. Suppose you want to teach a computer to recognize cats in photos. In traditional programming, you would write code that looks for whiskers, pointy ears and various other features. In machine learning, you would give the computer many pictures of cats and not cats and the model would learn to tell the difference by itself. ML systems can be classified according to the amount and type of supervision they get during training. There are three main types of ML:

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

In supervised learning, the training set that is fed into the model contains the desired solutions, which are called labels. The dataset which is used to train the model is called the training set (Géron 2023). The model will use this training set to learn all the underlying patterns to know about the data. Once its done, the model will be tested on a separate dataset called test set. The test set will be used to simulate the real-world images and it does not contain the desired solutions. There are two types of supervised learning: Regression and Classification. The output of the regression will be a numeric value, such as price of house, price of cars, etc. Classification will have a definite number of output such as either cat or dog, cancer or not.

In unsupervised learning, the dataset is not labeled, identifying patterns and relationships without explicit guidance or prior knowledge of desired outcomes. Unlike supervised learning,there is no human intervention or explicit feedback to guide the learning process (Géron 2023). One of the most widely used unsupervised learning is anomaly detection. For example, detecting unusual credit card payments to prevent fraud, detecting manufacturing defect in industries.

Reinforcement learning is very different from its other counterparts. The learning system called an agent can observe an environment, select and perform actions and receive rewards or penalties in return. It does not have any previous experience and learns everything by itself and decides the best strategy also known as policy, to get the best rewards over time. A policy tells the agent which action to take based on the situation (Hammoudeh 2018).

For example, many robots implement Reinforcement Learning algorithms to learn how to walk. DeepMind's AlphaGo program is also a good example of Reinforcement Learning. It made the headlines in May 2017 when it beat the world champion Ke Jie at the game of Go. It learned its winning policy by analyzing millions of games and then playing many games against itself (Géron 2023).

However, there are many challenges in Machine Learning tasks. Two things that can go wrong here: Bad algorithm and Bad data. Some examples of bad data: Insufficient quantity of data, Irrelevant features, Non-representative training data, Poor Quality data. Some examples of bad algorithms: Overfitting the training data, Underfitting the training data (Géron 2023) .

2.2 Artificial Neural Networks

Deep learning is a subset of machine learning that focuses on multi-layered neural networks to perform tasks such as classification, regression and representational learning. To understand in depth about deep learning, it helps to go back in time. In the 1950s and 60s, researchers tried to build systems that mimic the brain, using what we now call "Neural Networks". But back then, computers were too slow and data was limited, so the idea didn't work well. In the early 2000s, things changed. With faster computers, more data and better techniques, deep learning started to become practical (Pires et al. 2023). Some of the common deep learning architectures are fully connected networks, deep belief networks, recurrent neural networks, convolutional neural networks, generative adversarial networks, transformers and neural radiance fields. These architectures have been applied to fields such as Computer Vision (CV), Natural Language Processing (NLP), Machine Translation, Bioinformatics, Drug Design, Medical Image Analysis, speech recognition and various games.

ANNs are at the very core of Deep Learning. They are flexible, efficient and capable of handling large Machine learning tasks such as classifying billions of images (e.g., Google Images), recommending the best videos to watch to hundreds of millions of users every day (e.g., YouTube).

2.2.1 The Perceptron

The perceptron is one of the simplest ANN architectures, invented in 1957 by Franklin Rosenblatt and is considered as one of the building blocks of modern neural networks. At its very core, a perceptron takes several input values, applies weights to them, adds a bias and passes the result through an activation function to produce an output. A perceptron consists of a single layer of interconnected nodes where each node receives input. Since all input features are connected to each node in the output layer, this setup is referred to as a fully connected or dense layer. The inputs form the input layer, while the output of this computation becomes the final prediction.

A simple diagram of perceptron can be seen in Figure 2.1. The goal is to find a line or boundary that separates data into different classes, for example classifying whether an image is cat or not. The perceptron works well for linearly separable problems, where the classes can be separated by a straight line. However, it struggles with non-linear or complex pattern problems.

Each artificial neuron in an ANN takes some input, does a simple calculation, and passes the result to the next layer of neurons. These neurons are arranged in layers:

- The input layer receives the raw data.
- The hidden layers process the data through mathematical functions.
- The output layer gives the final result.

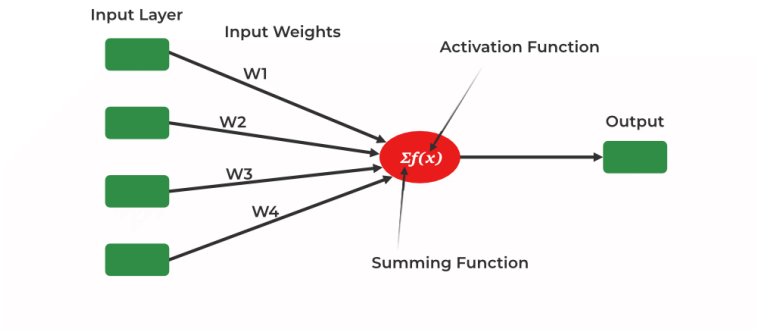

Figure 2.1: A Single layer perceptron

2.2.2 The Multilayer Perceptron

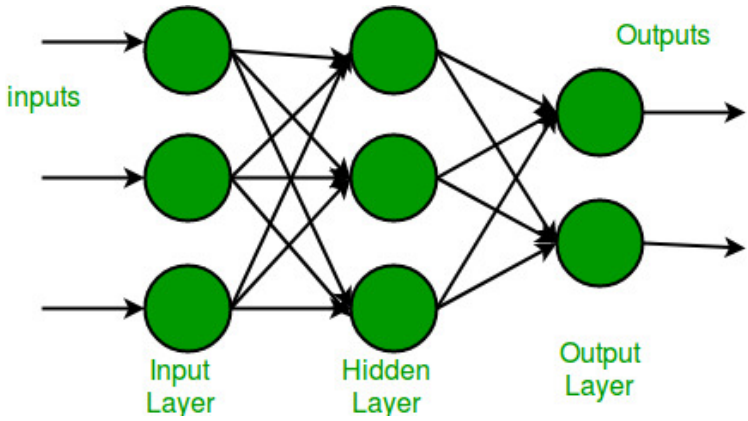

Figure 2.2: A Multi layer perceptron

A typical MLP is composed of an input layer, one or more hidden layers and one final layer known as the output layer (see Figure 2.2). The layers positioned near the input layer are commonly known as the lower layers and the layers at the top are called dense layers. When an ANN includes multiple hidden layers between the input layer and the output layer, it is called Deep Neural Network. This architecture allows the MLP to apply more complex patterns and functions than a single-layer perceptron (Vakalopoulou et al.

2023). The hidden layers enable the model to learn more and extract features from the data. As the number of hidden layers increases, the model learns to capture detailed representation of the input.

This architecture has been proven effective in many applications such as Hand-writing recognition, Image Classification, Speech recognition and more. However, MLPs have several limitations. They require huge amounts of data and computational power especially as the number of hidden layers increases. They are also sensitive to the hyper-parameters such as learning rate and hidden layers. Additionally, training deep MLPs can sometimes suffer from issues like vanishing gradients, making optimization more difficult.

2.2.3 Backpropagation

As neural networks grew in complexity with the introduction of multiple hidden layers in MLPs, a significant challenge was there: How to effectively train these architectures? Traditional methods lacked the ability to adjust the weights across all the layers in an efficient manner on its own. This made it very difficult for MLPs to learn complex, non-linear relationships in data.

To address this issue, Paul Werbos developed the backpropagation algorithms which gained widespread attention all over the world (Werbos 1990). Back-propagation provided a practical solution for training deep neural networks by applying the chain rule of calculus to efficiently calculate how much each parameter in the model contributes to the overall loss.

There are two main stages: a forward pass and a backward pass. In the forward pass, the input data flows through the network and produces an output as a result. The loss function calculates the error between the predicted output and the actual output. In the backward pass, this error is propagated backward through the network, computing the gradients layer

by layer (GeeksforGeeks 2021). These gradients are then used to update the weights using an optimization method called Stochastic Gradient Descent (SGD).

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \eta \frac{\partial L}{\partial w_{ij}}$$

Where:

$w_{ij}^{(t+1)}$: Updated weight between neuron i and j at time step $t+1$

$w_{ij}^{(t)}$: Current weight at time step t

η : Learning rate (step size)

$\frac{\partial L}{\partial w_{ij}}$: Gradient of the loss function L with respect to the weight w_{ij}

The above equation allows the network to learn by minimizing the loss function iteratively, thus improving the accuracy over time. Like every other algorithm out there, there are challenges with Backpropagation. Vanishing gradients is one of the major problems which occurs in Backpropagation. The gradients can become very small which makes it difficult for the lower layers to learn. The gradients can also become very large causing it to diverge. If the architecture is too large, it might also cause overfitting instead of generalizing. Despite its simplicity, the backpropagation is remarkably powerful and is one of the reasons of success of deep neural networks that is being used in current AI applications.

2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) is a specialized type of Deep learning techniques. It was first introduced in 1989 by LeCun (LeCun, Boser, et al. 1989) which became popular for object classification. CNNs consists of multiple layers of nodes which are also called neurons. One of the fundamental

concepts of CNNs is the convolutional layer from which the architecture gets its name.

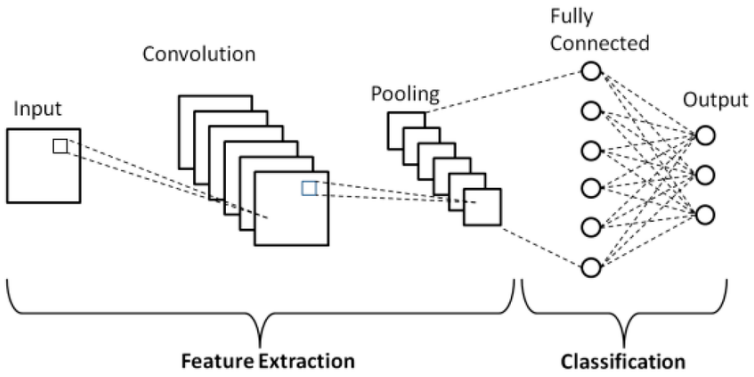

Figure 2.3: Diagram of a simple CNN Architecture
(Phung et al. 2019)

The architecture of a CNN typically consists of multiple layers including convolutional layers, activation layers (such as ReLUs), pooling layers and fully connected layers. Each layer plays a specific role in transforming the input data into meaningful representations that can be used for classification, detection and other tasks. In early layers, these filters commonly identify low-level features such as edges, corners, and textures. As the data progresses through deeper layers, the network begins to learn more abstract and complex features, such as shapes, objects or even faces depending on the task (Stabinger et al. 2016).

2.3.1 History of CNN

The history of Convolutional Neural Networks (CNNs) dates back to the 1980s with the introduction of the *Neocognitron* by Fukushima (Fukushima 1980). A significant milestone came with the development of LeNet-5 by LeCun et al (LeCun, Bottou, et al. 1998) which was successfully used for digit recognition tasks. Despite its success, CNNs were not widely used or popular

for a long time due to limited computational resources and dataset sizes. This changed in 2012 when Krizhevsky et al introduced AlexNet which achieved a substantial performance leap on the ImageNet dataset contributing to the deep learning revolution (Krizhevsky et al. 2012). This success was largely due to the use of GPUs, ReLU activations and dropout regularization.

Following AlexNet, several deeper and more powerful architectures were proposed. VGGNet (Simonyan et al. 2014) focused on simplicity through stacked small filters, while GoogLeNet (Szegedy et al. 2015) introduced Inception modules for efficient computation. ResNet (He et al. 2016) then addressed the vanishing gradient problem by using residual connections, enabling the training of networks with over 100 layers. These developments placed the CNNs as a foundational component in modern computer vision tasks such as image classification, object detection and semantic segmentation.

2.3.2 Convolution Layer

The input layer takes raw pixel values of images as input often in RGB (Red, Blue, Green) or grayscale format (other formats also exist) and their dimensions will be defined by height x width x channels. The image data is then fed into the convolutional layer, which starts the process of finding the patterns in the image (Oye et al. 2024). Its primary role is to automatically extract important features such as edges, corners and patterns. This is done through small learnable filters known as Kernels, which slides over the input image.

As the kernel (or filter) moves across the image, it performs an element-wise multiplication with the part of the image it overlaps which is later followed by a summation (Figure 2.4 for reference). This process is called the convolution operation and the result is a feature map that highlights specific patterns or structures present in the input data. Each filter is trained to detect a specific

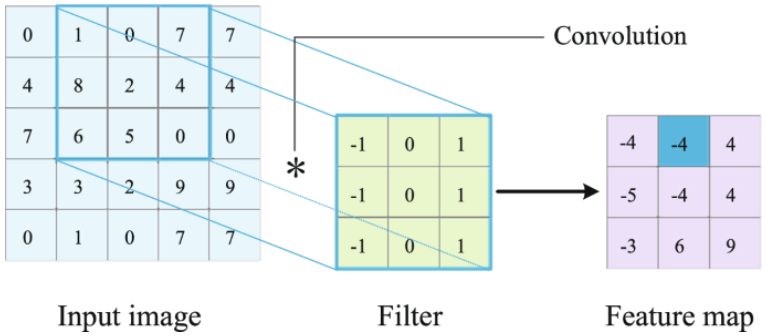

Figure 2.4: Visual representation of convolution operation

kind of feature and multiple filters can be applied in a single layer to capture a variety of patterns. The output of a convolutional layer depends on factors such as the size of the filters, the stride (how far the filter moves at each step), and the padding (how much extra space is added around the input image). These settings affect the size of the resulting feature map and how much detail is retained from the original image.

2.3.3 Pooling Layer

In many CNN architectures, this is an extensive use of down sampling operation that reduces the dimensionality of the representation and hence, reduces the number of parameters and the computational overload of the model (Vakalopoulou et al. 2023). Some of the most common downsampling methods are maximum pooling, average pooling and global average pooling. They are used to prevent the problem of overfitting. Overfitting occurs when a model learns the training data too well including the irrelevant details and noise, leading to poor performance on new and unseen data (Géron 2023).

The working of a pooling layer is as follows: The size of the pooling window is chosen first along with the Stride. Stride is the step size at which the window moves. A common size of the window is 2x2 window and the stride is set to 2, which reduces the size of the feature maps in half. The pooling

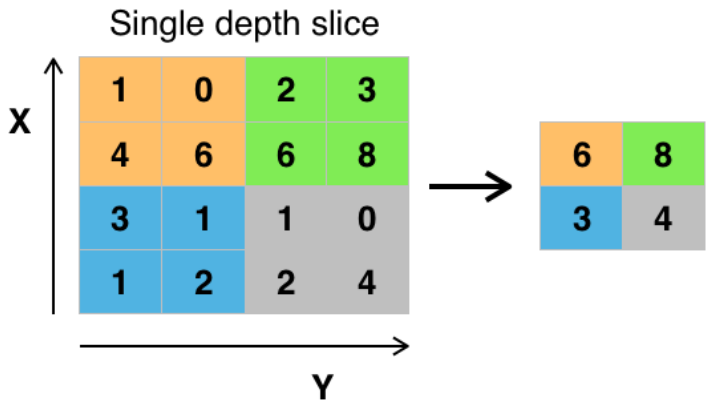

Figure 2.5: An Example of Max Pooling

operation is applied to each region of the input feature map covered by the window. Depending on the type of pooling (max, average and other types) the operation extracts the required value from each window (GeeksforGeeks contributors 2020). Max pooling is the most widely used pooling method in CNN architectures. It works by selecting the maximum value from each part of the feature map covered by the kernel of size 2x2 (Xu et al. 2015). An example is in Figure 2.5 for reference. This helps the model retain the most prominent features while reducing spatial dimensions.

2.3.4 Activation Layer

The activation function of a node in an artificial neural network is a function that calculates the output of the node based on its individual inputs and their weights. It is very crucial for its role of introducing non-linearity into CNNs (Hinkelmann 2018). Without the activation functions, CNNs will behave more like linear model, limiting its ability to learn non-complex patterns. Common activation functions include the Rectified Linear Unit (ReLU), sigmoid and tanh. Among these, ReLU is the most popular due to its simplicity and effectiveness (Dubey et al. 2022).

One of the major challenges in training neural networks is the vanishing gradient problem. The gradients become extremely small as they are back propagated to the lower layers. This issue is particularly severe when using activation functions like the sigmoid or tanh. ReLU reduces the vanishing gradient problem. But unfortunately, ReLUs are not perfect either. They suffer from what is known as Dying ReLUs. This occurs when neurons consistently output zero due to negative input values, effectively "killing" the neuron because it no longer updates during backpropagation. To resolve this, they introduced Leaky ReLUs. It introduces a small slope (typically 0.01) for negative input values instead of assigning them zero.

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$$

where α is a small constant. This ensures that the neuron remains active and continues to learn even for negative inputs, helping avoid dead neurons and improving gradient flow during training (Xu et al. 2015).

2.3.5 Fully Connected Layer

The fully connected layer is the last layer in CNNs. It serves as an interface between the learned features and the output prediction. The fully connected layer takes the flattened output from the previous layers and connects every neuron to every neuron in the next layer. This means that each output neuron receives information from all activated features extracted by the earlier layers (Géron 2023). After identifying edges and textures through convolution and selecting the most prominent ones through pooling, the fully connected layers uses all this information to decide, for instance, whether the image represents a "cat" or "dog" in a image classification problem.

Each neuron in a fully connected layer computes a weighted sum of its inputs and a bias and applies an activation function (typically ReLU or softmax

for classification tasks). The output of this layer is then used for the final prediction, such as assigning probabilities to classes in a classification task.

$$\mathbf{y} = f(\mathbf{W} \cdot \mathbf{x} + \mathbf{b})$$

where:

- $\mathbf{y}$ is the output vector of the fully connected layer,
- $\mathbf{W}$ is the weight matrix,
- $\mathbf{x}$ is the input vector (flattened from the previous layer),
- $\mathbf{b}$ is the bias vector,
- f is the activation function.

There are few challenges associated with this as well. If the input is high dimensional, they become computationally expensive and their parameters increases. To resolve this, techniques such as Dropout and Batch Normalisation are used (G. Chen et al. 2019). Let's take a look at those.

2.3.6 Dropout

Dropout is a regularization technique used to reduce the overfitting. When training, dropout randomly drops out (deactivates) neurons in the network during each training step. They are temporarily ignored, meaning they don't take part in forward pass or back propagation (Srivastava et al. 2014). By doing this, the network is forced to not depend too much on any single neuron and instead learn more robust features that are useful in combination with many different parts of the network. This leads to better generalization when the model is tested on unseen data.

For example, if dropout is set to 0.5 in a fully connected layer, half the neurons will be turned off randomly during each training iteration. At test

time, all neurons are used, but their output is scaled to account for the ones that were dropped during training (Srivastava et al. 2014). The below Figure 2.6 shows an example of how Dropout works in a neural network.

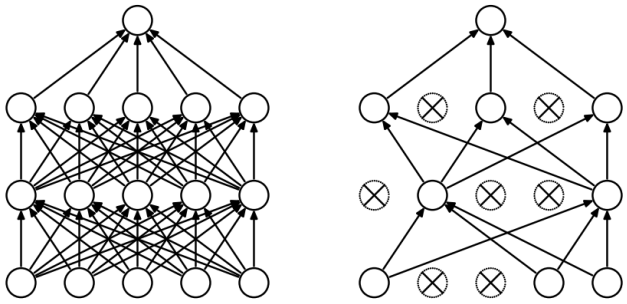

Figure 2.6: An Example of Dropout (Srivastava et al. 2014)

2.3.7 Batch Normalisation

Batch Normalization is used to reduce the problem of internal covariate shift in neural networks. In traditional neural networks, as the input data propagates through the network, the distribution of each layer's inputs changes. This phenomenon is known as internal covariate shift and it can slow down training process (Ioffe et al. 2015). Batch Normalization aims to reduce this issue by normalizing the inputs of each layer. It works by normalizing the data within each mini-batch. This means it calculates the mean and variance of data in a batch and then adjusts the values so that they have similar range. After that it scales and shifts the values so that model learn effectively. After standardizing, the method introduces two learnable parameters: one for scaling the input (gamma) and one for shifting it (beta). These allow the network to still represent complex functions even after normalization (Peerthum et al. 2023).

2.4 CNNs for Semantic Segmentation

In semantic segmentation, each pixel in the image is classified according to the class of the object it belongs to (e.g. road, car, sidewalk, building etc.) as shown in Figure 2.7. Note that different objects of the same class are not distinguished. For example, all the bicycles on the right side of the segmented image end up as one big lump of pixels (Géron 2023). Some of the use cases where semantic segmentation can be useful are: Autonomous vehicles use semantic segmentation to see the world around them and respond to them, medical diagnosis to detect tumors in MRI or CT scans.

Figure 2.7: Semantic Segmentation (Géron 2023)

The main difficulty in this task is that when images go through a regular CNN, they gradually lose their spatial resolution (due to the layers with strides greater than 1). So, a regular CNN may end up knowing that there's a person somewhere in the bottom left of the image, but it will not be much more precise than that.

2.4.1 History of Semantic Segmentation

Semantic segmentation has been through several phases starting from the traditional image processing. Techniques such as thresholding, edge detection, region growing and clustering (e.g., k-means) were used to group pixels based on low-level features like color and texture. But in complex scenes, it

was often brittle. To improve, probabilistic graphical models such as Conditional Random Fields (CRFs) were used (Krähenbühl et al. 2011). The machine learning methods such as support vector machines (SVM) and random forests offered improved classification. But they were still limited by the representational power of manually designed features.

One of the major advancements came with the introduction of deep learning particularly during the development of Fully Convolutional Networks (FCN). It transformed classification networks into dense prediction systems which is capable of learning end-to-end from input images to segmentation maps (Long et al. 2015). U-Net became a standard in biomedical segmentation tasks due to its encoder-decoder structure and skip connections (Ronneberger et al. 2015). In recent years, attention-based and transformers based models have further advanced this.

2.4.2 Working of Semantic Segmentation

Semantic segmentation typically begins with convolutional layers, where the model learns patterns such as edges, corners in early layers to detect object parts and full objects in deeper layers. But due to max pooling and strided convolutions the spatial resolution of the feature maps decreases, which can result in loss of fine details (Long et al. 2015). To improve segmentation performance and recover spatial details lost during downsampling, several enhancements have been proposed. The most significant among them include skip connections, dilated convolutions, autoencoder-based frameworks, and transformer-based models.

Skip connections

Skip connections, also known as shortcut connections connect earlier layers in a network to later layers, bypassing the intermediate layers. They are most commonly used in encoder-decoder architectures such as U-Net to transfer

spatial information from the encoder to the decoder (Ronneberger et al. 2015). This helps to preserve details such as edges and boundaries, which are lost during pooling operations. It improves segmentation accuracy and also helps to prevent vanishing gradients.

Dilated convolutions

Dilated convolutions, also known as atrous convolutions, are a variation of standard convolutions where the kernel is expanded by inserting spaces (or zeros) between its elements (L. C. Chen et al. 2018). This helps to increase the receptive field without increasing the number of parameters or reducing spatial resolution. This allows to capture more contextual information when identifying large or complex object.

Autoencoders

Autoencoders are neural networks trained to compress input data into a small and compressed form known as latent space and then reconstruct to its original form. Encoder compresses the input into a latent (compressed) representation whereas decoder attempts to reconstruct the input from this latent representation (Badrinarayanan et al. 2017). These are especially useful in applications like dietary analysis, where labeled data may be limited. They allow the network to focus on learning generalizable features while restoring spatial details during decoding.

Transformers

Transformer-based architectures use self-attention mechanisms to capture relationships between all parts of an image, allowing the model to understand long-range dependencies. Unlike traditional CNNs, which process local regions, transformers look at the entire image at once (Xie et al. 2021). This

helps in segmenting complex scenes where context matters, for example, distinguishing similar looking food items placed apart.

2.4.3 Semantic Segmentation Architectures

Semantic segmentation has seen rapid progress due to advancements in deep learning, especially convolutional neural networks (CNNs). Architectures have been proposed to address key challenges such as loss of spatial resolution, insufficient contextual information and difficulty in delineating object boundaries.

Fully Convolutional Networks

Fully Convolutional Networks (FCN) were the first deep learning models designed especially for semantic segmentation (Long et al. 2015). Unlike traditional CNNs used for classification, FCNs eliminate fully connected layers and replace them with 1×1 convolutions, allowing them to output spatially structured predictions of the same dimensions as the input image. Upsampling layers were used to restore the reduced-resolution feature maps back to the original input size. They also introduced skip connections from earlier layers to the final layers, helping recover spatial details lost during downsampling. While foundational, FCNs sometimes produce coarse outputs due to limited contextual awareness.

U-Net

U-Net was originally designed for biomedical image segmentation. It is based on an encoder-decoder structure with skip connections between layers in the encoder and decoder (Ronneberger et al. 2015). These skip connections help preserve fine grained spatial information and enable the network to accurately segment small structures. U-Net has been widely adopted in various domains

due to its simplicity and effectiveness, especially in scenarios with limited training data.

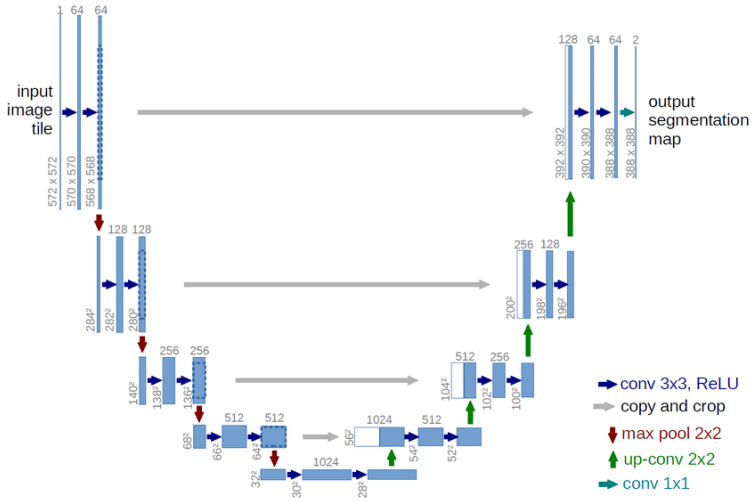

Figure 2.8: U-Net Architecture

SegNet

SegNet is another popular encoder-decoder architecture (Badrinarayanan et al. 2017). The encoder is typically a convolutional network such as VCG16 and the decoder uses the indices from max-pooling operations in the encoder to perform non-linear upsampling. This approach helps retain spatial information and reduces the number of trainable parameters in the decoder. SegNet is particularly effective for real-time applications due to its efficient design. Unlike U-Net or DeepLab, SegNet does not concatenate encoder features in the decoder which can limit the decoder's ability to recover fine details, especially for small or complex objects.

Attention-Based Models

Attention mechanisms have become a pivotal innovation in semantic segmentation by allowing models to focus solely on relevant features while ignoring

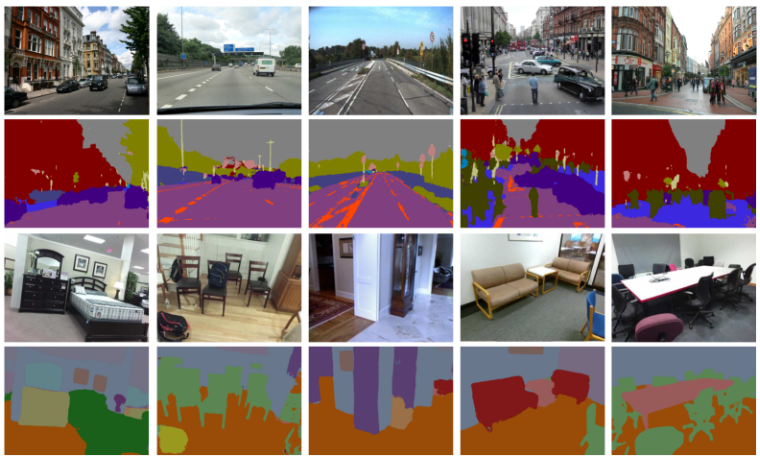

Figure 2.9: SegNet predictions on road scenes and indoor scenes.

the irrelevant information. They can be integrated into convolutional networks or form the core of Transformer-based designs. The three common types of attention used in semantic segmentation are Channel Attention, Spatial Attention, Self-Attention (Hu et al. 2018). Channel attention focuses on "what" is important by learning inter-dependencies between feature channels. For example, if the presence of certain textures (like rice or pasta) is crucial, channel attention will boost the relevant filters (Hu et al. 2018). Spatial attention focuses on "where" important features are located. It enhances spatial regions that are more likely to contain target objects while suppressing background noise. Self-Attention computes relationships between all pixel pairs in a feature map, enabling long-range dependency modeling. This is especially useful for segmenting objects that are spatially distant but semantically related (Wang et al. 2018).

SegFormer

SegFormer is a recent and a very powerful architecture developed specifically for semantic segmentation. Unlike traditional models that relies on convolution layers, SegFormer makes uses a Transformer based encoder along with

lightweight multilayer perception (MLP) (Xie et al. 2021). It can understand both the overall context of the image and fine granular details. It captures features at multiple scales without using fixed positional encodings which helps the model adapt better to different scenes. The decoder is lightweight and does not include complex components which makes the model fast and efficient while still maintaining high accuracy. SegFormer has shown strong performance on well known datasets like Cityscapes and ADE20K. It has many variants - B0, B1, B2, B3, B4, B5 ranging from small and fast (B0) to very large and better accuracy (B5). They differ from each other in their depth, number of layers and number of parameters. Its the balance between speed and accuracy that makes it especially useful for food segmentation, where identifying different ingredients clearly is important.

DeepLabV3

It is one of the most advanced and widely used architectures for semantic segmentation. It builds on the strengths of earlier DeepLab versions by combining atrous (dilated) convolutions and an encoder-decoder structure to capture both fine details and long-range context. The encoder uses a backbone network (such as Xception or ResNet) to extract features and incorporates the Atrous Spatial Pyramid Pooling (ASPP) module to aggregate multi-scale information. The decoder in DeepLabV3 upsamples low-resolution features and combines them with higher-resolution features from earlier layers, improving the segmentation of object boundaries. This dual-path design enables the model to retain spatial accuracy while also understanding complex scene layouts. DeepLabV3 achieved state-of-the-art performance on various dataset including PASCAL VOC and Cityscapes and is highly suitable for applications like food segmentation in dietetics, where distinguishing overlapping and diverse food items is critical.

Figure 2.10: DeepLabv3 Architecture with labeled blocks

YOLOv8 (You Only Look Once – Version 8)

YOLOv8 is the latest iteration in the YOLO family, originally designed for real-time object detection, but now also supporting instance and semantic segmentation (Pedro 2023). The key concept of this is decoupled head architecture, where classification, regression and segmentation outputs are processed independently, improving both accuracy and inference efficiency. In applications like dietary analysis, YOLOv8 could segment food items in real-time on a smartphone with impressive accuracy and low latency. Unlike traditional segmentation models like U-Net or DeepLabV3, YOLOv8 emphasizes speed and deployment efficiency, making it ideal for embedded systems or mobile applications.

2.4.4 Evaluation Metrics

Evaluating semantic segmentation models requires metrics that capture the quality of pixel-level predictions. Unlike image classification, where accuracy alone is often sufficient, semantic segmentation demands better evaluation to consider the class imbalance and boundary precision.

Accuracy

Accuracy refers to the percentage of pixels in an image that are correctly classified. The bounds are between 0 and 100 where 0 means no pixel is

correctly classified and 100 means all pixels are correctly classified.

$$\text{Pixel Accuracy (PA)} = \frac{TP + TN}{TP + TN + FP + FN}$$

Where:

- TP (True Positive): Pixels correctly predicted as belonging to a certain class.
- TN (True Negative): Pixels correctly predicted as not belonging to that class.
- FP (False Positive): Pixels incorrectly predicted as belonging to the class.
- FN (False Negative): Pixels incorrectly predicted as not belonging to the class.

Intersection over Union (IoU)

IoU measures the percentage overlap between the predicted segmentation and the ground truth. Higher IoU means better overlap between prediction and actual segmentation. The bounds are between 0 and 100 where 0 means no overlap between prediction and ground truth and 100 means perfect overlap.

$$\text{IoU} = \frac{TP}{TP + FP + FN}$$

Mean Intersection over Union (mIoU)

Averaged IoU over all classes, this is the most commonly used metric for evaluating semantic segmentation models where K is the number of classes.

$$\text{mIoU} = \frac{1}{K} \sum_{i=1}^{K} \frac{TP_i}{TP_i + FP_i + FN_i}$$

Frequency Weighted IoU (FWIoU)

FWIoU calculates the percentage IoU weighted by the frequency of each class in the dataset. This helps address class imbalance by giving more weight to classes that appear more frequently.

$$\text{FWIoU} = \sum_{i=1}^{K} \frac{n_i}{N} \times \frac{TP_i}{TP_i + FP_i + FN_i}$$

Where

- n_i = number of pixels belonging to class i
- N = total number of pixels across all classes

Choice of evaluation

Pixel accuracy measures the overall percentage of correctly classified pixels but it can be misleading in imbalanced datasets. For example, if the background occupies most of an image, the model can achieve high accuracy by simply predicting the background everywhere, even if it completely fails on smaller or less frequent objects. IoU and mIoU provide a more balanced evaluation because they directly measure the overlap between predicted and actual regions for each class. mIoU averages the IoU across all classes and gives equal importance to both frequent and rare categories which makes it more reliable for evaluating segmentation performance in datasets like FoodSeg103.

2.5 Datasets and Published results

One of the most critical components of any research is availability and identification of suitable datasets. In the context of dietetics, segmentation

is primarily applied to food images for identifying and labeling individual food items, estimating portion sizes. This section presents a comprehensive overview of the publicly available datasets relevant to semantic segmentation, particularly in the domain of food and nutrition, along with a summary of results achieved using state-of-the-art models on these datasets.

2.5.1 FoodSeg103

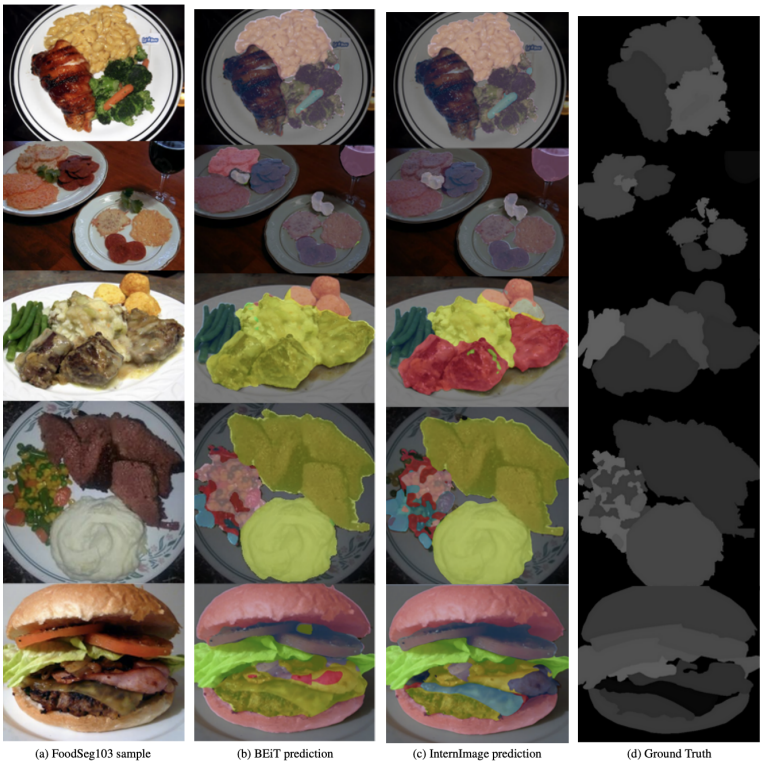

Figure 2.11: Performance of BEiT v2 and InternImage-B model on FoodSeg103 (Sinha et al. 2023).

It is one of the largest and most widely referenced datasets for pixel-level food image segmentation. It consists of over 7100 images annotated at the pixel level across 103 food categories, ranging from common fruits and veg-

etables to cooked dishes and processed items. The dataset is diverse in food types and image conditions which involves real life scenarios such as different lightning, background or even overlapping. The dataset emphasizes the challenge of intra-class variation and class imbalance, as some categories (e.g., rice, noodles) appear much more frequently than others. FoodSeg103 is publicly available and designed to support real-world applications such as calorie estimation, meal logging and food recognition in health apps. A convolutional model and a BEiT transformer were compared, reporting a new high of 49.4% mIoU using BEiT on FoodSeg103 (Sinha et al. 2023).

2.5.2 UNIMIB2016

Another important contribution to the semantic segmentation landscape was given by the UNIMIB2016 dataset. It was was introduced by Ciocca et al. (2016) as an extension of their prior UNIMIB2015 dataset, It was developed for university cafetaria environment in Milan-Bicocca university canteen, containing 1,027 tray images featuring approximately 3,616 segmented food instances across 73 food categories in real-life self-service contexts (Ciocca et al. 2017). Each image is annotated with pixel-level ground truth masks for 65 food categories, making it a valuable resource for dietetics-related applications. The additional challenges in this dataset includes overlapping, mixed dishes and variability in food presentation, which represent the real world conditions often than using studio quality datasets. The dataset emphasizes real-world challenges such as diverse lighting, overlapping and visually similar food items and backgrounds such as white plates on white trays, making it more demanding than standard studio datasets. The authors benchmarked the dataset using a CNN-based tray analysis pipeline, achieving around 79% recognition accuracy across all food classes. One of the limitations to this is that the food diversity is limited to the western culture making it less cultural diversity.

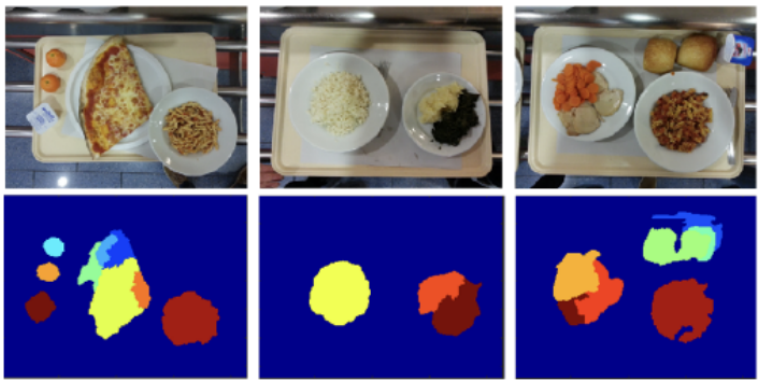

Figure 2.12: Examples of segmentation results of some UNIMIB2016 images.

2.6 Summary

This literature review explored the evolution of semantic segmentation techniques and their relevance to dietetics. It begun with a foundational overview, reviewing the core principles of Artificial Intelligence (AI), Machine Learning (ML) and Artificial Neural Networks (ANNs) including key concepts such as the perceptron, multilayer perceptrons (MLPs) and the critical role of backpropagation in training deep networks. These concepts laid the groundwork for understanding more advanced deep learning techniques such as Convolutional Neural Networks (CNNs) which have become the cornerstone of modern semantic segmentation tasks.

CNN architectures were explored including convolutional layers, pooling, activation functions, fully connected layers and enhancements such as dropout and batch normalization for regularization and optimization. Then we moved on to CNNs for semantic segmentation, introducing the encoder-decoder structure and essential enhancements like skip connections, dilated convolutions, autoencoders and transformer-based models. These techniques enable precise pixel-level labeling, which is critical for segmenting overlapping or visually complex food items in dietetics. Key architectures such as Fully

Convolutional Networks (FCNs), U-Net, SegNet, SegFormer and DeepLabV3 were discussed, each offering specific advantages in accuracy, efficiency and contextual awareness. A significant part of the review focused on publicly available datasets such as FoodSeg103 and UNIMIB2016.

In conclusion, the literature reveals a clear progression from basic neural architectures to highly specialized segmentation models capable of handling the complex requirements of dietetic analysis.

Chapter 3

Prototypes

This chapter provides an overview of the implementation of DeepLabV3, U-Net and SegFormer on the FoodSeg103 dataset. It covers the development environment, the architectural characteristics of each model and a few strategies such as WarmupPolyLR, Transfer learning, Focal Loss, Dice Loss and the implementation details of the training pipeline.

3.1 Introduction

Accurate segmentation of food items is essential for applications like nutrition monitoring which can lead to poor nutrition and disease if not done properly (Sinha et al. 2023). Traditional Convolutional segmentation architectures such as U-Net and DeepLabV3 have shown promising performance in pixel level segmentation tasks, but they often struggle with complex food scenes involving fine boundaries, overlapping food items and class imbalance. In the recent times, transformers based models such as SegFormer are being used as powerful alternatives due to their superior global context modeling and robust segmentation performance pixel level. Motivated by the above, this work presents a comparative study of DeepLabV3, U-Net and SegFormer on the FoodSeg103 dataset showcasing the strength and weakness in tackling and

overcoming the unique difficulties in food segmentation. FoodSeg103 is the most suitable dataset for training and evaluating food image segmentation models because it provides detailed pixel level annotations at the ingredient level which allows for more precise identification of each ingredient within a dish.

3.2 Development Environment

All development and experimentation of this comparison was made on University of Limerick's GPU using an "NVIDIA A100 80GB PCIe". The environment ran with Python version `3.10.18` and PyTorch version `2.7.1+cu128`. Additional libraries such as `segmentation_models_pytorch`, `transformers` and `torchvision` were used to load the predefined models. `NumPy`, `Pandas` and `scikit-image` were used for data manipulation and `Matplotlib` for visualization. All the experiments, whether successful or not, were saved in a private repository in GitHub for version control.

3.3 Architecture Details

3.3.1 DeepLabV3

```
1 from torchvision.models.segmentation import deeplabv3_resnet101,
  DeepLabV3_ResNet101_Weights
2
3 weights = DeepLabV3_ResNet101_Weights.DEFAULT
4 deeplab_model = deeplabv3_resnet101(weights=weights)
5 deeplab_model.classifier[4] = nn.Conv2d(256, 104, kernel_size=1)
6 deeplab_model = deeplab_model.to(DEVICE)
```

Figure 3.1: DeepLabV3 model

DeepLabV3 model with ResNet101 backbone was imported from torchvision package as seen in Line 1 in Figure 3.1. deeplabv3_resnet101 is a function that

returns the DeepLabV3 architecture and DeepLabV3_ResNet101_Weights gives us the pretrained weights. Line 3 loads the pretrained weights for the model. "DEFAULT" usually refers to the weights trained on the COCO dataset. These weights are trained to segment objects like people, animals, vehicles and various others from COCO. They do not know anything about food ingredients or classes in FoodSeg103 yet. The pretrained weights are then attached to the DeepLab model as seen in Line 4. This creates a DeepLabV3 model instance with the pretrained ResNet-101 backbone and ASPP module, initialized with the weights from COCO. The encoder i.e ResNet101 extracts the feature maps and the decoder predicts the segmentation masks for the 21 classes in COCO. But FoodSeg103 is different from COCO, it has 104 classes (103 food classes and 1 background class) rather than 21 present in COCO. So, the top layer has to be replaced to be compatible with FoodSeg103 segmentation as seen in Line 5. The model is then moved to the HARDWARE i.e CPU, GPU or TPU. To summarize, DeepLabV3 was originally trained on COCO dataset which contains generic objects. The encoder is retained to reuse its feature extractions and final classification layer is replaced with 104 classes (103 food classes and one background class). 256 is the number of input feature channels from the previous decoder layer and kernel_size=1 means a 1×1 convolution as seen in Line 5. The new classifier layer starts with random weights, while the encoder has pretrained weights that get fine-tuned on FoodSeg103.

3.3.2 U-Net

U-Net with ResNet101 backbone was imported from Segmentation Model PyTorch (SMP) library as seen in Line 1 in Figure 3.2. U-Net follows an encoder-decoder structure, where the encoder captures high-level features while the decoder reconstructs pixel-level segmentation maps. The encoder is set to "resnet101" as seen in Line 4 which is a deep residual network trained on "imagenet" as seen in Line 5. The number of input channels is set to 3

```python
import segmentation_models_pytorch as smp

unet_model = smp.Unet(
    encoder_name="resnet101",
    encoder_weights="imagenet",
    in_channels=3,
    classes= 104
)
```

Figure 3.2: U-Net model

as seen in Line 6 because it specifies the model that the input image will be RGB images. Again, the number of classes is set to 104 because FoodSeg103 has 103 food classes plus an additional class for the background. This is done because U-Net was originally designed for medical binary segmentation tasks, typically one or two classes. By changing the output layer to 104 channels, we match it to our tasks of 104 classes. While the encoder benefits from ImageNet pretraining, the decoder layers of U-Net are initialized with random values and trained to learn the different food images such as irregular shapes, overlapping ingredients.

3.3.3 SegFormer

```python
from transformers import SegformerForSemanticSegmentation
segformer_model = SegformerForSemanticSegmentation.from_pretrained(
    "nvidia/segformer-b2-finetuned-ade-512-512",
    num_labels= 104,
    ignore_mismatched_sizes=True
)
```

Figure 3.3: DeepLabV3 model

The last and final model for comparison is the SegFormer model which was introduced by NVIDIA in 2021. It is a hierarchical vision transformer encoder with a light MLP decoder. It comes in different variants - B0, B1,

B2, B3, B4, B5 ranging from small and fast (B0) to very large and accurate (B5). They vary from each other in their depth, number of layers, hidden dimensions and number of parameters. B0 is designed for mobile and real-time applications with fewer parameters, while B4 and B5 are the largest models, offering state-of-the-art accuracy at the cost of significantly higher computational and memory requirements. In our implementation, we selected the B2 variant, which provides an optimal balance between accuracy and GPU memory consumption, as the larger variants (B3, B4, B5) could not fit into our GPU memory during training due to limited hardware resources. The SegFormer model was pretrained on ADE20K dataset which consists of around 150 classes. The encoder (hierarchical vision transformer) was pretrained on ImageNet for general image understanding. The final layer was replaced with 104 classes i.e 103 food classes and background as seen in Line 4. The ignore_mismatched_sizes is set to True as seen in Line 5 because the model was initially trained on ADE20K dataset which consists of 150 classes whereas our dataset consists of 104 classes. This means that final layer in the pretrained model does not match with the newly defined classification layer. By setting it to True, all layers except the final segmentation head are loaded from the pretrained model, while the last layer is randomly initialized. If it is set to False (default), it will throw an error.

3.4 Implementation

This section explains the strategies and methods used to train and improve the semantic segmentation models - DeepLabV3, U-Net and SegFormer on FoodSeg103 Dataset.

3.4.1 Transfer Learning

Figure 3.4 is a custom function written by the author to freeze or unfreeze the encoder of any of the three models DeepLabV3, U-Net and SegFormer. The

```python
# A common function for freezing and unfreezing layers in
# DeepLab, U-Net and SegFormer
def freeze_layers(model, should_freeze):
    should_require_grad = not should_freeze

    if should_freeze:
        print("Freezing layers...")
    else:
        print("Unfreezing layers...")

    if hasattr(model, 'backbone'):  # DeepLab
        for param in model.backbone.parameters():
            param.requires_grad = should_require_grad
    elif hasattr(model, 'encoder'):  # U-Net
        for param in model.encoder.parameters():
            param.requires_grad = should_require_grad
    elif hasattr(model, 'segformer'):  # SegFormer
        for param in model.segformer.encoder.parameters():
            param.requires_grad = should_require_grad
```

Figure 3.4: A custom function for freezing/unfreezing the encoders in a model

should_freeze parameter decides whether the encoder layers will be frozen or unfrozen. If freezing is required, the requires_grad attribute of the encoder parameters is set to False, otherwise it is set to True as seen in Line 4. The function checks the model type by checking its attributes, such as 'backbone' for DeepLabV3, 'encoder' for U-Net, and 'segformer' for SegFormer. Based on this check, the encoder layers are selectively frozen or unfrozen according to the training requirement. During the first five epochs, the encoder (backbone) layers are frozen, meaning their weights remain unchanged while only the decoder or segmentation head is trained. This prevents the randomly initialized decoder from producing unstable gradients that could disrupt the useful pretrained features learned on datasets like ImageNet or ADE20K. After the 5 epochs, the encoders are unfrozen, allowing the entire network to fine-tune and adapt to the specific characteristics of the FoodSeg103 dataset, such as complex textures, small ingredients and irregular boundaries such as overlapping. This leads to stable training, faster convergence, improved accuracy and mIoU compared to training all layers from scratch or keeping

the encoder frozen throughout.

```
1  def train_deeplab(num_epochs, model, optimizer, train_loader, scheduler,
   freeze_epochs=5):
2
3      model.train()
4      total_loss = []
5      start_time = time.time()
6      freeze_layers(model, True)
7
8      for epoch in range(num_epochs):
9
10         if epoch == freeze_epochs:
11             freeze_layers(model, False)
12
13         print(f"\nEpoch {epoch+1}/{num_epochs}")
14         epoch_start = time.time()
15         train_loss = 0.0
16
17         progress_bar = tqdm(train_loader, desc= f"Epoch: {epoch + 1} / {num_epochs}",
18                             unit="batch")
19
20         for i, (images, labels) in enumerate(progress_bar, 1):
21             images, labels = images.to(DEVICE), labels.to(DEVICE).long()
22             optimizer.zero_grad()
23             outputs = model(images)['out']
24             loss = combined_loss(outputs, labels)
25             loss.backward()
26             optimizer.step()
27             train_loss += loss.item()
28
29             progress_bar.set_postfix(avg_loss= train_loss/i)
30
31         scheduler.step()
32
33         avg_loss = train_loss / len(train_loader)
34         total_loss.append(avg_loss)
35         epoch_end = time.time()
36         epoch_time = epoch_end - epoch_start
37         print(f"Epoch {epoch+1} completed in {epoch_time/60:.2f} min | Avg Loss:
   {avg_loss:.4f}")
38
39     total_time = time.time() - start_time
40     print(f"\nTotal Training Time: {total_time:.2f} seconds ({total_time/60:.2f}
   minutes)")
```

Figure 3.5: Freezing / Unfreezing in DeepLabV3 Implementation

In Figure 3.5, the training begins with the encoder layers frozen as seen in line 6 to preserve the pretrained features in the encoder. Once the specified number of freezing epochs is reached, the layers are unfrozen as seen in line 11 allows the entire model to be fine-tuned on the FoodSeg103 dataset.

3.4.2 WarmupPolyLR

The WarmupPolyLR class (Figure 3.6) defines a custom learning rate scheduler that combines a linear warmup phase followed by a polynomial decay strategy to control the learning rate during training (Xie et al. 2021). It inherits from "_LRScheduler" and adjusts the learning rate at each epoch.

```
1  class WarmupPolyLR(torch.optim.lr_scheduler._LRScheduler):
2      def __init__(self, optimizer, warmup_epochs, total_epochs, base_lr, end_lr=0.0,
   power=0.9, last_epoch=-1):
3          self.warmup_epochs = warmup_epochs
4          self.total_epochs = total_epochs
5          self.base_lr = base_lr
6          self.end_lr = end_lr
7          self.power = power
8          super(WarmupPolyLR, self).__init__(optimizer, last_epoch)
9
10     def get_lr(self):
11         current_epoch = self.last_epoch + 1
12         if current_epoch <= self.warmup_epochs:
13             # Linear warmup
14             return [1e-6 + (self.base_lr - 1e-6) * current_epoch / self.warmup_epochs for
   _ in self.base_lrs]
15         else:
16             # Poly decay. Obtained from https://arxiv.org/pdf/1706.05587 in the 4.1
   Training Protocol section
17             iter = current_epoch - self.warmup_epochs
18             max_iter = self.total_epochs - self.warmup_epochs
19             progress = iter / max_iter
20             poly_factor = (1 - progress) ** self.power
21             return [self.end_lr + (self.base_lr - self.end_lr) * poly_factor for _ in
   self.base_lrs]
```

Figure 3.6: A custom class for modifying the learning rate

In the initalisation phase, parameters such as warmup_epochs, total_epochs, base_lr,end_lr, power and last_epoch determine how the learning rate will change throughout training. The warmup epochs determines the warmup phase. During this phase, the learning rate starts from a very small value and increases linearly as seen in Line 12 - 14. The base_lr denotes the initial main learning rate after warmup. The power parameter determines the rate of decay. The last epoch allows resuming training from a specific epoch if needed. A default value of -1 means the scheduler starts fresh from epoch 1. In the get_lr() method, if the current epoch is within the warmup phase, the learning rate increases linearly from a very small value up to base_lr, ensuring a smooth start and avoiding unstable updates in the early epochs as seen in Line 12 - 14. The value 1e-6 is an arbitrary small number chosen to ensure that the learning rate starts from a non-zero value. Starting with zero would prevent any weight updates in the initial steps. Therefore, using a small positive value helps initiate training smoothly while keeping the updates minimal during the warmup phase. After the warmup period, the learning rate decays polynomially according to the equation 3.1.

$$poly_{factor} = (1 - iter/max\_iter)^{power} \tag{3.1}$$

The iter determines the number of epochs have passed since warmup finished as seen in Line 17. The max_iter determines the total number of epochs remaining after warmup as seen in Line 18. The progress tells how far are we along in the decay phase as seen in Line 19. The poly factor (See Equation 3.1) is calculated obtained from the values. If the optimizer has multiple parameter groups, then it is applied to each group as seen in Line 21.

3.4.3 Focal and Dice Loss

In segmentation datasets like FoodSeg103, some classes dominate (such as background or large food items) while other classes (such as small ingredients) are very rare. Generally in food image segmentation, the distribution of ingredients is highly imbalanced with many ingredients appearing rarely. As a result, the model struggles to accurately predict these less frequent and long tail categories. Generally, we use CrossEntropyLoss which is good for most of the cases. But it treats all the classes same and in cases like long-tailed datasets the small classes often get ignored which can cause the model to predict poorly. To resolve this issue, Focal and Dice loss are preferred. Focal Loss is a variant of CrossEntropyLoss.

Focal Loss

$$FL(p_t) = -\alpha(1 - p_t)^{\gamma} \log(p_t) \tag{3.2}$$

- p_t is the predicted probability for the correct class. If the model is confident ($p_t = 0.90$), the loss should be small. If the model is less confident ($p_t = 0.30$), then the loss will be large
- $(1 - p_t)$ is called the focusing term. The higher the p_t value, the less the value of focusing term, contributing less to the loss. On the other

hand, the lower the p_t value, the higher the value of focusing term, contributing more to the loss.

- γ denotes how much we should focus on the hard examples
- α is the balancing factor that adjusts the weight of different classes to handle class imbalance. For instance, if background pixels dominate, we can give smaller α to the background class to reduce its impact.
- $\log(p_t)$ is the standard Cross Entropy part that penalizes the wrong predictions

```
1  class FocalLoss(nn.Module):
2      def __init__(self, alpha=1.0, gamma=2.0, ignore_index=255, reduction='mean'):
3          super(FocalLoss, self).__init__()
4          self.alpha = alpha
5          self.gamma = gamma
6          self.ignore_index = ignore_index
7          self.reduction = reduction
8
9      def forward(self, inputs, targets):
10         ce_loss = F.cross_entropy(inputs, targets, ignore_index=self.ignore_index,
   reduction='none')
11         pt = torch.exp(-ce_loss)  # Probability of correct class
12         focal_loss = self.alpha * (1 - pt) ** self.gamma * ce_loss
13         return focal_loss.mean()
```

Figure 3.7: Focal Loss Code

Figure 3.7 shows the custom FocalLoss class implementation. The parameters in the initialisation such as alpha (α), Gamma (γ) are explained above in the equation 3.2. Ignore index ignores certain index such as 255 and reduction with mean averages the loss across pixels. In the forward method, the standard cross entropy is calculated as seen in Line 10 and 11. Once it is found, then it is used in the focal loss formula (See equation 3.2) in the line 12. Finally in line 13, the mean loss is computed across all the pixels.

Dice Loss

$$DiceLoss = 1 - \frac{2|P \cap G| + \epsilon}{|P| + |G| + \epsilon} \tag{3.3}$$

- P is the predicted mass probabilities
- G is the ground truth mask
- ϵ is the smoothing term to avoid division by zero

```
1 def dice_loss(pred, target, smooth=1.0):
2     pred = torch.softmax(pred, dim=1)
3     target_onehot = F.one_hot(target, num_classes=pred.shape[1]).permute(0, 3, 1,
  2).float()
4     intersection = (pred * target_onehot).sum(dim=(2, 3))
5     union = pred.sum(dim=(2, 3)) + target_onehot.sum(dim=(2, 3))
6     return 1 - ((2 * intersection + smooth) / (union + smooth)).mean()
```

Figure 3.8: Dice Loss Code

Dice loss measures how much the predicted segmentation mask overlaps with the ground truth mask. If it is a perfect 100% overlap then loss is 0, if it is 0% overlap then the loss is 1. In line 2, the model's raw outputs prediction are passed through a softmax function along dim=1 to convert the logits into probabilities for each class. In line 3, the ground truth target, which contains class IDs for each pixel, is converted into a one-hot encoded tensor using F.one_hot, with shape [N, C, H, W] (where N is the batch size, C is the number of classes, and H and W are the height and width of the image). This conversion ensures that each class has its own channel, making it easier to compare with pred. The intersection term is calculated by multiplying the predicted probabilities with the one-hot ground truth and summing over all pixels, representing the overlapping regions. The union is the total predicted probabilities plus the total ground truth pixels for each class. The values of intersection and union along with the smooth factor (ϵ) is substituted in the equation 3.3

Combined Loss

$$CombinedLoss = 0.5 * FocalLoss + 0.5 * DiceLoss \tag{3.4}$$

- FocalLoss is the value of the Focal Loss

- DiceLoss is the value of the Dice Loss

```
1 def combined_loss(outputs, labels):
2     focal_loss = FocalLoss(ignore_index=255)
3     f_loss = focal_loss(outputs, labels)
4     d_loss = dice_loss(outputs, labels)
5     return 0.5 * f_loss + 0.5 * d_loss
```

Figure 3.9: Combination of both Focal and Dice Loss

The combined_loss function in the Figure 3.9 combines both Focal Loss and Dice Loss. The Focal Loss class is instantiated with ignore index set to 255 as seen in Line 2. The value of focal loss and dice loss is obtained as seen in Line 3 and 4. The function returns the average of both focal and dice loss. Focal Loss is used to overcome the class imbalance problem by focusing more on difficult pixels (which are misclassified or rare classes) and reduces the impact of easy classes. Dice Loss measures the quality of overlap and shape matching between predicted and ground truth masks, ensuring better boundary accuracy and overall segmentation quality. By combining both, we are leveraging the power of both worlds to generate higher mIoU and accuracy especially in long tailed datasets like FoodSeg103.

3.5 Training

```
1 outputs = model(images)
2 logits = outputs.logits
3 logits = F.interpolate(logits, size=labels.shape[1:], mode="bilinear",align_corners=False)
4 loss = combined_loss(logits, labels)
```

Figure 3.10: SegFormer output interpolation code

DeepLabV3 and U-Net follow a conventional training loop but SegFormer requires a slightly different approach due to the structure of its output. In the case of DeepLabV3 and U-Net, the model directly returns the segmentation

logits, which can be passed immediately to the loss function as seen in Line 23 and 24 in Figure 3.5.

On the other hand, SegFormer returns the output under the .logits attribute as seen in Line 2 in Figure 3.10. Additionally, the output resolution from SegFormer might not always match the target label resolution. Hence, an interpolation step is added to resize the predictions before computing the loss as seen in Line 3. The remaining training for SegFormer is same as the conventional training in DeepLabV3 and U-Net.

3.6 Summary

This chapter outlined the implementation setup used to train and evaluate the models - DeepLabV3, U-Net and SegFormer on the FoodSeg103 dataset. It began with the prepossessing steps applied including resizing, normalization and data augmentation techniques like random flipping. The dataset was setup using a custom PyTorch Dataset class to handle both input images and pixel-level labels. Pretrained models such as DeepLabV3, U-Net and SegFormer were initialized with pretrained weights and adapted to the desired 104 classes in the FoodSeg103 dataset. Key training strategies like encoder freezing/unfreezing, the use of custom loss functions including CrossEntropy, Dice, Focal and learning rate scheduling through WarmupPolyLR were implemented to improve model performance. The evaluation included the computation of metrics such as Pixel Accuracy and mean Intersection over Union (mIoU) along with visual prediction outputs. This chapter forms the basis for the experimental comparisons discussed in the next chapter.

Chapter 4

Empirical Studies

4.1 Introduction

In this chapter, we present a empirical analysis comparing three different widely used semantic segmentation models - DeepLabV3, U-Net and SegFormer on the FoodSeg103 Dataset. The purpose is to find the generalization capability of these models when applied to complex food images such as FoodSeg103. Each experiment investigates different combinations of training strategies, hyperparameters and loss functions to understand their impact on segmentation performance.

4.1.1 Dataset

The dataset used for comparison here is **FoodSeg103**. As the name suggests, the dataset has 103 finely labeled food categories and the background class and is known for its complex visual structure, long-tailed distributions and overlapping ingredients. The dataset is split into 4983 training set images and 2135 validation images. Each image is of size 512x384. Transformations and data augmentations were applied during the training to improve the generalisation.

4.1.2 Data Transformations

Before training, several image transformations were applied to make sure that the model can generalize well from the FoodSeg103 dataset. Normalisation was performed using mean and standard deviation values, (0.485, 0.456, 0.406) for the mean and (0.229, 0.224, 0.225) for the standard deviation. The three values for mean and standard deviation correspond to the Red, Green, and Blue channels of an image. Each channel is normalized separately to match the distribution of the data the pretrained models were trained on. This was done to scale the pixel values so that the input distribution matched the one used when training the pretrained encoders on ImageNet. This helps to converge faster and stabilize training. Random horizontal flipping with a probability of 0.5 was applied to allow the model to see the inverted images. This helps to improve the model and able to recognize regardless of orientation. All the images were resized to 512x384 to ensure uniformity across batches and compatibility with the model architectures. These steps not only acts as a way to standardize the inputs but also acts as form of augmentation, reducing overfitting and also improve the performance of the model.

4.2 Experiments

4.2.1 Experiment 1

Objectives: The objective of this experiment is to set a baseline performance for the three models. This experiment was done to evaluate the performance without introducing any special loss functions, learning rate or training enhancements. The results from this experiment serves as a base against which the subsequent experiments can be measured.

Models: DeepLabV3 (ResNet-101), U-Net (ResNet-101), SegFormer (B2

variant).

Loss functions: CrossEntropy Loss

Scheduler: PolyLR scheduler

Batch Size: 8 for training and 4 for validation

Hyper-parameters

Table 4.1: Hyperparameter Configuration for Experiment 1

Hyper-parameter / Transformations	**Values**
Learning Rate	6e-5
Input Size	512×384
Flip Probability	0.5
Mean	(0.485, 0.456, 0.406)
Standard Deviation	(0.229, 0.224, 0.225)

Results

Table 4.2: Evaluation Metrics for Experiment 1

Model	**Accuracy (%)**	**mIoU (%)**	**Runtime**
DeepLabV3	77.84	26.83	6 hours
U-Net	75.29	16.32	7 hours
SegFormer	77.74	28.82	7 hours

Evaluation

SegFormer performed the best with mIoU of 28.82% which is closely followed by DeepLabV3 with mIoU of 26.83% while U-Net has low mIoU of 16.32%. In terms of Accuracy, both DeepLabV3 (77.84%) and SegFormer (77.74%) performed almost identical. The loss curves in Figure 4.1 shows the smooth convergence of all three models. SegFormer shows faster and early convergence which shows it learns the feature more quicker. U-Net on the other hand, showed convergence more slowly and had high loss at the end of the training which shows its low segmentation accuracy. The Figure 4.2 shows

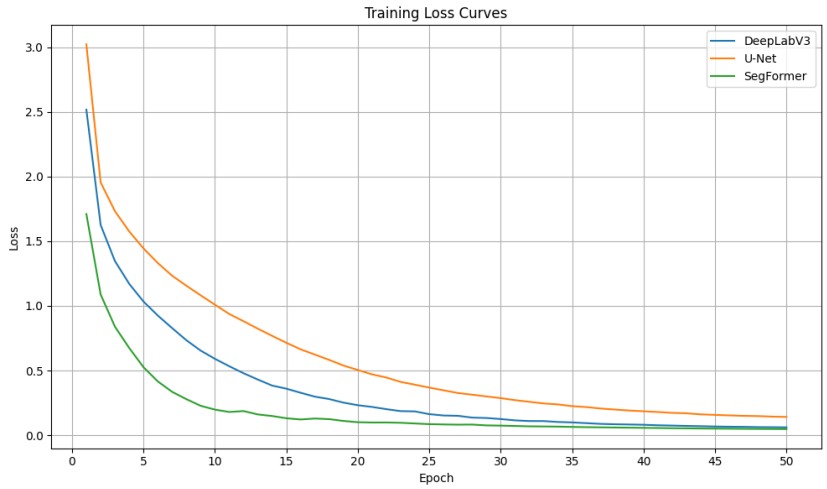

Figure 4.1: Loss progression for DeepLabV3, U-Net and SegFormer during training for Experiment 1

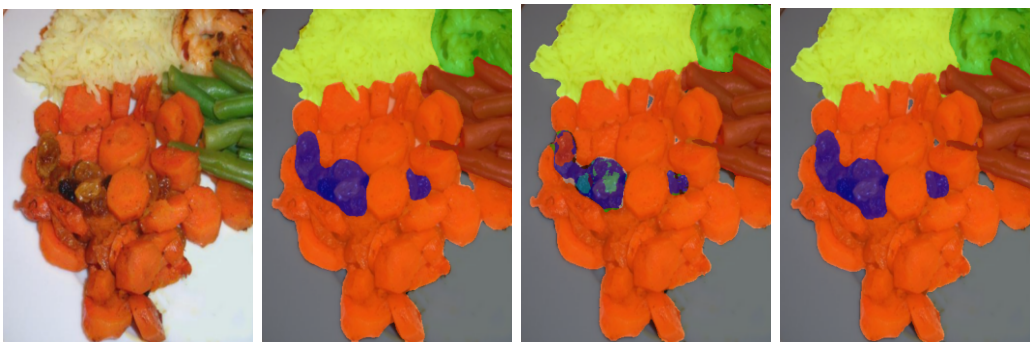

Figure 4.2: Original Image, DeepLabV3, U-Net, SegFormer (Left to Right) for Experiment 1

the SegFormer produces more precise and complete segmentation masks especially around complex object boundaries, whereas DeepLabV3 also performs good but with slightly less detail preservation. U-Net struggles around the overlapping ingredients. Overall, transformer based model SegFormer and advanced convolution model DeepLabV3 performs better than U-Net. This acts as baseline for improvement in the subsequent experiments.

4.2.2 Experiment 2

Objectives: The objective of this experiment was to explore whether using the hyperparameters and transformations from the research paper *"Transferring Knowledge for Food Image Segmentation using Transformers and Convolutions"* by Sinha et al. (2023) could improve the performance of DeepLabV3, U-Net and SegFormer. The training setup is same as Experiment 1 but the settings are modified to check whether it would improve the performance.

Models: DeepLabV3 (ResNet101), U-Net (ResNet101), SegFormer (B2 variant)
Loss functions: CrossEntropy Loss
Scheduler: PolyLR scheduler
Optimizer: AdamW with a base learning rate of 6e-5
Batch Size: 8 for training and 4 for validation

Hyper-parameters

Table 4.3: Hyperparameter Configuration for Experiment 2

Hyper-parameter / Transformations	**Values**
Learning Rate	6e-5
Input Size	512×384
Flip Probability	0.5
Mean	(0.485, 0.456, 0.406)
Standard Deviation	(0.229, 0.224, 0.225)

Results

Table 4.4: Evaluation Metrics for Experiment 2

Model	**Accuracy (%)**	**mIoU (%)**	**Runtime**
DeepLabV3	77.99	27.37	6 hours
U-Net	75.00	16.73	7 hours
SegFormer	78.80	32.00	7 hours

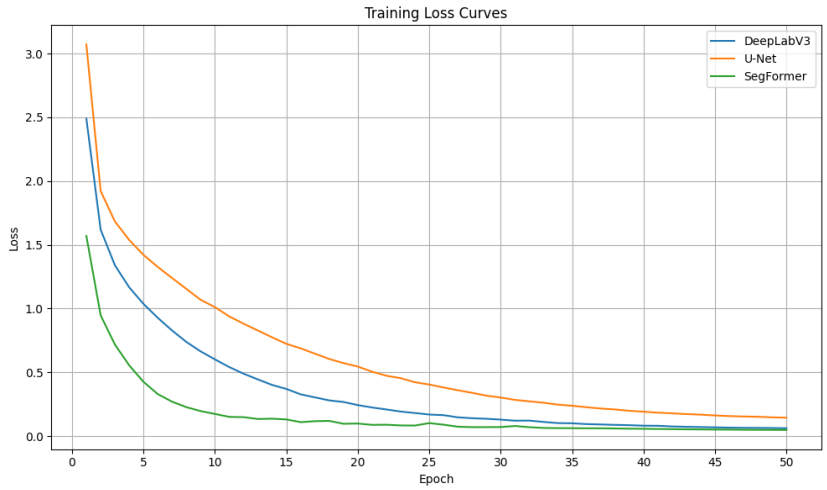

Figure 4.3: Loss progression for DeepLabV3, U-Net and SegFormer during training for Experiment 2

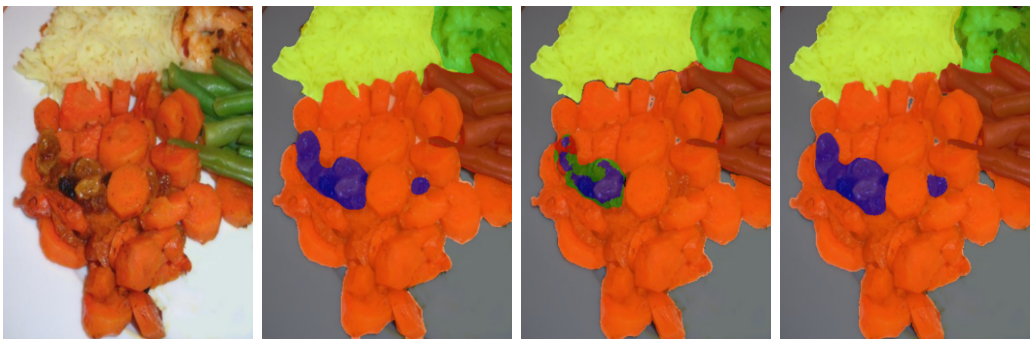

Figure 4.4: Original Image, DeepLabV3, U-Net, SegFormer (Left to Right) for Experiment 2

Evaluation

Just like the previous experiment, SegFormer achieved the highest mIoU of 32.00% and additionally slightly better with accuracy of 78.80%. It is then followed by DeepLabV3 with the mIoU of 27.37% and accuracy of 77.99%. U-Net performance was the lowest showing only slightest improvement over the previous experiment (from 16.32% to 16.73%). The loss curves in Figure 4.3 shows that SegFormer converged quickly with a steep drop in loss in the initial phase of training. DeepLabV3 also benefited from the updated hyperparameters showing a slightly smoother and more stable convergence. U-Net's loss curve though improved still lagged behind which suggests its architecture is less suited to the complexities of FoodSeg103. From Figure 4.4, we can see that SegFormer produced a cleaner segmentation boundaries and fewer misclassifications than Experiment 1, while DeepLabV3 also showed some improvements. U-Net's masks remained inconsistent just like previous experiment showing inconsistency across overlapping ingredients. SegFormer and DeepLabV3 showed better accuracy and mIoU when trained with chosen parameters from the research paper.

A comparison of the accuracy and mIoU results for all ten experiments is presented in section 4.3.

4.2.3 Experiment 3

Objectives: The objective of this experiment was to check whether combining Cross Entropy Loss with Dice Loss could improve segmentation performance. While Cross-Entropy Loss focuses on pixel-wise classification accuracy, Dice Loss optimises the overlap between predicted and ground-truth segment. By applying a weighted combination of 0.7 of Cross Entropy and 0.3 of Dice Loss, this experiment was to determine whether strengths of both loss could be leveraged to improve better mIoU and accuracy.

Models: DeepLabV3 (ResNet101), U-Net (ResNet101), SegFormer (B2 variant)
Loss functions: CrossEntropy & Dice Loss
Scheduler: PolyLR scheduler
Optimizer: AdamW with a base learning rate of 6e-5
Batch Size: 8 for training and 4 for validation

Hyper-parameters

Table 4.5: Hyperparameter Configuration for Experiment 3

Hyper-parameter / Transformations	**Values**
Learning Rate	6e-5
Input Size	512×384
Flip Probability	0.5
Mean	(0.485, 0.456, 0.406)
Standard Deviation	(0.229, 0.224, 0.225)

Results

Table 4.6: Evaluation Metrics for Experiment 3

Model	**Accuracy (%)**	**mIoU (%)**	**Runtime**
DeepLabV3	78.14	32.37	6 hours
U-Net	74.67	21.46	6 hours
SegFormer	77.91	37.18	7 hours

Evaluation

The combination of Dice Loss and Cross Entropy loss showed a notable performance for all the three models compared to Experiment 2. SegFormer achieved the highest mIoU of 37.18% and a good pixel accuracy at 77.91%. DeepLabV3 showed increase in mIoU of 32.37% while U-Net improved to 21.46%. The loss curves in Figure 4.5 showed smooth convergence for all

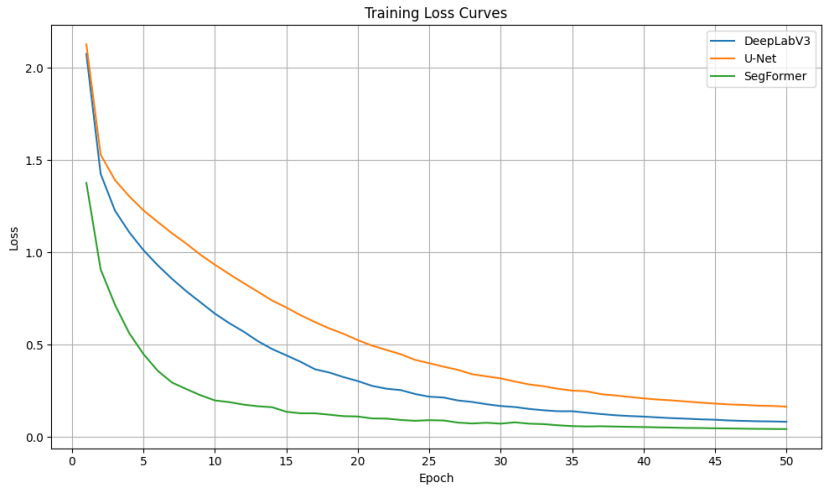

Figure 4.5: Loss progression for DeepLabV3, U-Net, and SegFormer during training for Experiment 3

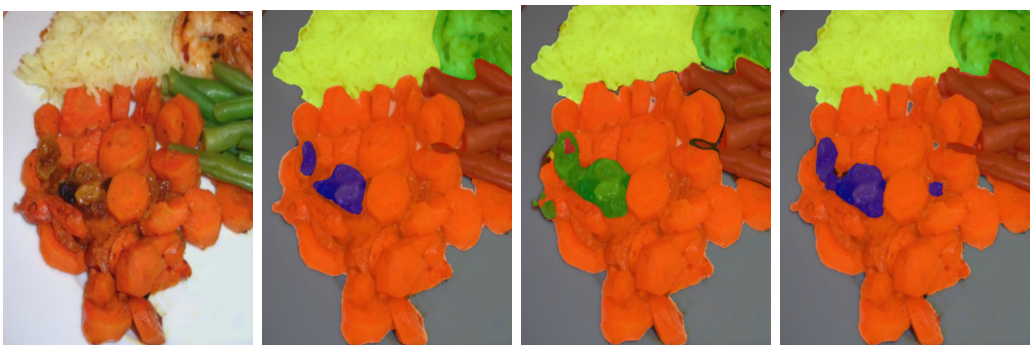

Figure 4.6: Original Image, DeepLabV3, U-Net, SegFormer (Left to Right) for Experiment 3

the three models compared to Experiment 2 showing more stable optimization. SegFormer showed the highest improvement of about +5.18% and DeepLabV3 of about +5.00% in mIoU from the previous experiments. The Figure 4.6 showed that models showed improvement around the boundaries and the overlapping ingredients. Overall, this experiment showed the effectiveness of combination of both Cross Entropy Loss and Dice Loss improving

the efficiency of all three models.

4.2.4 Experiment 4

Objectives: The objective of this experiment was to check whether increasing the training time from 50 to 100 epochs would improve SegFormer's performance. Since SegFormer consistently outperformed DeepLabV3 and U-Net in the earlier experiments, only SegFormer was retained for the next few experiments. The optimizers, learning rate scheduler and other settings are same as previous experiment. This will show the effects of training for longer time,

Models: SegFormer (B2 variant)
Loss functions: CrossEntropy & Dice Loss
Scheduler: PolyLR scheduler
Optimizer: AdamW with a base learning rate of 6e-5
Batch Size: 8 for training and 4 for validation

Hyper-parameters

Table 4.7: Hyperparameter Configuration for Experiment 4

Hyper-parameter / Transformations	**Values**
Learning Rate	6e-5
Input Size	512×384
Flip Probability	0.5
Mean	(0.485, 0.456, 0.406)
Standard Deviation	(0.229, 0.224, 0.225)

Results

Table 4.8: Evaluation Metrics for Experiment 4

Model	**Accuracy (%)**	**mIoU (%)**	**Runtime**
SegFormer	79.08	36.57	7 hours

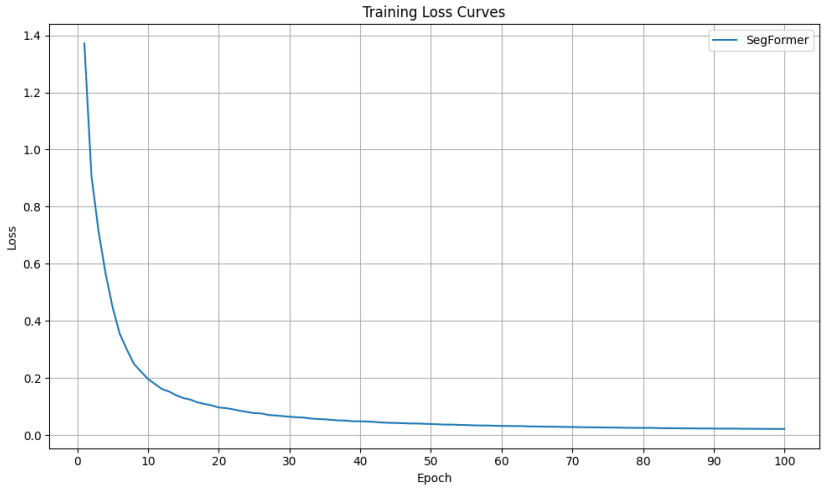

Figure 4.7: Loss progression for SegFormer during training for Experiment 4

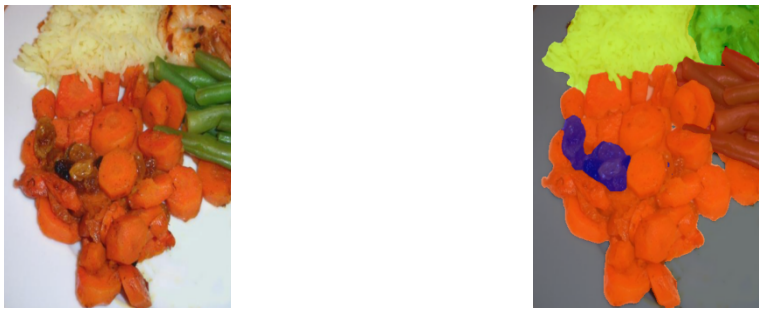

Figure 4.8: Original Image and SegFormer prediction for Experiment 4

Evaluation

The extended training duration with SegFormer achieved of an accuracy of 79.08% and an mIoU of 36.57% when trained for 100 epochs. Compared

to experiment 1 which has an accuracy of 77.74% and 28.82% represents a noticeable improvement, shows the benefit of extended training duration. The loss curves in Figure 4.7 showed a steep decline during the first 15 epochs which is followed by a slow plateau showing that most learning happened early in the training process. However, the improvement in accuracy were relatively small (increase in 2% from base experiment), shows that longer training alone does not improve generalization. In Figure 4.8, we can see that the predictions were sharper than baseline experiment.

4.2.5 Experiment 5

Objectives: The objective of this experiment is to check whether replacing the learning rate scheduler would improve the performance of the SegFormer model. WarmupPolyLR scheduler was used in place of Poly LR scheduler with an initial warmup phase to polynomial decay and the training time was kept for 50 epochs.

Models: SegFormer (B2 variant)
Loss functions: CrossEntropy & Dice Loss
Scheduler: WarmupPolyLR scheduler
Optimizer: AdamW with a base learning rate of 6e-5
Batch Size: 8 for training and 4 for validation

Hyper-parameters

Table 4.9: Hyperparameter Configuration for Experiment 5

Hyper-parameter / Transformations	**Values**
Learning Rate	6e-5
Input Size	512×384
Flip Probability	0.5
Mean	(0.485, 0.456, 0.406)
Standard Deviation	(0.229, 0.224, 0.225)

Results

Table 4.10: Evaluation Metrics for Experiment 5

Model	**Accuracy (%)**	**mIoU (%)**	**Runtime**
SegFormer	78.55	38.22	6 hours

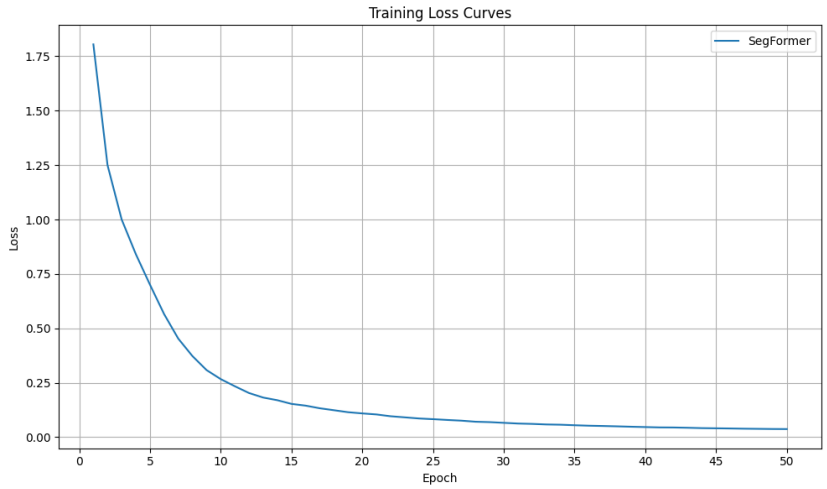

Figure 4.9: Loss progression for SegFormer during training for Experiment 5

Figure 4.10: Original Image and SegFormer prediction for Experiment 5

Evaluation

When applying only WarmupPolyLR scheduler, SegFormer achieved 78.55% accuracy and a higher 38.22% mIoU. Even though there is a drop in accuracy compared to Experiment 4, the increase in mIoU indicates the better segmentation quality and better overlap with ground truth masks. This shows that scheduling helped the model focus in regions with overlapping or complex food items. The loss curves in Figure 4.9 showed a steep decline during the first 15 epochs which is followed by a slow plateau showing that most learning happened early in the training process.

4.2.6 Experiment 6

Objectives: The objective of this experiment was to check whether combining both the previous two experiment's changes i.e increasing the training duration from 50 to 100 epochs and the use of WarmupPolyLR would improve the performance of the model.

Models: SegFormer (B2 variant)
Loss functions: CrossEntropy & Dice Loss
Scheduler: WarmupPolyLR scheduler
Optimizer: AdamW with a base learning rate of 6e-5
Batch Size: 8 for training and 4 for validation

Hyper-parameters

Table 4.11: Hyperparameter Configuration for Experiment 6

Hyper-parameter / Transformations	**Values**
Learning Rate	6e-5
Input Size	512×384
Flip Probability	0.5
Mean	(0.485, 0.456, 0.406)
Standard Deviation	(0.229, 0.224, 0.225)

Results

Table 4.12: Evaluation Metrics for Experiment 6

Model	**Accuracy (%)**	**mIoU (%)**	**Runtime**
SegFormer	78.86	36.80	7 hours

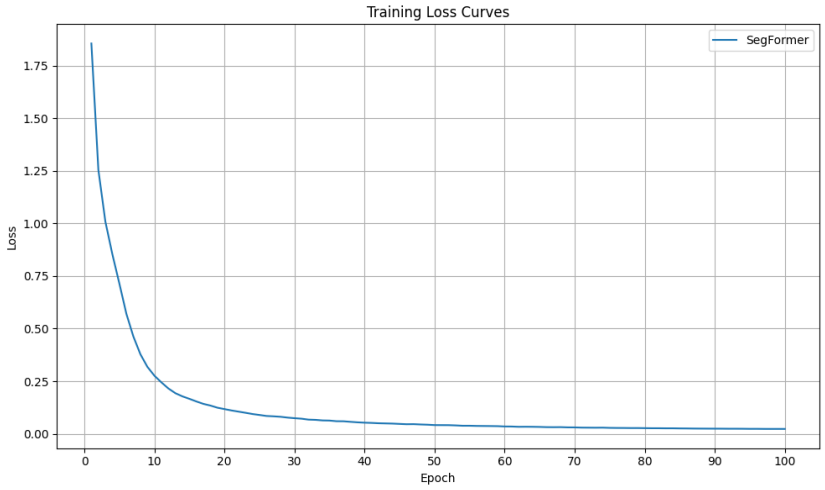

Figure 4.11: Loss progression for SegFormer during training for Experiment 6

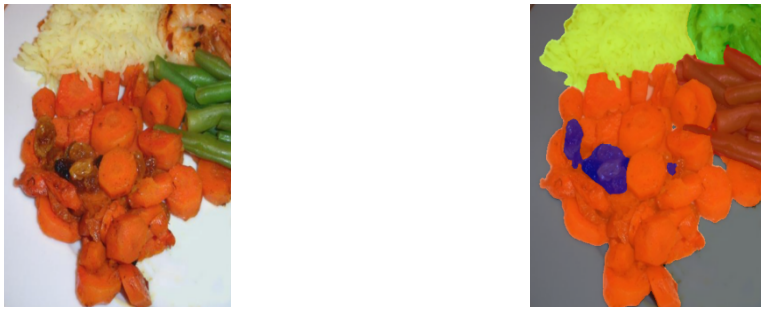

Figure 4.12: Original Image and SegFormer prediction for Experiment 6

Evaluation

The extended training of 100 epochs along with WarmupPolyLR achieved accuracy of 78.86% accuracy and 36.80% mIoU. This performed better than Experiment 4 but did not perform better than Experiment 5. This shows that combining more epochs together with scheduling achieved only limited improvements. Compared to earlier experiments, SegFormer still outperformed DeepLabV3 and U-Net but within the SegFormer trials, Experiment 5 gave the best results which shows that learning rate scheduling was more effective than simply training longer.

4.2.7 Experiment 7

Objectives: The objective of this experiment is to check whether freezing-unfreezing approach together with WarmupPolyLR would improve the segmentation performance. For the first five epochs, the encoder layers of each model were frozen to retain its pretrained features while allowing only the decoder layers to learn the specific patterns. After this phase, the encoders were unfrozen to enable full fine-tuning. This gradual adaptation was intended to stabilize early training, avoid catastrophic forgetting and allow for more refined learning in later epochs.

Models: DeepLabV3 (ResNet101), U-Net (ResNet101), SegFormer (B2 vari-

ant)
Loss functions: CrossEntropy & Dice Loss
Scheduler: WarmupPolyLR scheduler
Optimizer: AdamW with a base learning rate of 6e-5
Batch Size: 8 for training and 4 for validation

Hyper-parameters

Table 4.13: Hyperparameter Configuration for Experiment 7

Hyper-parameter / Transformations	**Values**
Learning Rate	6e-5
Input Size	512×384
Flip Probability	0.5
Mean	(0.485, 0.456, 0.406)
Standard Deviation	(0.229, 0.224, 0.225)

Results

Table 4.14: Evaluation Metrics for Experiment 7

Model	**Accuracy (%)**	**mIoU (%)**	**Runtime**
DeepLabV3	78.47	33.36	5 hours
U-Net	75.00	19.90	5 hours
SegFormer	79.87	40.76	8 hours

Evaluation

The freezing–unfreezing approach combined with WarmupPolyLR showed notable improvements compared to previous experiments. SegFormer achieved the highest mIoU of 40.76% and accuracy of 79.87% across all experiments beating even its strong performance in Experiment 5 of 38.22%. DeepLabV3 also recorded its best mIoU of 33.36% so far, while U-Net showed a modest performance of 19.90% mIoU. The loss curves in Figure 4.13 shows that freezing helped stabilise early training, with models avoiding the sharp changes

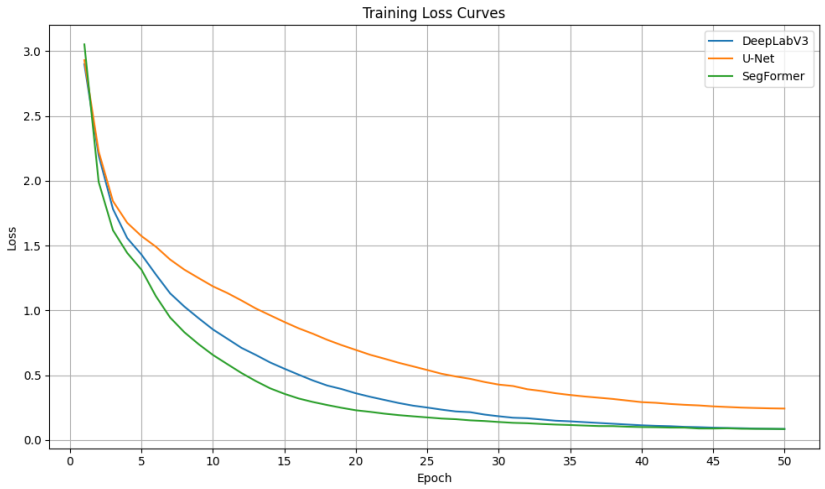

Figure 4.13: Loss progression for DeepLabV3, U-Net, and SegFormer during training for Experiment 7

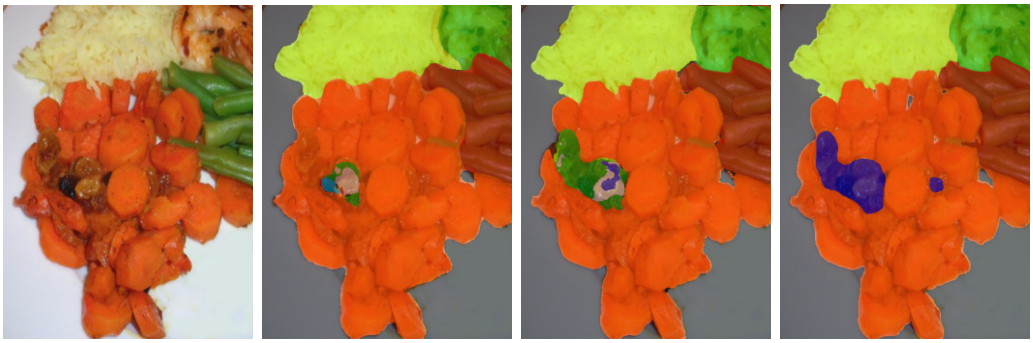

Figure 4.14: Original Image, DeepLabV3, U-Net, SegFormer (Left to Right) for Experiment 7

observed in some earlier experiments. The Figure 4.14 shows that SegFormer was able to capture finer boundaries and reduce misclassifications of small food items. Compared to Experiment 5, where only SegFormer with WarmupPolyLR alone, this approach of combining freezing-unfreezing and WarmupPolyLR proved more effective.

4.2.8 Experiment 8

Objectives: The objective of this experiment is to check whether combining Focal Loss and Dice Loss would improve the segmentation performance. Focal Loss is used to address the class imbalance. Both are given equal importance in calculating the loss.

Models: DeepLabV3 (ResNet101), U-Net (ResNet101), SegFormer (B2 variant)
Loss functions: Dice Loss & Focal Loss
Scheduler: WarmupPolyLR scheduler
Optimizer: AdamW with a base learning rate of 6e-5
Batch Size: 8 for training and 4 for validation

Hyper-parameters

Table 4.15: Hyperparameter Configuration for Experiment 8

Hyper-parameter / Transformations	**Values**
Learning Rate	6e-5
Input Size	512×384
Flip Probability	0.5
Mean	(0.485, 0.456, 0.406)
Standard Deviation	(0.229, 0.224, 0.225)

Results

Table 4.16: Evaluation Metrics for Experiment 8

Model	**Accuracy (%)**	**mIoU (%)**	**Runtime**
DeepLabV3	77.73	35.51	8 hours
U-Net	75.21	22.71	7 hours
SegFormer	79.18	41.14	11 hours

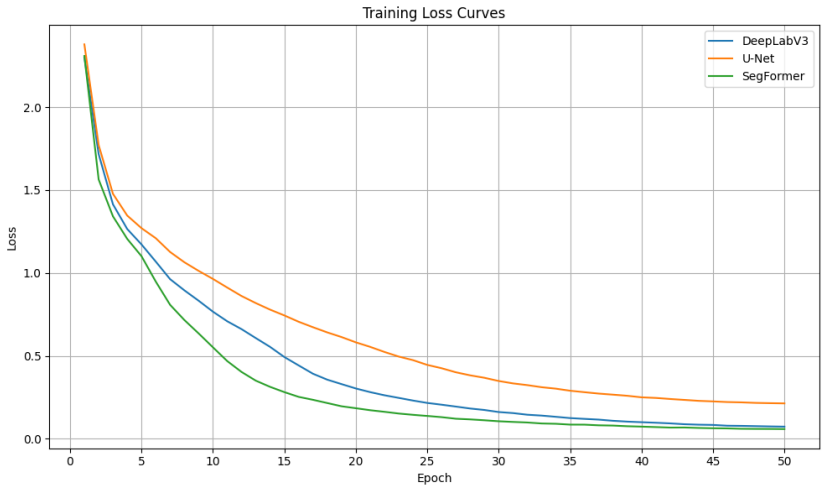

Figure 4.15: Loss progression for DeepLabV3, U-Net, and SegFormer during training for Experiment 8

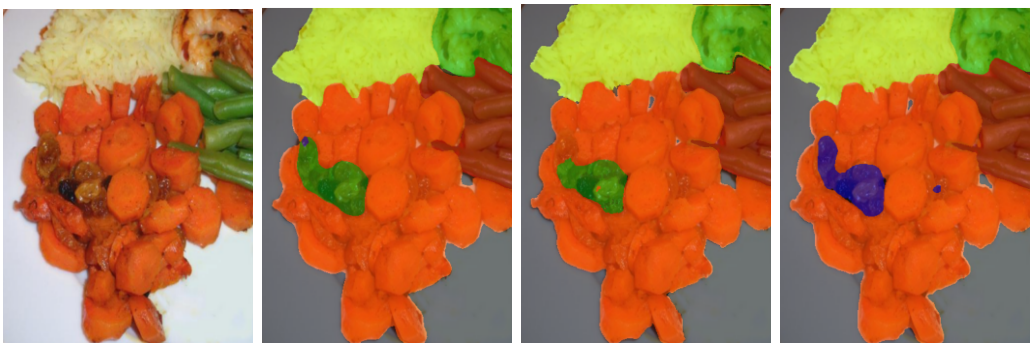

Figure 4.16: Original Image, DeepLabV3, U-Net, SegFormer (Left to Right) for Experiment 8

Evaluation

The results showed a clear performance gain compared to previous experiments. SegFormer achieved its highest mIoU of 41.14% across all experiments surpassing the 40.76% in Experiment 7. DeepLabV3 also recorded its best mIoU so far (35.51%), while U-Net improved to 22.71%, indicating that the combined loss formulation was useful for all models. The loss curves in Figure 4.15 showed steady convergence without instability. The Figure 4.16 showed

that both SegFormer and DeepLabV3 showed improvement around boundary and reduced over-segmentation errors. Compared to Experiment 7, the improvement suggests that combining Focal and Dice Loss effectively helps the WarmupPolyLR schedule and freezing–unfreezing strategy, resulting in more effective models.

4.2.9 Experiment 9

Objectives: The objective of this experiment is to check whether combining Focal Loss and Dice Loss without the freezing/unfreezing approach would improve the segmentation performance. The WarmupPolyLR scheduler and other training settings remained the same as the earlier experiments.

Models: DeepLabV3 (ResNet101), U-Net (ResNet101), SegFormer (B2 variant)
Loss functions: Dice Loss & Focal Loss
Scheduler: WarmupPolyLR scheduler
Optimizer: AdamW with a base learning rate of 6e-5
Batch Size: 8 for training and 4 for validation

Hyper-parameters

Table 4.17: Hyperparameter Configuration for Experiment 9

Hyper-parameter / Transformations	**Values**
Learning Rate	6e-5
Input Size	512×384
Flip Probability	0.5
Mean	(0.485, 0.456, 0.406)
Standard Deviation	(0.229, 0.224, 0.225)

Results

Table 4.18: Evaluation Metrics for Experiment 9

Model	**Accuracy (%)**	**mIoU (%)**	**Runtime**
DeepLabV3	77.47	34.48	12 hours
U-Net	74.89	23.73	9 hours
SegFormer	78.70	40.39	10 hours

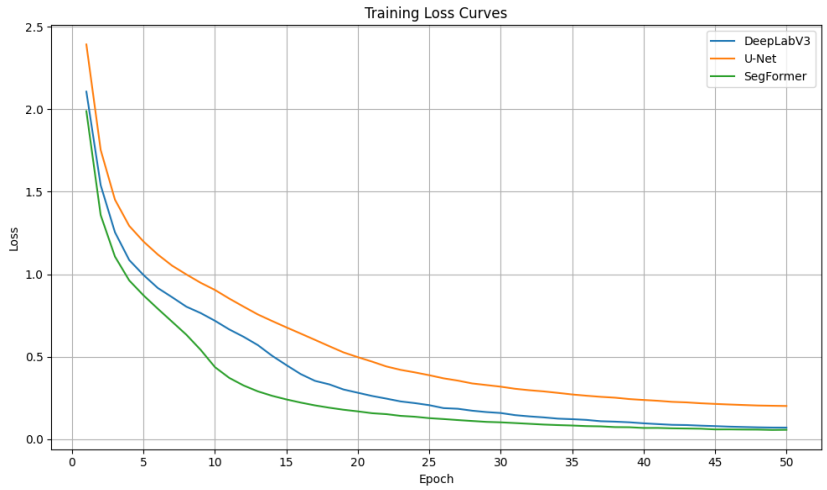

Figure 4.17: Loss progression for DeepLabV3, U-Net, and SegFormer during training for Experiment 9

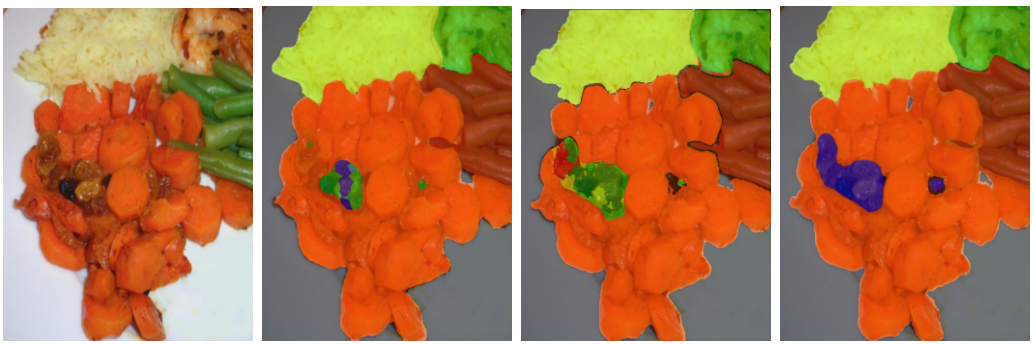

Figure 4.18: Original Image, DeepLabV3, U-Net, SegFormer (Left to Right) for Experiment 9

Evaluation

The table 4.18 shows that SegFormer achieved an mIoU of 40.39%, only marginally below its highest value of 41.14% from Experiment 8 where freezing–unfreezing was applied. DeepLabV3 and U-Net achieved 34.48% and 23.73% mIoU, both showing improvements over their baseline results but not better than their best performance in Experiment 8. This shows that while the combination of loss significantly improves performance compared to the previous experiments, the additional use of freezing–unfreezing (seen in Experiment 7) still provides a small improvement in segmentation accuracy. The loss curves in 4.17 shows smooth and stable convergence suggesting that the combined loss remains effective. The Figure 4.18 shows that sharper object boundaries and better handling of smaller food components compared to Experiments 1–3 while not better than its best performance.

4.2.10 Experiment 10

Objectives: The objective of this experiment is to evaluate the performance of U-Net++ under the same training settings but without DeepLabV3 and SegFormer. This is done to check whether the architectural improvements of U-Net++ over the standard U-Net particularly its nested and dense skip connections could improve segmentation accuracy and mIoU.

Models: U-Net++ (ResNet101)
Loss functions: Dice Loss & Focal Loss
Scheduler: WarmupPolyLR scheduler
Optimizer: AdamW with a base learning rate of 6e-5
Batch Size: 8 for training and 4 for validation

Hyper-parameters

Table 4.19: Hyperparameter Configuration for Experiment 10

Hyper-parameter / Transformations	**Values**
Learning Rate	6e-5
Input Size	512×384
Flip Probability	0.5
Mean	(0.485, 0.456, 0.406)
Standard Deviation	(0.229, 0.224, 0.225)

Results

Table 4.20: Evaluation Metrics for Experiment 10

Model	**Accuracy (%)**	**mIoU (%)**	**Runtime**
U-Net++	75.59	23.68	11 hours

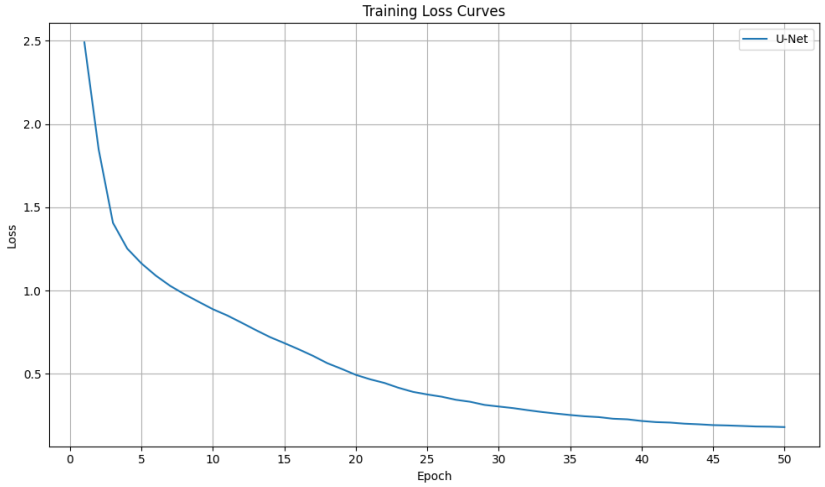

Figure 4.19: Loss progression for U-Net++ during training for Experiment 10

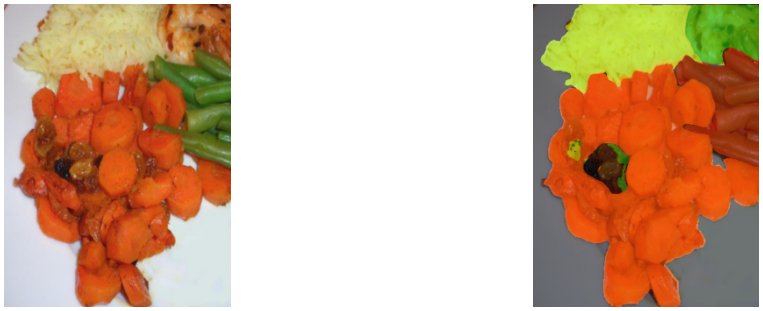

Figure 4.20: Original Image and U-Net++ for Experiment 10

Evaluation

U-Net++ achieved an accuracy of 75.59% and an mIoU of 23.68%, representing a slight improvement in mIoU compared to the U-Net's best result of 23.73% (Experiment 9). While the improvement is measurable, it is not substantial enough to compete with SegFormer or DeepLabV3 in earlier experiments which consistently performed with mIoU values above 32%. The loss curves in Figure 4.19 shows smooth and stable convergence, showing that the model trained effectively without overfitting. The Figure 4.20 shows slightly sharper boundaries and better handling of smaller objects compared to U-Net. This shows that even though U-Net++ has architectural advantages, FoodSeg103 dataset may require more advanced feature extraction as seen in transformer based models.

4.3 Final Evaluation

Table 4.21: Accuracy and mIoU Scores Across 10 Experiments

Exp	**DeepLab Accuracy (%)**	**U-Net Accuracy (%)**	**SegFormer Accuracy (%)**	**DeepLab mIoU (%)**	**U-Net mIoU (%)**	**SegFormer mIoU (%)**
1	**77.84**	75.29	77.74	26.83	16.32	**28.82**
2	77.99	75.00	**78.80**	27.37	16.73	**32.00**
3	**78.14**	74.67	77.91	32.37	21.46	**37.18**
4	–	–	**79.08**	–	–	**36.57**
5	–	–	**78.55**	–	–	**38.22**
6	–	–	**78.86**	–	–	**36.80**
7	78.47	75.00	**79.87**	33.36	19.90	**40.76**
8	77.73	75.21	**79.18**	35.51	22.71	**41.14**
9	77.47	74.89	**78.70**	34.48	23.73	**40.39**
10	–	**75.59**	–	–	23.68	–
Mean	75.09	77.94	**78.74**	20.65	31.65	**36.88**
Median	75.00	77.91	**78.80**	21.46	32.86	**37.18**
Std Dev	0.30	0.35	**0.65**	3.12	3.69	**4.13**

The experimental results in table 4.21 shows the performance of the three models across the 10 experiments. We begun with a baseline experiment setup, then we moved on to the effects of various architectural choices, training strategies, loss functions, learning rate schedulers and finetuning techniques such as freezing and unfreezing the encoder. Overall, SegFormer out-

performed both the models in terms of both accuracy and mIoU in all the 10 experiments especially when trained with WarmupPolyLR and advanced loss functions like Dice and Focal Loss. It achieved its highest accuracy of 79.19% and 41.14% mIoU in experiment 8 where we used Focal and Dice Loss together with the freezing/unfreezing approach. DeepLabV3 showed good accuracy but its mIoU was often lower than SegFormer showing its limitations in capturing overlapping boundaries. U-Net had the lowest performance among the three models showing its inability to perform in complex datasets like FoodSeg103. However, U-Net++ in Experiment 10 showed some slight improvement over U-Net.

In order to improve the evaluation, four statistical tests were conducted to understand the relationship and significance between pixel accuracy and mIoU across the 10 experiments. Both Pearson's correlation (0.897) and Spearman's rank correlation (0.830) showed a strong positive correlation which suggests that higher pixel accuracy generally aligns with higher mIoU. This shows that both metrics support each other in evaluating segmentation quality. Additionally, Mann-Whitney U test was used to compare SegFormer and DeepLabV3, which gave a p-value of 0.016 showing a statistically significant difference in mIoU between the two models. However, the t-test produced a p-value of 0.154, which is above the typical significance threshold of 0.05. This suggests that the difference might not be statistically significant. This discrepancy is likely due to the small number of samples (only 7 - 10 results for each model) which can affect the sensitivity of parametric tests like the t-test. Overall, SegFormer remained the best in both mIoU and accuracy across the experiments supported by strong correlations and non-parametric significance testing, showing its effectiveness in food image segmentation.

Comparison to Existing Models

Table 4.22: mIoU comparison against existing FoodSeg103 models

Model	mIoU	Reference
BEiT v2	49.4	Sinha et al. (2023)
ReLeM-SeTR (LSTM)	43.9	Wu et al. (2021)
SegFormer-B2	41.14	Ours
InternImage-B	41.1	Sinha et al. (2023)
DeepLabV3+	34.2	Wu et al. (2021)

The table 4.22 shows the comparison of our best performing model (SegFormer-B2 with 41.14 mIoU) against the state-of-the-art models. While the best models like BEiT v2 and ReLeM-SeTR (LSTM) achieved high scores, our model came close with InternImage-B with an mIoU of 41.14. Compared to DeepLabV3+ by Wu et al. (2021), SegFormer-B2 performed better. This shows the true power of transformers.

4.4 The Research Questions Revisited

This section revisits the research questions and explains how the experiments answered the questions. ***RQ1****: Comparative Study of DeepLabV3, U-Net and SegFormer for Semantic Segmentation in FoodSeg103.* All the three models were trained and evaluated under the same conditions in all the 10 experiments. The results clearly showed that SegFormer consistently outperformed DeepLabV3 and U-Net in terms of both accuracy and mIoU in all experiments. Specifically, Experiment 8 results as in table 4.16 shows the best performance for SegFormer achieving 79.18% accuracy and 41.14 mIoU making it the best model. ***RQ2****: Which of the following loss functions performs better - CrossEntropy, Focal, and Dice Loss separately or combinations thereof.* Each experiment used different combinations of loss functions. From the experiments it was observed that the combination of both Dice Loss and Focal Loss provided better results. This combination helped the

model handle class imbalance and improve segmentation in complex regions like overlapping. Best performance of highest mIoU of 41.14 was achieved through this combination. ***RQ3****: Does WarmupPolyLR improve training stability compared to standard decay schedules?* WarmupPolyLR which was used in several experiments showed better training stability and performance when compared to the default poly decay. When it is combined with proper loss functions and training settings, it led to more consistent improvements in model performance especially with SegFormer. This suggests that learning rate scheduling played a key role in achieving better results. Overall, the research successfully answered all the research questions by evaluating the three models, loss functions and training strategies. The findings also highlight the strengths of transformer based models like SegFormer and offers some valuable insights for future work in food analysis.

4.5 Cityscapes Revisited

Objectives: The objective of this experiment is the check the generalization capability of the best performing experiment from FoodSeg103 (Experiment 8) on a different domain dataset. The Cityscapes dataset consists of urban street scenes and was chosen to evaluate whether the models would perform just as good as it did in Experiment 8. It was also evaluated to check whether SegFormer would still perform better than DeepLabV3 and U-Net in Cityscapes using the same training configurations.

Models: DeepLabV3 (ResNet101), U-Net (ResNet101), SegFormer (B2 variant)
Loss functions: Dice Loss & Focal Loss
Scheduler: WarmupPolyLR scheduler
Optimizer: AdamW with a base learning rate of 6e-5
Batch Size: 8 for training and 4 for validation

Hyper-parameters

Table 4.23: Hyperparameter Configuration for Cityscapes

Hyper-parameter / Transformations	**Values**
Learning Rate	6e-5
Input Size	512×512
Flip Probability	0.5
Mean	(0.485, 0.456, 0.406)
Standard Deviation	(0.229, 0.224, 0.225)

Results

Table 4.24: Evaluation Metrics for for Cityscapes

Model	**Accuracy (%)**	**mIoU (%)**	**Runtime**
DeepLabV3	90.07	46.00	6 hours
U-Net	78.64	29.97	4 hours
SegFormer	91.35	50.84	5 hours

Evaluation

The SegFormer model showed the strongest performance in Cityscapes dataset achieving the highest accuracy (91.35%) and mIoU (50.84%) while the training time took only 5 hours. This shows that the SegFormer which is a transformer based model not only performed well on FoodSeg103 but also generalized well on Cityscapes. It was followed by DeepLabV3 with 90.07% accuracy and 46.00% mIoU although it required an additional one hour of runtime. While U-Net maintained a good accuracy of 78.64% it struggled with overlap quality which is shown in its low mIoU of 29.97%. The loss curves of each of the models can be seen in Figure 4.21. All the three models

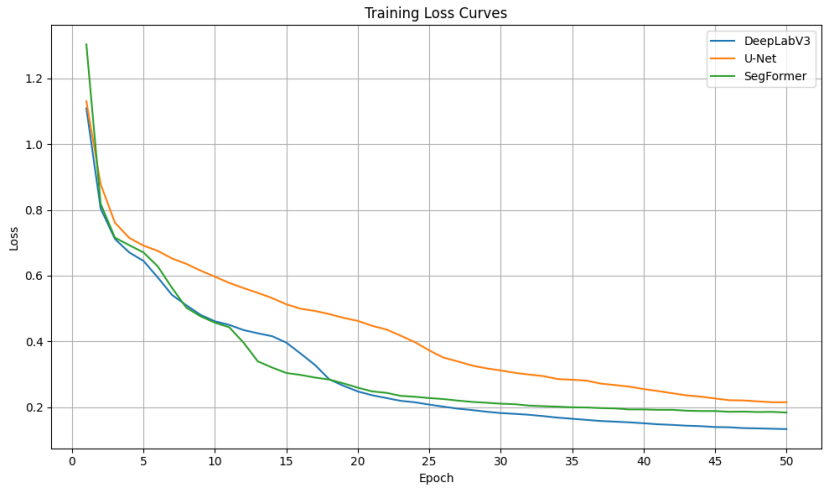

Figure 4.21: Loss progression for DeepLabV3, U-Net, and SegFormer during training for Cityscapes

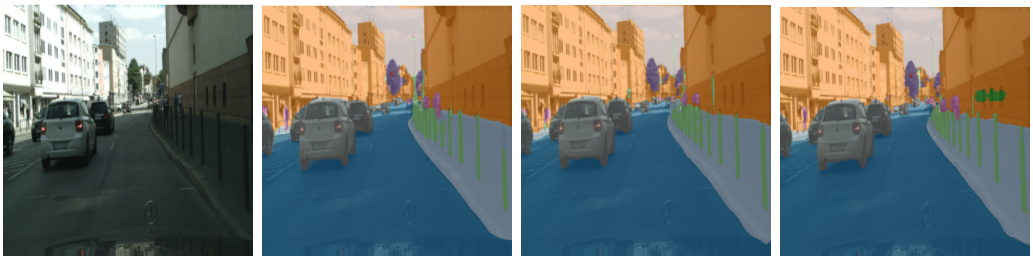

Figure 4.22: Original Image, DeepLabV3, U-Net, SegFormer (Left to Right) for Cityscapes

showed a steep decline and then stabilized. The Figure 4.22 shows that the SegFormer produced cleaner boundaries.

This experiment shows that the SegFormer model is well suited for semantic segmentation tasks achieving high accuracy and generalization. Its efficiency in both performance and training time shows its potential for real world applications.

Comparison to Existing Models

Table 4.25: mIoU comparison against existing Cityscapes models

Model	mIoU	Reference
SegFormer-B5	84.0	Xie et al. (2021)
SegFormer-B2	**50.84**	**Ours**
SETR	48.6	Xie et al. (2021)
SegFormer-B2	46.5	Xie et al. (2021)
DeeplabV3+/R101	44.1	Xie et al. (2021)
SegFormer-B0	37.4	Xie et al. (2021)
FCN-R50	36.1	Xie et al. (2021)

The SegFormer-B2 model (Our model) achieved a mIoU of 50.84% on the Cityscapes dataset, outperforming the original SegFormer-B2 (46.5%) and other models like SETR (48.6%) and DeepLabV3+ (44.1%). Although it did not reach the performance of the larger SegFormer-B5 (84.0%), it still showed strong results considering its light architecture. This shows that with the right training strategies, even smaller models can deliver good performance in real-world segmentation tasks.

4.6 Threats to Validity

The FoodSeg103 dataset which is large and diverse suffers from class imbalance, overlapping food items and long tail distribution. These issues may cause models to perform better on frequent classes while under performing on rare ones. Collecting more balanced datasets, applying advanced sampling strategies could result in improvement in class performance. Limited GPU memory and training time restricted the scale of experimentation. Some models (e.g., SegFormer B4 or B5) could not be trained due to resource limits, potentially missing stronger performance. Using more powerful hardware

would allow experimentation with larger models, longer training runs and higher input resolutions for improved results. Only three models DeepLabV3, U-Net and SegFormer were chosen for comparison. Although they represent different architectures such as CNN, encoder-decoder and transformer, other state of the art models were not included, which might restrict generalization. Evaluating with a wider range of architectures would back the conclusions and provide a broader benchmark for food segmentation.

Addressing these issues in future work would strengthen the reliability of the findings. Such improvements could provide a more comprehensive understanding of semantic segmentation in the food domain

Chapter 5

Discussion and Conclusions

This dissertation explored the performance of the state-of-the-art semantic segmentation models including DeepLabV3, U-Net and SegFormer on the domain specific FoodSeg103 dataset. The models were evaluated across 10 experiments to understand their strengths and weaknesses in this food segmentation task. The results from the 10 experiments showed that SegFormer which is a transformed based model performed consistently and outperformed its counterparts DeepLabV3 and U-Net (Convolution based models) in capturing fine details and complex boundaries achieving an highest accuracy of 79.18% and mIoU of 41.14% in Experiment 8. DeepLabV3 also showed some good performance in multiple experiments. U-Net, on the other hand only performed well in terms of accuracy. Across the all 10 experiments, SegFormer performed the best consistently. The architectures explored in this dissertation especially SegFormer were not part of my masters curriculum, so I had to put extra effort in gaining an in-depth understanding of them. It involved studying research papers, implementation details and practical challenges such as training large scale models.

One of the major challenges was the high computational demand of each of these models. Each experiment required significant runtime ranging from

several hours to days. The limited access to university GPU resources added to the complexity which required good planning and adjustments of batch sizes, resolutions of the images and training schedules based on the availability of GPU. Despite these challenges, every experiment helped to gain a deeper understanding of how these architectures impact segmentation performance especially in a complex dataset like FoodSeg103.

This has very strong potential for real-time applications such as diet monitoring, calorie estimation, nutrition tracking. By accurately identifying food items and ingredients in a plate, it can help users or dietitians better understand the nutritional content. These models could be integrated into mobile apps but in order for this to work in real time these models have to be lightweight, fast and optimized for mobile devices. SegFormer B0, which is the lightest model could be used for mobile devices. In future, the models can be trained with more diverse and larger datasets to generalize better across diverse foods and representing more real world. Another improvement could be adding object detection or instance segmentation for handling overlapping dishes. Overall, this dissertation provides the groundwork for building better semantic segmentation models that could help people in maintaining healthier lifestyles and improve their diet.

Appendices

I have saved all the experiments and the results in the GitHub Link and all the trained models are stored in a public Google Drive link due to the size of the models.

GitHub Link: Click here to redirect to Experiment code base

Google Drive Link: Click here to redirect to Trained Models links

References

Badrinarayanan, Vijay, Alex Kendall, and Roberto Cipolla (2017). "SegNet: A deep convolutional encoder-decoder architecture for image segmentation". In: *IEEE transactions on pattern analysis and machine intelligence* 39.12, pp. 2481–2495. DOI: 10.1109/TPAMI.2016.2644615 (cit. on pp. 24, 26).

Chen, Guangyong et al. (2019). "Rethinking the Usage of Batch Normalization and Dropout in the Training of Deep Neural Networks". In: *arXiv preprint arXiv:1905.05928*. URL: https://arxiv.org/abs/1905.05928 (cit. on p. 20).

Chen, Liang Chieh et al. (2018). "DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs". In: *IEEE transactions on pattern analysis and machine intelligence* 40.4, pp. 834–848. URL: https://arxiv.org/abs/1606.00915 (cit. on p. 24).

Chen, Liang-Chieh et al. (2017). "Rethinking Atrous Convolution for Semantic Image Segmentation". In: *arXiv preprint arXiv:1706.05587*. URL: https://arxiv.org/abs/1706.05587 (cit. on pp. 2, 4).

Ciocca, Gianluigi, Paolo Napoletano, and Raimondo Schettini (2017). "Food Recognition: A New Dataset, Experiments, and Results". In: *IEEE Journal of Biomedical and Health Informatics* 21.3, pp. 588–598. DOI: 10.1109/JBHI.2016.2636441. URL: https://ieeexplore.ieee.org/document/7776769 (cit. on p. 33).

Diary, Data Scientist's (2021). *Implementation of Dice Loss - Vision (PyTorch)*. https://medium.com/data-scientists-diary/implementation-of-dice-loss-vision-pytorch-7eef1e438f68 (cit. on p. 2).

Dubey, Shiv Ram, Satish Kumar Singh, and Bidyut Baran Chaudhuri (2022). "Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark". In: *Neurocomputing* 503, pp. 92–108. DOI: 10.1016/j.neucom.2022.06.111. URL: https://doi.org/10.1016/j.neucom.2022.06.111 (cit. on p. 18).

Fukushima, Kunihiko (1980). "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological Cybernetics* 36.4, pp. 193–202. DOI: 10.1007/BF00344251. URL: https://doi.org/10.1007/BF00344251 (cit. on p. 15).

GeeksforGeeks (2021). *Backpropagation in Neural Network*. https://www.geeksforgeeks.org/backpropagation-in-neural-network/ (cit. on p. 14).

GeeksforGeeks contributors (2020). *CNN — Introduction to Pooling Layer*. https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/ (cit. on p. 18).

Géron, Aurélien (2023). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 3rd. O'Reilly Media. ISBN: 9781098125974 (cit. on pp. 8–10, 17, 19, 22).

Hammoudeh, Ahmad (2018). *A Concise Introduction to Reinforcement Learning*. https://www.researchgate.net/publication/323657509_A_Concise_Introduction_to_Reinforcement_Learning. DOI: 10.13140/RG.2.2.31027.53285 (cit. on p. 9).

He, Kaiming et al. (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern*

recognition, pp. 770–778. DOI: 10.1109/CVPR.2016.90. URL: https://arxiv.org/abs/1512.03385 (cit. on p. 16).

Hinkelmann, Knut (2018). *Neural Networks*. University of Applied Sciences Northwestern Switzerland, p. 7. URL: http://didattica.cs.unicam.it/old/lib/exe/fetch.php?media=didattica:magistrale:kebi:ay_1718:ke-11_neural_networks.pdf (cit. on p. 18).

Hu, Jie, Li Shen, and Gang Sun (2018). "Squeeze-and-excitation networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7132–7141. URL: https://arxiv.org/abs/1709.01507 (cit. on p. 27).

Huynh, Nghi (2023). *Understanding Evaluation Metrics in Medical Image Segmentation*. https://medium.com/@nghihuynh_37300/understanding-evaluation-metrics-in-medical-image-segmentation-d289a373a3f (cit. on pp. 3, 5).

Ioffe, Sergey and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *arXiv preprint arXiv:1502.03167*. URL: https://arxiv.org/abs/1502.03167 (cit. on p. 21).

Krähenbühl, Philipp and Vladlen Koltun (2011). "Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 24, pp. 109–117. URL: https://arxiv.org/abs/1210.5644 (cit. on p. 23).

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "ImageNet classification with deep convolutional neural networks". In: *Advances in Neural Information Processing Systems*. Vol. 25, pp. 1097–1105. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf (cit. on p. 16).

LeCun, Yann, Bernhard Boser, et al. (1989). "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4, pp. 541–

551. URL: https://galileo-unbound.blog/wp-content/uploads/2025/02/lecun.neco_.1989.1.4.541.pdf (cit. on p. 14).

LeCun, Yann, Léon Bottou, et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324. DOI: 10.1109/5.726791. URL: https://doi.org/10.1109/5.726791 (cit. on p. 15).

Long, Jonathan, Evan Shelhamer, and Trevor Darrell (2015). "Fully Convolutional Networks for Semantic Segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 3431–3440. DOI: 10.1109/CVPR.2015.7298965. URL: https://ieeexplore.ieee.org/document/7298965 (cit. on pp. 1, 23, 25).

McCarthy, John (1955). *What is Artificial Intelligence?* Available at http://jmc.stanford.edu/articles/whatisai/whatisai.pdf (cit. on p. 7).

Mitchell, Tom M. (1997). *Machine Learning*. McGraw-Hill. ISBN: 9780070428072 (cit. on p. 7).

Oye, Emma and Rebekah Lucas (2024). *Convolutional Neural Networks (CNNs)*. URL: https://www.researchgate.net/publication/387278938_Convolutional_Neural_Networks_CNNs (cit. on p. 16).

Pedro, Juan (2023). "Detailed Explanation of YOLOv8 Architecture". In: *Medium Blog*. URL: https://medium.com/@abinpedro.bc22/detailed-explanation-of-yolov8-architecture-part-1-6da9296b954e (cit. on p. 29).

Peerthum, Yashna and Mark Stamp (2023). "An Empirical Analysis of the Shift and Scale Parameters in BatchNorm". In: *arXiv preprint arXiv:2303.12818*. URL: https://arxiv.org/abs/2303.12818 (cit. on p. 21).

Phung, Van Hiep and Eun Joo Rhee (2019). "A High-Accuracy Model Average Ensemble of Convolutional Neural Networks for Classification of Cloud Image Patches on Small Datasets". In: *Applied Sciences* 9.21, p. 4500. DOI: 10.3390/app9214500. URL: https://www.mdpi.com/2076-3417/9/21/4500 (cit. on p. 15).

Pires, Paulo Botelho, José Duarte Santos, and Inês Veiga Pereira (2023). "Artificial Neural Networks: History and State of the Art". In: *Encyclopedia of Information Science and Technology, Sixth Edition*. IGI Global. DOI: 10.4018/978-1-6684-7366-5.ch037. URL: https://www.researchgate.net/publication/374723059_Artificial_Neural_Networks_History_and_State_of_the_Art (cit. on p. 10).

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer, pp. 234–241. URL: https://arxiv.org/abs/1505.04597 (cit. on pp. 2, 4, 23–25).

Russell, Stuart and Peter Norvig (2021). *Artificial Intelligence: A Modern Approach*. 4th. Pearson. ISBN: 9780134610993 (cit. on p. 8).

Simonyan, Karen and Andrew Zisserman (2014). "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556*. URL: https://arxiv.org/abs/1409.1556 (cit. on p. 16).

Sinha, Grant et al. (2023). "Transferring Knowledge for Food Image Segmentation using Transformers and Convolutions". In: *arXiv preprint arXiv:2306.09203*. URL: https://arxiv.org/abs/2306.09203 (cit. on pp. 1, 32, 33, 36, 53, 76).

Srivastava, Nitish et al. (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56, pp. 1929–1958. URL: https://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf (cit. on pp. 20, 21).

Stabinger, Sebastian and Antonio Rodriguez-Sanchez (2016). "25 Years of CNNs: Can We Compare to Human Abstraction Capabilities?" In: *Artificial Neural Networks and Machine Learning – ICANN 2016*. Ed. by A. Villa, P. Masulli, and A.J. Pons Rivero. Vol. 9887. Lecture Notes in Computer Science. Springer, pp. 380–387. DOI: 10.1007/978-3-319-44781-0_45. URL: https://www.researchgate.net/publication/305702676_

25_Years_of_CNNs_Can_We_Compare_to_Human_Abstraction_Capabilities (cit. on p. 15).

Szegedy, Christian et al. (2015). "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9. URL: https://arxiv.org/abs/1409.4842 (cit. on p. 16).

Vakalopoulou, Maria et al. (2023). "Deep Learning: Basics and Convolutional Neural Networks (CNNs)". In: *Machine Learning for Brain Disorders*. Chapter available under CC BY 4.0 license. Springer, pp. 77–115. DOI: 10.1007/978-1-0716-3195-9_3. URL: https://doi.org/10.1007/978-1-0716-3195-9_3 (cit. on pp. 12, 17).

Wang, Xiaolong et al. (2018). "Non-local Neural Networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. URL: https://arxiv.org/abs/1711.07971 (cit. on p. 27).

Werbos, Paul J (1990). "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE*. Vol. 78. 10. IEEE, pp. 1550–1560 (cit. on p. 13).

Wizard, Vision (2020). *Understanding Focal Loss: A Quick Read*. https://medium.com/visionwizard/understanding-focal-loss-a-quick-read-b914422913e7 (cit. on p. 2).

Wu, Xiongwei et al. (2021). *A Large-Scale Benchmark for Food Image Segmentation*. https://arxiv.org/abs/2105.05409. DOI: 10.1145/3474085.3475201 (cit. on pp. 1, 2, 4, 76).

Xie, Enze et al. (2021). "SegFormer: Simple and efficient design for semantic segmentation with transformers". In: *Advances in Neural Information Processing Systems* 34, pp. 12077–12090. URL: https://arxiv.org/abs/2105.15203 (cit. on pp. 2–4, 24, 28, 42, 80).

Xu, Bing et al. (2015). "Empirical Evaluation of Rectified Activations in Convolutional Network". In: *arXiv preprint arXiv:1505.00853*. URL: https://arxiv.org/abs/1505.00853 (cit. on pp. 18, 19).